Bioinformatics

UNIVERSITY OF
CAMBRIDGE

Computer Laboratory

Computer Science Tripos Part II

Pietro Lio`

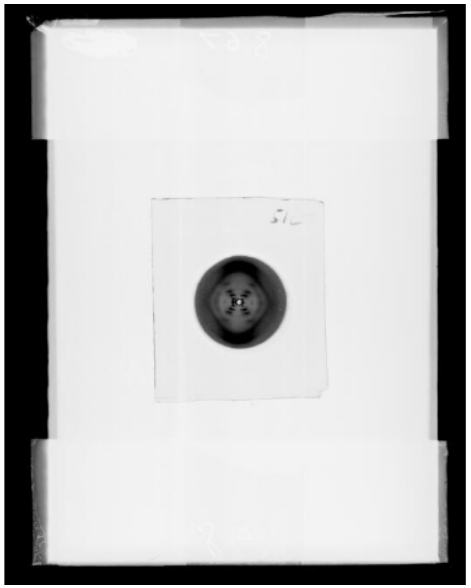
BioInformatics 2019-2020

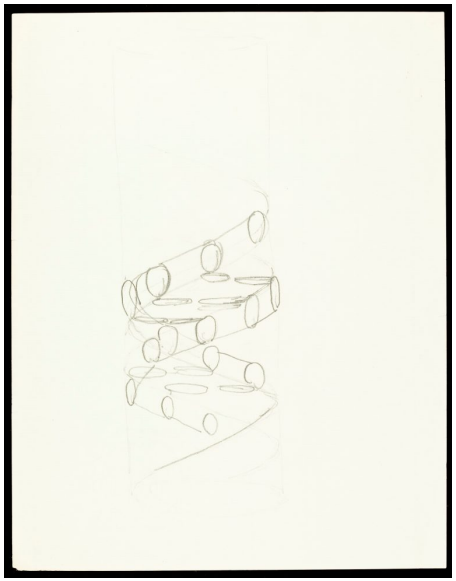
At the core of life there is a sort of programming; the DNA sequence contains both the code for the structure of the 3d parts (usually proteins, programmed self assembly process) and the code that represents the manual of instructions -how much, where, when a certain part should be produced.

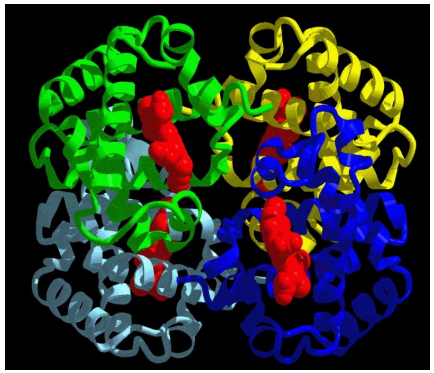
Bioinformatics is about algorithms and machine learning methods to identify the coding elements in the DNA sequences and characterise the parts.

Both DNA sequence and protein structure research have adopted good abstractions: 'DNA-as-string' (a mathematical string is a finite sequence of symbols) and 'a protein-as-a three-dimensional-labelled-graph'.

Models of DNA and proteins





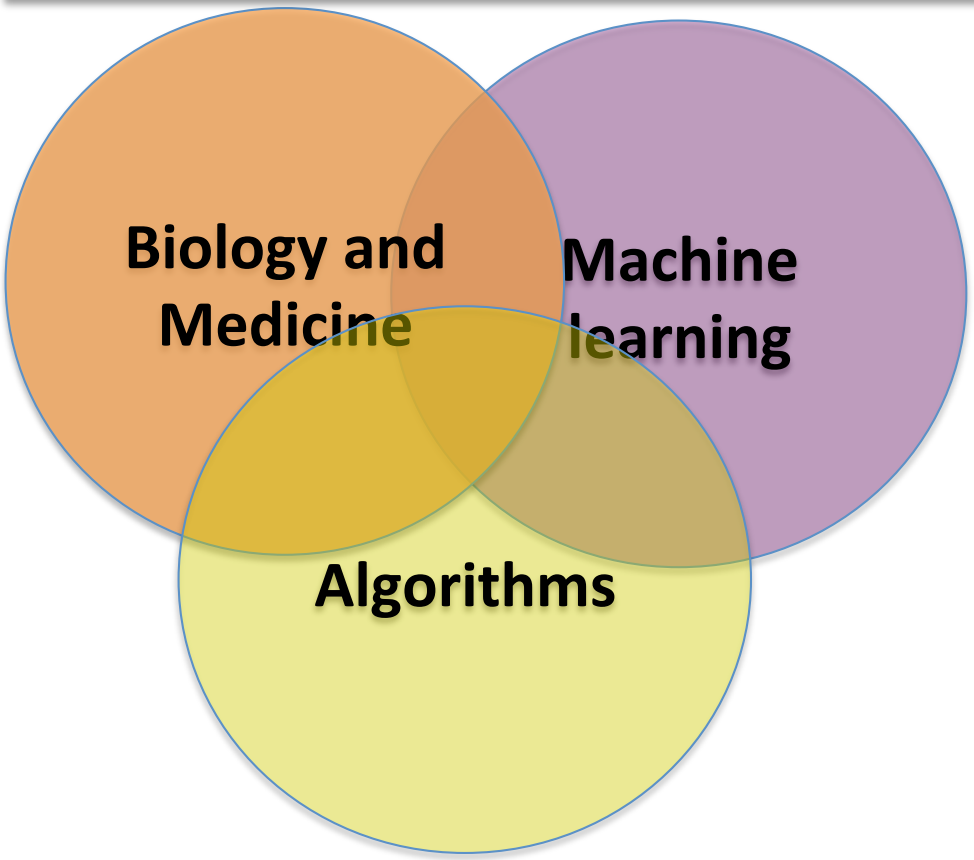


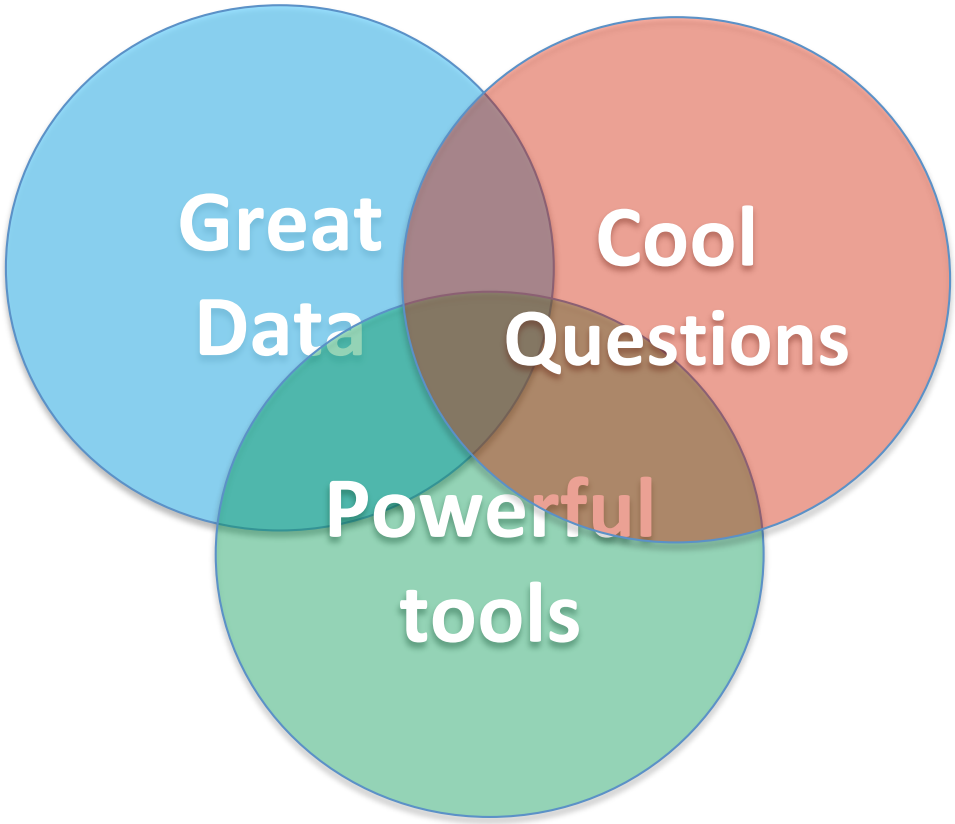
5-CCTGAGCCAACTATTGATGAA-3
3-GGACTCGGTTGATAACTACTT-5

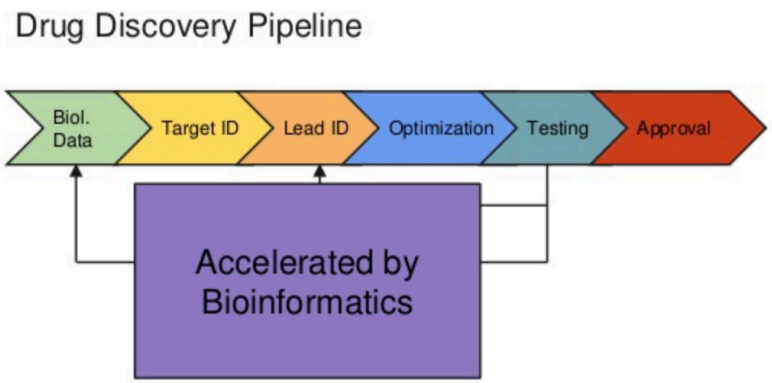
ABSTRACTIONS:
DNA AS A STRING,
PROTEIN AS A LABELLED GRAPH
DNA AND PROTEINS AS NETWORKS

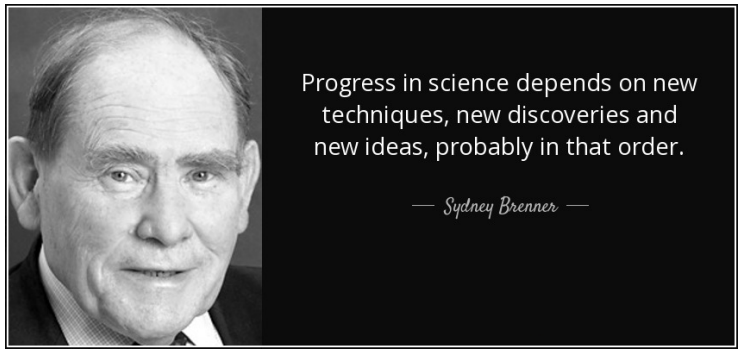
sources: Photograph 51', March 1953, by Rosalind Franklin; Pencil sketch of the DNA double helix by Francis Crick; Replica of Crick and Watson's 1953 DNA Double Helix Model, https://blog.sciencemuseum.org.uk/why-the-double-helix-is-still-relevant/

What is BioInformatics

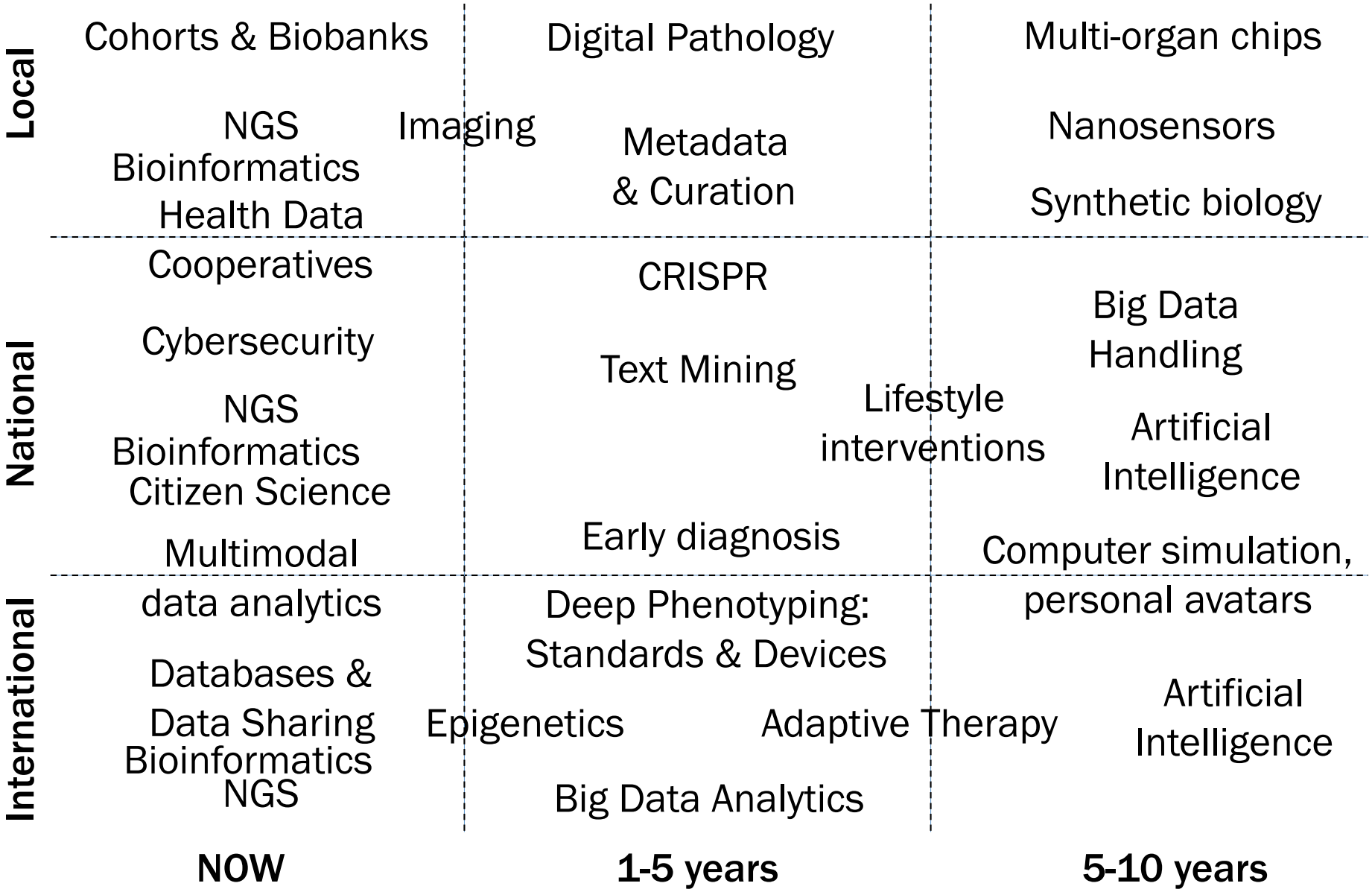






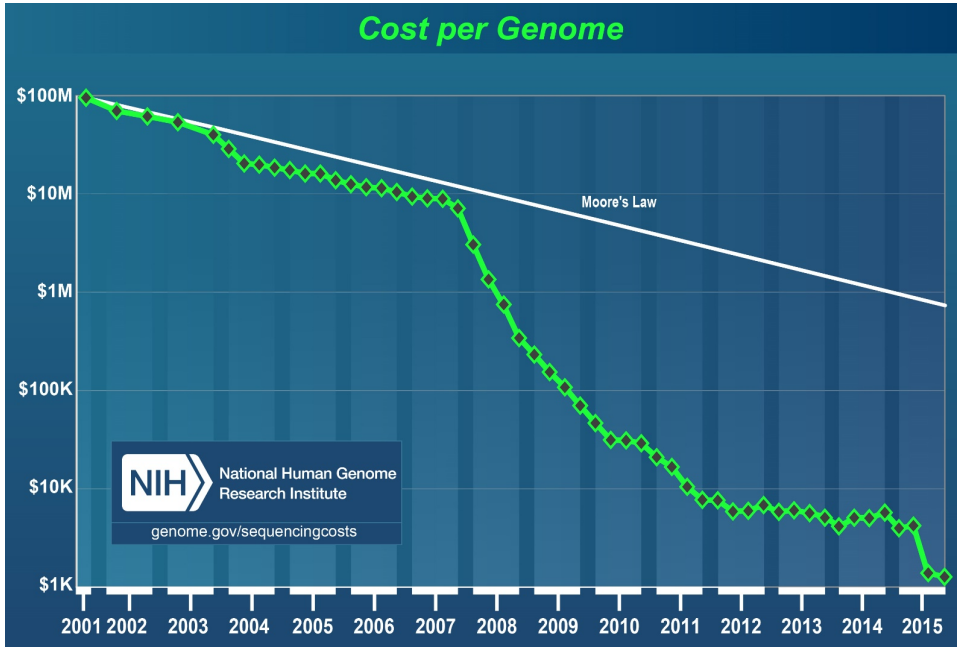


Bioinformatics: a central position in medicine



NGS= next generation sequenging

DNA for genomic diagnostics



Impact on Personalised Medicine

- **Cancer**: Disease stratification based on driver mutations
- **Rare diseases**: Most patients now receive a genetic diagnosis
- **Drugs**: Patient-specific prediction of efficacy and side effects

https://www.genome.gov/sequencingcosts

High-performance computing

1979

today

Who has a computer?

- 1960s: Major research institutes
- 1970s: University departments
- 1980s: Companies and schools
- 2019: Almost everybody & always

Genome sequencing

2006

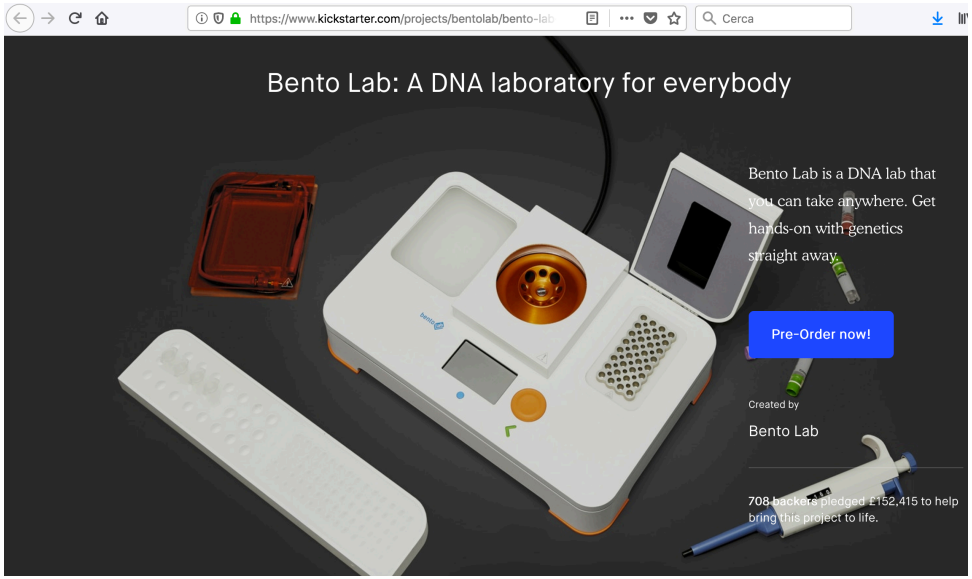
today

Whose genome has been sequenced?

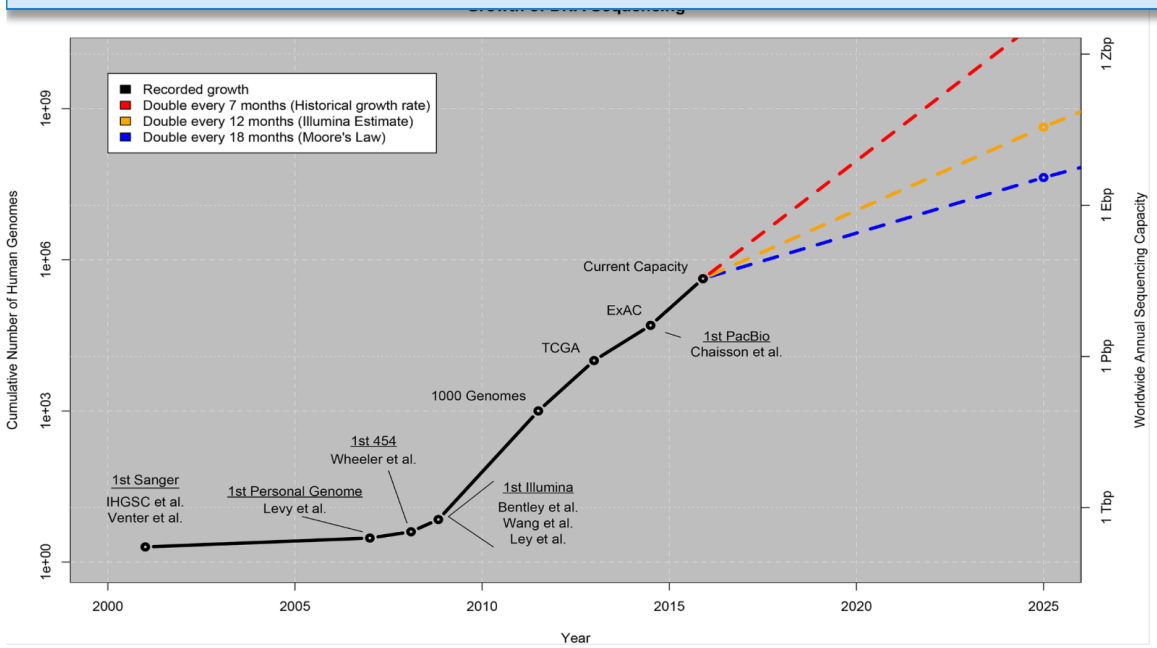
- 1996: First bacterium (E. coli)
- 2001: Human reference genome
- 2007: First personal genomes
- 2019: Millions personal genomes

Garage genomics

Oxford nanopore



DNA is big data



Data Repository: http://www.ebi.ac.uk; http://www.ncbi.nlm.nih.gov/ ; http://genome.ucsc.edu/ www.ensembl.org

Data Phase	Astronomy	Twitter	YouTube	Genomics
Acquisition	25 zetta-bytes/year	0.5–15 billion tweets/year	500–900 million hours/year	1 zetta-bases/year
Storage	1 EB/year	1–17 PB/year	1–2 EB/year	2–40 EB/year
Analysis	In situ data reduction	Topic and sentiment mining	Limited requirements	Heterogeneous data and analysis
	Real-time processing	Metadata analysis		Variant calling, ~2 trillion central processing unit (CPU) hours
	Massive volumes			All-pairs genome alignments, ~10,000 trillion CPU hours
Distribution	Dedicated lines from antennae to server (600 TB/s)	Small units of distribution	Major component of modern user's bandwidth (10 MB/s)	Many small (10 MB/s) and fewer massive (10 TB/s) data movement

doi:10.1371/journal.pbio.1002195.t001

How much DNA in the body and in the biosphere

Each base pair take a couple of bits to encode (because you have to choose between G, A, T and C.

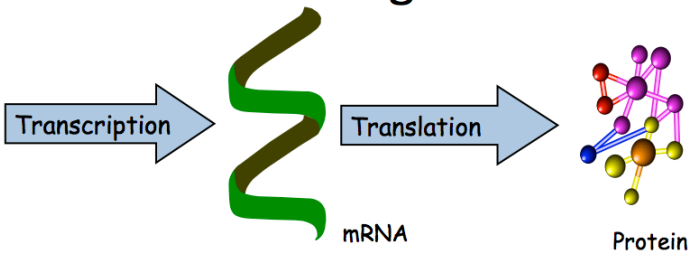
You have 46 chromosomes in each (autosomal) cell (3 billion base pairs, 2 meters long, 2nm thick, folded into a 6μm ball). If you teased out those 46 strands and placed them end to end they'd be about 2 metres long - but that's just one cell. Every time a cell replicates it has to copy 2 meters of DNA reliably.

As there are about 3.7×10^{13} cells in the human body (and hence 1.7×10^{15} chromosomes or strands), your entire DNA would stretch about 7.4×10^{10} km or fifty thousand million miles (133 Astronomical Units long) — DNA in human population 20 million light years long (the Andromeda Galaxy is 2.5 Million light years).

Lower bound on the total information content in the biosphere: 5.3×10^{31} ($\pm 3.6 \times$ 1031) megabases (Mb) of DNA. Taking the rate of DNA transcription as an analogy for processing speed, they further estimated Earth's computational power: 10^{15} yottaNOPS (1024 Nucleotide Operations Per Seconds).

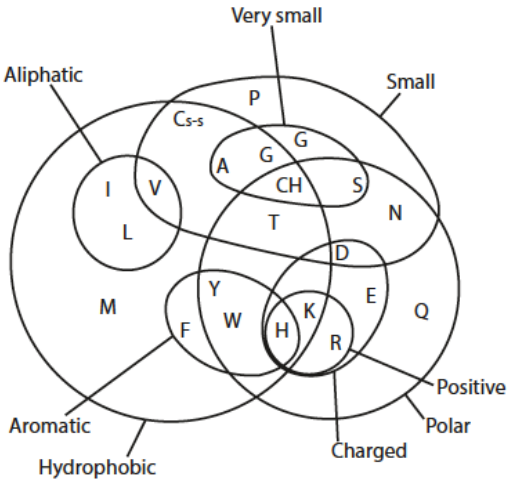
Genetic Code

Central Dogma



Gene

cells express **different** subset of the genes
In different tissues and under different conditions



CCTGAGCCAACTATTGATGAA
GCACTCGGTTGATAACTACTT

DNA

transcription

CCUGAGCCAACUAUUGAUGAA

mRNA

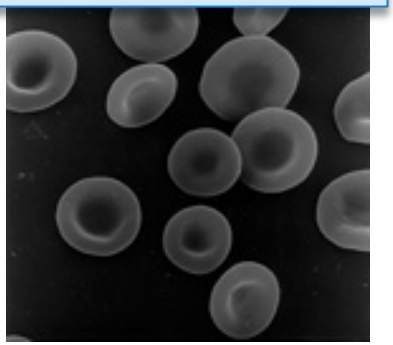
translation

PEPTIDE

Protein

1st position (5' end)	2nd position U	2nd position C	2nd position A	2nd position G	3rd position (3' end)
U	Phe	Ser	Tyr	Cys	U
	Phe	Ser	Tyr	Cys	C
	Leu	Ser	STOP	STOP	A
	Leu	Ser	STOP	Trp	G
C	Leu	Pro	His	Arg	U
	Leu	Pro	His	Arg	C
	Leu	Pro	Gln	Arg	A
	Leu	Pro	Gln	Arg	G
A	Ile	Thr	Asn	Ser	U
	Ile	Thr	Asn	Ser	C
	Ile	Thr	Lys	Arg	A
	Met	Thr	Lys	Arg	G
G	Val	Ala	Asp	Gly	U
	Val	Ala	Asp	Gly	C
	Val	Ala	Glu	Gly	A
	Val	Ala	Glu	Gly	G

Healthy Individual



sequences
in Fasta format

>gi|28302128|ref|NM_000518.4| Homo sapiens hemoglobin, beta (HBB), mRNA

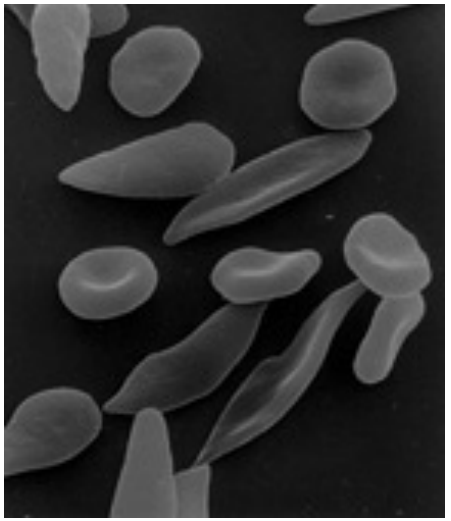
ACATTTGCTTCTGACACAACTGTGTTCACTAGCAACCTCAAACAGACACCATGGTGCATCTGACTCCTGA

GGAGAAGTCTGCCGTTACTGCCCTGTGGGGCAAGGTGAACGTGGATGAAGTTGGTGGTGAGGCCCTGGGC
AGGCTGCTGGTGGTCTACCCTTGGACCCAGAGGTTCTTTGAGTCCTTTGGGGATCTGTCCACTCCTGATG
CTGTTATGGGCAACCCTAAGGTGAAGGCTCATGGCAAGAAAGTGCTCGGTGCCTTTAGTGATGGCCTGGC
TCACCTGGACAACCTCAAGGGCACCTTTGCCACACTGAGTGAGCTGCACTGTGACAAGCTGCACGTGGAT
CCTGAGAACTTCAGGCTCCTGGGCAACGTGCTGGTCTGTGTGCTGGCCCATCACTTTGGCAAAGAATTCA
CCCCACCAGTGCAGGCTGCCTATCAGAAAGTGGTGGCTGGTGTGGCTAATGCCCTGGCCCACAAGTATCA
CTAAGCTCGCTTTCTTGCTGTCCAATTTCTATTAAAGGTTCCTTTGTTCCCTAAGTCCAACTACTAAACT
GGGGGATATTATGAAGGGCCTTGAGCATCTGGATTCTGCCTAATAAAAAACATTTATTTTCATTGC

>gi|4504349|ref|NP_000509.1| beta globin [Homo sapiens]

MVHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWTQRFFESFGDLSTPDAVMGNPKVKAHGKKVLG
AFSDGLAHLDNLKGTFATLSELHCDKLHVDPENFRLLGNVLVCVLAHHFGKEFTPPVQAAYQKVVAGVAN
ALAHKYH

Individual with Sickle Cell Anemia



>gi|28302128|ref|NM_000518.4| Homo sapiens hemoglobin, beta (HBB), mRNA

ACATTTGCTTCTGACACAACTGTGTTCACTAGCAACCTCAAACAGACACCATGGTGCATCTGACTCCTGA

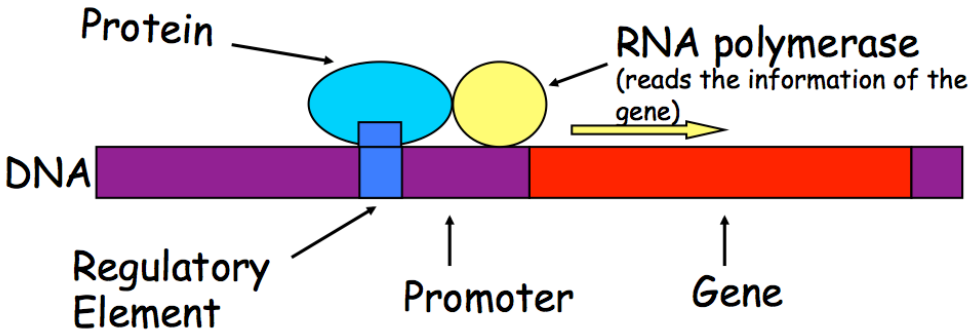
GGTGAAGTCTGCCGTTACTGCCCTGTGGGGCAAGGTGAACGTGGATGAAGTTGGTGGTGAGGCCCTGGGC
AGGCTGCTGGTGGTCTACCCTTGGACCCAGAGGTTCTTTGAGTCCTTTGGGGATCTGTCCACTCCTGATG
CTGTTATGGGCAACCCTAAGGTGAAGGCTCATGGCAAGAAAGTGCTCGGTGCCTTTAGTGATGGCCTGGC
TCACCTGGACAACCTCAAGGGCACCTTTGCCACACTGAGTGAGCTGCACTGTGACAAGCTGCACGTGGAT
CCTGAGAACTTCAGGCTCCTGGGCAACGTGCTGGTCTGTGTGCTGGCCCATCACTTTGGCAAAGAATTCA
CCCCACCAGTGCAGGCTGCCTATCAGAAAGTGGTGGCTGGTGTGGCTAATGCCCTGGCCCACAAGTATCA
CTAAGCTCGCTTTCTTGCTGTCCAATTTCTATTAAAGGTTCCTTTGTTCCCTAAGTCCAACTACTAAACT
GGGGGATATTATGAAGGGCCTTGAGCATCTGGATTCTGCCTAATAAAAAACATTTATTTTCATTGC

>gi|4504349|ref|NP_000509.1| beta globin [Homo sapiens]

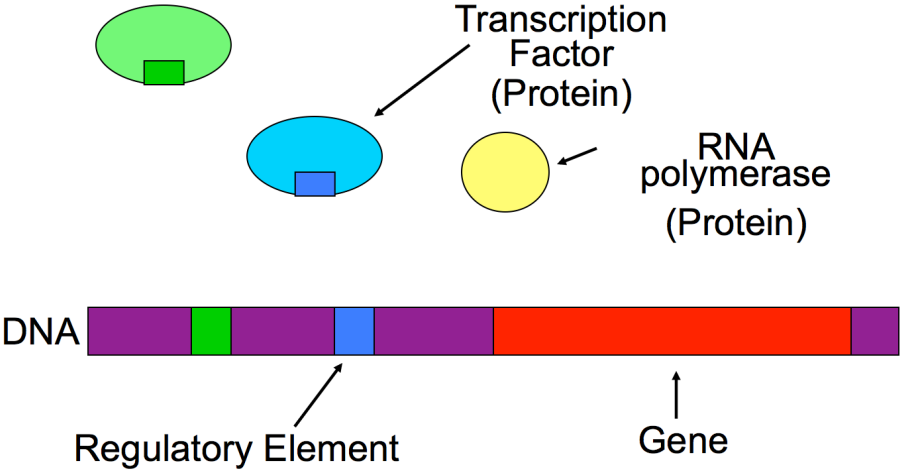
MVHLTPVEKSAVTALWGKVNVDEVGGEALGRLLVVYPWTQRFFESFGDLSTPDAVMGNPKVKAHGKKVLG
AFSDGLAHLDNLKGTFATLSELHCDKLHVDPENFRLLGNVLVCVLAHHFGKEFTPPVQAAYQKVVAGVAN
ALAHKYH

• Gene and protein interactions as graphs

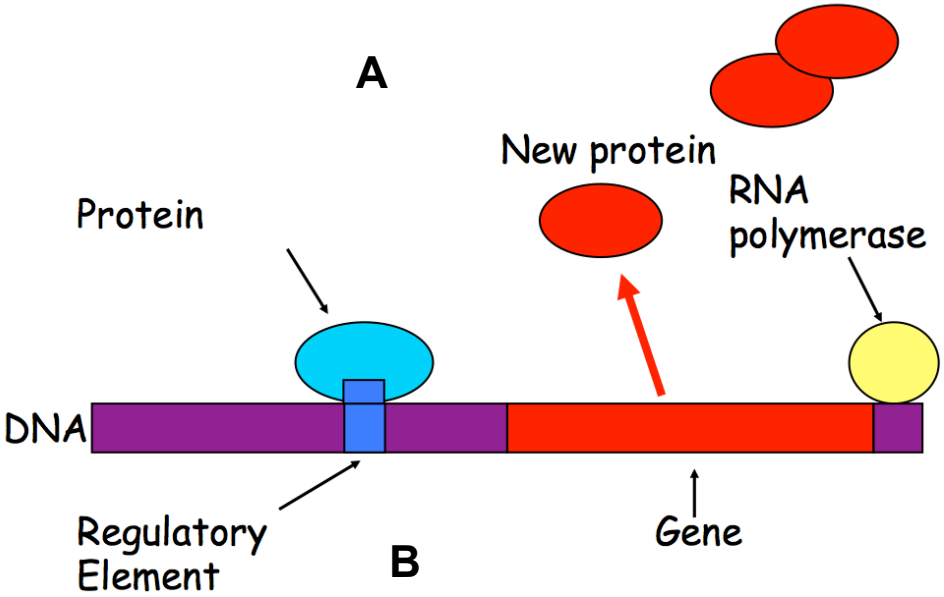
Genes are activated or repressed by regulatory proteins which bind to gene flanking sequences (promoter) and are coded by the same or other genes.



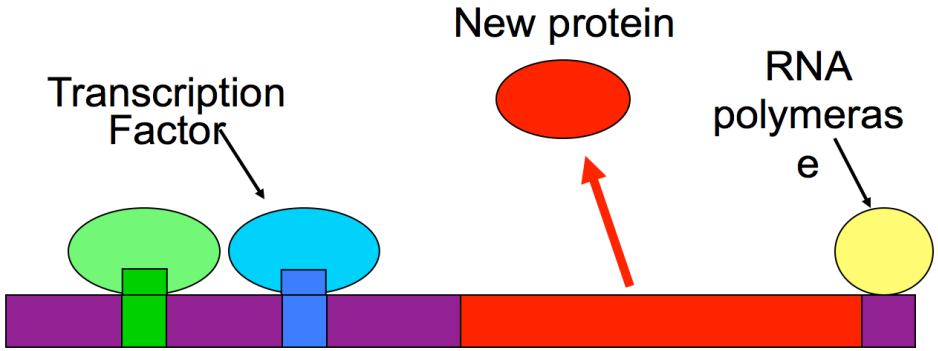
A



C

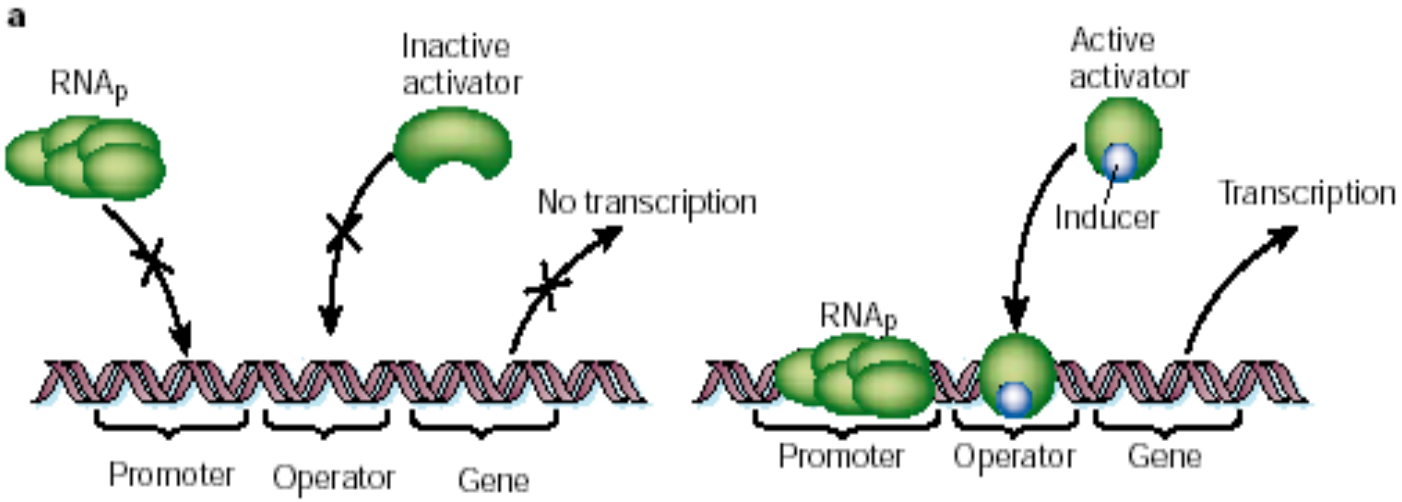


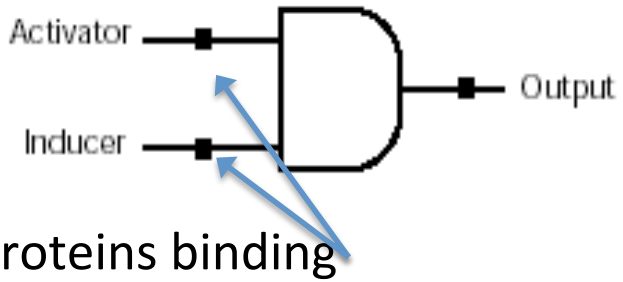
B



D

Logic gates: The Cell as an information processing device

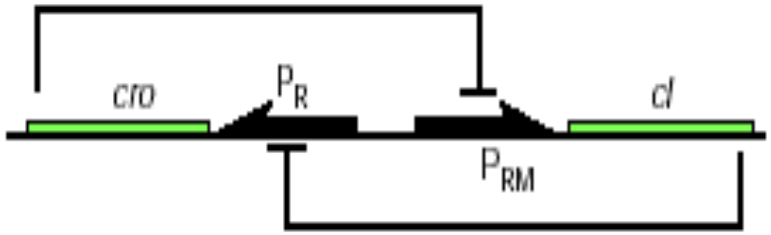




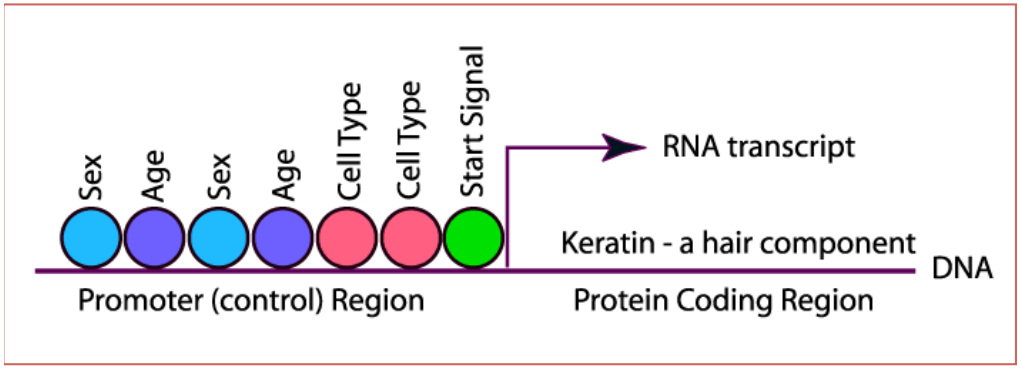
Activator	Inducer	Output
0	0	0
0	1	0
1	0	0
1	1	1

proteins binding regulatory elements

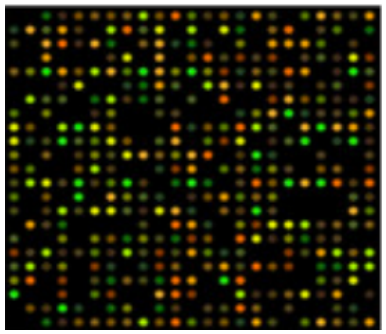
Toggle switch (cro and cl are genes; Pr and Prm are binding sites for the proteins of genes cro and cl)

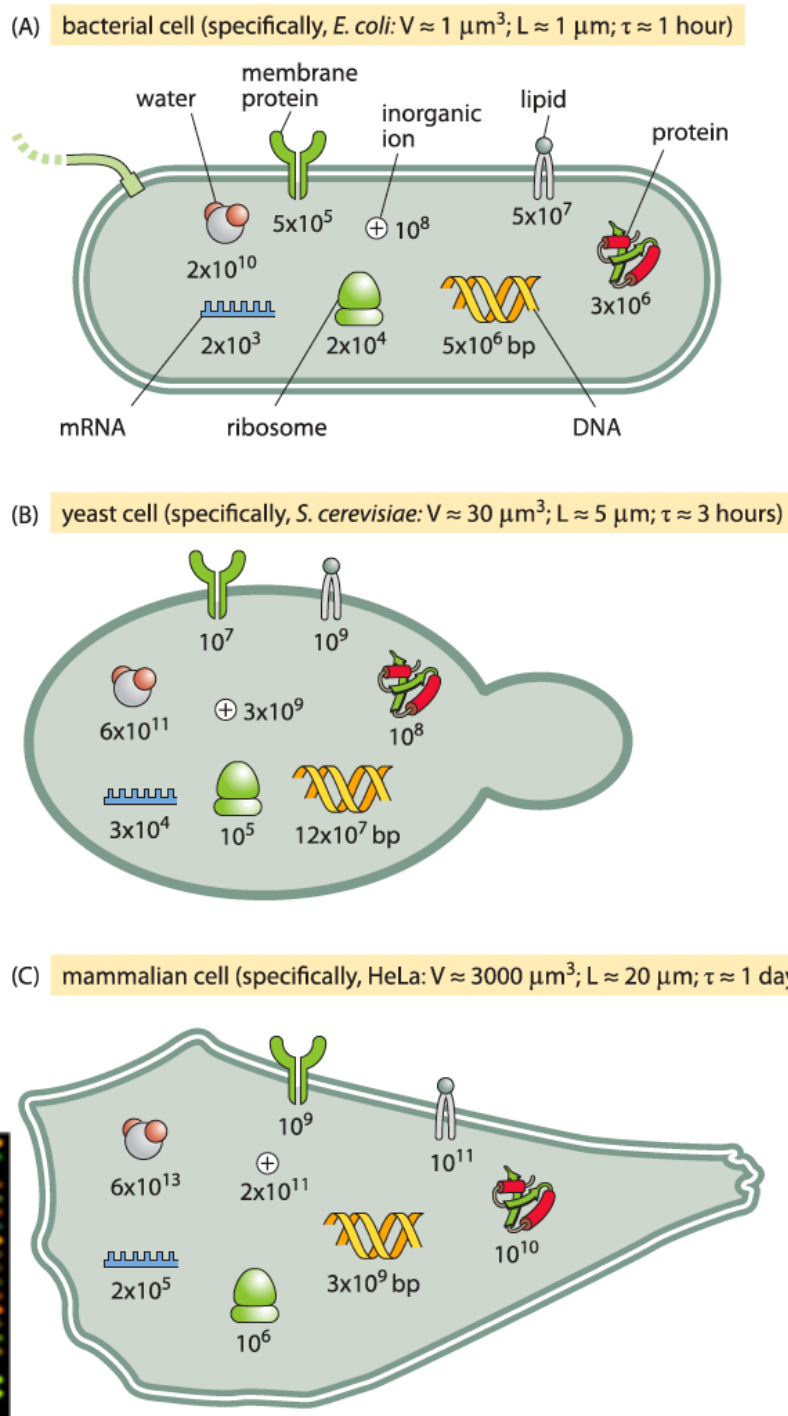


The Cell is a Computer in Soup



ABOVE: Idealized promoter for a gene involved in making hair. Proteins that bind to specific DNA sequences in the promoter region together turn a gene on or off. These proteins are themselves regulated by their own promoters leading to a gene regulatory network with many of the same properties as a neural network. We use chips (right) to monitor the activity of all the genes in different conditions (gene expression).





Cells versus Computers

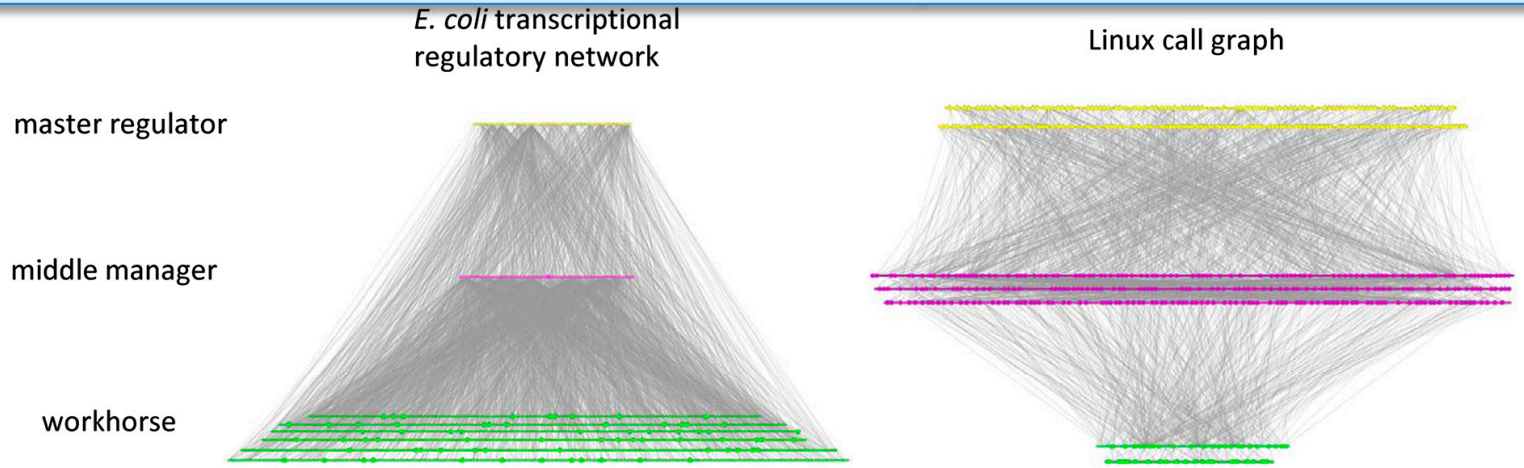
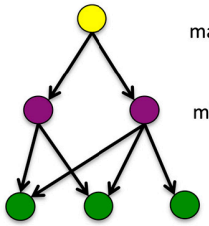


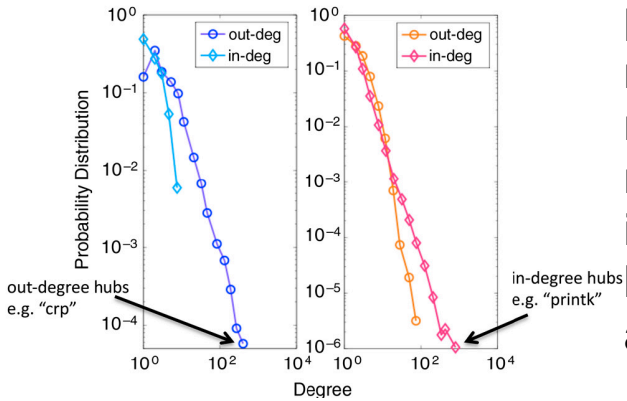
Fig. 1. The hierarchical layout of the *E. coli* transcriptional regulatory network and the Linux call graph. (*Left*) The transcriptional regulatory network of *E. coli*. (*Right*) The call graph of the Linux Kernel. Nodes are classified into three categories on the basis of their location in the hierarchy: master regulators (nodes with zero in-degree, *Yellow*), workhorses (nodes with zero out-degree, *Green*), and middle managers (nodes with nonzero in- and out-degree, *Purple*). Persistent genes and persistent functions (as defined in the main text) are shown in a larger size. The majority of persistent genes are located at the workhorse level, but persistent functions are underrepresented in the workhorse level. For easy visualization of the Linux call graph, we sampled 10% of the nodes for display. Under the sampling, the relative portion of nodes in the three levels and the ratio between persistent and nonpersistent nodes are preserved compared to the original network. The entire *E. coli* transcriptional regulatory network is displayed.

A



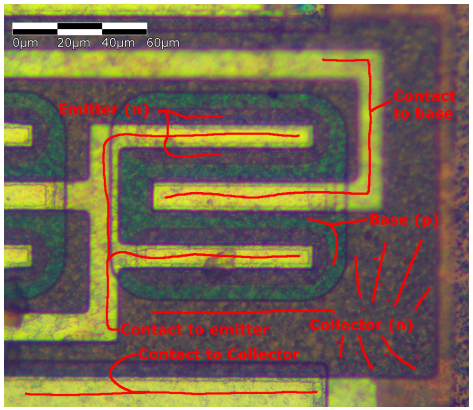
	percentage in E. coli regulatory network	percentage in Linux call graph
master regulator	4.6	29.6
middle manager	5.1	58.2
workhorse	90.2	12.3

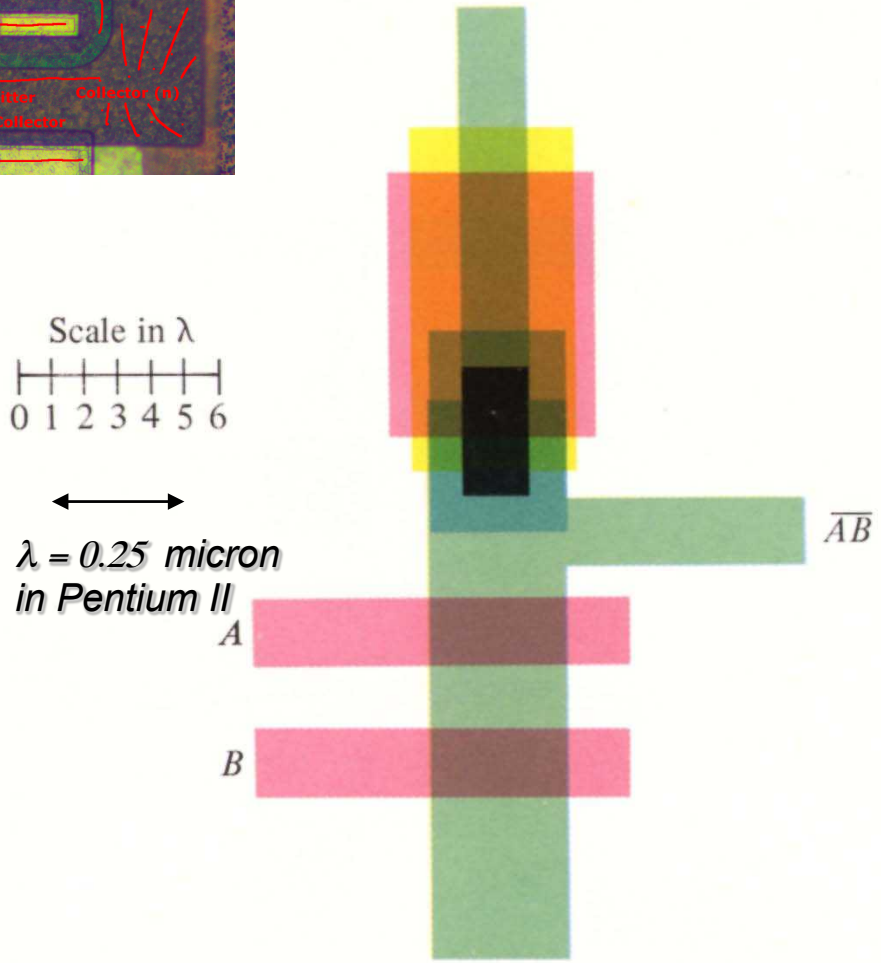
B



The transcriptional regulatory network (1,378 nodes) follows a conventional hierarchical picture, with a few top regulators and many workhorse proteins. The Linux call graph (12,391 nodes), on the other hand, possesses many regulators; the number of workhorse routines is much lower in proportion. The regulatory network has a broad out-degree distribution but a narrow in-degree distribution. The situation is reversed in the call graph, where we can find in-degree hubs, but the out-degree distribution is rather narrow. Yan et al. PNAS 2010, 107, 20.

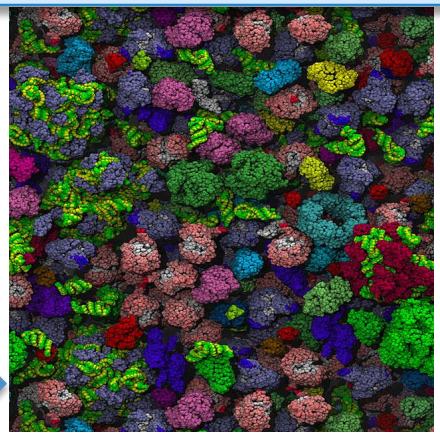
Scales of electronic and bio devices





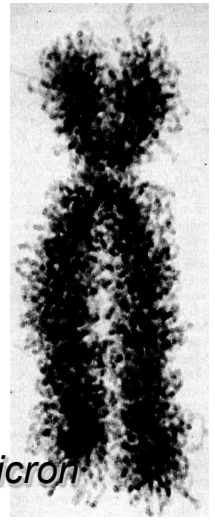
(a) NAND gate layout geometry.

proteins inside a bacterium



1 micron

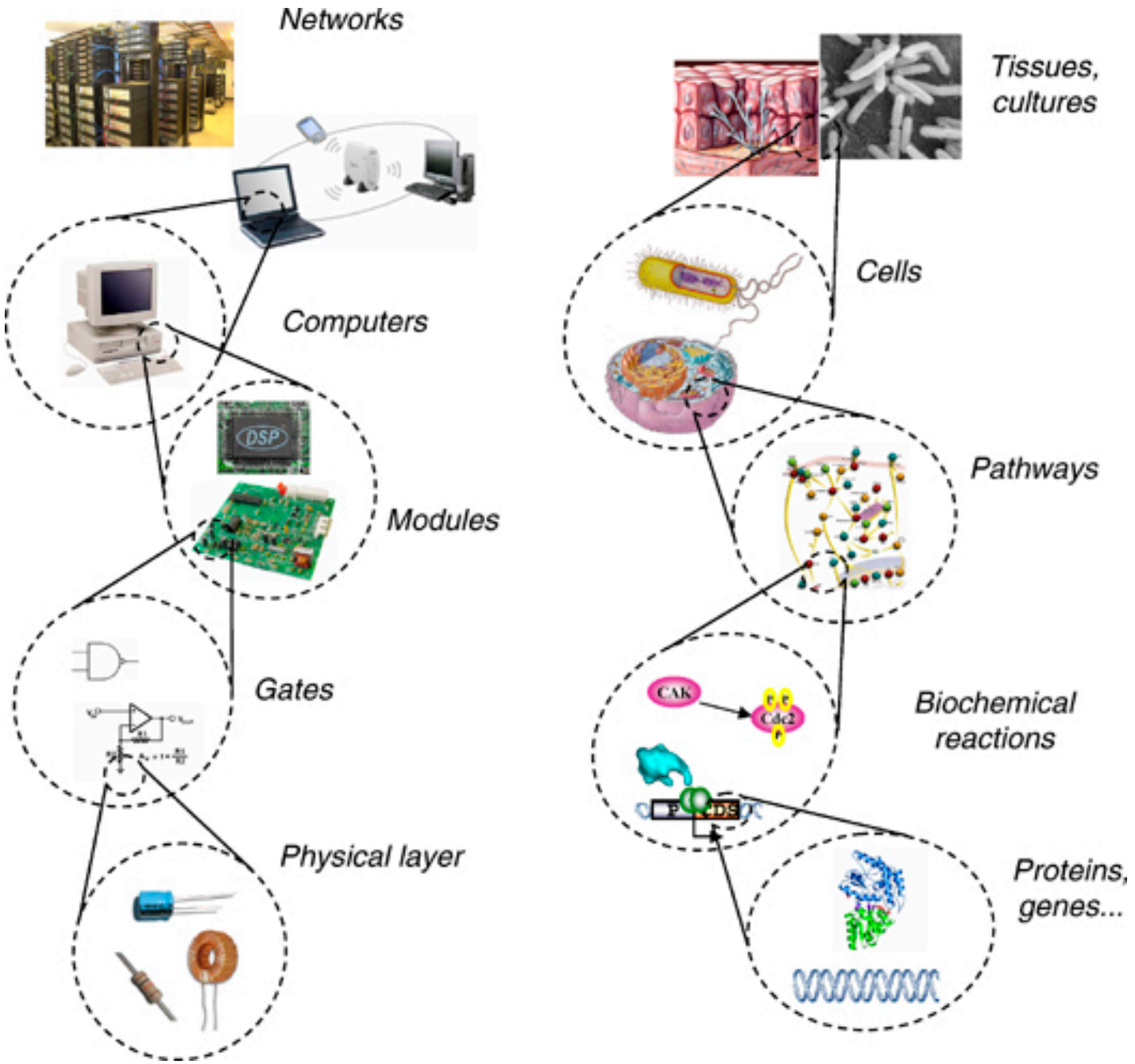
Bacterium



1 micron

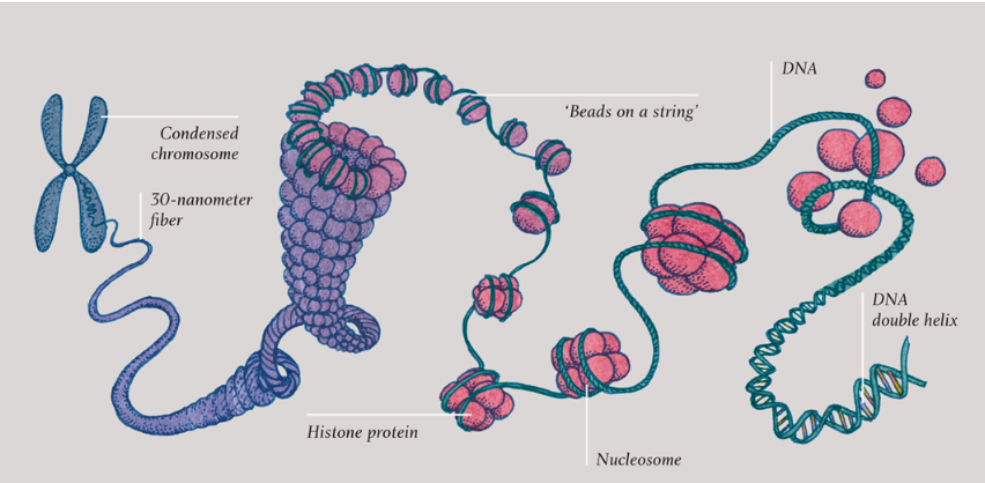
Human chromosome.

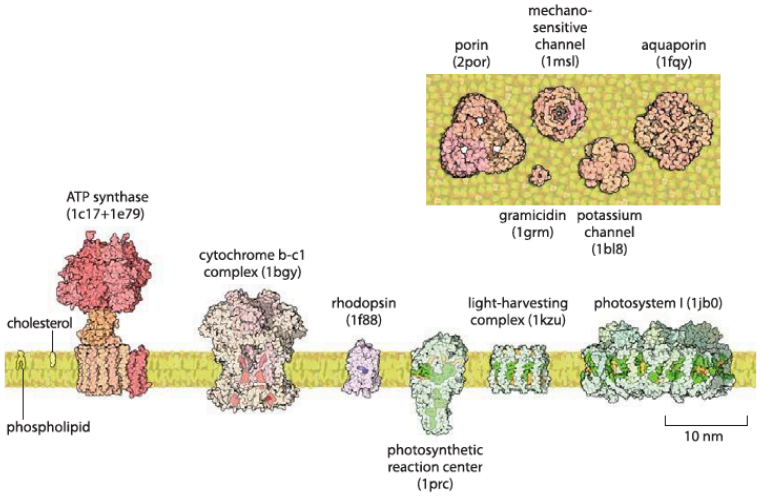
The network level: can you spot the difference?



Nature is programmed for self-assemble; Bioinformatics is needed to identify the key elements

- DNA, RNA and proteins can:
- Organize themselves to self assemble different types of devices (mechanisms such rotors, motors) or structures with different shapes across time and space scales.
- Organise other types of molecules such as lipids, sugars and artificial ones.
- Organise large set of reactions (such as metabolic networks) and Execute different kinetics
- Self-Assemble control devices



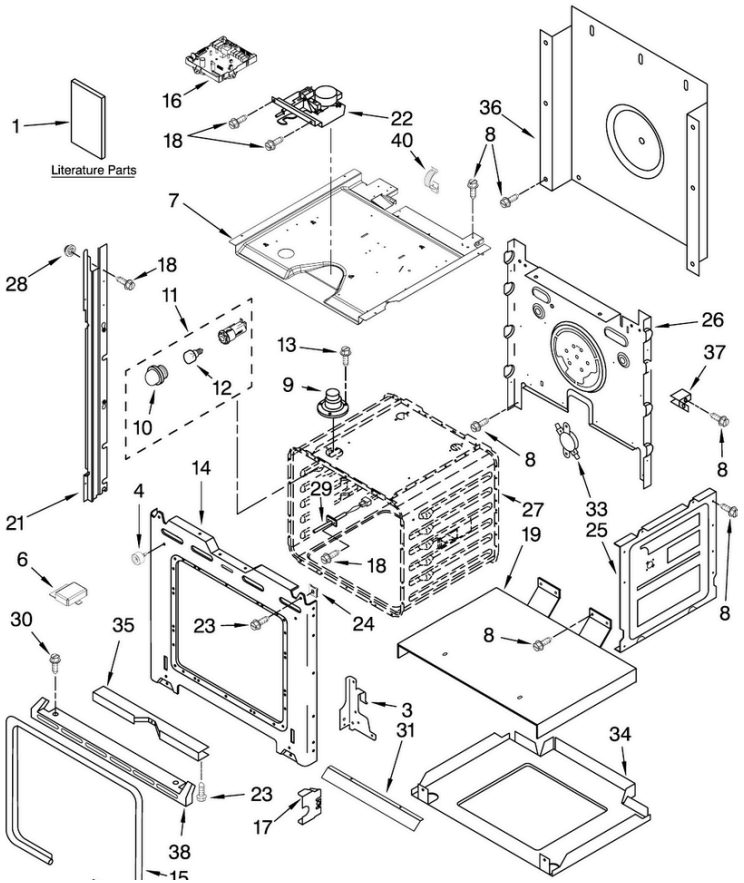


IKEA®

OVEN PARTS

For Models: IBS550PWW00, IBS550PWS00
(White) (Stainless)

30" BUILT–IN
ELECTRIC OVEN
SELF CLEAN
THERMAL CONVECTION



OVEN PARTS

For Models: IBS550PWW00, IBS550PWS00
(White) (Stainless)

Illus. No.	Part No.	DESCRIPTION
1		Literature Parts
	W10210919	Instructions, Installation
	W10075750	Instruction, Installer
	W10270566	Owners Manual
	W10201489	Tech Sheet
	8304572	Installation/ Undercounter
3		Hinge Receiver
	4455606	Right
	4455605	Left
4	W10105790	Bumper, Door
6	8285593	Cover, Wire
7	8303099	Top Chassis
8	4449809	Screw
9	4452158	Tube, Vent
10	W10067870	Lens, Light
11	W10009930	Assembly, Oven Light
12	4452166	Bulb, Light
13	3196176	Screw
14	W10115831	Frame, Front
15	4455382	Gasket, Door
16	W10158972	Control
17	4452152	Bracket, Mounting
18	4449154	Screw
19	4451478	Support, Insulation
21		Rail, Mounting
	W10158955	Right
	W10158956	Left
22	9760888	Latch, Motorized
23	4449040	Screw
24	4450118	Nut
25	4455641	Side, Chassis
26	W10161110	Back, Chassis
27		Liner, Oven (Not serviceable)
28	4448444	Grommet
29	W10131825	Sensor
30	3400805	Screw
31	9760677	Deflector, Vent
33	9759243	Thermostat
34	9759713	Base, Chassis
35	4448933	Retainer, Gasket
36	4451747	Cover, Back
37	4449845	Support, Bracket
38	W10163965	Trim, Bottom
40	98997	Clip

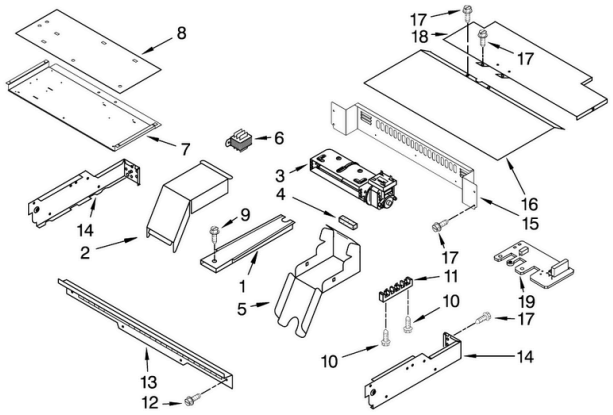
FOLLOWING PARTS NOT ILLUSTRATED

INSULATION

4449316	Insulation, Wrap
4450365	Insulation, Back

TOP VENTING PARTS

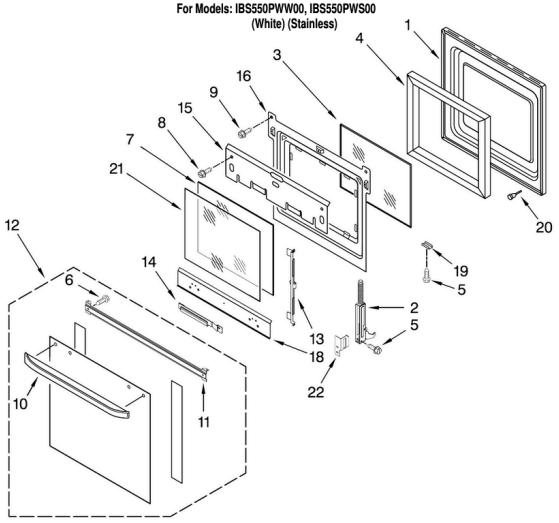
For Models: IBS550PWW00, IBS550PWS00
(White) (Stainless)



Illus. No.	Part No.	DESCRIPTION
1	4451759	Vent, Exhaust
2	4455178	Elbow, Vent–Top
3	8300223	Blower
	4455352	Bracket
4	4451758	Block, Foam
5	4455177	Elbow, Vent–Bottom
6	9760587	Transformer, Control
7	W10171405	Mounting Plate
8	8303076	Insulation
9	4449040	Screw
10	3400080	Screw
11	7401P038–60	Block, Terminal
12	4450038	Screw
13	8303497	Vent, Control Panel
14		Side, Control Top
	4451597	Right
	4451598	Left
15	8303100	Back, Control
16	4450551	Cover, Top Rear
17	4449809	Screw
18	4452367	Cover, Top Front
19	4451985	Suppressor

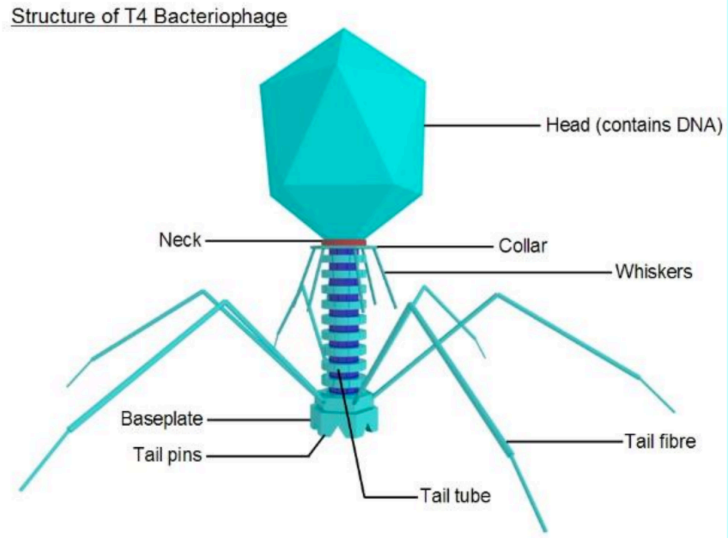
OVEN DOOR PARTS

For Models: IBS550PWW00, IBS550PWS00
(White) (Stainless)



Illus. No.	Part No.	DESCRIPTION
1	9759413	Door Liner
2		Hinge, Door
	9760577	Right
	9760578	Left
3	4449259	Glass, Inner
4	4451722	Insulation, Door
5	4449743	Screw
6	W10169394	Screw
7	9759225	Glass, Inner
8	4449809	Screw
9	3400832	Screw
10	W10158977	Door Handle
11	W10159515	Bracket, Handle
12		Door Glass
	W10163962	White
	W10163961	Stainless
13	9759229	Bracket(2nd Glass)
14	9759230	Bracket(3rd Glass)
15	4457104	Shield, Heat
16	4457132	Retainer, Glass
18	4457114	Shield, Heat (Btm)
19	W10158978	Retainer, Clip
20	4448520	Bumper, Door (3)
21		Glass, Inner
	9759228	White
	9759225	Stainless
22		Cover, Hinge
	W10137044	Right
	W10137043	Left

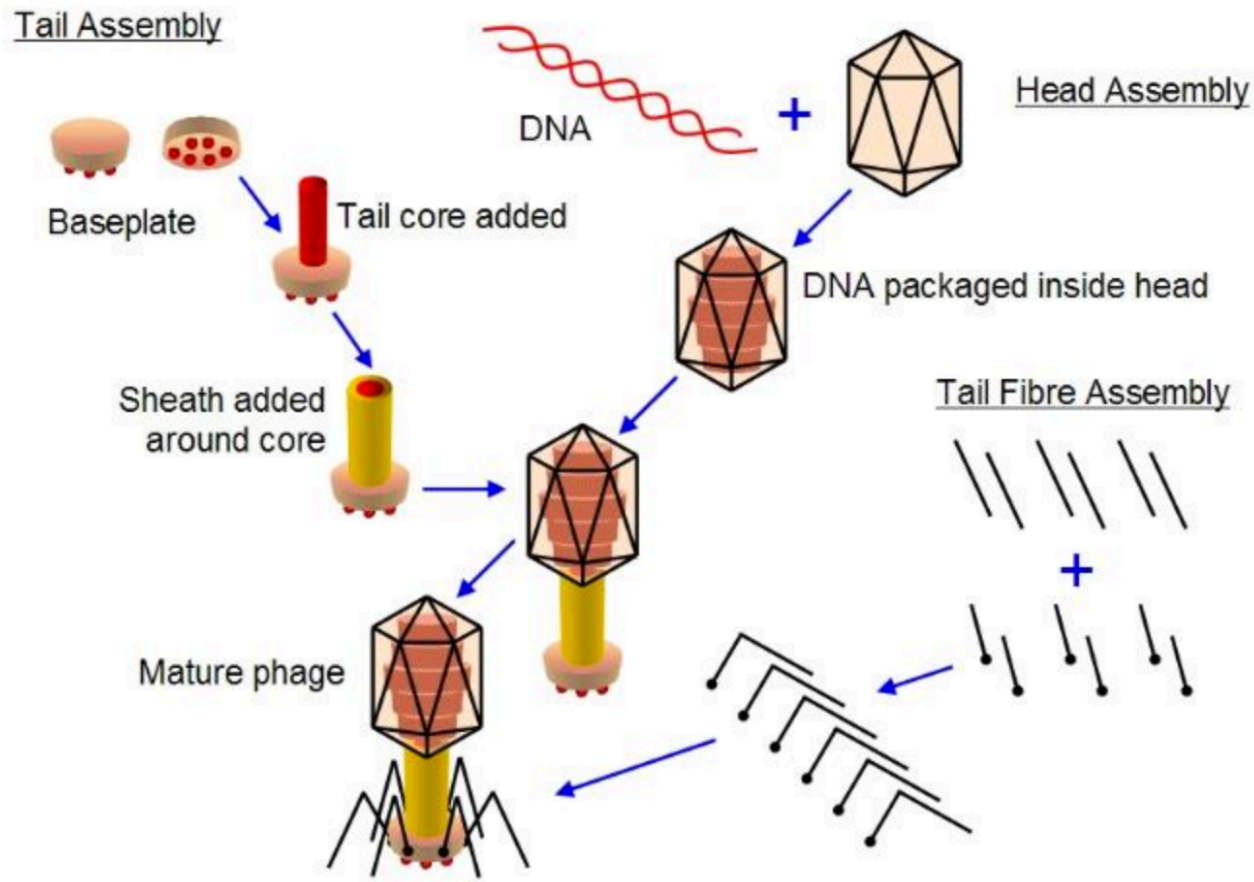
Macroscale IKEA: not self assembly



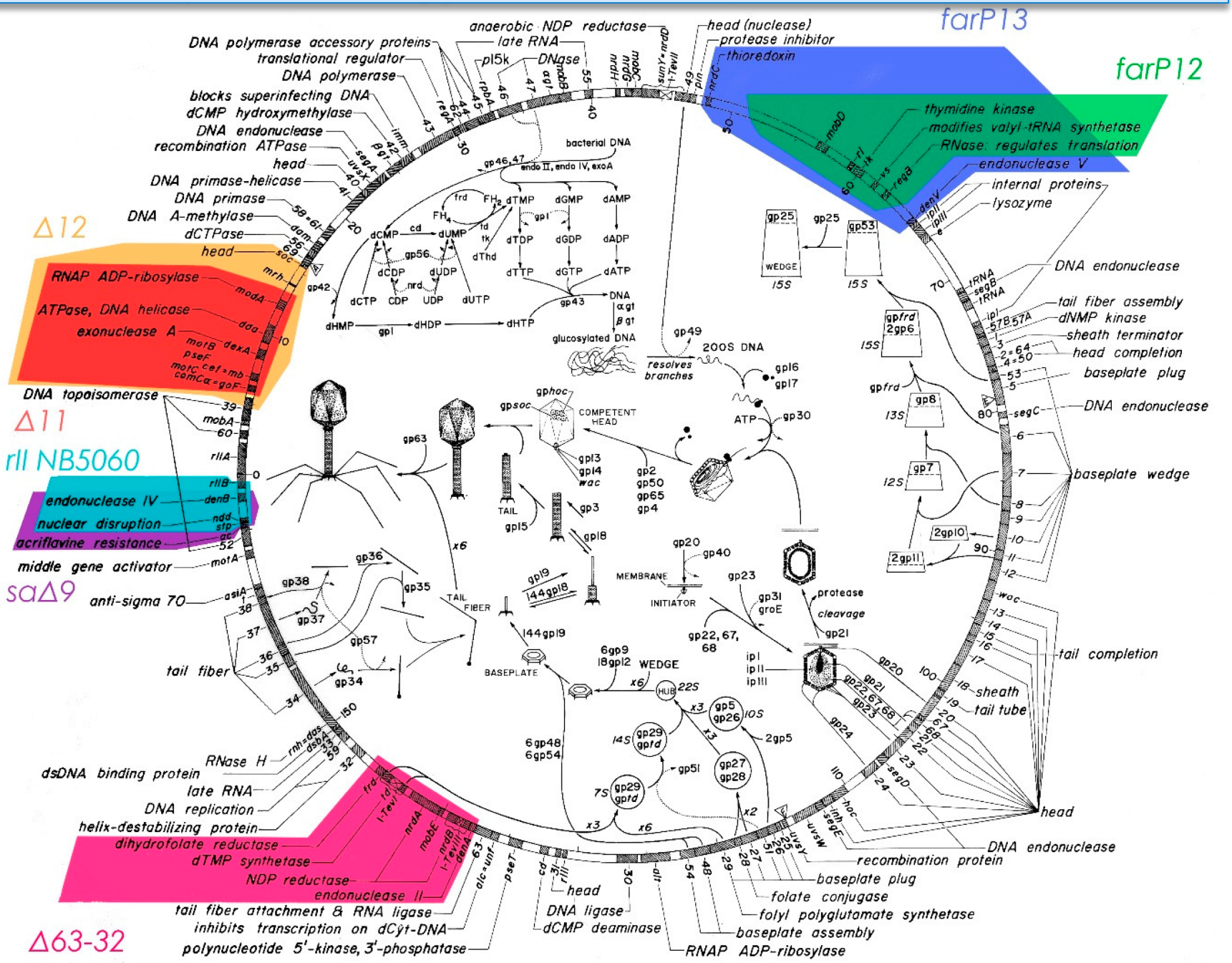
microscale IKEA: Nature is programmed for self assembly

24 to 200 nanometers they're 10 to 100 times smaller than the average bacterium, much too small to see with an ordinary light microscope.

5. We absorb about 30 billion phages into our bodies every day. They form an integral part of our microbial ecosystem.



The genome contains both the instructions for assembly and for the parts and it is shipped with the virus



Cells versus Computers

- Base-4 (ACGT)
- DNA
- Bases
- Codons (triplets of bases for each amino acid)
- Genetic Code (translate codons into amino acids)
- Gene/Protein
- Chromosome
- Genome Size

- Base-2 (101010)
- Magnetic tape/Disk
- Bits/Transistors
- Bytes
- Instruction Set

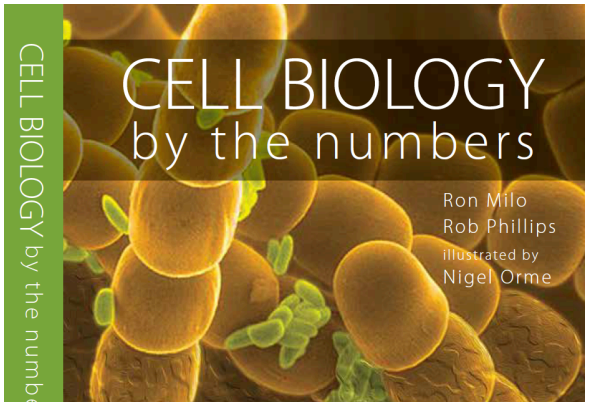
- File, Program
- Hard Disk
- Disk Capacity

Cells versus Computers

Biology	Computer science
1. Digital alphabet consists of bases A, C, T, G	1. Digital alphabet consists of 0, 1
2. Codons consist of three bases	2. Computer bits form bytes
3. Genes consist of codons	3. Files consist of bytes
4. Promoters indicate gene locations	4. File-allocation table indicates file locations
5. DNA information is transcribed into hnRNA and processed into mRNA	5. Disc information is transcribed into RAM
6. mRNA information is translated into proteins	6. RAM information is translated onto a screen or paper
7. Genes may be organized into operons or groups with similar promoters	7. Files are organized into folders
8. "Old" genes are not destroyed; their promoters become nonfunctional	8. "Old" files are not destroyed; references to their location are deleted
9. Entire chromosomes are replicated	9. Entire discs can be copied
10. Genes can diversify into a family of genes through duplication	10. Files can be modified into a family of related files
11. DNA from a donor can be inserted into host chromosomes	11. Digital information can be inserted into files
12. Biological viruses disrupt genetic instructions	12. Computer viruses disrupt software instructions
13. Natural selection modifies the genetic basis of organism design	13. Natural selection procedures modify the software that specifies a machine design
14. A successful genotype in a natural population outcompetes others	14. A successful website attracts more "hits" than others

If you want to know more about biology

A free book is this: cell biology by the numbers
http://book.bionumbers.org/



- Genetics for Computer Scientists

https://www.cs.helsinki.fi/group/genetics/Genetics_for_CS_March_04.pdf

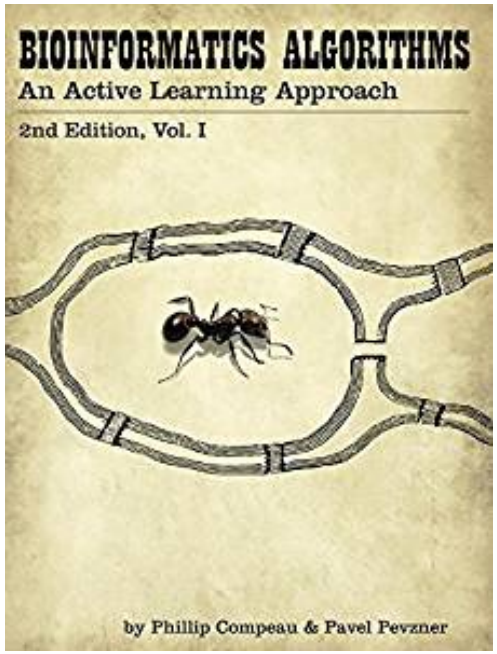
- Molecular Biology for Computer Scientists:

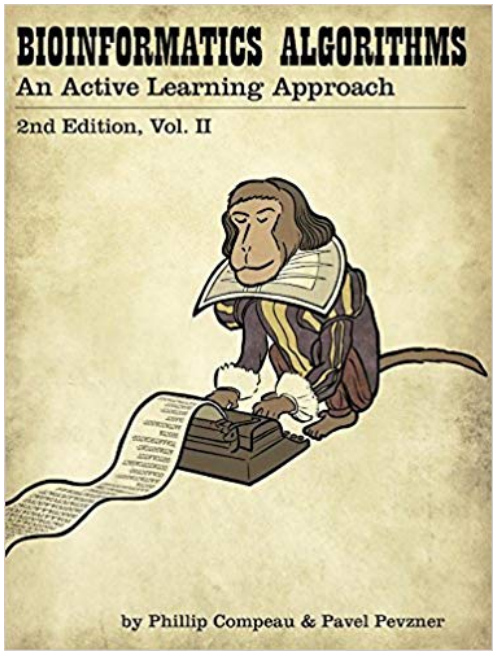
http://tandy.cs.illinois.edu/Hunter_MolecularBiology.pdf

Biology and Computers: A lesson in what is possible
https://ethw.org/

https://www.wehi.edu.au/wehi-tv/

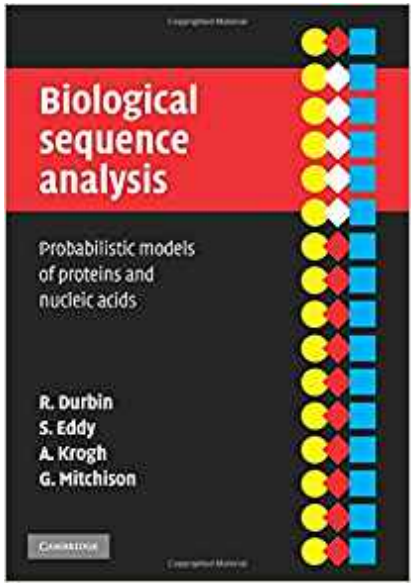
General references for course





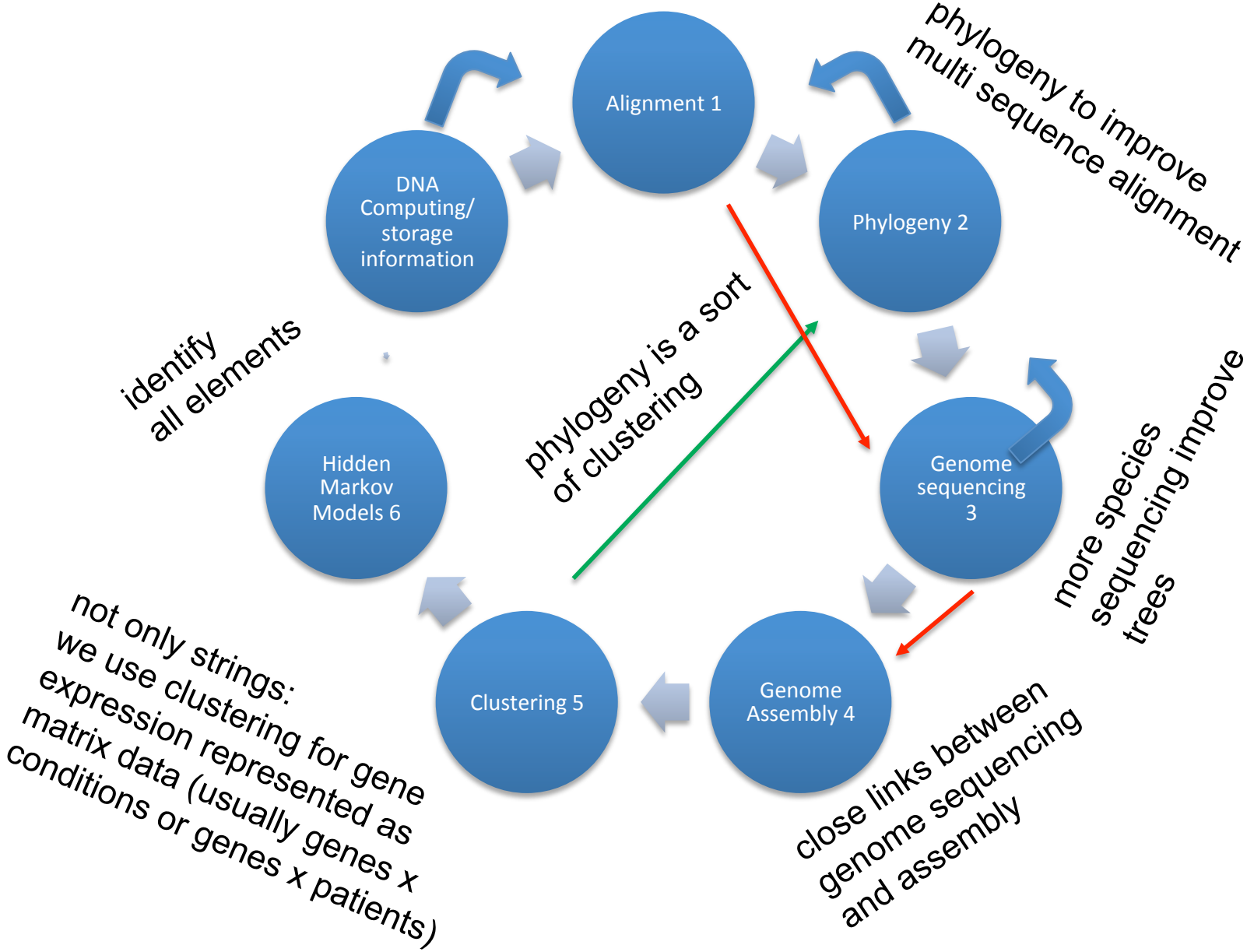
Partly based on book: Compeau and Pevzner Bioinformatics algorithms (chapter 3,5,7-10 chapter).

also Richard Durbin, Sean R. Eddy, Anders Krogh, Graeme Mitchison
Biological Sequence Analysis:
Probabilistic Models of Proteins and Nucleic Acids



No biology in the exam questions (You need to know only the reason of the algorithms).

Structure of the course



Aligning DNA and Protein Sequences

- how to align two sequences?
- Trees (what is the relationships of multiple sequences and what has to do with species evolutionary history)
- Genome sequence (how to analyse a genome)

How Do We Compare Biological Sequences?

- From Sequence Comparison to Biological Insights
- The Alignment Game and the Longest Common Subsequence
- Dynamic Programming and Backtracking Pointers
- From Global to Local Alignment
- Penalising Insertions and Deletions in Sequence Alignment
- Space-Efficient Sequence Alignment
- Nussinov folding algorithm (RNA 2dimensional folding)

Summary for alignment lectures

Algorithms in this lecture: Longest common subsequence, Needleman-Wunsch, Smith-Waterman, Affine gap, Hirschberg, Nussinov RNA folding. Typical tasks: align genome and protein sequences; we want to detect all differences at the single base to block of bases levels. In the RNA folding problem we want to align a molecule with itself.

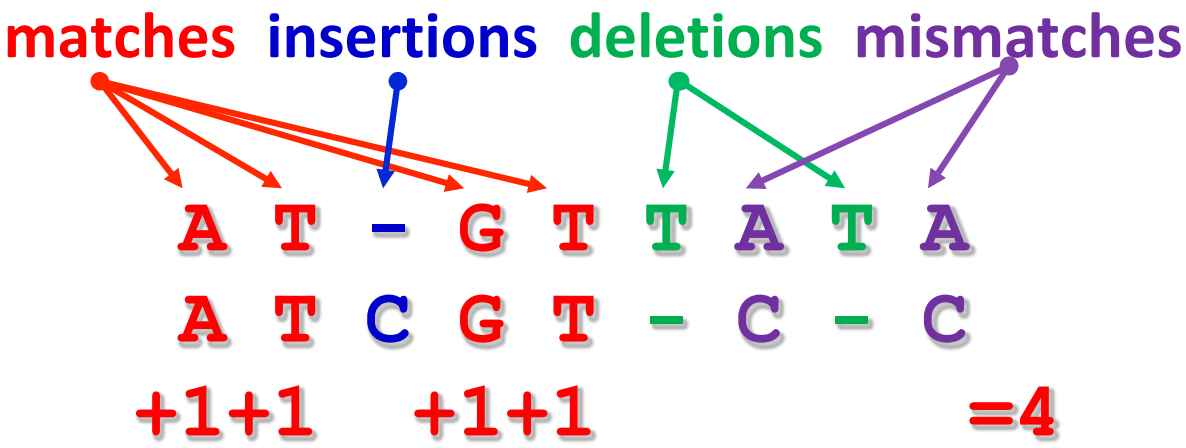
Data: DNA or protein (amino acid) sequences considered as strings; input: two strings (Nussinov accepts one string in input and search for internal similarities). Output: a set of aligned positions that makes easy the identification of conserved patterns. Note that each string belongs to a double helix so the information could be related to one of the two strands and read in one or the opposite orientation.

Many events (mutations) could lead to sequence changes. Therefore the conservation of a substring between two strings may suggest to a crucial functional role for the cell.

The dynamic programming algorithms could be used to detect similarities within a single string (last section of the lecture). This is particularly useful to find the folding of RNA moleculaes (in a RNA molecule the T is replaced by U).

Main question in this lecture: how similar are these two sequences?

What Is the Sequence Alignment?



Alignment of two sequences is a two-row matrix:

1st row: symbols of the 1st sequence (in order) interspersed by "-"
2nd row: symbols of the 2nd sequence (in order) interspersed by "-"

Longest Common Subsequence

```
A T - G T T A T A
A T C G T - C - C
```

Matches in alignment of two sequences (**ATGT**) form their **Common Subsequence**

Longest Common Subsequence Problem: Find a longest common subsequence of two strings.

- **Input:** Two strings.
- **Output:** A longest common subsequence of these strings.

Alignment: 2 row representation

Given 2 DNA sequences **v** and **w**:

v : A T G T T A T **m** = 7
w : A T C G T A C **n** = 7

Alignment : 2 * **k** matrix (**k** > **m**, **n**)

letters of **v**	A	T	--	G	T	T	A	T	--
letters of **w**	A	T	C	G	T	--	A	--	C

4 matches | 2 insertions | 2 deletions

Longest Common Subsequence

Longest Common Subsequence (LCS) –the simplest form of sequence alignment – allows only insertions and deletions (no mismatches). In the LCS Problem, we scored 1 for matches and 0 for indels; in real analysis we consider penalising indels and mismatches with negative scores.

- Given two sequences $\mathbf{v} = v_1 v_2 \ldots v_m$ and $\mathbf{w} = w_1 w_2 \ldots w_n$
- The LCS of **v** and **w** is a sequence of positions in

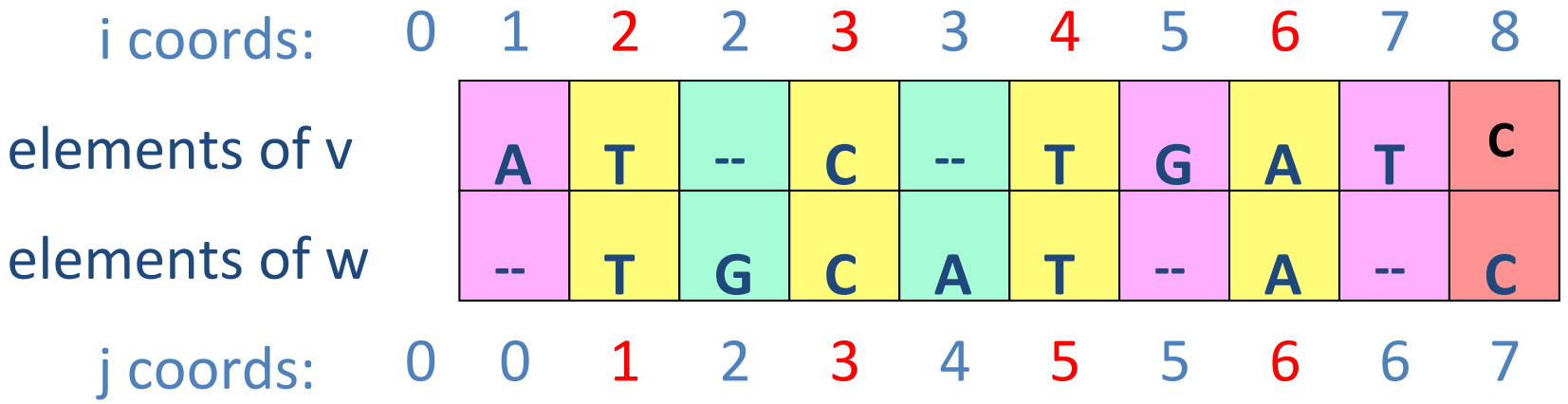
$$\mathbf{v}: 1 \leq i_1 < i_2 < \ldots < i_t \leq m$$

and a sequence of positions in

$$\mathbf{w}: 1 \leq j_1 < j_2 < \ldots < j_t \leq n$$

such that i_t -th letter of **v** equals to j_t-th letter of **w** and **t** is maximal.

Longest Common Subsequence



(0,0)→ (1,0)→ (2,1)→ (2,2)→ (3,3)→ (3,4)→ (4,5)→ (5,5)→ (6,6)→ (7,6)→ (8,7)

Matches shown in red

positions in v: 2 < 3 < 4 < 6 < 8

positions in w: 1 < 3 < 5 < 6 < 7

Every common subsequence is a path in 2-D grid

Longest Common Subsequence

The Edit distance between two strings is the minimum number of operations (insertions, deletions, and substitutions) to transform one string into the other

Hamming distance always compares **i**-th letter of **v** with **i**-th letter of **w**

v = ATATATAT

w = TATATATA

Just one shift

Make it all line up

Edit distance may compare **i**-th letter of **v** with **j**-th letter of **w**

v = -ATATATAT

w = TATATATA-

Hamming distance:

d(**v**, **w**)=8

Computing Hamming distance is a trivial task

Edit distance:

d(**v**, **w**)=2

Computing edit distance is a non-trivial task

Edit Distance: Example

TGCATAT → ATCCGAT in 4 steps

TGCATAT → (insert A at front)

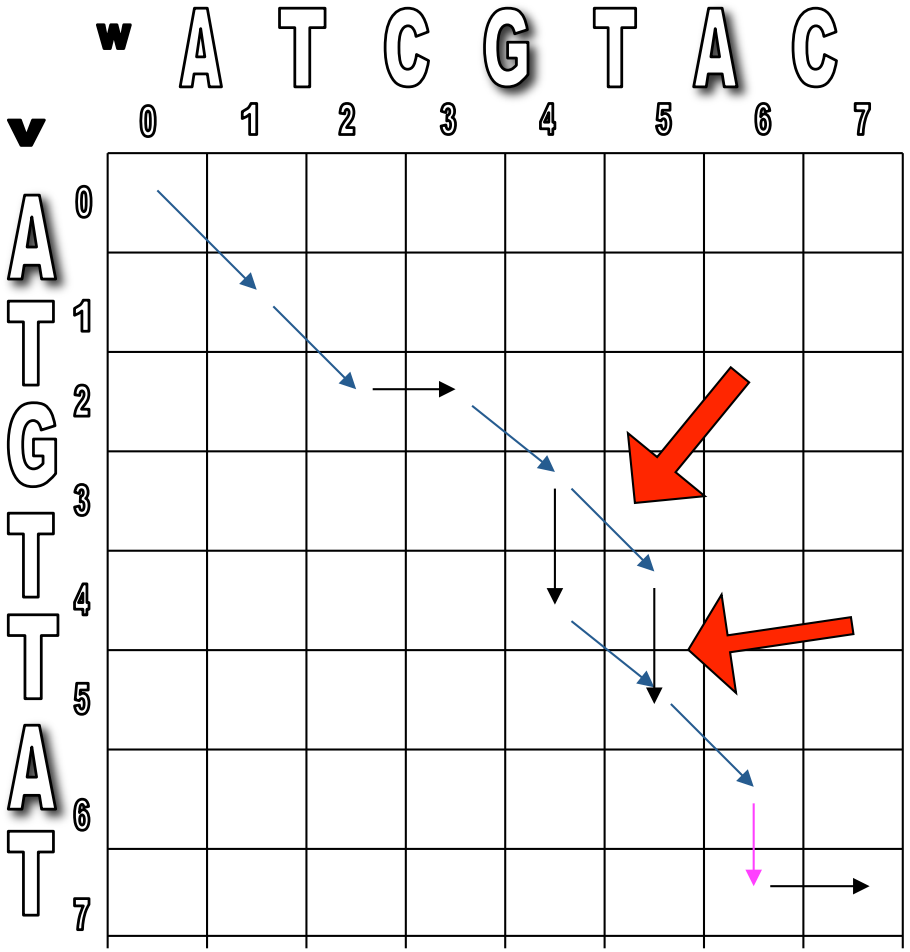
ATGCATAT → (delete 6th T)

ATGCATA → (substitute G for 5th A)

ATGCGTA → (substitute C for 3rd G)

ATCCGAT (Done)

Alignment as a Path in the Edit Graph



Old Alignment

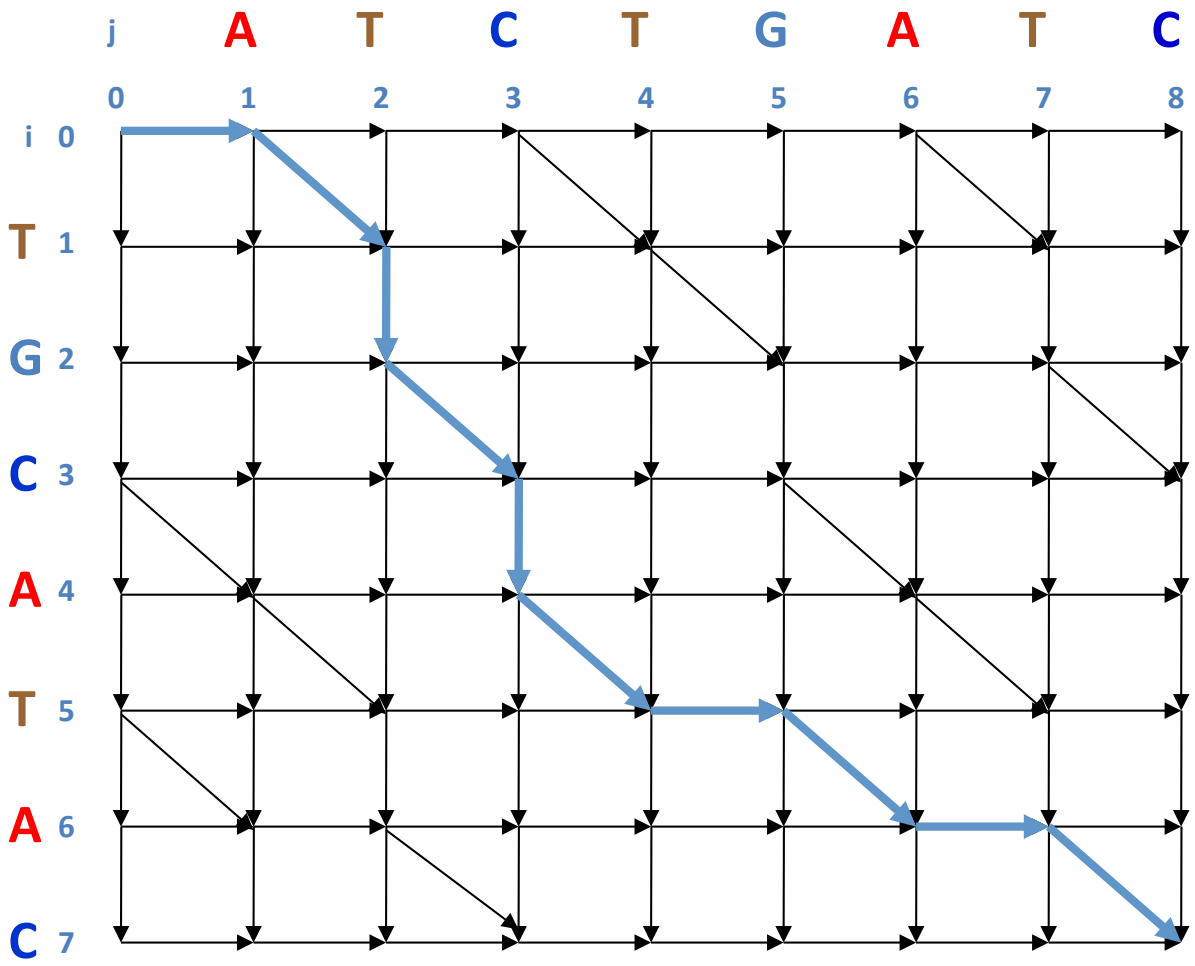
```
    01223456677
v=   AT_GTTAT_
w=   ATCGT_A_C
    01234556667
```

New Alignment

```
    01223456677
v=   AT_GTTAT_
w=   ATCG_TA_C
    01234456667
```

Two similar alignments; the score is 5 for both the alignment paths.

LCS Problem as - Edit Graph



Every path is a common subsequence.

Every diagonal edge adds an extra element to common subsequence

LCS Problem: Find a path with maximum number of diagonal edges

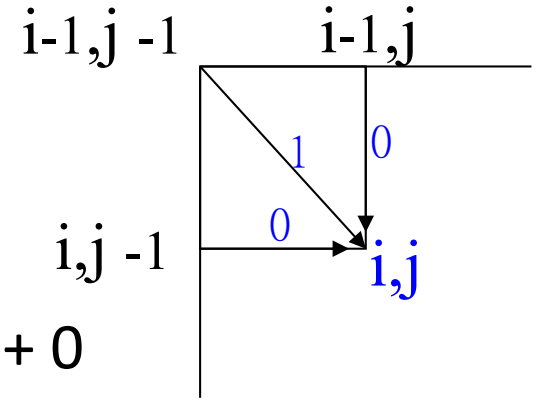
Computing LCS

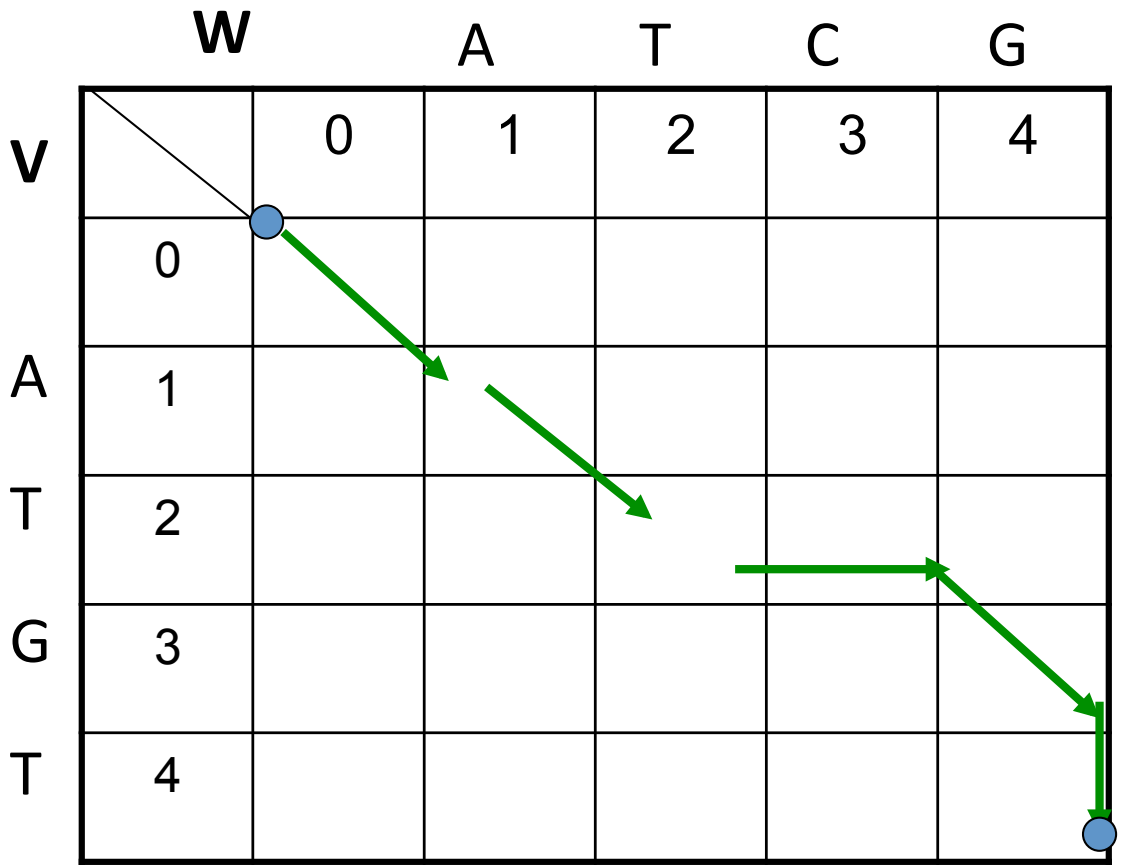
Let $\mathbf{v}_i$ = prefix of **v** of length i: $v_1 \ldots v_i$

and $\mathbf{w}_j$ = prefix of **w** of length j: $w_1 \ldots w_j$

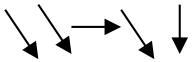
The length of LCS($\mathbf{v}_i$,$\mathbf{w}_j$) is computed by:

$$s_{i,j} = \text{MAX} \begin{cases} s_{i-1,j} + 0 \\ s_{i,j-1} + 0 \\ s_{i-1,j-1} + 1, \quad \text{if } v_i = w_j \end{cases}$$





Every Path in the Grid Corresponds to an Alignment



0 1 2 2 3 4

V = A T - G T

| | |

W= A T C G –

0 1 2 3 4 4

LCS Algorithm

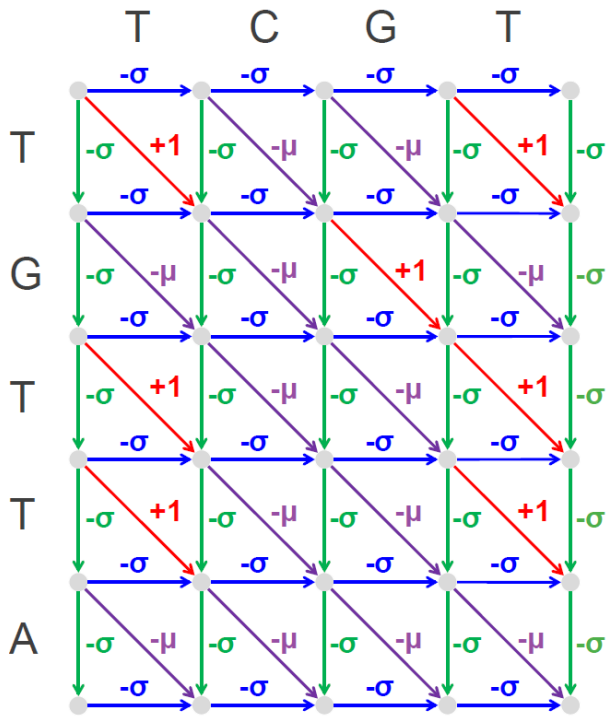
```
LCS(v, w)
1  for i ← 0 to n
2      s_{i,0} ← 0
3  for j ← 1 to m
4      s_{0,j} ← 0
5  for i ← 1 to n
6      for j ← 1 to m
7          s_{i,j} ← max { s_{i-1,j}
                         { s_{i,j-1}
                         { s_{i-1,j-1} + 1,   if v_i = w_j
8          b_{i,j} ← { "↑"   if s_{i,j} = s_{i-1,j}
                     { "←"   if s_{i,j} = s_{i,j-1}
                     { "↖",  if s_{i,j} = s_{i-1,j-1} + 1
9  return (s_{n,m}, b)
```

```
PRINTLCS(b, v, i, j)
1  if i = 0 or j = 0
2      return
3  if b_{i,j} = "↖"
4      PRINTLCS(b, v, i − 1, j − 1)
5      print v_i
6  else
7      if b_{i,j} = "↑"
8          PRINTLCS(b, v, i − 1, j)
9      else
10         PRINTLCS(b, v, i, j − 1)
```

The above recursive program prints out the longest common subsequence using the information stored in b. The initial invocation that prints the solution to the problem is PRINTLCS(b, v, n,m).

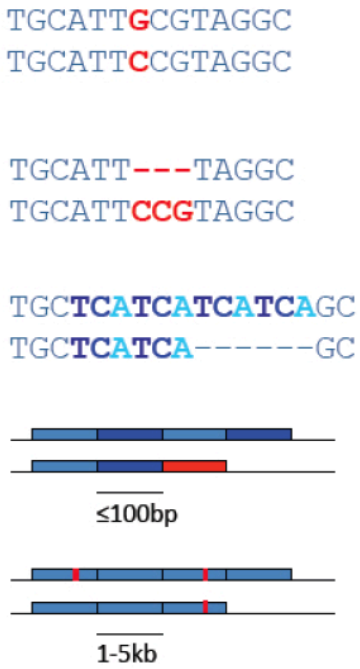
Alignment Graph

$$s_{i,j} = \max \begin{cases} s_{i-1,j} - \sigma \\ s_{i,j-1} - \sigma \\ s_{i-1,j-1} + 1, \text{ if } v_i = w_j \\ s_{i-1,j-1} - \mu, \text{ if } v_i \neq w_j \end{cases}$$



All genomes are littered with repeats so alignment of large sequences is difficult

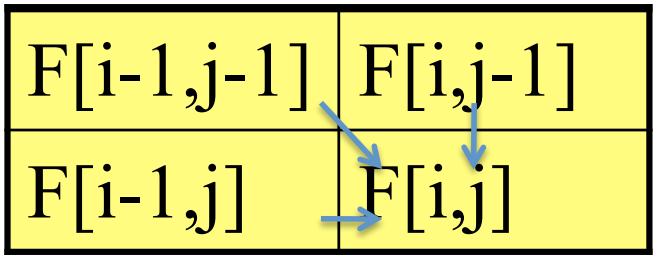
- Single nucleotide polymorphisms (SNPs)
  - 1 every few hundred bp, mutation rate* $\approx 10^{-9}$
- Short indels (=insertion/deletion)
  - 1 every few kb, mutation rate v. variable
- Microsatellite (STR) repeat number
  - 1 every few kb, mutation rate $\leq 10^{-3}$
- Minisatellites
  - 1 every few kb, mutation rate $\leq 10^{-1}$
- Repeated genes
  - rRNA, histones
- Large deletions, duplications, inversions
  - Rare, e.g. Y chromosome



increased difficulty with a puzzle with many repetitions

Figure : Type and frequency of mutations (replacements, insertions, deletions) in the human genome per generation; mutations change single DNA bases (SNP polymorphism) or rearrange DNA strings at different length scales. In sequence alignment we compare sequences that are different because of mutations.

Towards an algorithm to align biological sequences **(note I am using a DIFFERENT NOTATION!)**



Notice three possible cases:

1. x_i aligns to y_j

   $x_1 \ldots\ldots x_{i-1}$ x_i

   $y_1 \ldots\ldots y_{j-1}$ y_j

   $$F(i,j) = F(i-1, j-1) + \begin{cases} m, \text{ if } x_i = y_j \\ -s, \text{ if not} \end{cases}$$

2. x_i aligns to a gap

   $x_1 \ldots\ldots x_{i-1}$ x_i

   $y_1 \ldots\ldots y_j$ -

   $$F(i,j) = F(i-1, j) - d$$

3. y_j aligns to a gap

   $x_1 \ldots\ldots x_i$ -

   $y_1 \ldots\ldots y_{j-1}$ y_j

   $$F(i,j) = F(i, j-1) - d$$

Alignment

- How do we know which case is correct?

F[i-1,j-1]	F[i,j-1]
F[i-1,j]	F[i,j]

Inductive assumption:

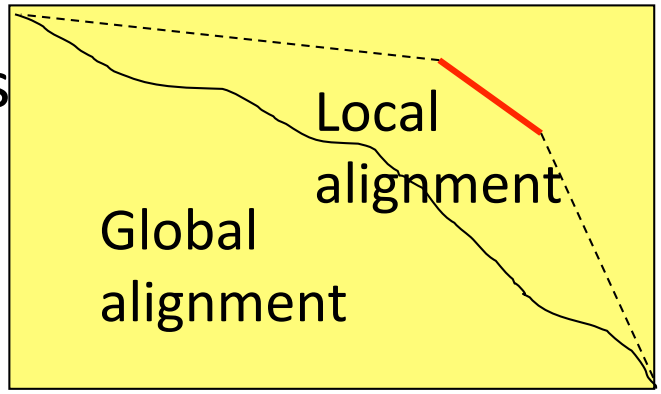
F(i, j-1), F(i-1, j), F(i-1, j-1) are optimal

Then,

$$F(i, j) = \max \begin{cases} F(i-1, j-1) + s(x_i, y_j) \\ F(i-1, \ \ j) - d \\ F(\ \ i, j-1) - d \end{cases}$$

Where $F(x_i, y_j) = m$, if $x_i = y_j$; -s, if not

- The Global Alignment Problem tries to find the longest path between vertices *(0,0)* and (*n,m*) in the edit graph.
- The Local Alignment Problem tries to find the longest path among paths between **arbitrary vertices** (*i,j*) and (*i'*, *j'*) in the edit graph.

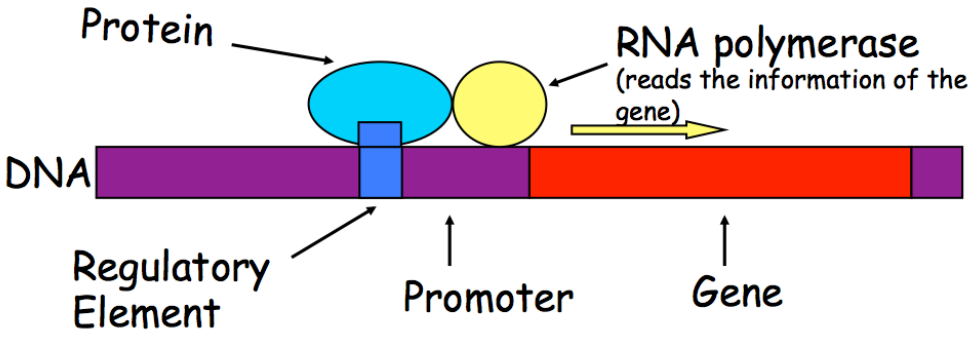


- Global Alignment

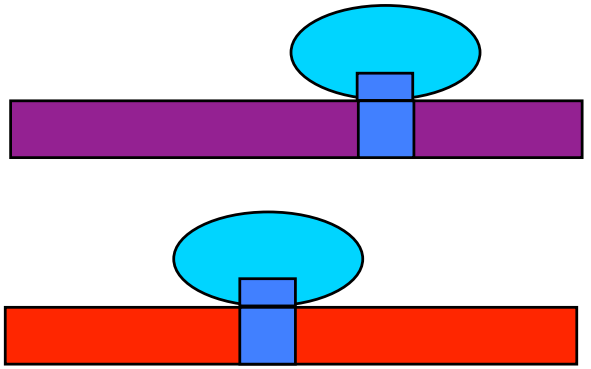
```
--T--CC-C-AGT--TATGT-CAGGGGACACG-A-GCATGCAGA-GAC
  |   || |   ||   | | | |||     || | | |   | ||||    |
AATTGCCGCC-GTCGT-T-TTCAG----CA-GTTATG-T-CAGAT--C
```

- Local Alignment—better alignment to find highly conserved segments

```
                    tccCAGTTATGTCAGgggacacgagcatgcagagac
                       ||||||||||||
aattgccgccgtcgttttcagCAGTTATGTCAGatc
```



local alignment to detect regulatory sites



Global Alignment

Global Alignment Problem: Find the highest-scoring alignment between two strings by using a scoring matrix.

- **Input:** Strings *v* and *w* as well as a matrix ***score***.
- **Output:** An alignment of *v* and *w* whose alignment score (as defined by the scoring matrix ***score***) is maximal among all possible alignments of *v* and *w*.

The Needleman-Wunsch Algorithm (Global alignment)

1. Initialization.
   a. $F(0, 0) = 0$
   b. $F(0, j) = -j \times d$
   c. $F(i, 0) = -i \times d$

2. Main Iteration. Filling-in partial alignments

d is a penalty

   a. For each $i = 1 \ldots\ldots M$
      For each $j = 1 \ldots\ldots N$

$$F(i, j) = \max \begin{cases} F(i-1, j) - d & \text{[case 1]} \\ F(i, j-1) - d & \text{[case 2]} \\ F(i-1, j-1) + s(x_i, y_j) & \text{[case 3]} \end{cases}$$

$$Ptr(i,j) = \begin{cases} \text{UP}, & \text{if [case 1]} \\ \text{LEFT} & \text{if [case 2]} \\ \text{DIAG} & \text{if [case 3]} \end{cases}$$

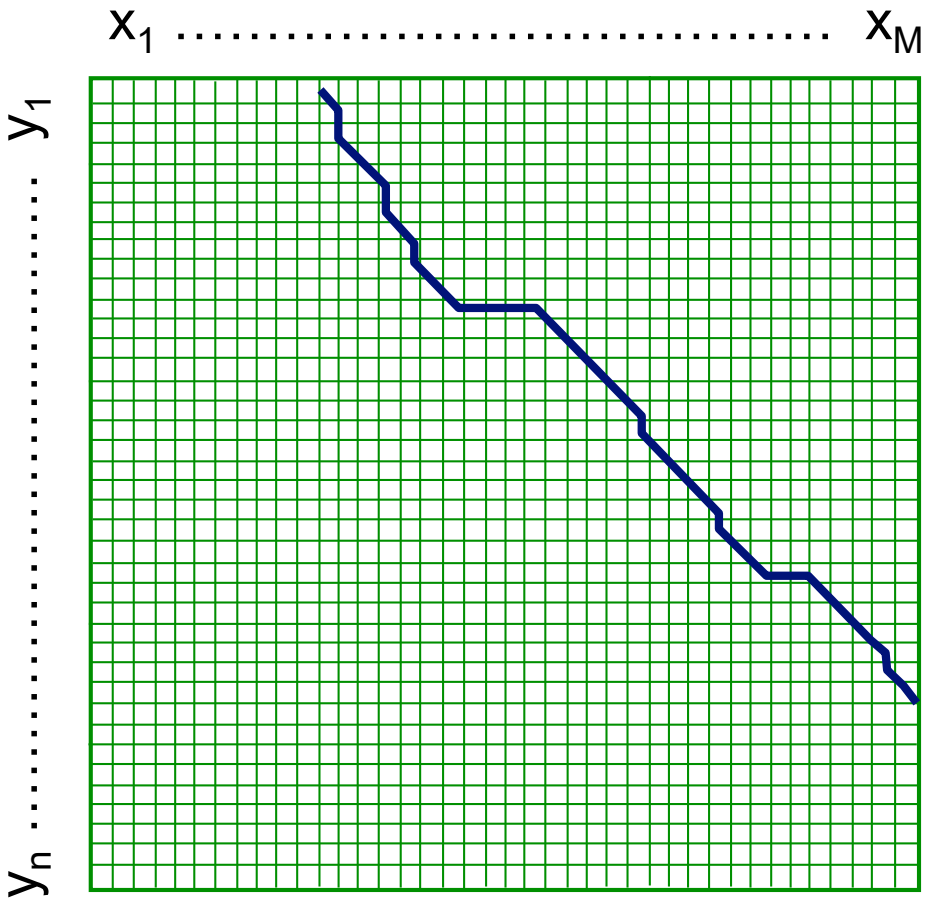
3. Termination. F(M, N) is the optimal score, and from Ptr(M, N) can trace back optimal alignment

Complexity: Space: O(mn); Time: O(mn)
Filling the matrix O(mn)
Backtrace O(m+n)

The Overlap Detection variant

Maybe it is OK to have an unlimited # of gaps in the beginning and end:

```
----------CTATCACCTGACCTCCAGGCCGATGCCCCTTCCGGC
GCGAGTTCATCTATCAC--GACCGC--GGTCG--------------
```



Changes:

1. Initialization

   For all i, j,

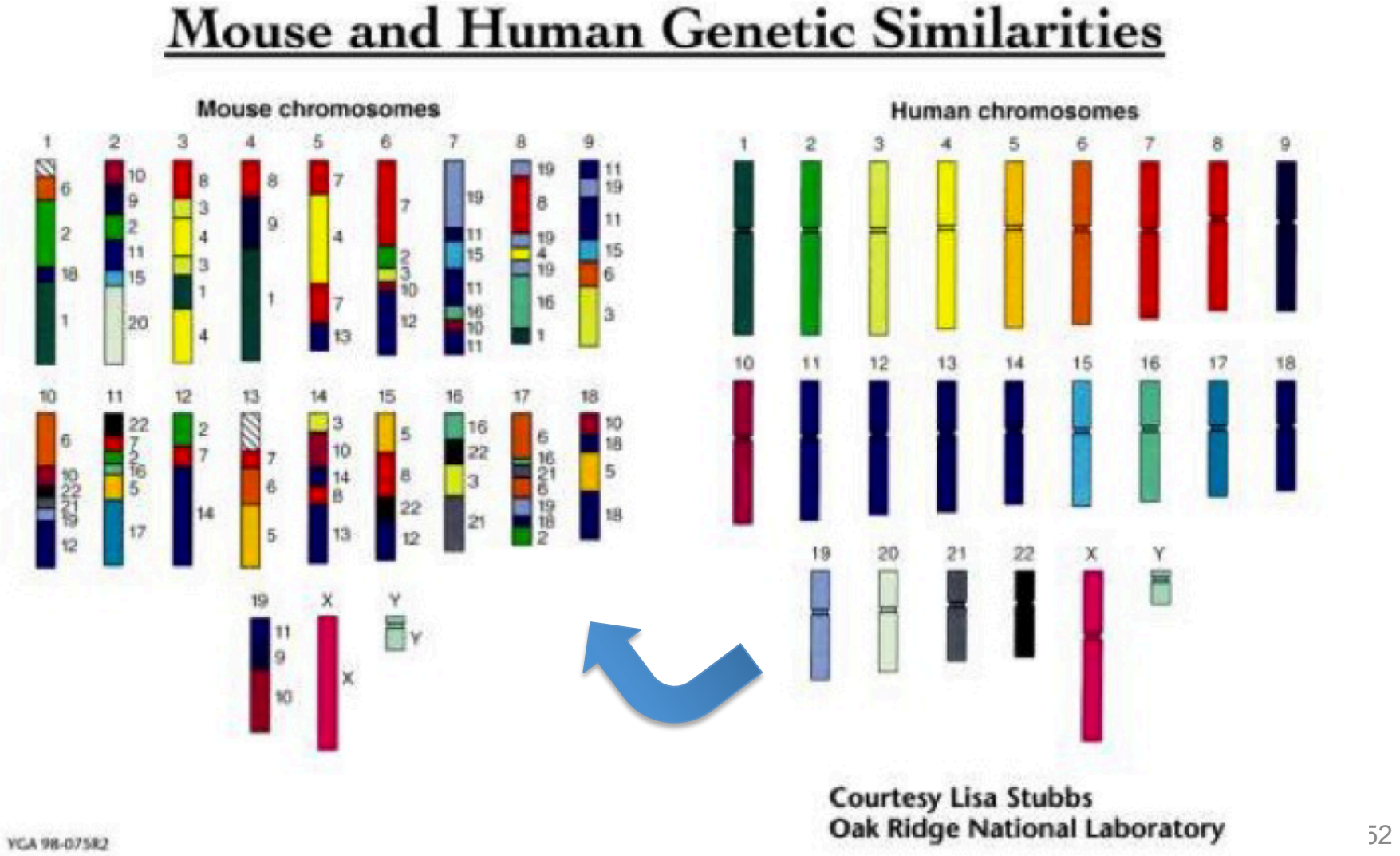
   $F(i, 0) = 0$

   $F(0, j) = 0$

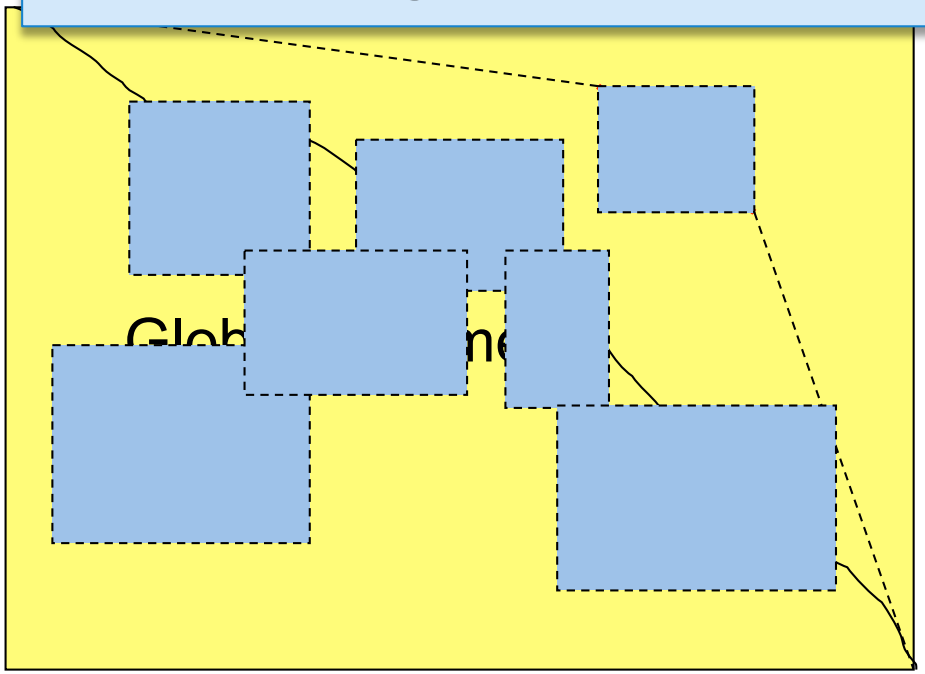
2. Termination

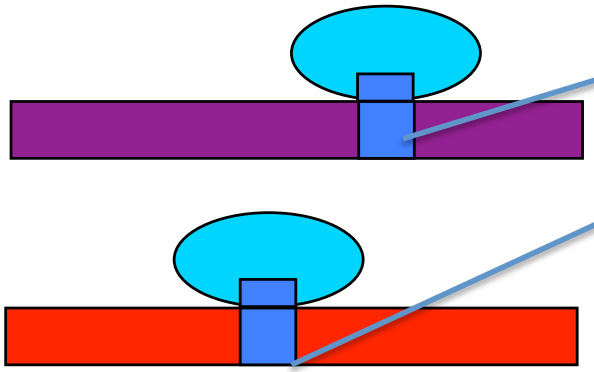
$$F_{OPT} = \max \begin{cases} \max_i F(i, N) \\ \max_j F(M, j) \end{cases}$$

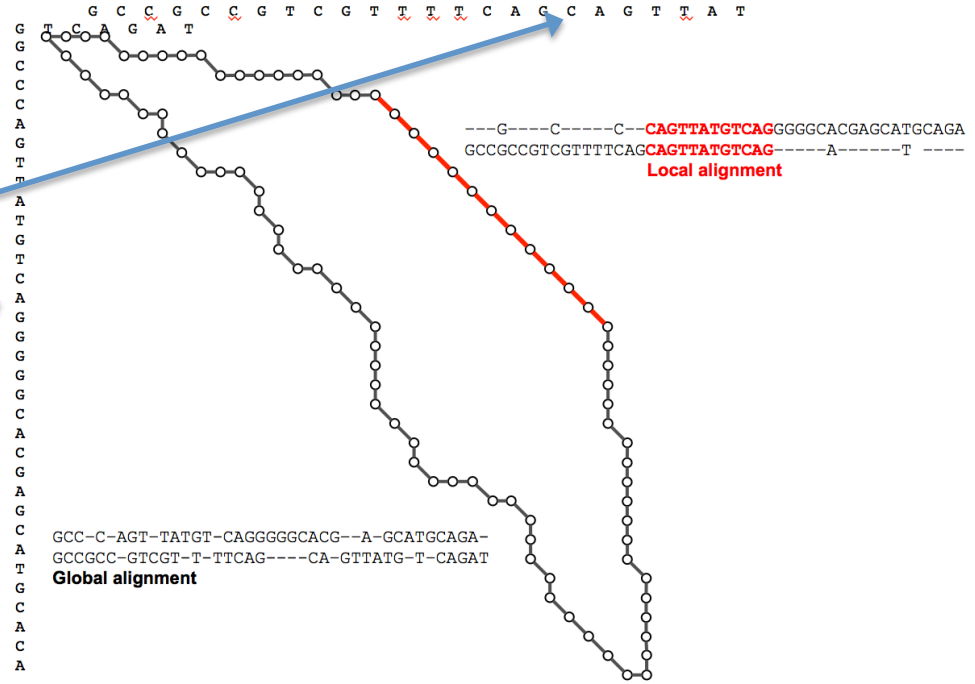
Can we use a similar algorithm to align entire genomes?



Local Alignment= Global Alignment in a subrectangle







local alignment to detect regulatory sites

Local Alignment Problem

Local Alignment Problem: Find the highest-scoring local alignment between two strings.

- **Input:** Strings *v* and *w* as well as a matrix ***score***.
- **Output:** Substrings of *v* and *w* whose global alignment (as defined by the matrix ***score***), is maximal among all global alignments of all substrings of *v* and *w*.

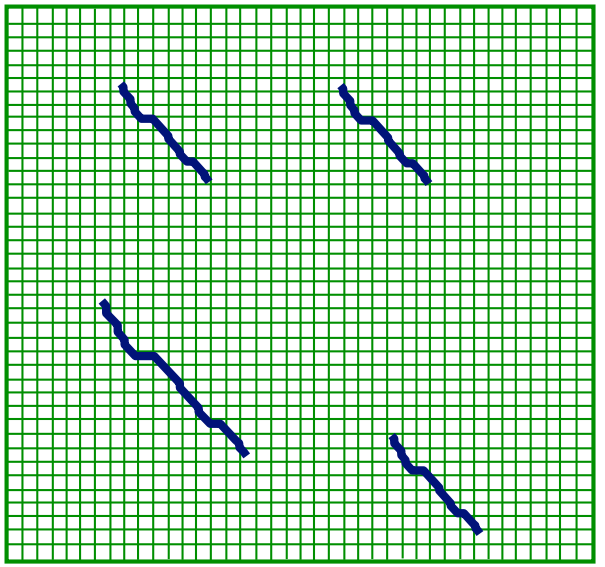
The local alignment: Smith-Waterman algorithm

T.F. Smith, M.S.Waterman, Identification of common molecular subsequences, J Mol Biol vol 147,195-197, 1981.

Idea: Ignore badly aligning regions: Modifications to Needleman-Wunsch

e.g. x = aaaacc**cccgggg**

y = **cccggg**aaccaacc



Initialization: $F(0,0)=F(0,j)=F(i,0)=0$

Iteration:

$$F(i,j) = \max \begin{cases} 0 \\ F(i-1,j) - d \\ F(i,j-1) - d \\ F(i-1,j-1) + s(x_i, y_j) \end{cases}$$

Termination:

1. If we want the best local alignment…

   $F_{OPT} = \max_{i,j} F(i,j)$

2. If we want all local alignments scoring > t

   For all i, j find F(i, j) > t, and trace back

David Waterman

Which Alignment is Better?

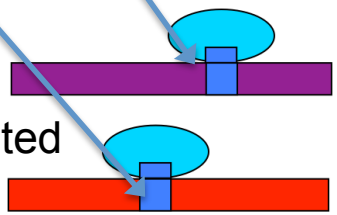
- Alignment 1: score = 22 (matches) - 20 (indels)=2.

```
GCC-C-AGT--TATGT-CAGGGGGCACG--A-GCATGCAGA-
GCCGCC-GTCGT-T-TTCAG----CA-GTTATG--T-CAGAT
```

- Alignment 2: score = 17 (matches) - 30 (indels)=-13.

```
---G----C-----C--CAGTTATGTCAGGGGGCACGAGCATGCAGA
GCCGCCGTCGTTTTCAGCAGTTATGTCAG-----A------T-----
```

local alignment



the local alignment detects a biological finding: two genes are regulated by he same protein

Scoring Gaps

- We previously assigned a fixed penalty σ to each indel.
- However, this fixed penalty may be too severe for a series of 100 consecutive indels.
- A series of k indels often represents a single evolutionary event (**gap**) rather than k events:

two gaps
(lower score)

```
GATCCAG
GA-C-AG
```

```
GATCCAG
GA--CAG
```

a single gap
(higher score)
or maybe 2 events

Mismatches and Indel Penalties

#matches – μ · #mismatches – σ · #indels

```
A T - G T T A T A
A T C G T - C - C
+1+1-2+1+1-2-3-2-3=-7
```

	A	C	G	T	–
A	+1	$-\mu$	$-\mu$	$-\mu$	$-\sigma$
C	$-\mu$	+1	$-\mu$	$-\mu$	$-\sigma$
G	$-\mu$	$-\mu$	+1	$-\mu$	$-\sigma$
T	$-\mu$	$-\mu$	$-\mu$	+1	$-\sigma$
–	$-\sigma$	$-\sigma$	$-\sigma$	$-\sigma$	

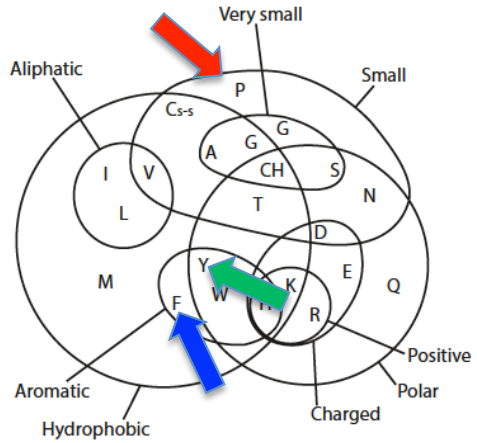
Scoring matrix

	A	C	G	T	–
A	+1	−3	−5	−1	−3
C	−4	+1	−3	−2	−3
G	−9	−7	+1	−1	−3
T	−3	−5	−8	+1	−4
–	−4	−2	−2	−1	

Even more general scoring matrix

How to compare amino acids: scoring matrices

C	Cys	12																		
S	Ser	0	2																	
T	Thr	-2	1	3																
P	Pro	-3	1	0	6															
A	Ala	-2	1	1	1	2														
G	Gly	-3	1	0	-1	1	5													
N	Asn	-4	1	0	-1	0	0	2												
D	Asp	-5	0	0	-1	0	1	2	4											
E	Glu	-5	0	0	-1	0	0	1	3	4										
Q	Gln	-5	-1	-1	0	0	-1	1	2	2	4									
H	His	-3	-1	-1	0	-1	-2	2	1	1	3	6								
R	Arg	-4	0	-1	0	-2	-3	0	-1	-1	1	2	6							
K	Lys	-5	0	0	-1	-1	-2	1	0	0	1	0	3	5						
M	Met	-5	-2	-1	-2	-1	-3	-2	-3	-2	-1	-2	0	0	6					
I	Ile	-2	-1	0	-2	-1	-3	-2	-2	-2	-2	-2	-2	-2	2	5				
L	Leu	-6	-3	-2	-3	-2	-4	-3	-4	-3	-2	-2	-3	-3	4	2	6			
V	Val	-2	-1	0	-1	0	-1	-2	-2	-2	-2	-2	-2	-2	2	4	2	4		
F	Phe	-4	-3	-3	-5	-5	-5	-4	-6	-5	-5	-2	-4	-5	0	1	2	-1	9	
Y	Tyr	0	-3	-3	-5	-3	-5	-2	-4	-4	-4	0	-4	-4	-2	-1	-1	-2	7	10
W	Trp	-8	-2	-5	-6	-6	-7	-4	-7	-7	-5	-3	2	-3	-4	-5	-2	-6	0	0	17
		C	S	T	P	A	G	N	D	E	Q	H	R	K	M	I	L	V	F	Y	W



Margaret Dayhoff

example: Y (Tyr) often mutates into F (score +7) but rarely mutates into P (score -5)

More Adequate Gap Penalties

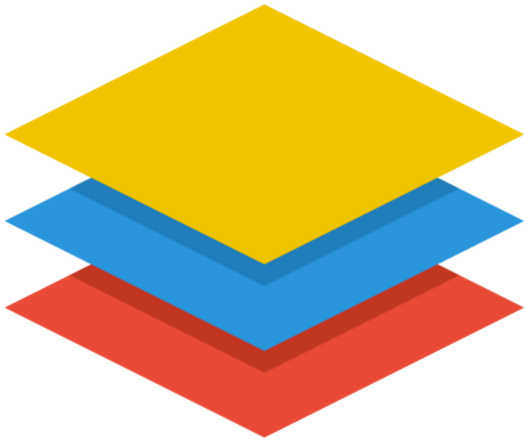
Affine gap penalty for a gap of length *k*: $\sigma+\varepsilon\cdot(k-1)$

σ - the **gap opening penalty**

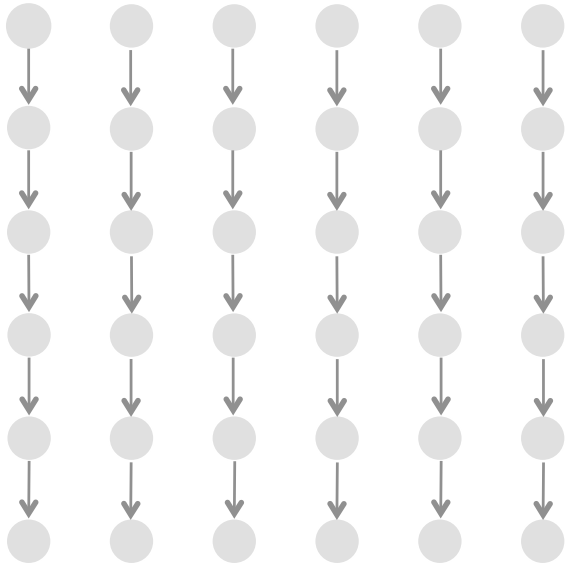
ε - the **gap extension penalty**

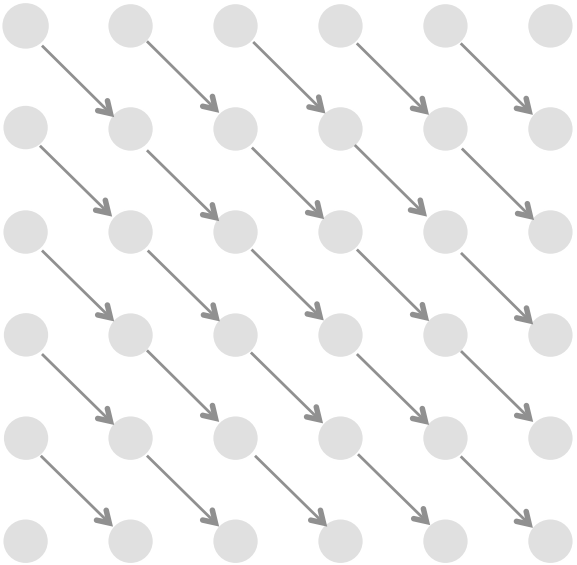
$\sigma > \varepsilon$, since starting a gap should be penalized more than extending it.

- Thinking on 3 levels

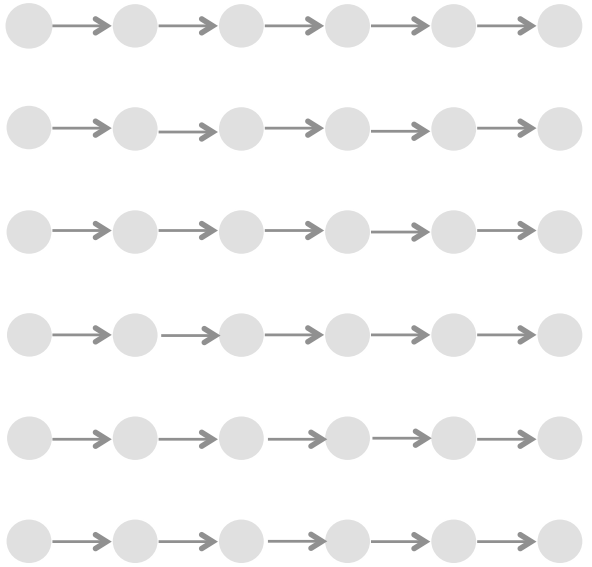


bottom level
(insertions)

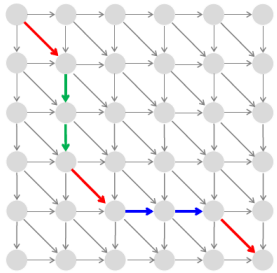




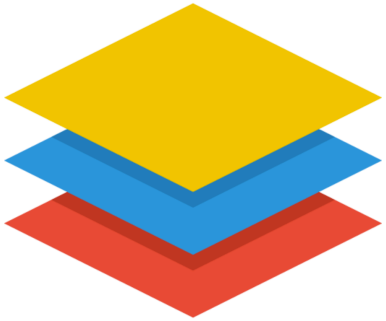
middle level
(matches/mismatches)

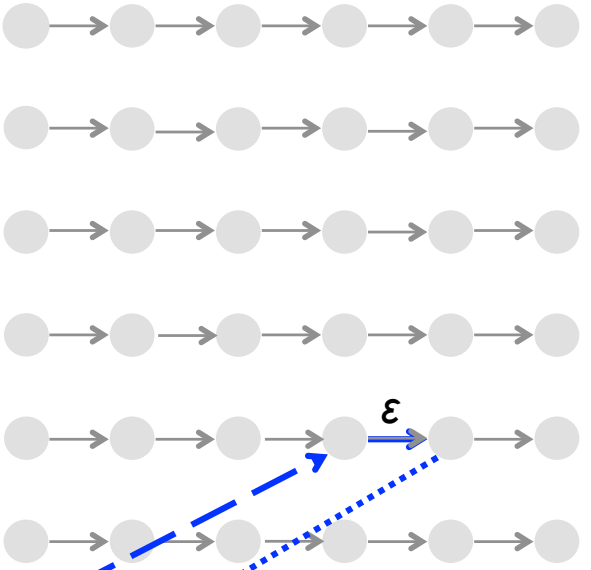


upper level
(deletions)



How can we emulate this path in the 3-level?





$lower_{i,j} = \max \{ lower_{i-1,j} - \varepsilon,\ middle_{i-1,j} - \sigma \}$

$upper_{i,j} = \max \{ upper_{i,j-1} - \varepsilon,\ middle_{i,j-1} - \sigma \}$

$middle_{i,j} = \max \{ lower_{i,j},\ middle_{i-1,j-1} + score(v_i, w_j),\ upper_{i,j} \}$

ε

σ

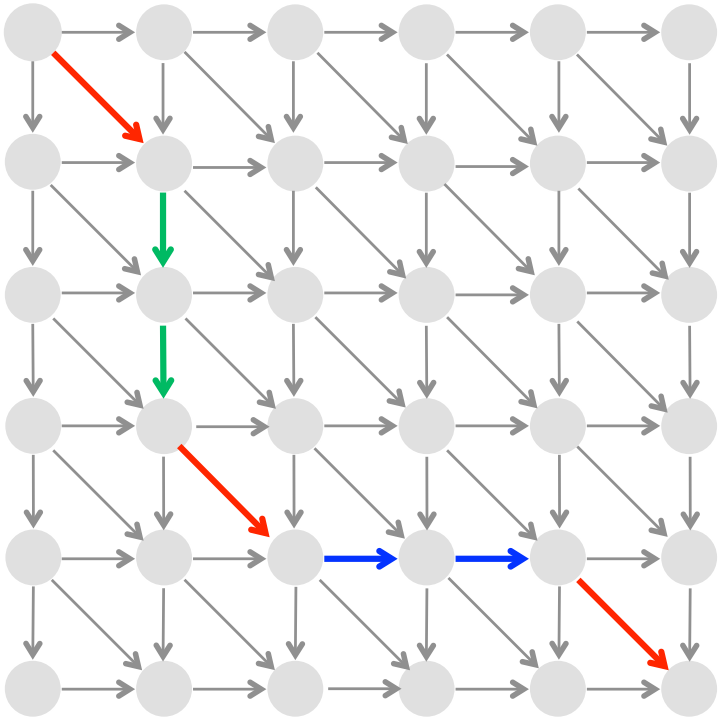
0

ε

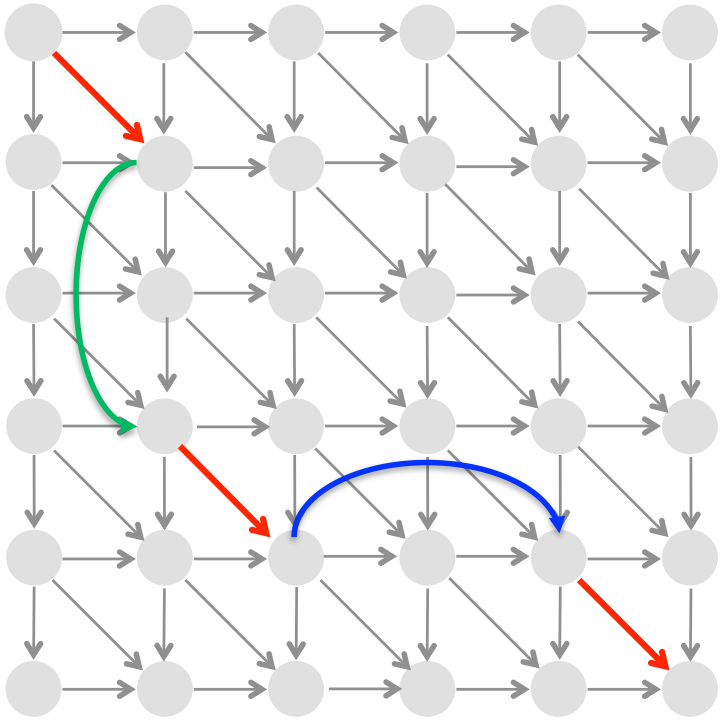
σ

0

• Modelling Affine Gap Penalties by Long Edges



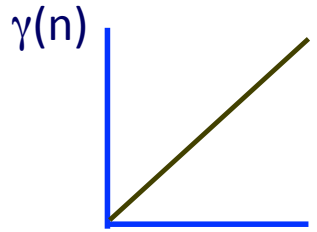
double gap: 2 events



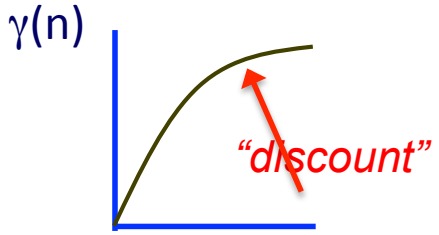
double gap: 1 event

Alignment with gaps

Current model: a gap of length n incurs penalty $n \times d$
Gaps usually occur in bunches so we use a convex gap penalty function:
$\gamma(n)$: for all n, $\gamma(n+1) - \gamma(n) \leq \gamma(n) - \gamma(n-1)$



Initialization: same



Iteration:

$$F(i,j) = \max \begin{cases} F(i-1, j-1) + s(x_i, y_j) \\ \max_{k=0\ldots i-1} F(k,j) - \gamma(i-k) \\ \max_{k=0\ldots j-1} F(i,k) - \gamma(j-k) \end{cases}$$

Termination: same

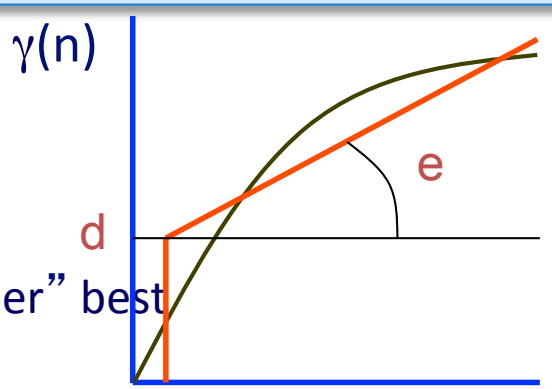
Running Time: $O(N^2M)$ (assume N>M)

Space: $O(NM)$

A compromise: affine gaps

$\gamma(n) = d + (n - 1) \times e$

gap open | gap extend



To compute optimal alignment, at position i,j, need to "remember" best score if gap is open and best score if gap is not open

F(i, j):score of alignment $x_1 \ldots x_i$ to $y_1 \ldots y_j$ if x_i aligns to y_j
G(i, j):score if x_i, or y_j, aligns to a gap

Initialization: $F(i, 0) = d + (i - 1) \times e$; $F(0, j) = d + (j - 1) \times e$

Iteration:

$$F(i, j) = \max \begin{cases} F(i-1, j-1) + s(x_i, y_j) \\ G(i-1, j-1) + s(x_i, y_j) \\ F(i-1, j) - d \\ F(i, j-1) - d \end{cases}$$

$$G(i, j) = \max \begin{cases} G(i, j-1) - e \\ G(i-1, j) - e \end{cases}$$

Termination: same

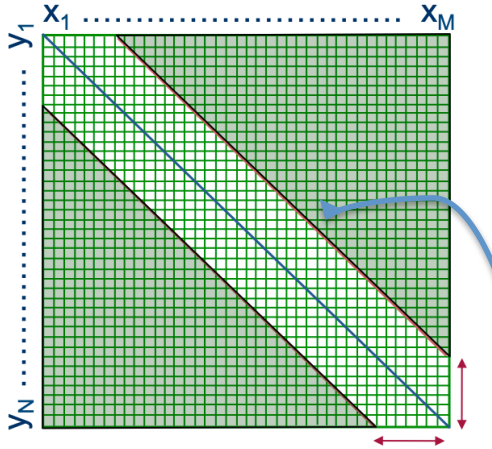
Banded DP: a special case

Assume we know that x and y are very similar; If the optimal alignment of x and y has few gaps, then the path of the alignment will be close to the diagonal

Assumption: # gaps(x, y) < k(N) (say N>M)

x_i
| implies | i – j | < k(N)
y_j

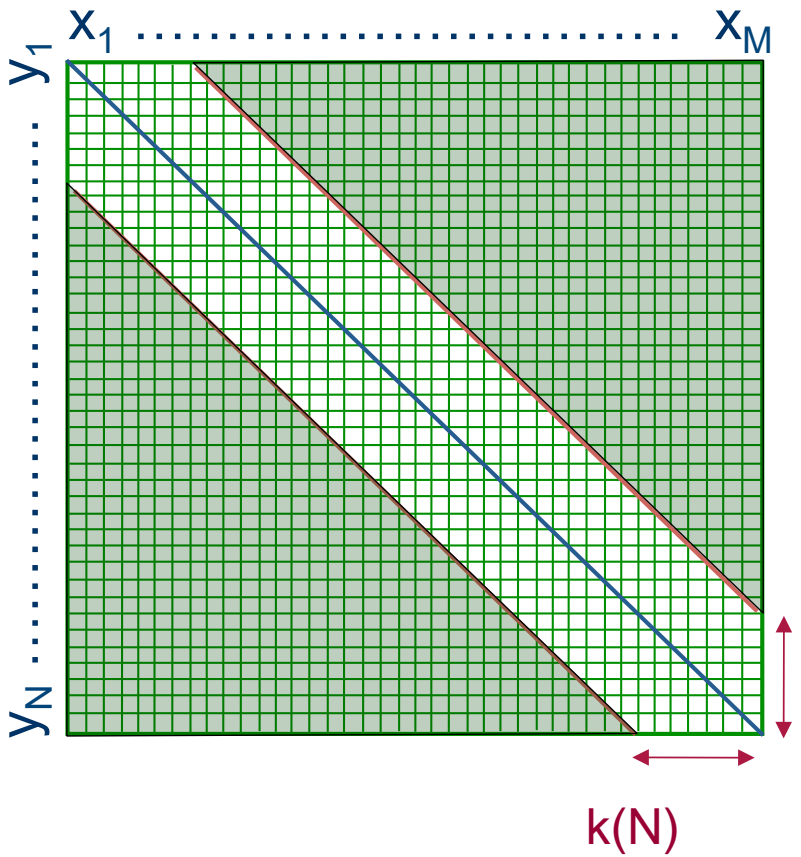
Time, Space: O(N × k(N)) << $O(N^2)$



F[i,i+k/2]	Out of range
F[i+1, i+k/2]	F[i+1, i+k/2 +1]

Note that for diagonals, i-j = constant.

Banded Dynamic Programming



Initialization:

F(i,0), F(0,j) undefined for i, j > k

Iteration:

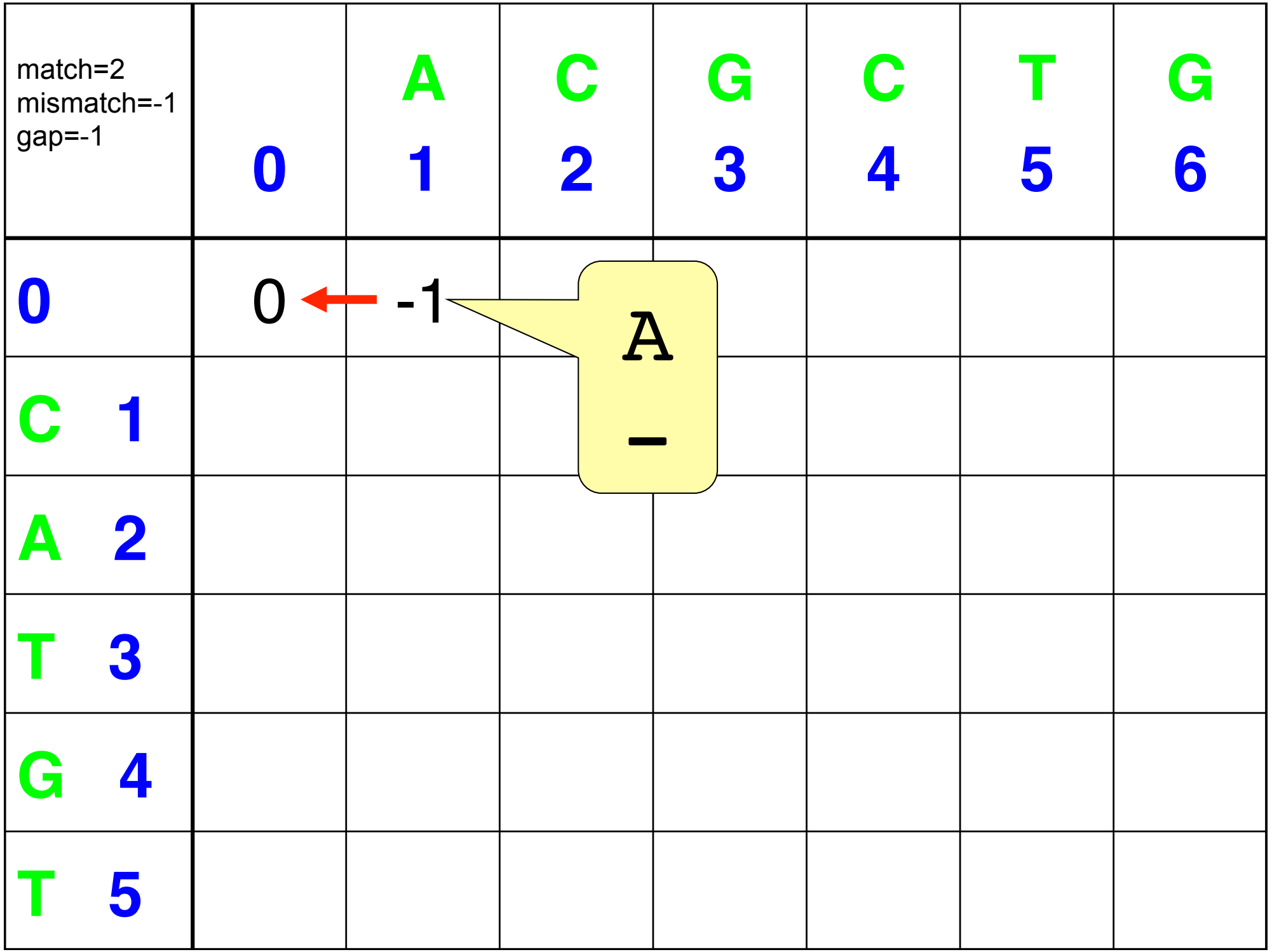
For i = 1…M
  For j = max(1, i – k)…min(N, i+k)

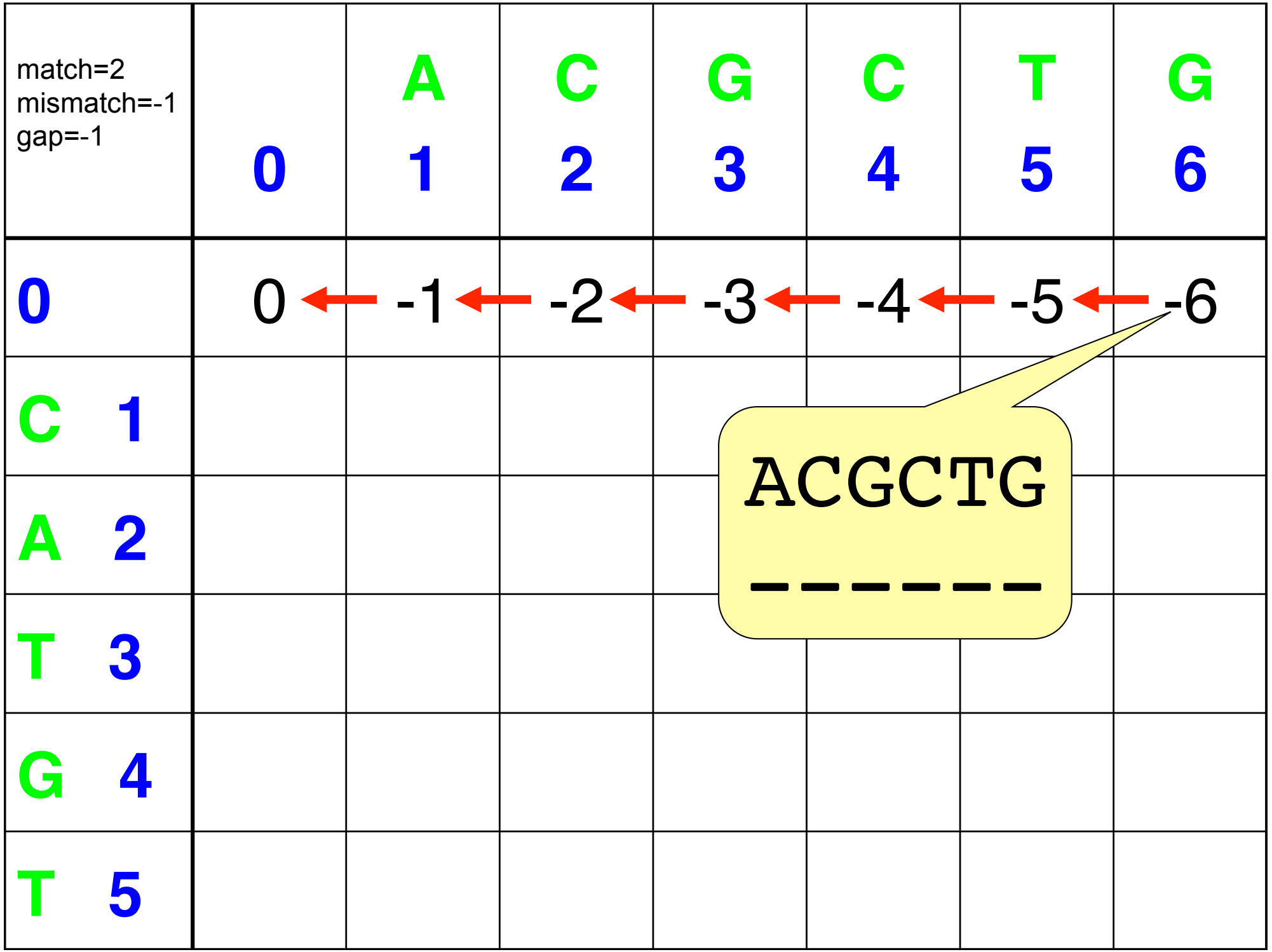
$$F(i, j) = \max \begin{cases} F(i-1, j-1) + s(x_i, y_j) \\ F(i, j-1) - d, \text{ if } j > i - k(N) \\ F(i-1, j) - d, \text{ if } j < i + k(N) \end{cases}$$

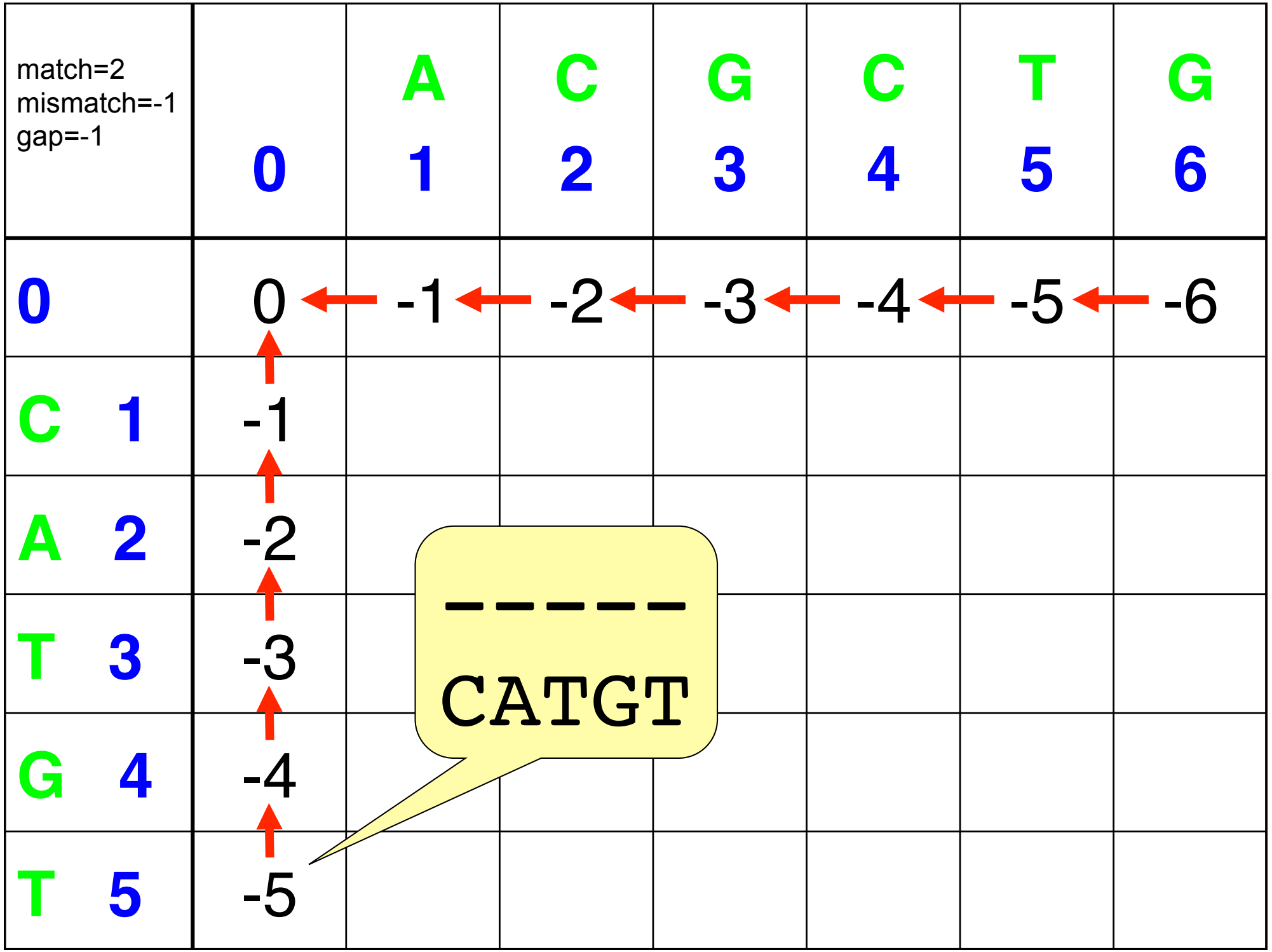
Termination: same

Easy to extend to the affine gap case

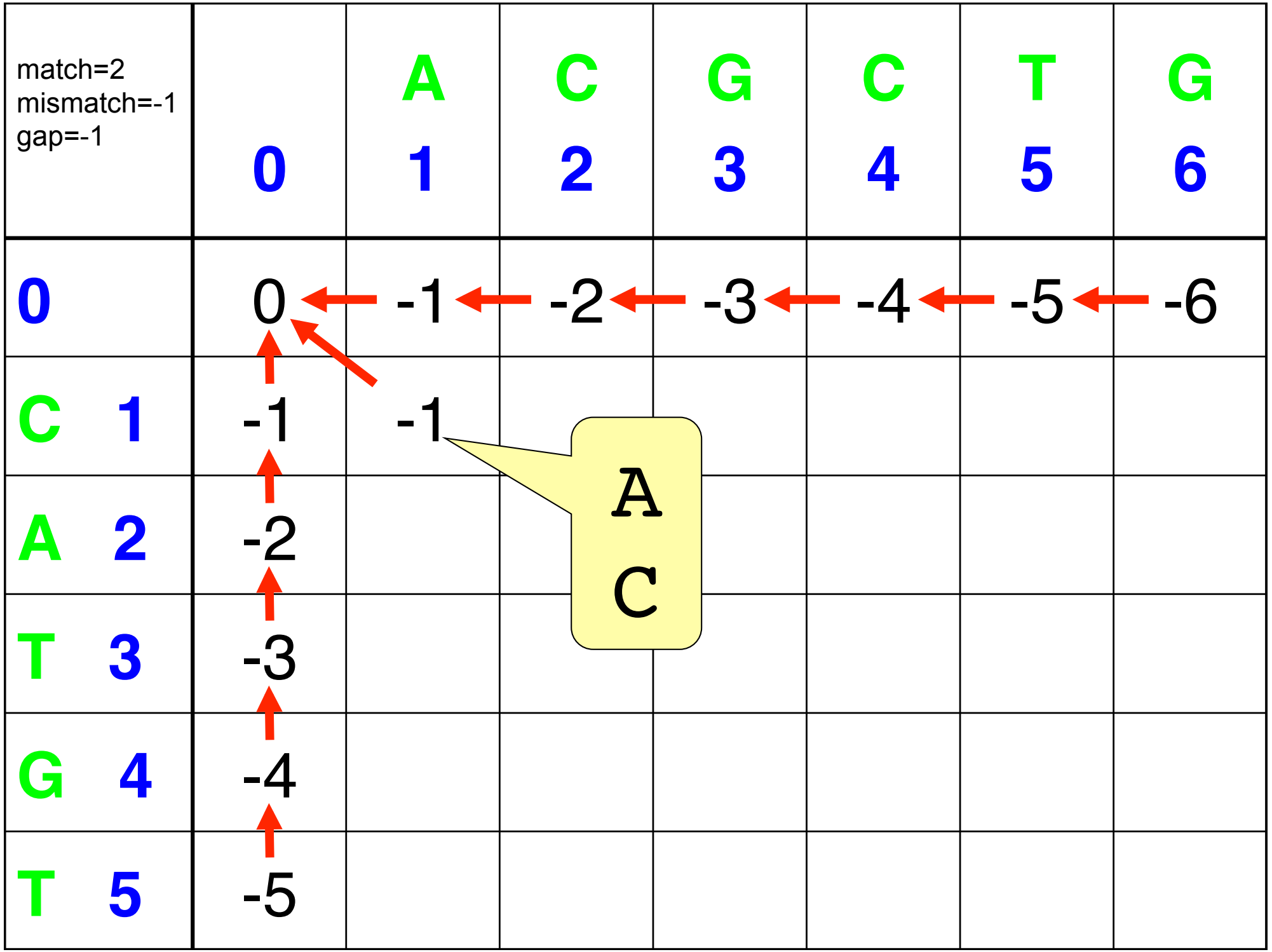
Example global alignment

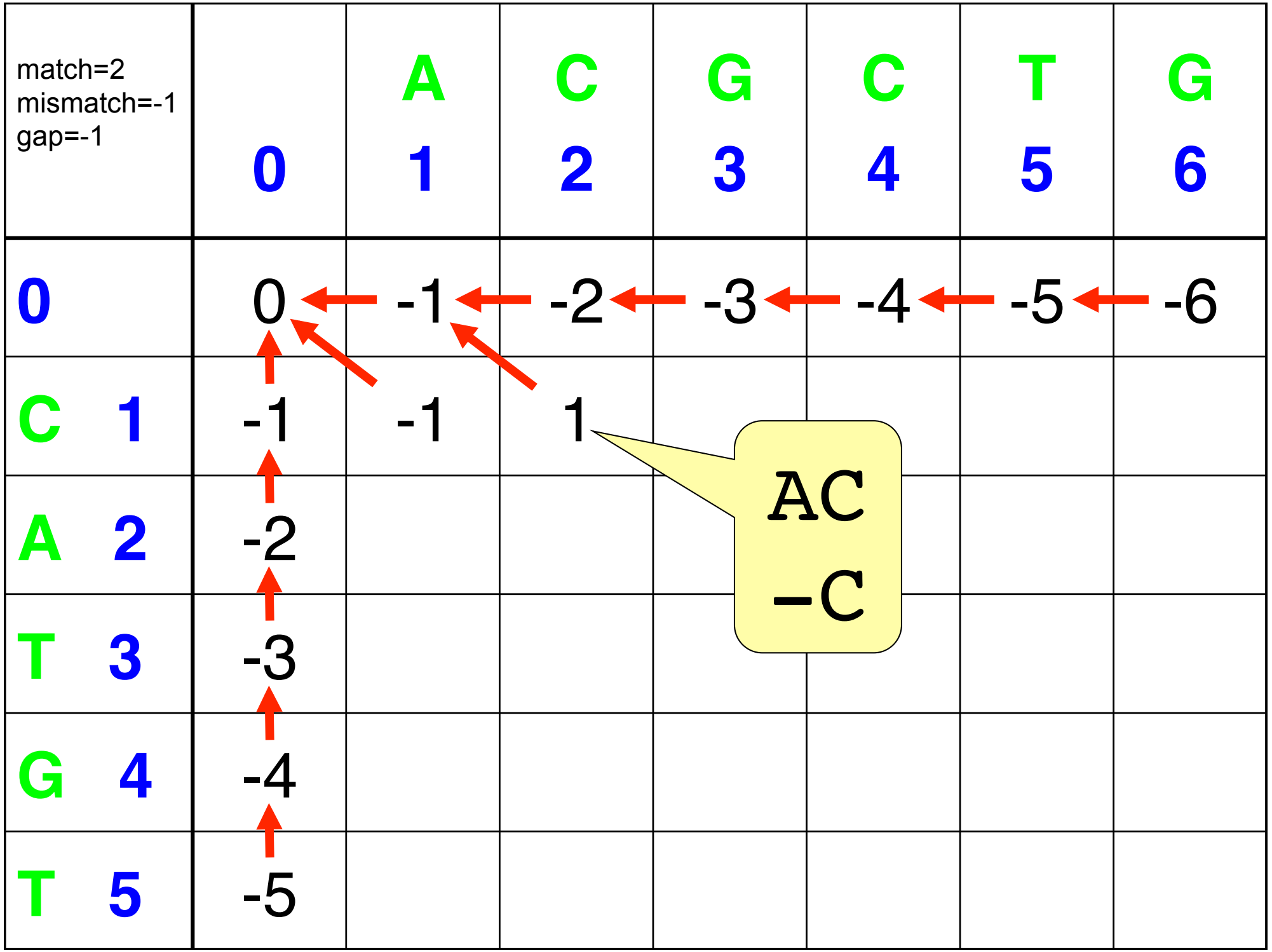

match=2
mismatch=-1
gap=-1
A
C
G
C
T
G
0
1
2
3
4
5
6
0
C 1
A 2
T 3
G 4
T 5
0
-1
A
–

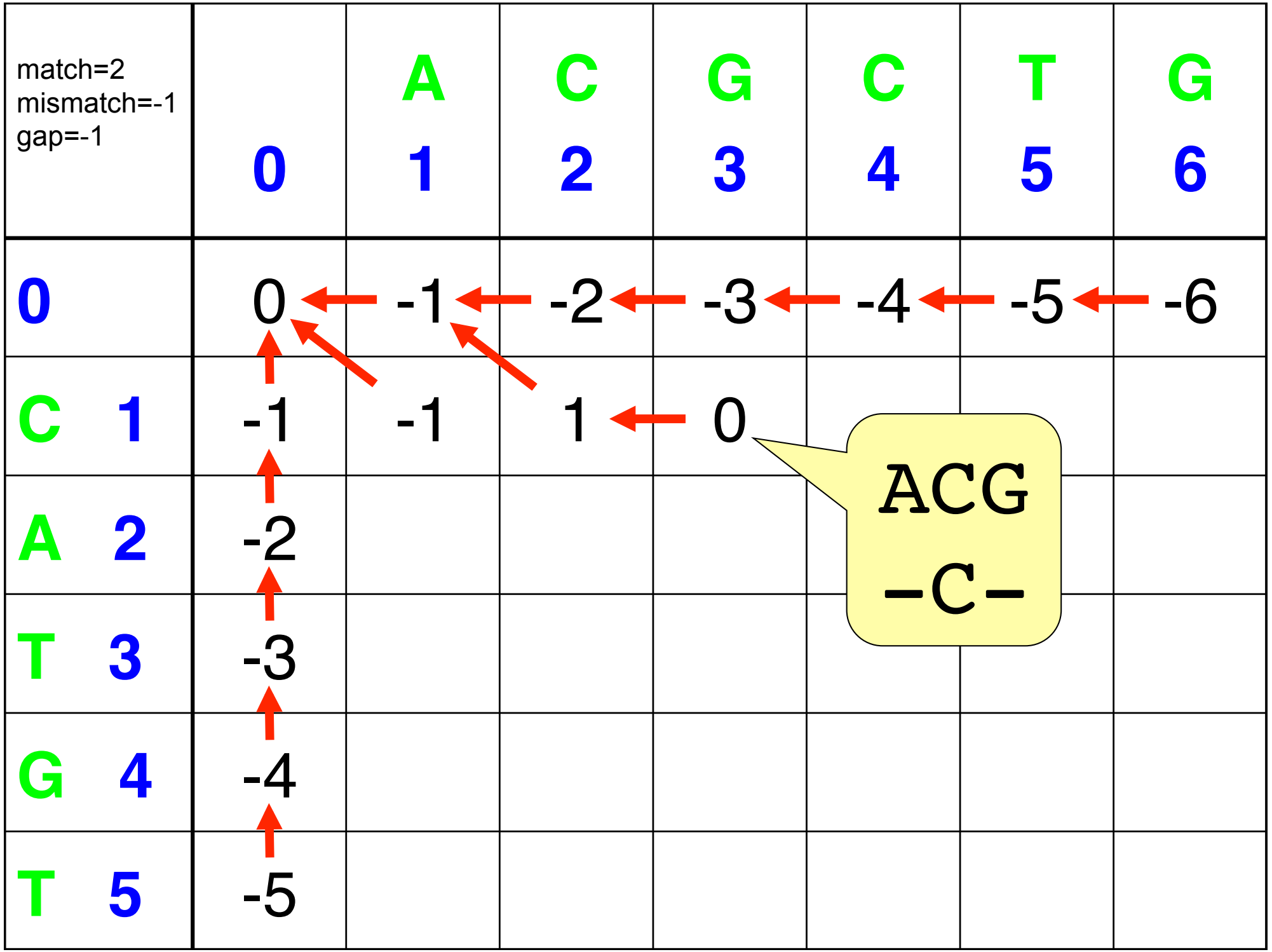

match=2
mismatch=-1
gap=-1
0
A 1
C 2
G 3
C 4
T 5
G 6
0
0 ← -1 ← -2 ← -3 ← -4 ← -5 ← -6
C 1
A 2
T 3
G 4
T 5
ACGCTG

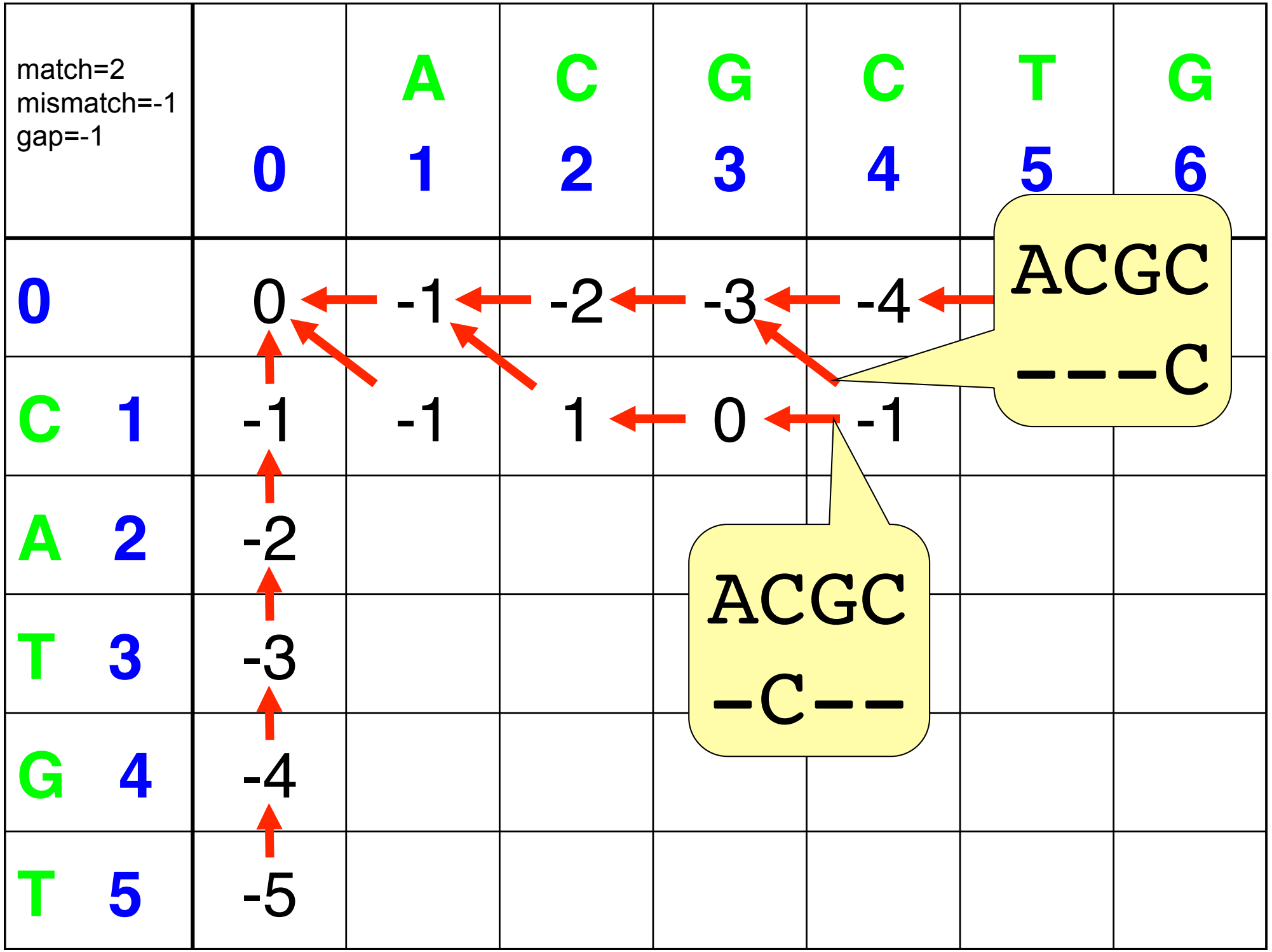

match=2
mismatch=-1
gap=-1
0
A 1
C 2
G 3
C 4
T 5
G 6
0
0 ← -1 ← -2 ← -3 ← -4 ← -5 ← -6
C 1
-1
A 2
-2
T 3
-3
G 4
-4
T 5
-5

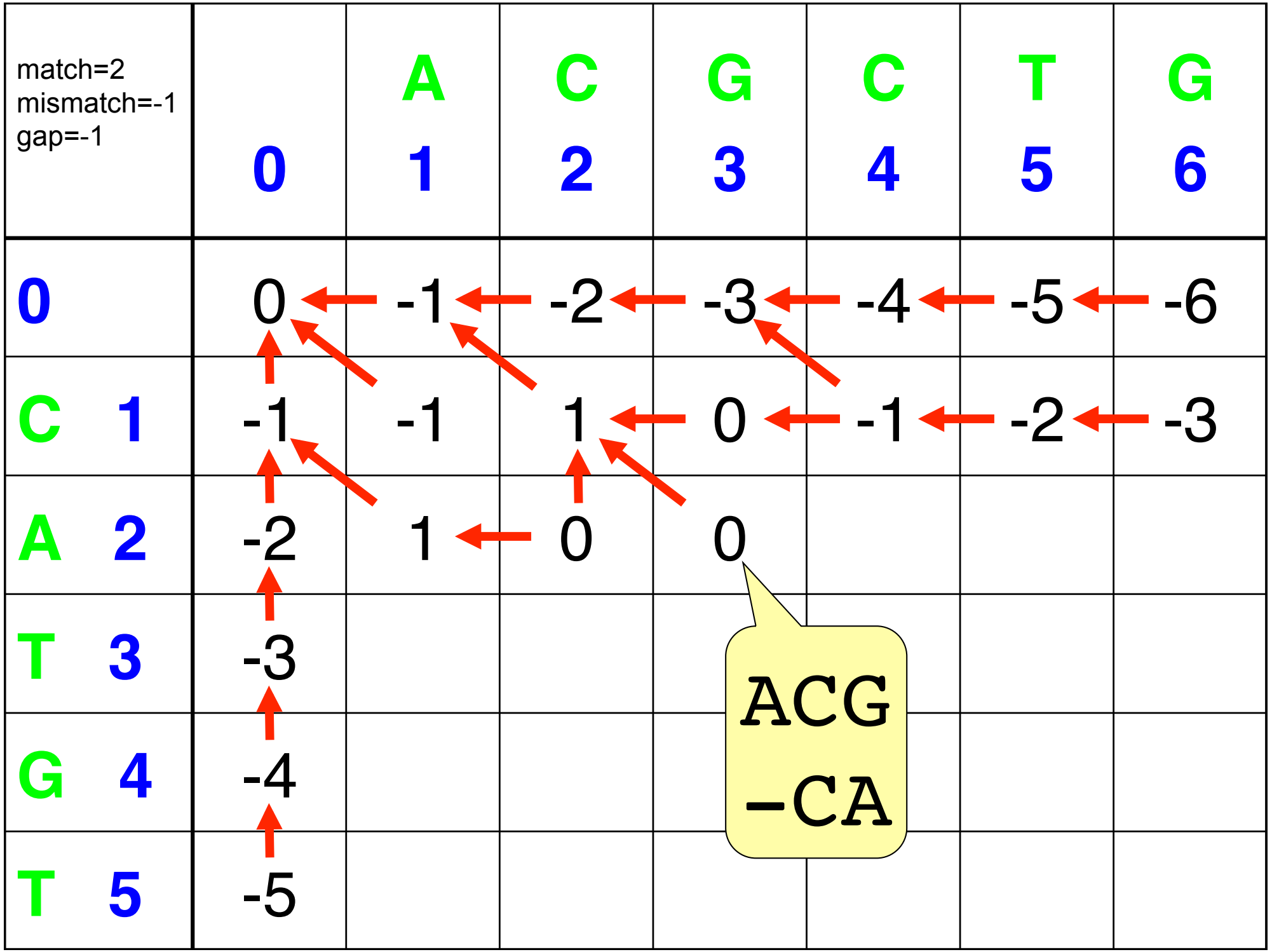
CATGT

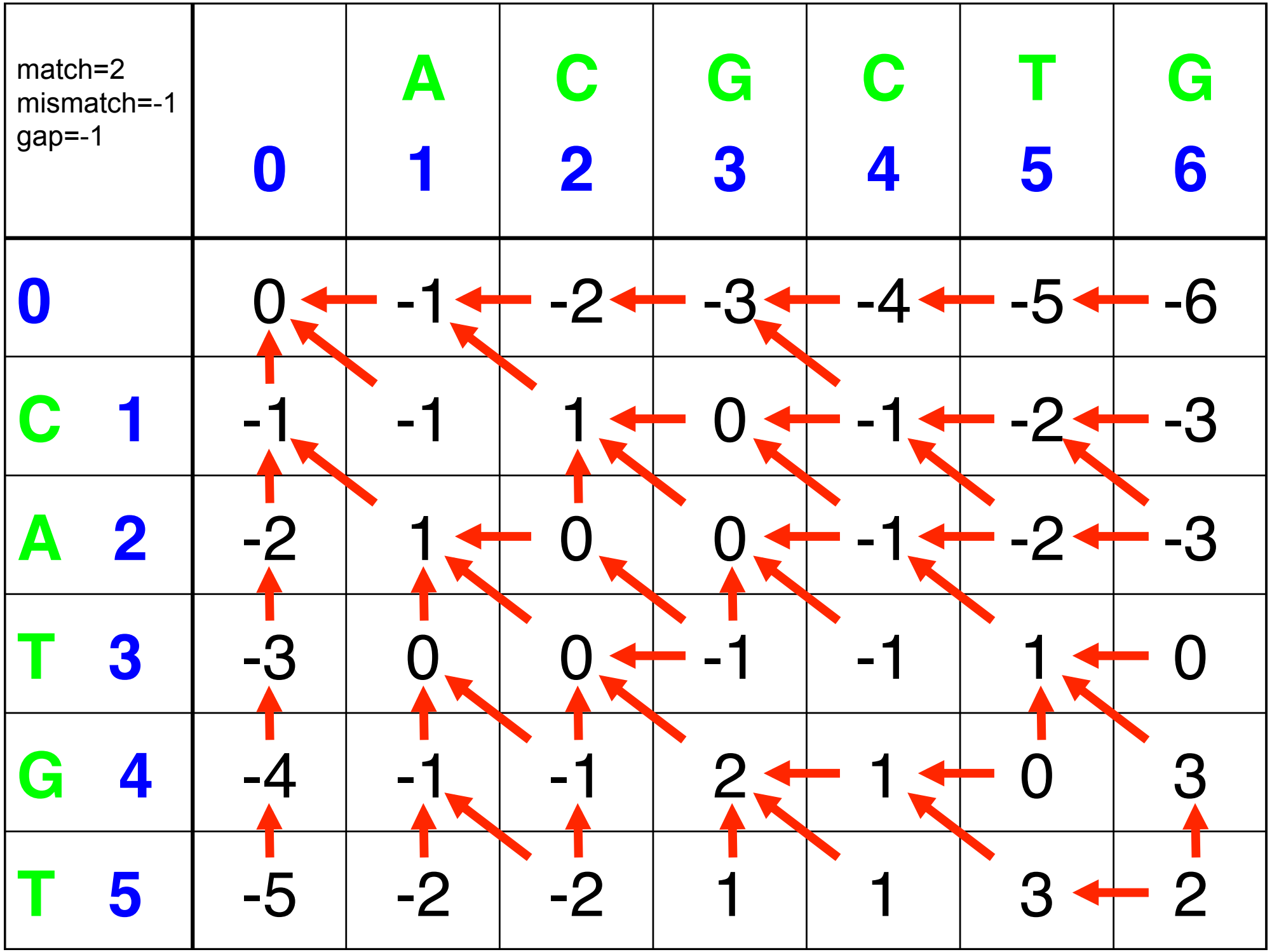

match=2
mismatch=-1
gap=-1
A C G C T G
0 1 2 3 4 5 6
0 0 -1 -2 -3 -4 -5 -6
C 1 -1 -1
A 2 -2
T 3 -3
G 4 -4
T 5 -5
A
C

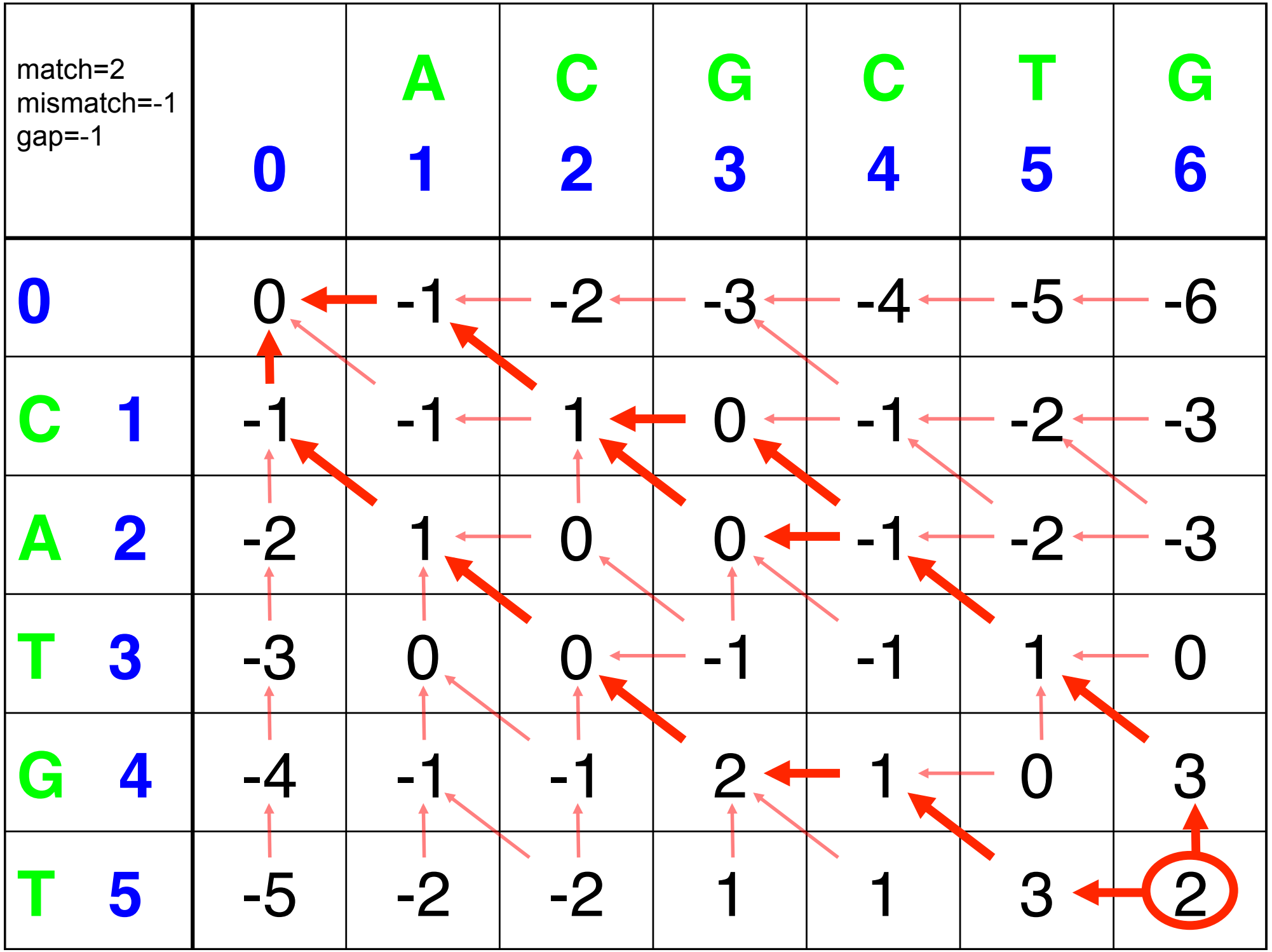

match=2
mismatch=-1
gap=-1
A C G C T G
0 1 2 3 4 5 6
0 0 -1 -2 -3 -4 -5 -6
C 1 -1 -1 1
A 2 -2
T 3 -3
G 4 -4
T 5 -5
AC
–C

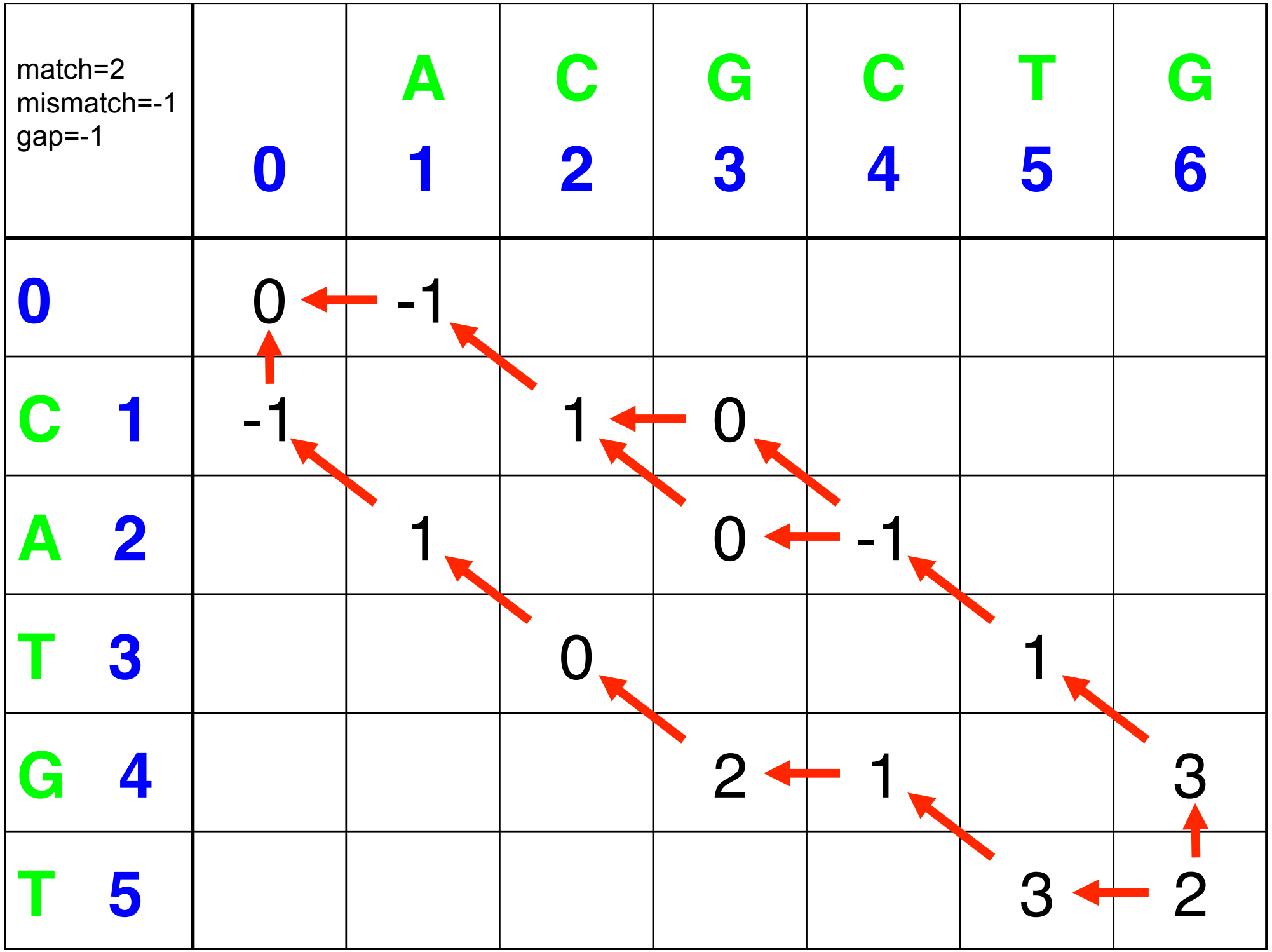

match=2
mismatch=-1
gap=-1
0
A 1
C 2
G 3
C 4
T 5
G 6
0
0 -1 -2 -3 -4 -5 -6
C 1
-1 -1 1 0
A 2
-2
T 3
-3
G 4
-4
T 5
-5
ACG
–C–

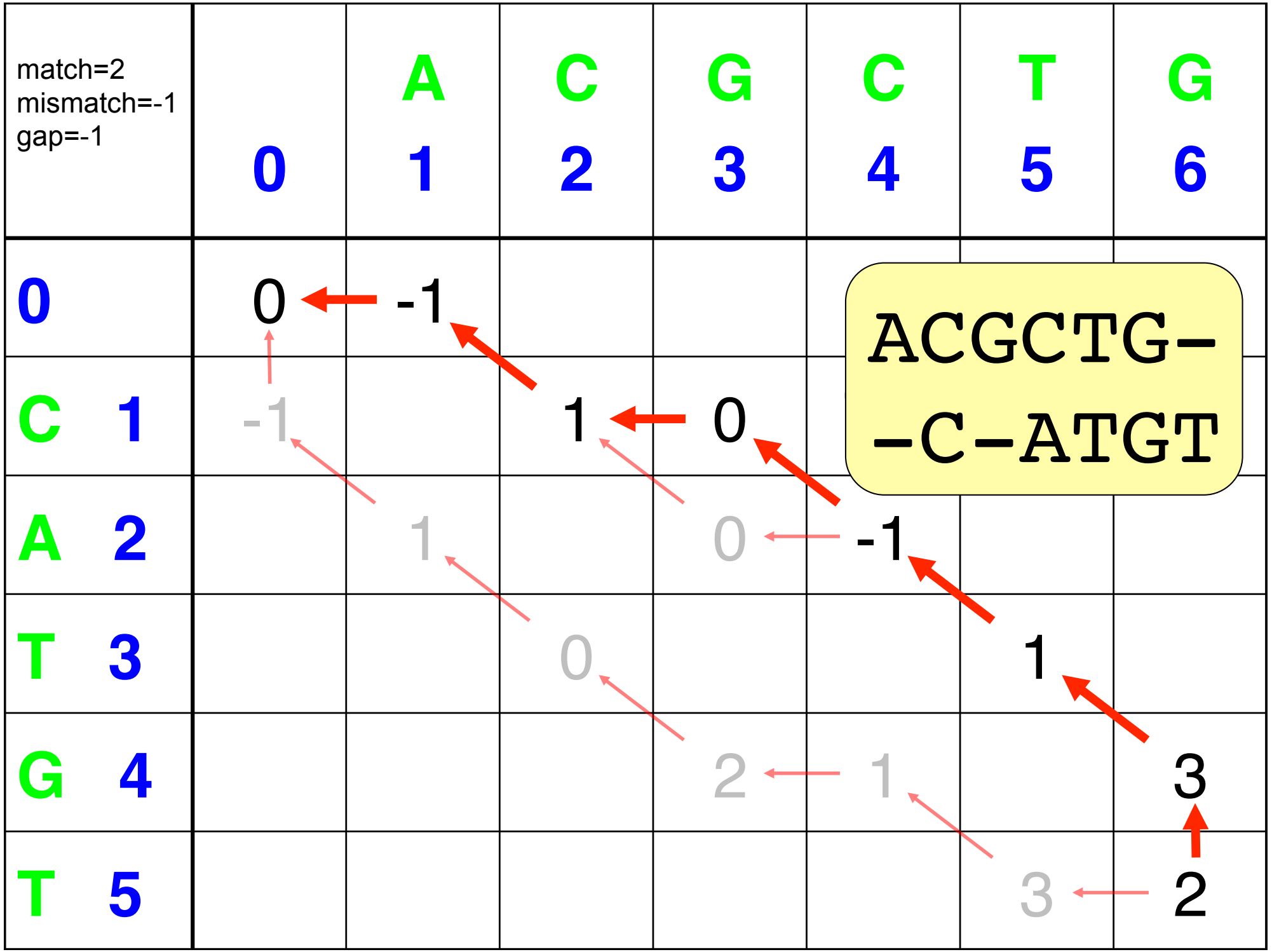

match=2
mismatch=-1
gap=-1
A C G C T G
0 1 2 3 4 5 6
0 0 -1 -2 -3 -4
C 1 -1 -1 1 0 -1
A 2 -2
T 3 -3
G 4 -4
T 5 -5
ACGC
---C
ACGC
-C--

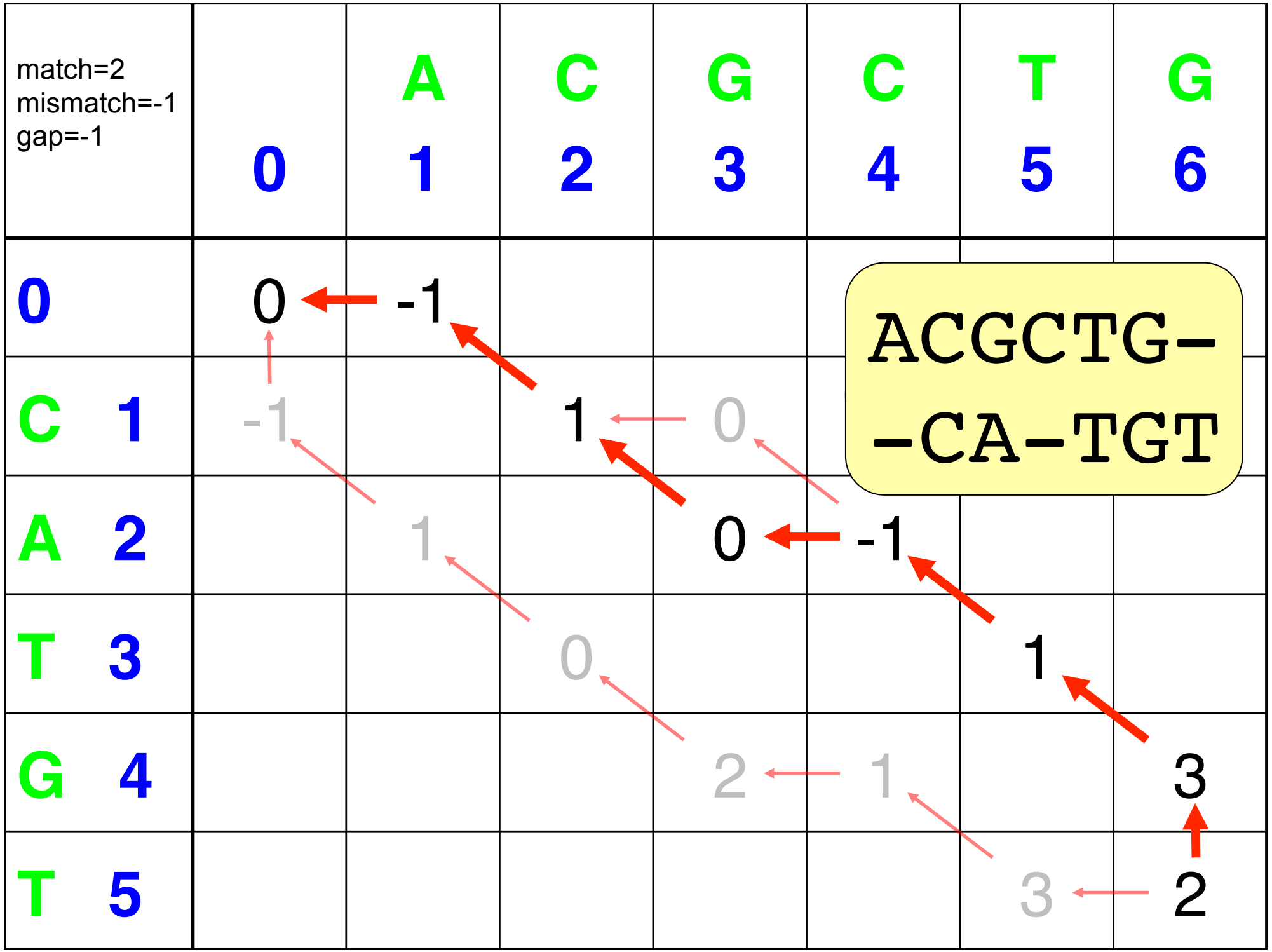

match=2
mismatch=-1
gap=-1
A C G C T G
0 1 2 3 4 5 6
0 0 -1 -2 -3 -4 -5 -6
C 1 -1 -1 1 0 -1 -2 -3
A 2 -2 1 0 0
T 3 -3
G 4 -4
T 5 -5
ACG
–CA

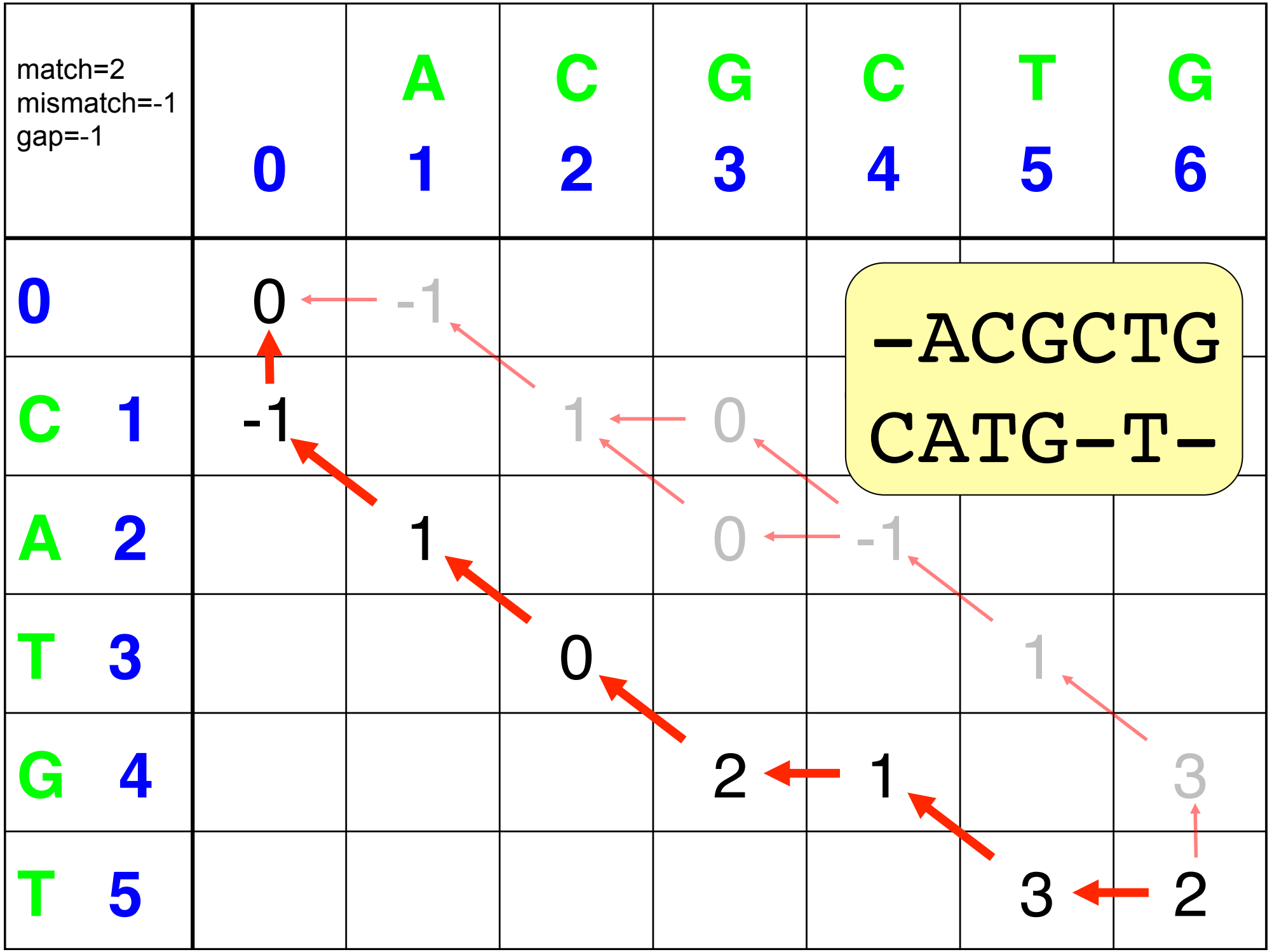

match=2
mismatch=-1
gap=-1
A C G C T G
0 1 2 3 4 5 6
0 0 -1 -2 -3 -4 -5 -6
C 1 -1 -1 1 0 -1 -2 -3
A 2 -2 1 0 0 -1 -2 -3
T 3 -3 0 0 -1 -1 1 0
G 4 -4 -1 -1 2 1 0 3
T 5 -5 -2 -2 1 1 3 2


match=2
mismatch=-1
gap=-1
A C G C T G
0 1 2 3 4 5 6
0 0 -1 -2 -3 -4 -5 -6
C 1 -1 -1 1 0 -1 -2 -3
A 2 -2 1 0 0 -1 -2 -3
T 3 -3 0 0 -1 -1 1 0
G 4 -4 -1 -1 2 1 0 3
T 5 -5 -2 -2 1 1 3 2


match=2
mismatch=-1
gap=-1
0 A C G C T G
0 1 2 3 4 5 6
0 0 -1
C 1 -1 1 0
A 2 1 0 -1
T 3 0 1
G 4 2 1 3
T 5 3 2


match=2
mismatch=-1
gap=-1
0
A 1
C 2
G 3
C 4
T 5
G 6
0
C 1
A 2
T 3
G 4
T 5
0
-1
-1
1
0
1
0
-1
0
1
2
1
3
3
2
ACGCTG-
-C-ATGT


match=2
mismatch=-1
gap=-1
0
A 1
C 2
G 3
C 4
T 5
G 6
0
C 1
A 2
T 3
G 4
T 5
0 -1
-1 1 0
1 0 -1
0 1
1 0
2 1 3
3 2
ACGCTG-
-CA-TGT


match=2
mismatch=-1
gap=-1
0
A 1
C 2
G 3
C 4
T 5
G 6
0
C 1
A 2
T 3
G 4
T 5
-ACGCTG
CATG-T-

Example local alignment

match=1
mismatch=-1
gap=-1

y = TAATA

x = TACTAA

y \ x		A	T	C	T	A	A
	0	**1**	**2**	**3**	**4**	**5**	**6**
0	0	0	0	0	0	0	0
T 1	0						
A 2	0						
A 3	0						
T 4	0						
A 5	0						

Local Alignment Example

match=1
mismatch=-1
gap=-1

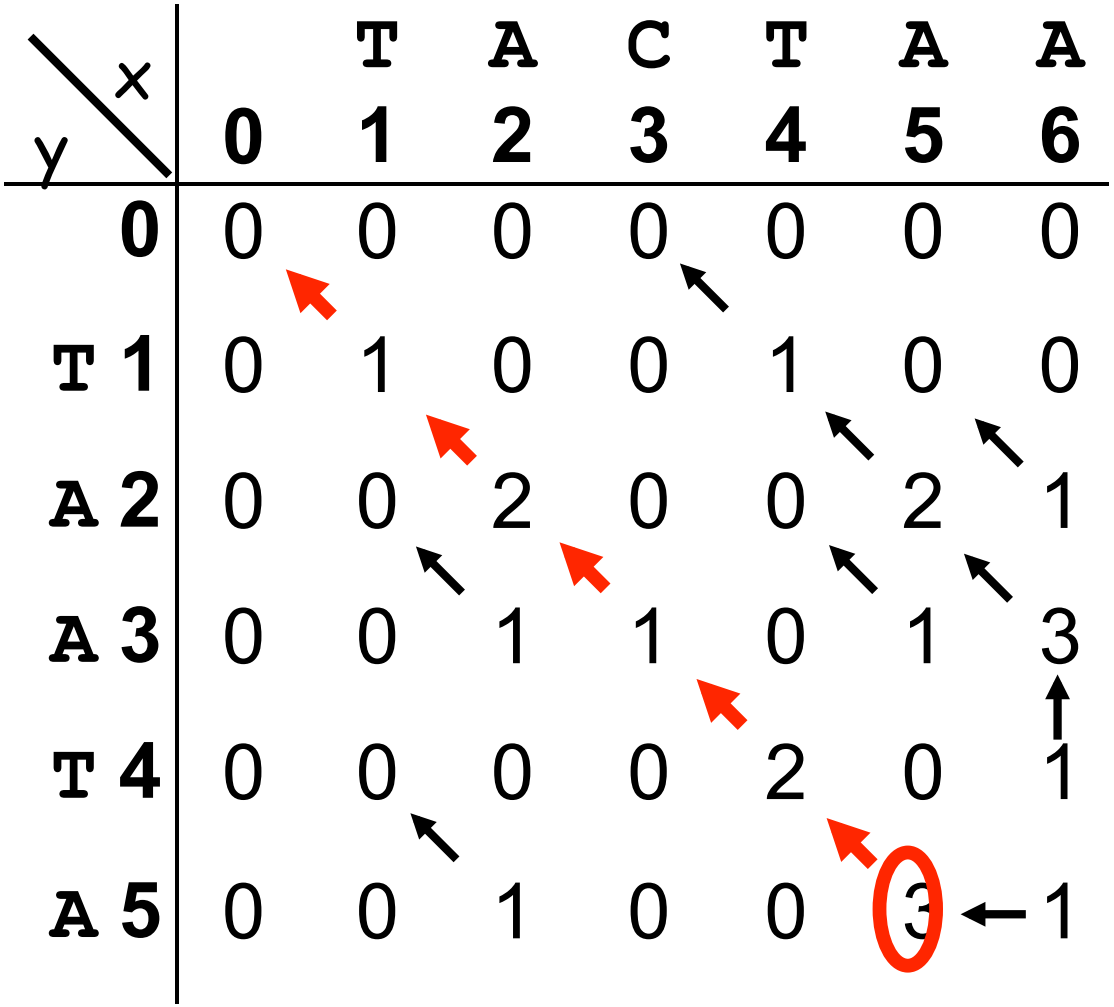
y = TAATA
x = TACTAA

y \ x		T	A	C	T	A	A
	0	**1**	**2**	**3**	**4**	**5**	**6**
0	0	0	0	0	0	0	0
T 1	0	1	0	0	1	0	0
A 2	0	0	2	0	0	2	1
A 3	0						
T 4	0						
A 5	0						

Local Alignment Example

match=1
mismatch=-1
gap=-1

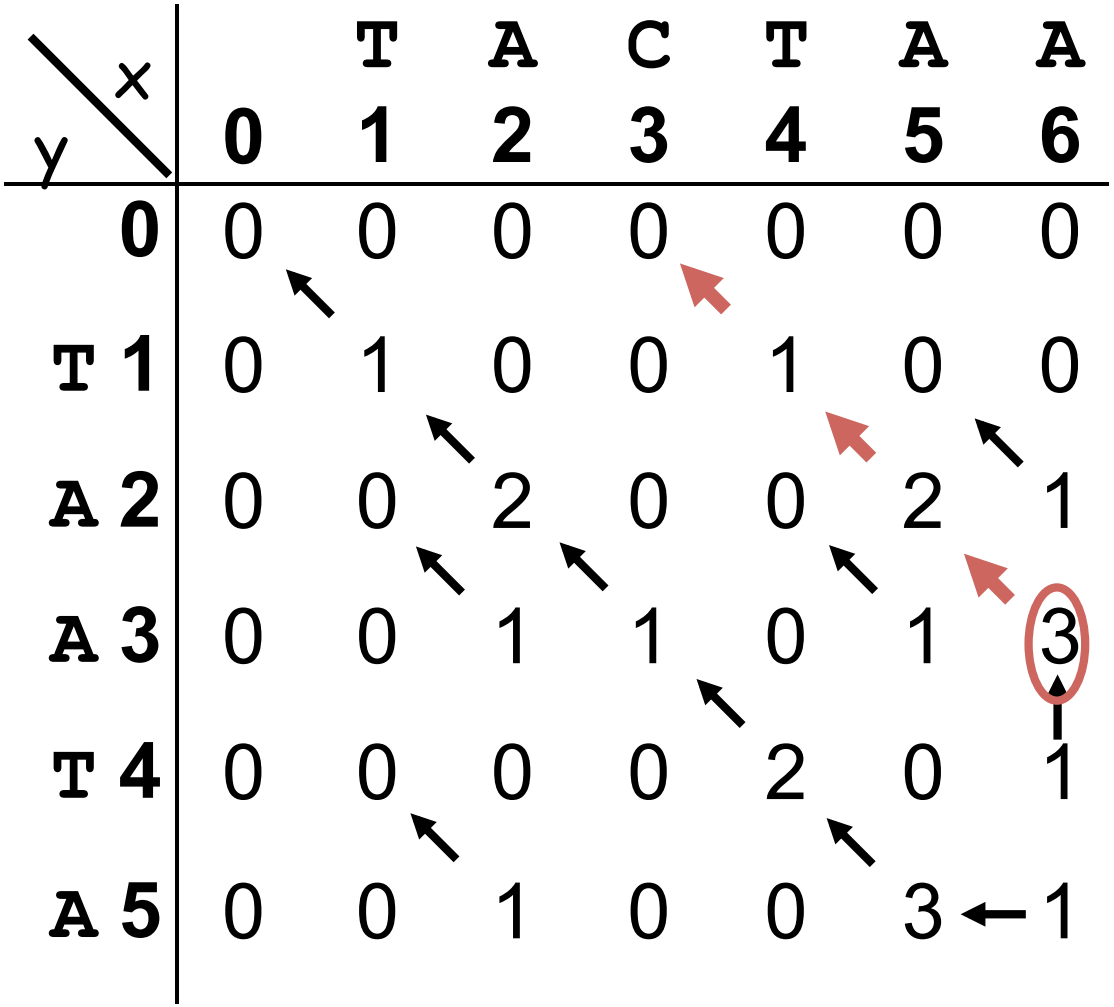
y = TAATA-
x = TACTAA



Local Alignment Example

match=1
mismatch=-1
gap=-1

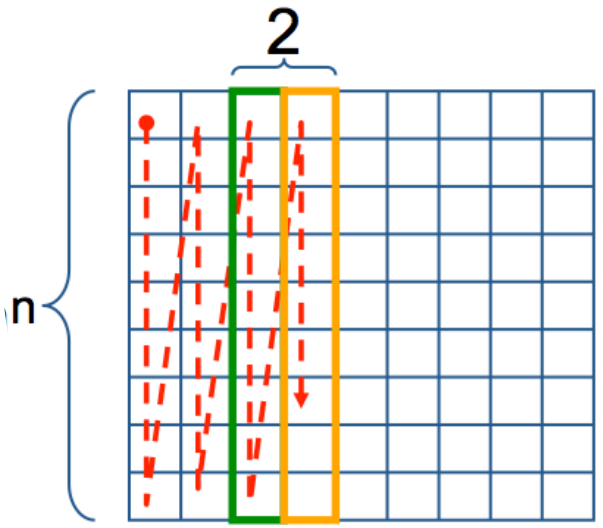
y = ---TAATA
x = TACTAA--



Computing Alignment Score with Linear Memory

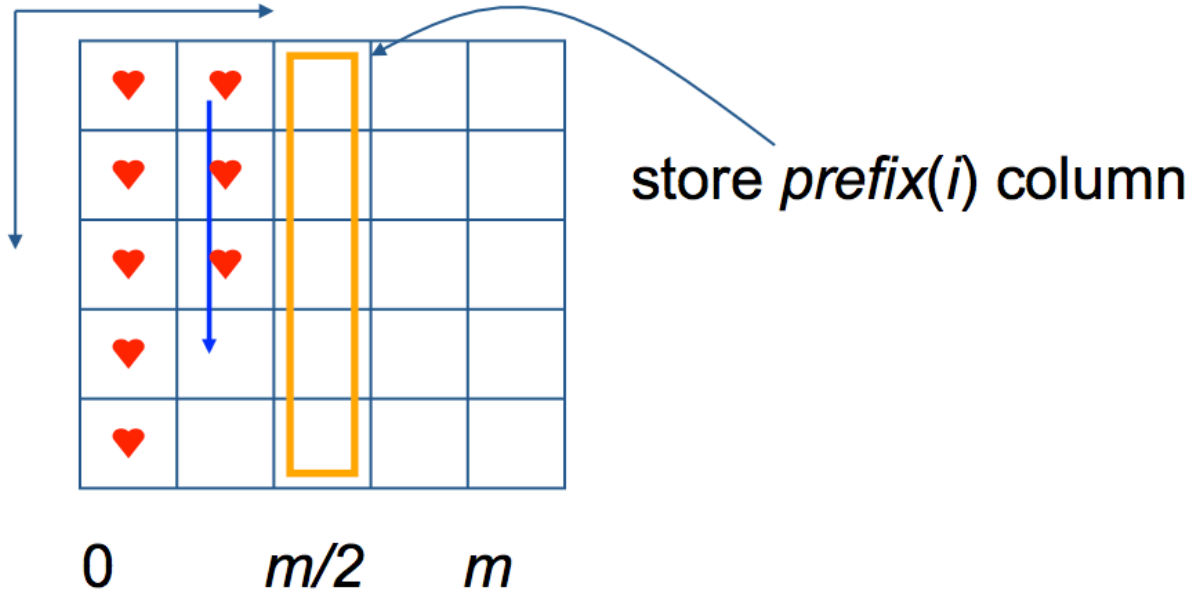
Alignment Score

- Space complexity of computing just the score itself is O(*n*)
- We only need the previous column to calculate the current column, and we can then throw away that previous column once we're done using it



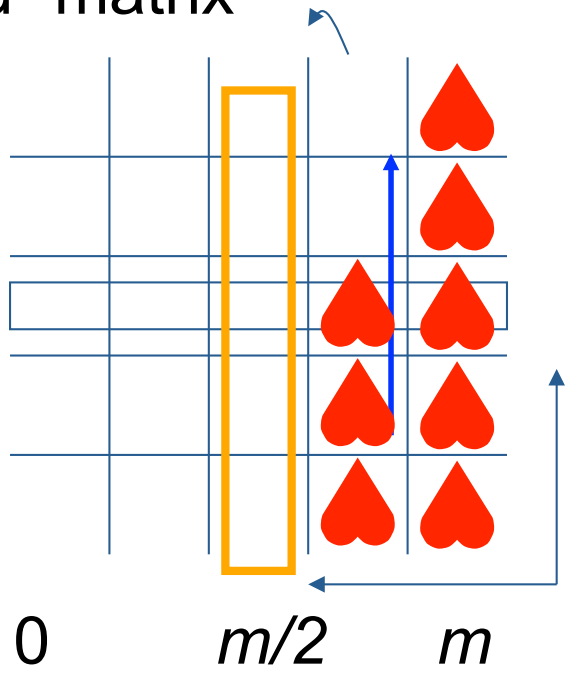
Computing Prefix(*i*)

- *prefix*(*i*) is the length of the longest path from (0,0) to (*i*,*m*/2)
- Compute *prefix*(*i*) by dynamic programming in the left half of the matrix



Computing Suffix(*i*)

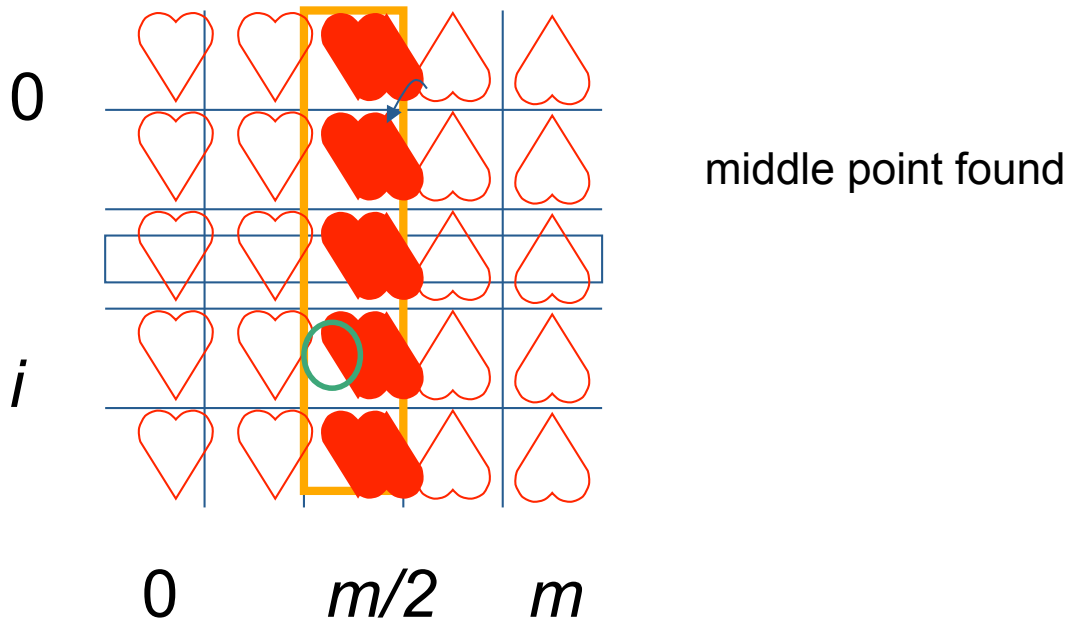
- *suffix*(*i*) is the length of the longest path from (*i*,*m*/2) to *(n,m)*
- *suffix*(*i*) is the length of the longest path from (*n*,*m*) to (*i*,*m*/2) with all edges reversed
- Compute *suffix*(*i*) by dynamic programming in the right half of the “reversed” matrix



store *suffix*(*i*) column

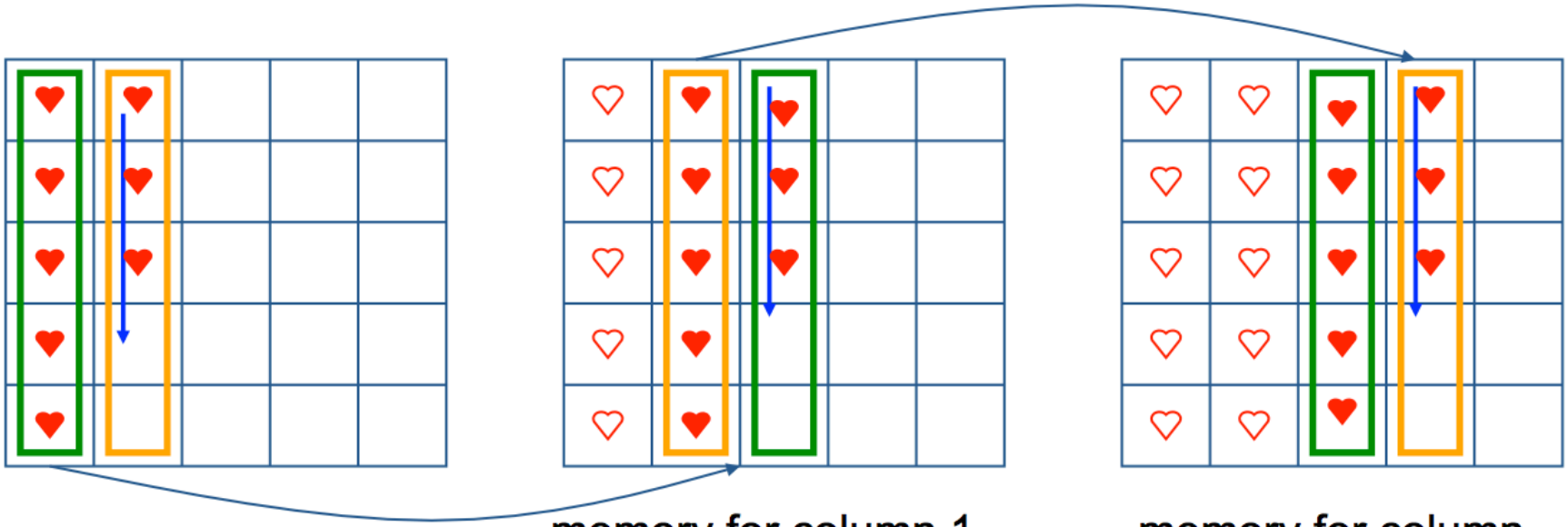
Length(i) = Prefix(i) + Suffix(i)

- Add *prefix*(*i*) and *suffix*(*i*) to compute *length(i):*
  - $length(i) = prefix(i) + suffix(i)$
- You now have a middle vertex of the maximum path (*i*,*m*/2) as maximum of *length(i)*



Computing Alignment Score: Recycling Columns

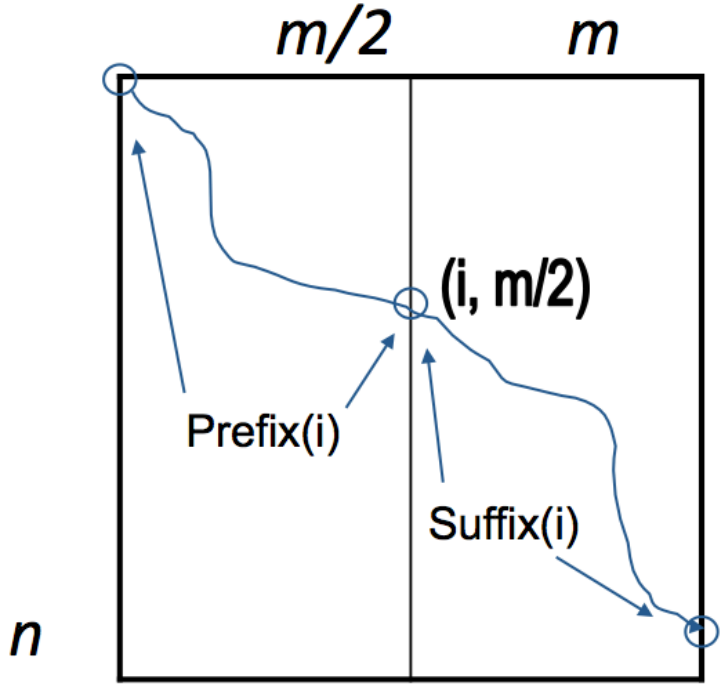
Only two columns of scores are saved at any given time



memory for column 1 is used to calculate column 3

memory for column 2 is used to calculate column 4

Crossing the Middle Line



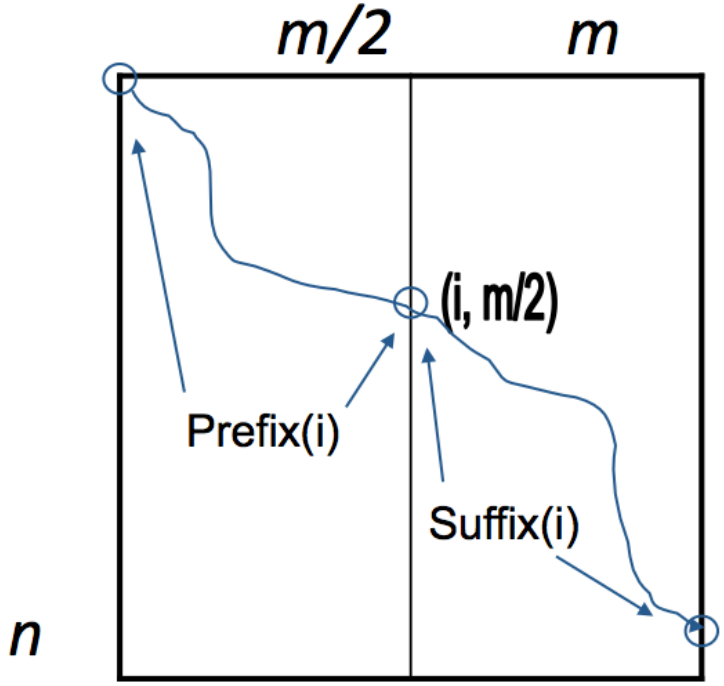
We want to calculate the longest path from (0,0) to (*n*,*m*) that passes through (*i*,*m*/2) where *i* ranges from 0 to *n* and represents the *i*-th row

Define

length(*i*)

as the length of the longest path from (0,0) to (*n*,*m*) that passes through vertex (*i*, *m*/2)

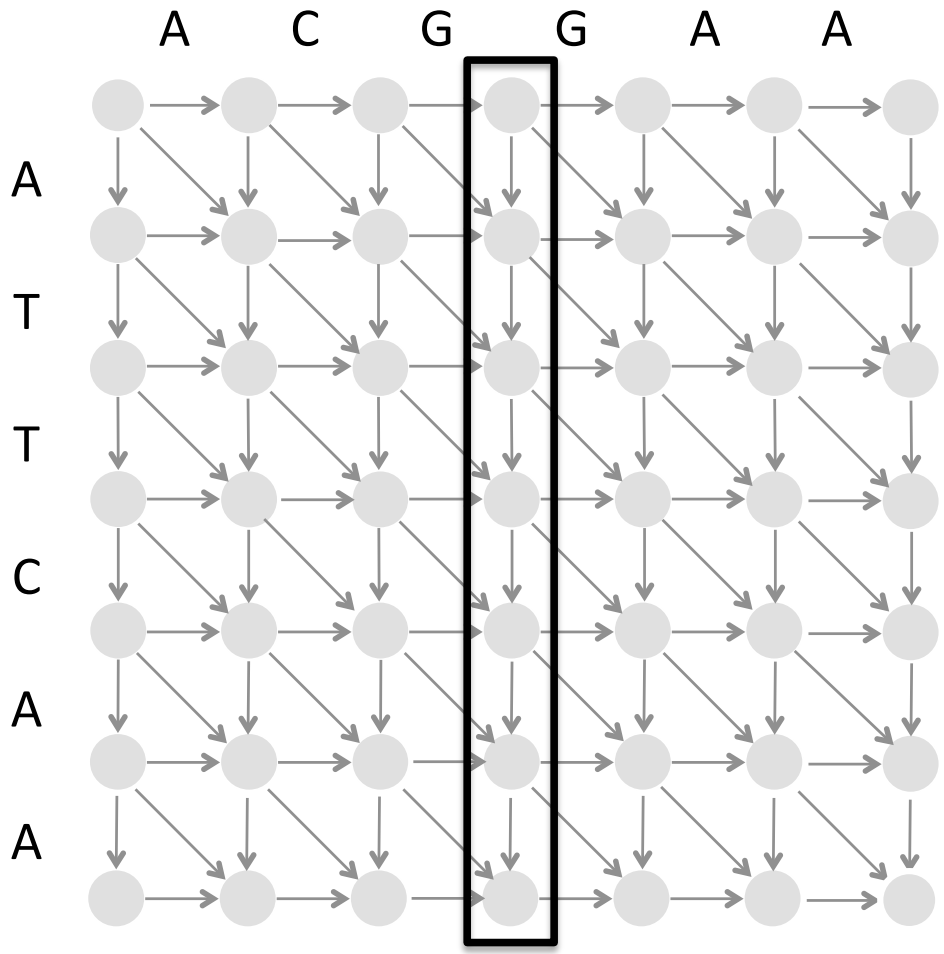
Crossing the Middle Line



Define (*mid*,*m*/2) as the vertex where the longest path crosses the middle column.

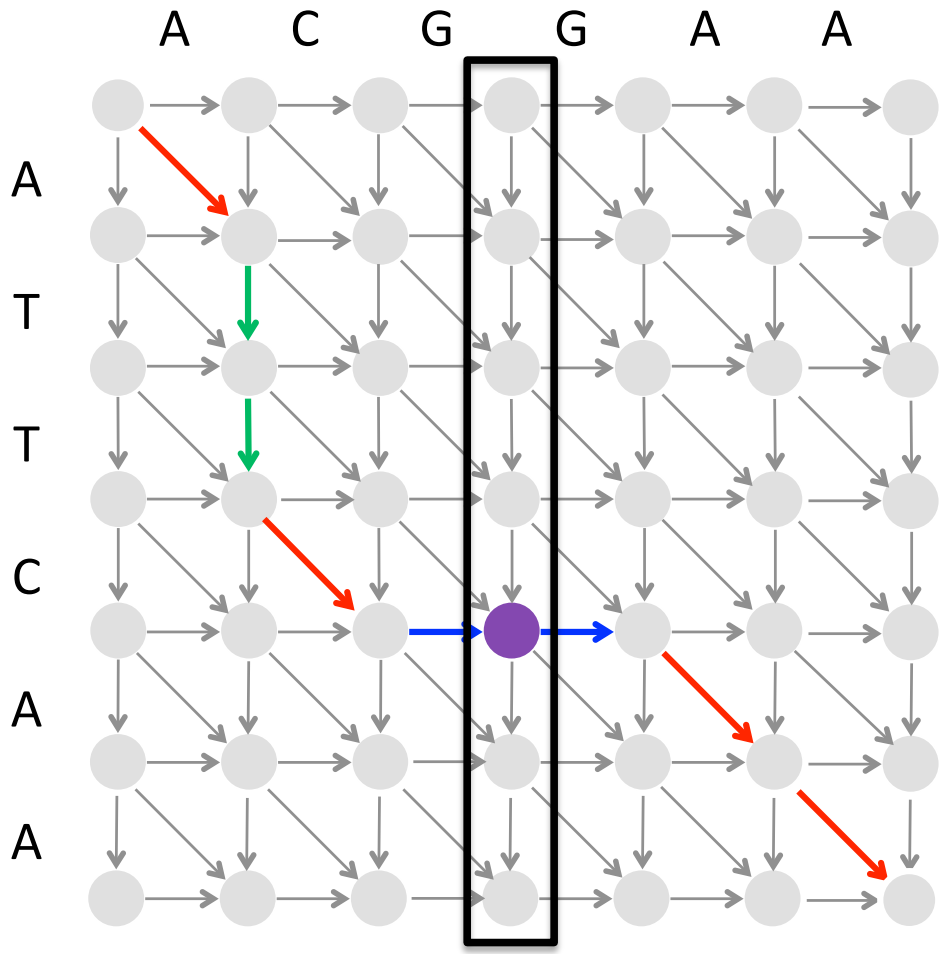
length(*mid*) = optimal length = $\max_{0 \le i \le n}$ *length(i)*

Middle Column of the Alignment



middle column
(*middle*=#columns/2)

Middle Node of the Alignment



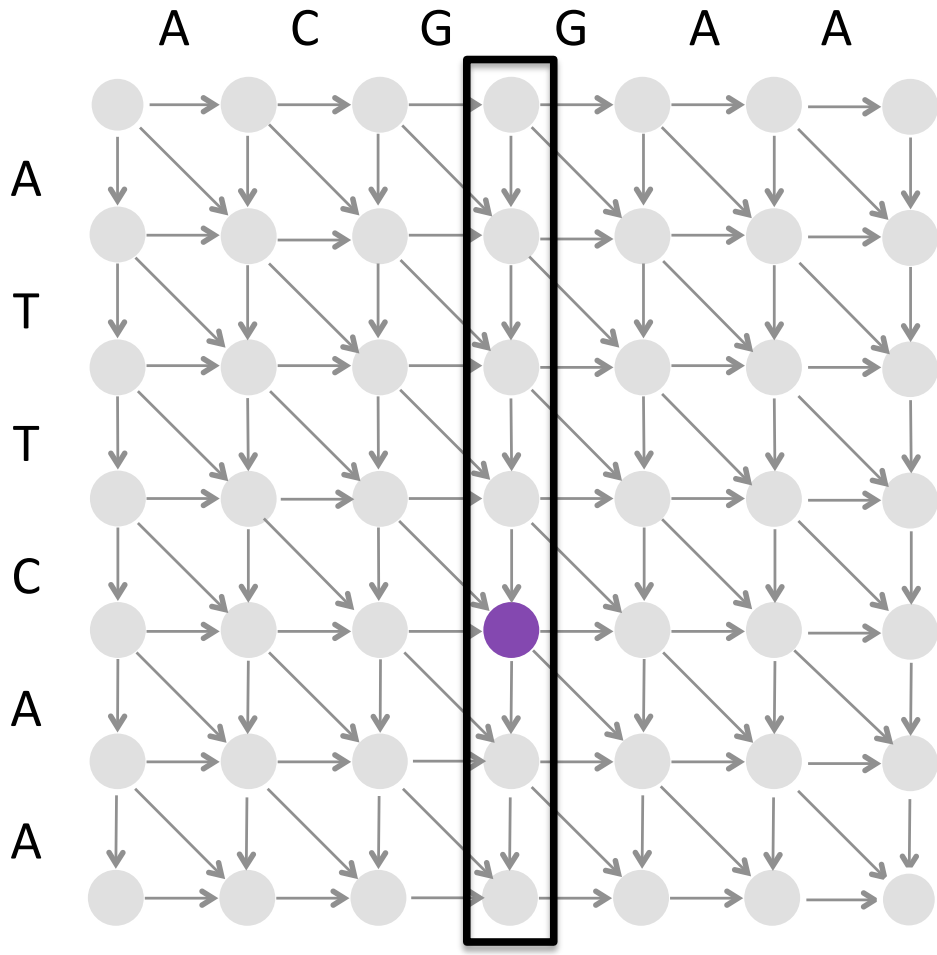
middle node

(a node where an optimal alignment path crosses the middle column; note that different longest paths may have different middle nodes, and a given longest path may have more than one middle node.)

Divide and Conquer Approach to Sequence Alignment

AlignmentPath(*source, sink*)

find *MiddleNode*

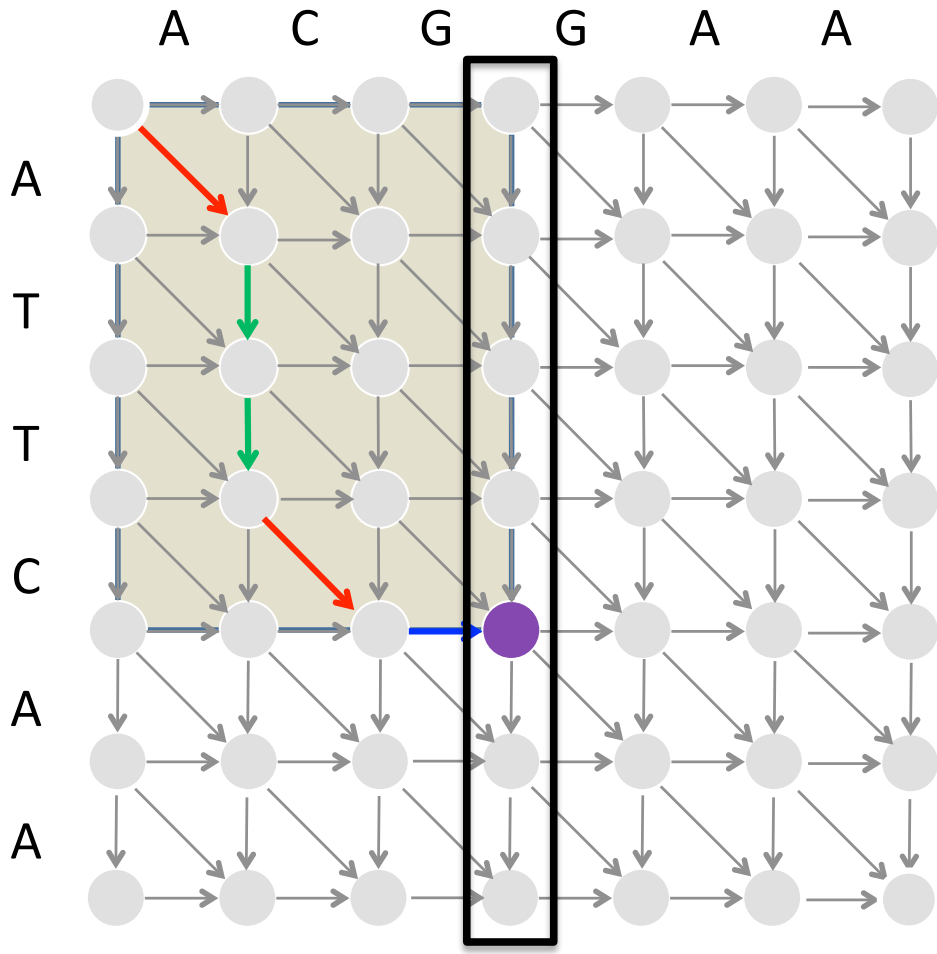


Divide and Conquer Approach to Sequence Alignment

AlignmentPath(*source, sink*)

find *MiddleNode*

AlignmentPath(*source, MiddleNode*)



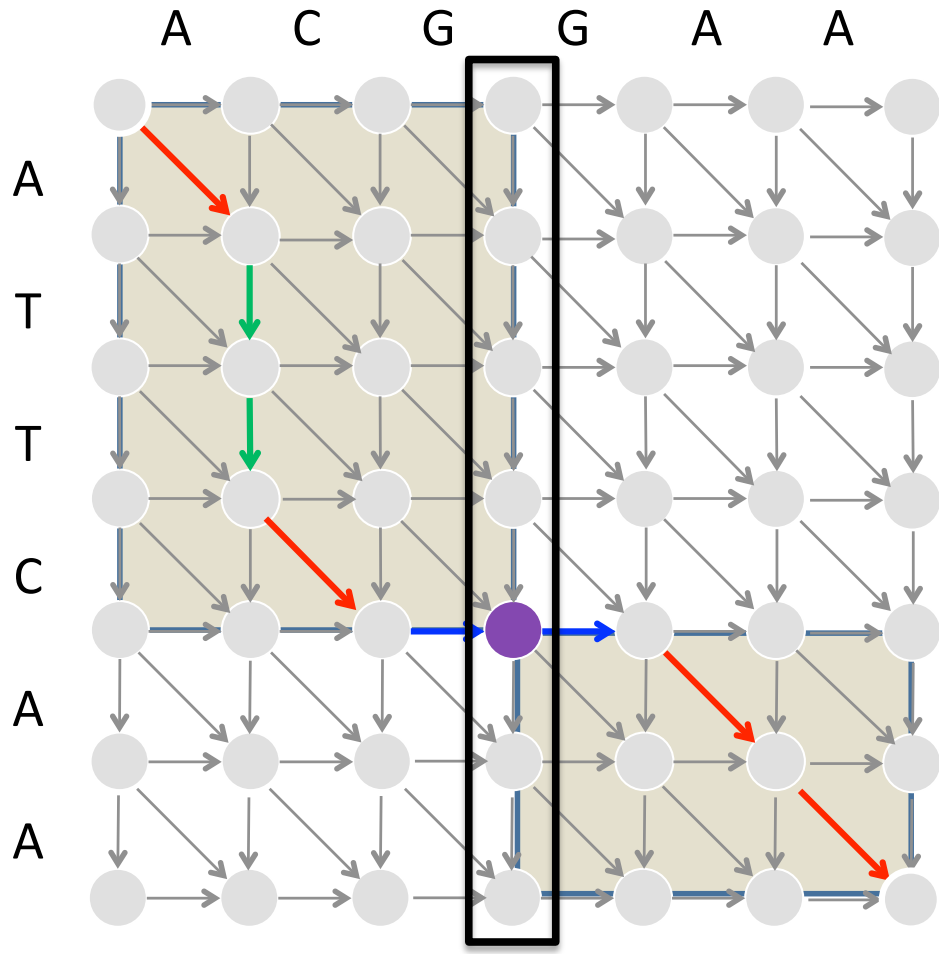
Divide and Conquer Approach to Sequence Alignment

AlignmentPath(*source, sink*)

find *MiddleNode*

AlignmentPath(*source, MiddleNode*)

AlignmentPath(*MiddleNode, sink*)



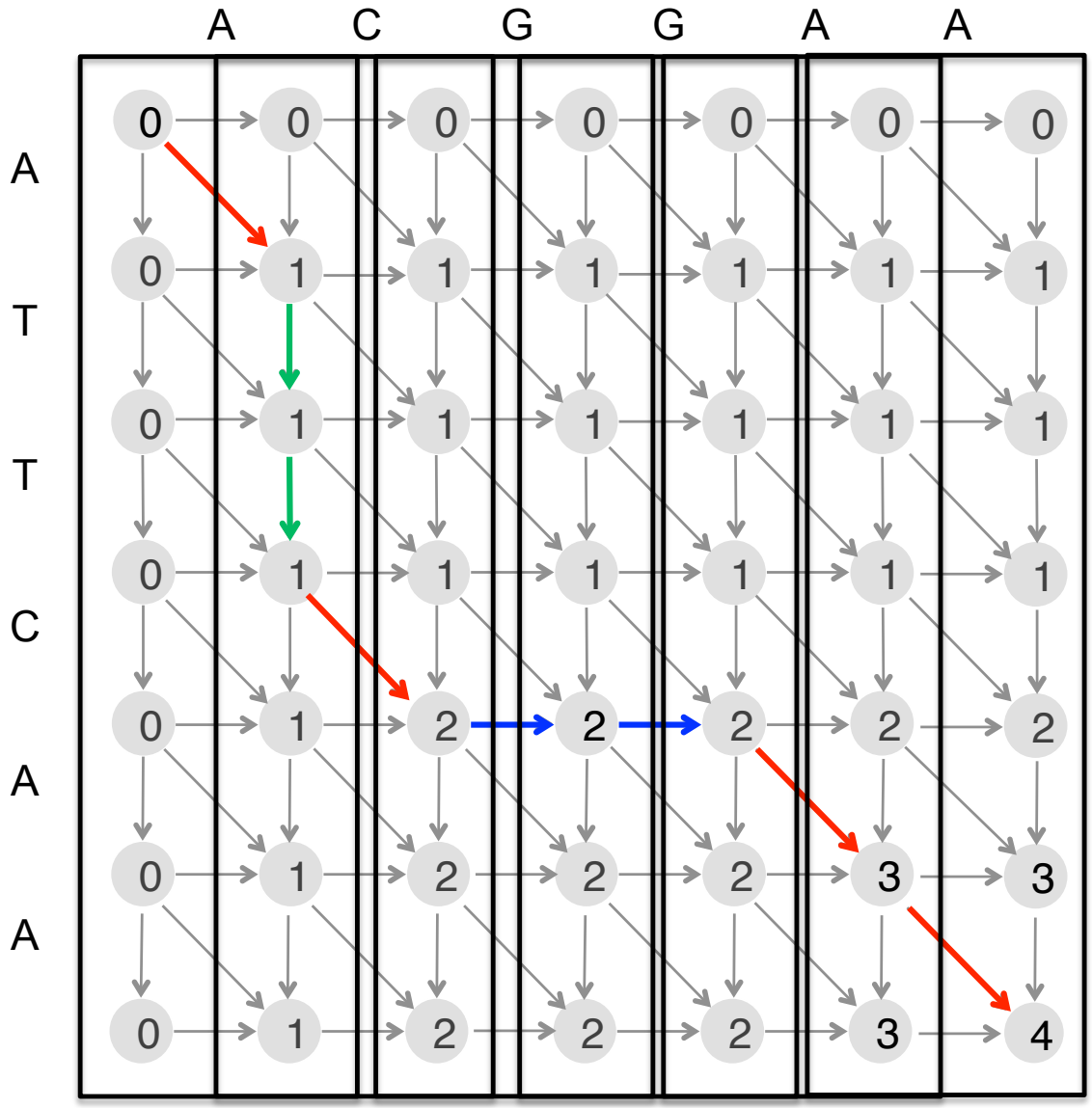
The only problem left is how to find this middle node in **linear space**!

Computing Alignment Score in Linear Space

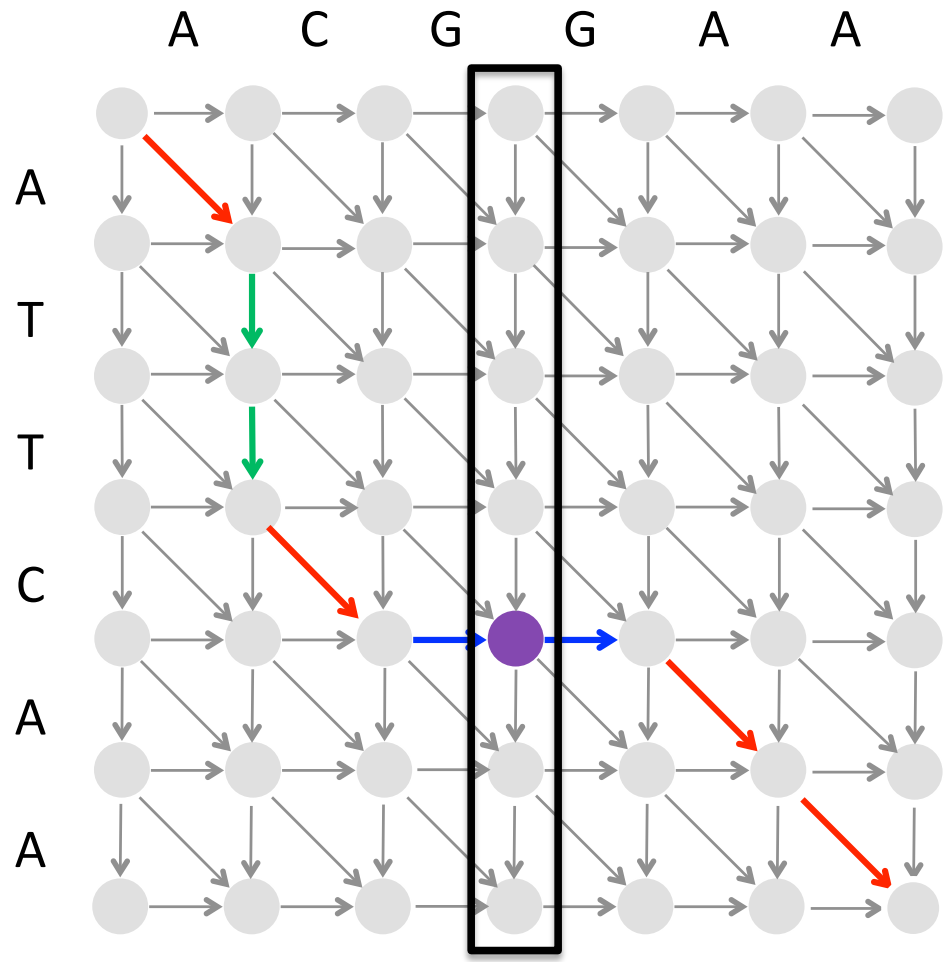
Finding the **longest path** in the alignment graph **requires** storing all backtracking pointers – O(*nm*) memory.

Finding the **length of the longest path** in the alignment graph **does not require** storing any backtracking pointers – O(*n*) memory.

Recycling the Columns in the Alignment Graph



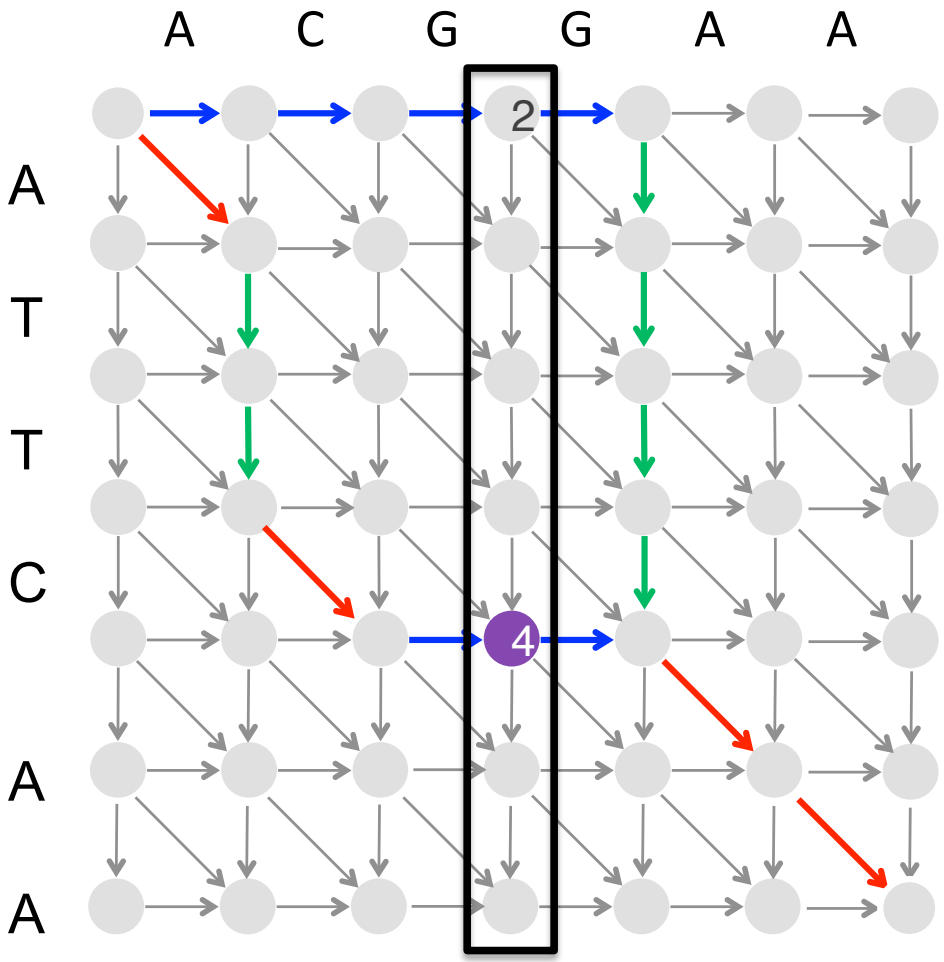
Can We Find the Middle Node without Constructing the Longest Path?



4-path that visits the node (4,middle) In the middle column

***i*-path** – a longest path among paths that visit the *i*-th node in the middle column

Can We Find The Lengths of All *i*-paths?

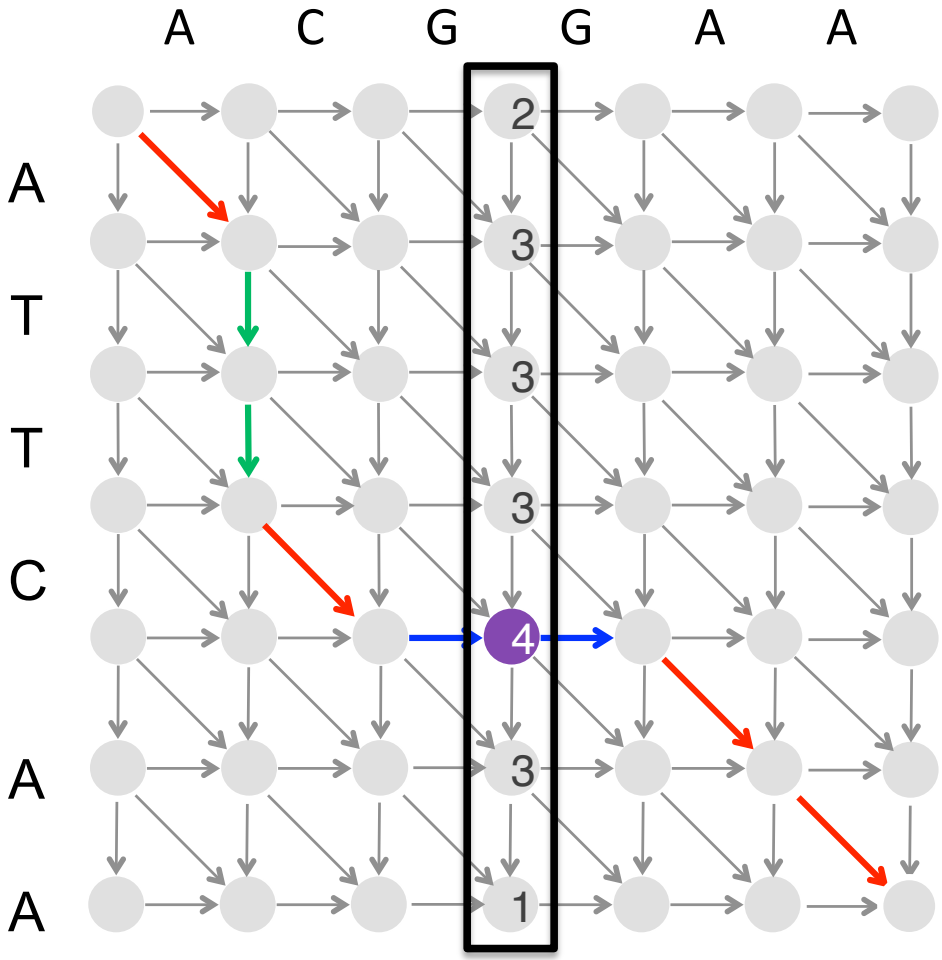


length(i): length of an *i*-path:

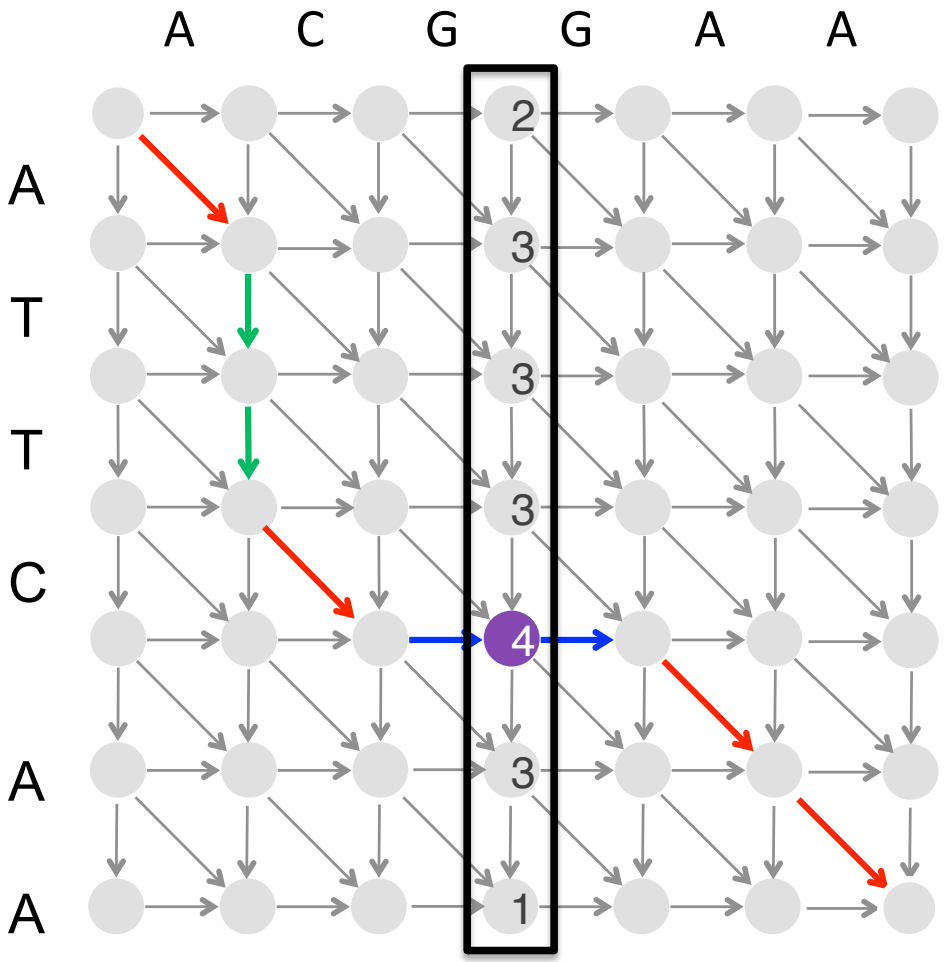
length(0)=2

length(4)=4

Can We Find The Lengths of All *i*-paths?



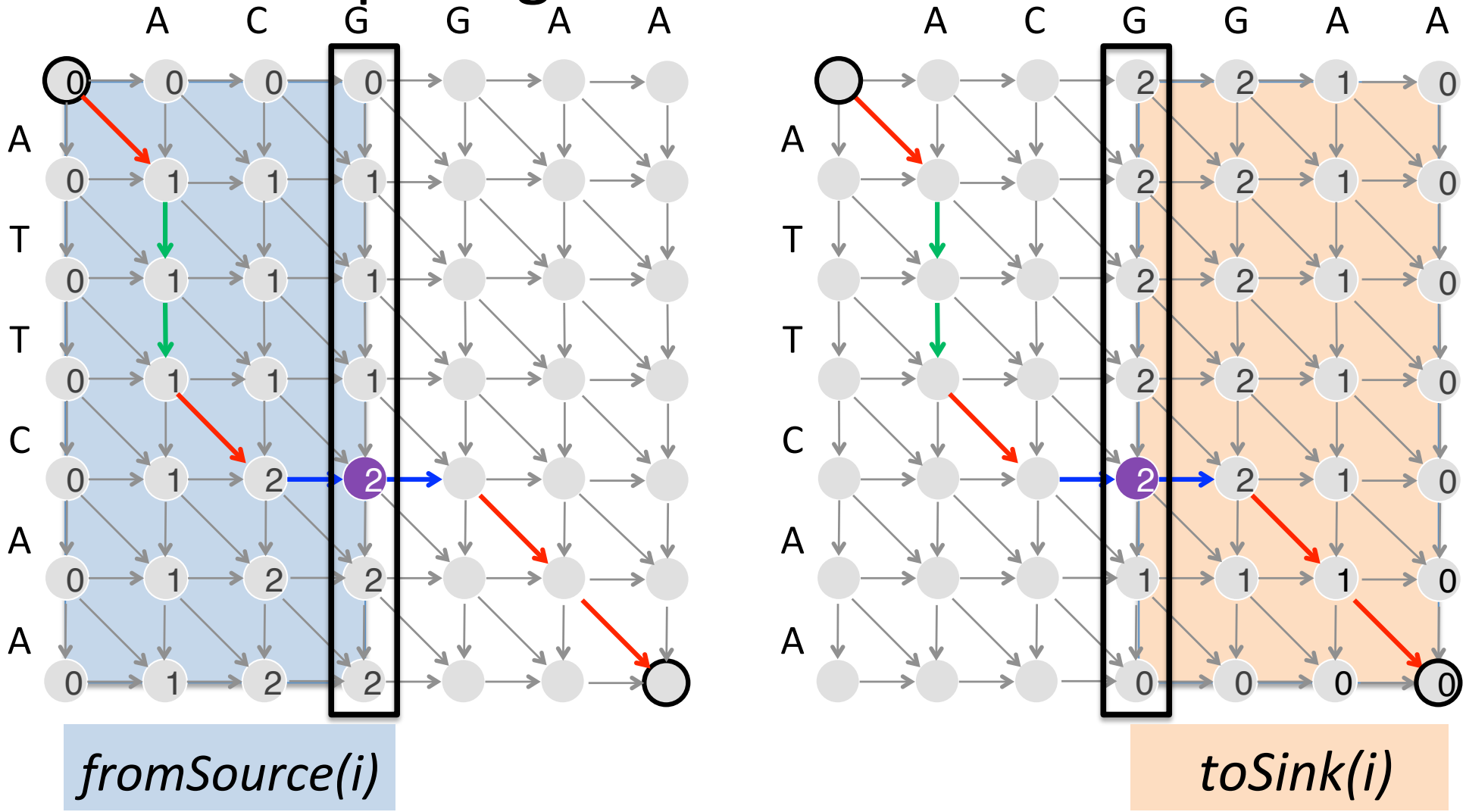
Can We Find The Lengths of *i*-paths?



length(i):
length of an *i*-path

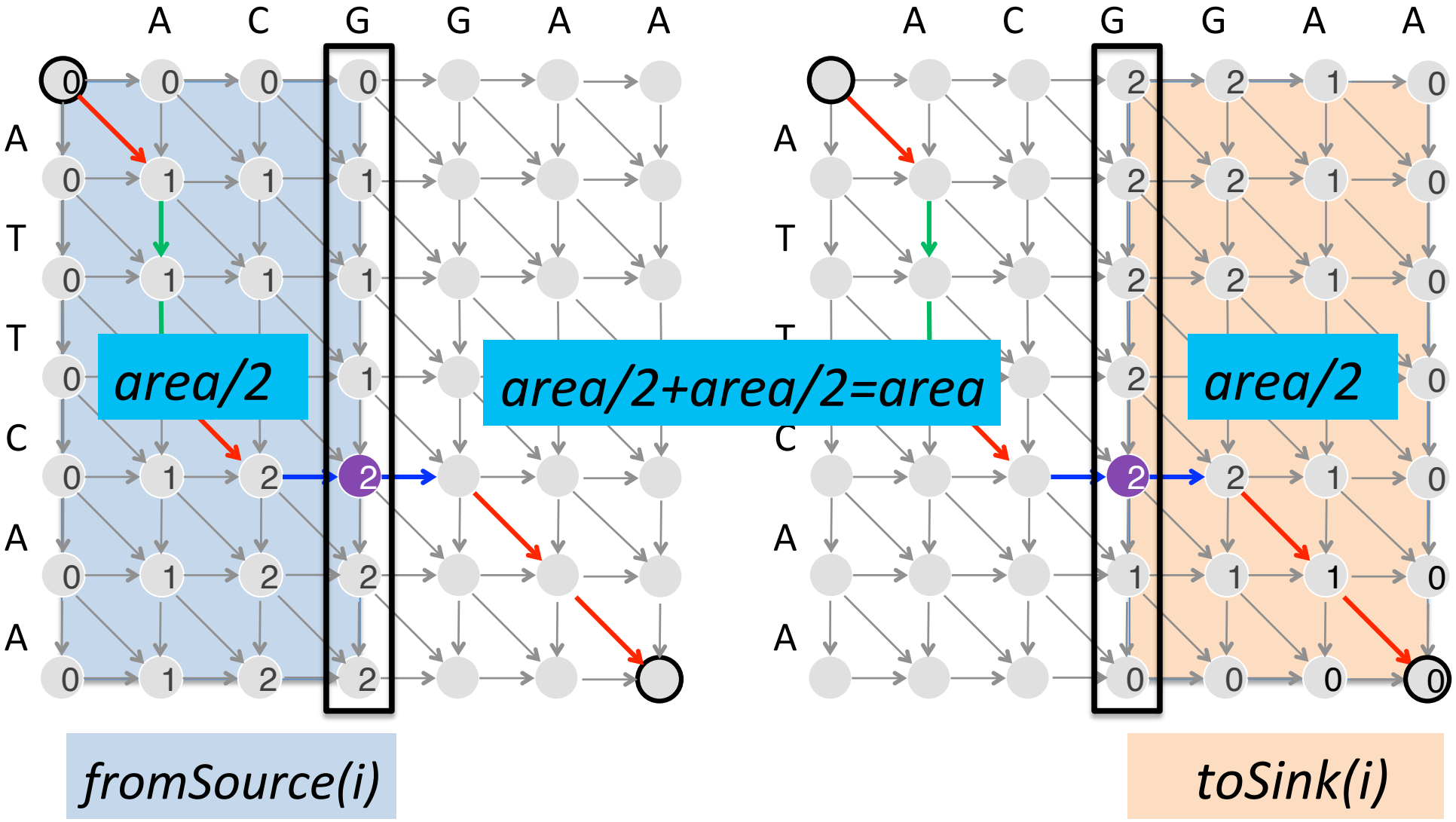
length(i)=fromSource(i)+toSink(i)

Computing *FromSource* and *toSink*



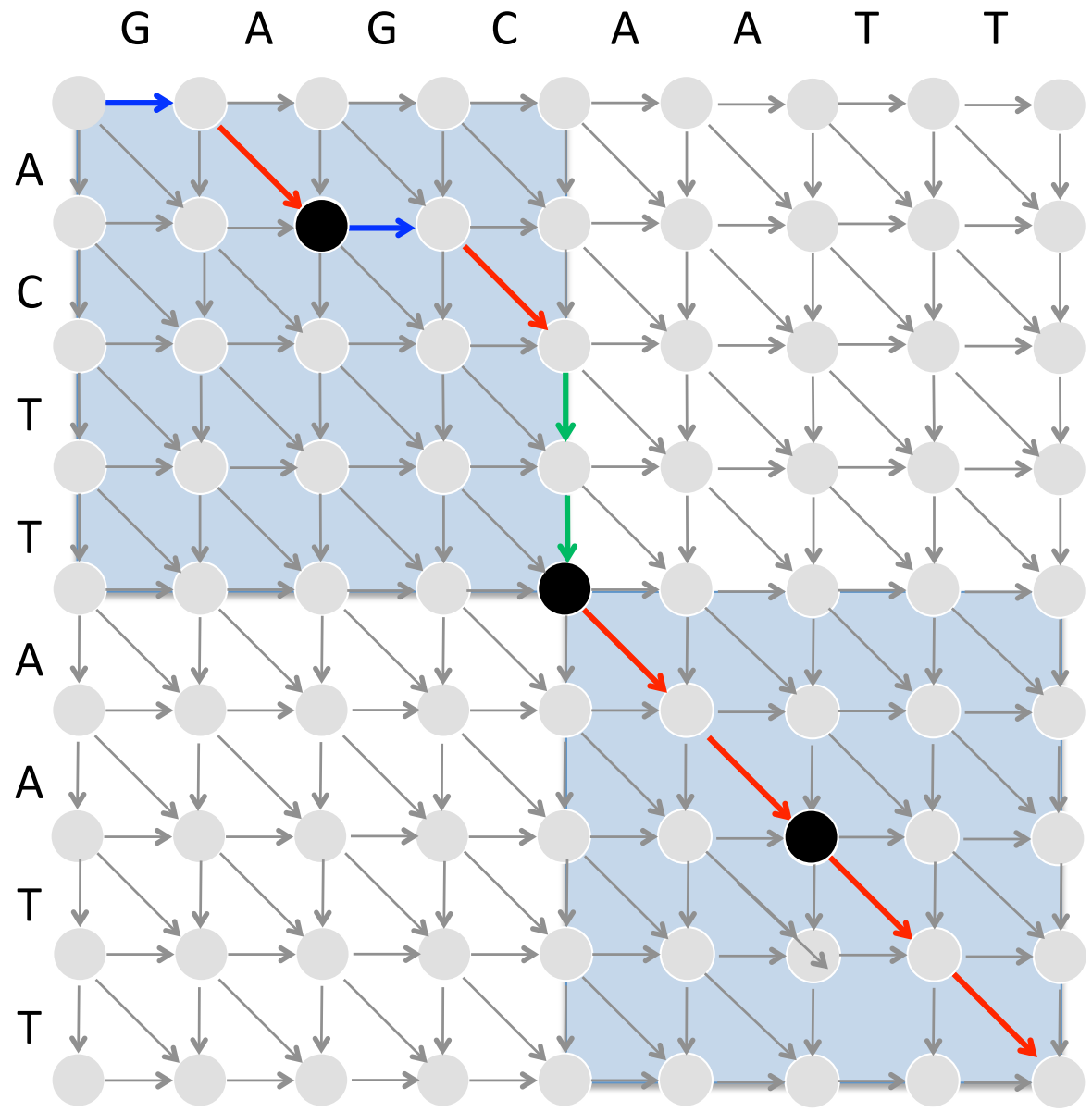
Computing FROMSOURCE(i) for all i can be done in O(n) space and O(n ·m/2) time. Computing TOSINK(i) for all i can also be done in O(n) space and O(n ·m/2) time; this requires reversing the direction of all edges and treating the sink as the source. Instead of reversing the edges, we can reverse the strings $v = v_1 \ldots v_n$ and $w = w_1 \ldots w_m$ and find $s_{n\text{-}i,m\text{-}middle}$ in the alignment graph for $v_n \ldots v_1$ and $w_m \ldots w_1$.

How Much Time Did It Take to Find the Middle Node ?



In total, we can compute all values LENGTH(i) = FROMSOURCE(i) + TOSINK(i) in linear space with runtime proportional to n · m/2 + n · m/2 = n · m, which is the total area of the alignment graph.

Laughable Progress: O(*nm*) Time to Find **ONE** Node!

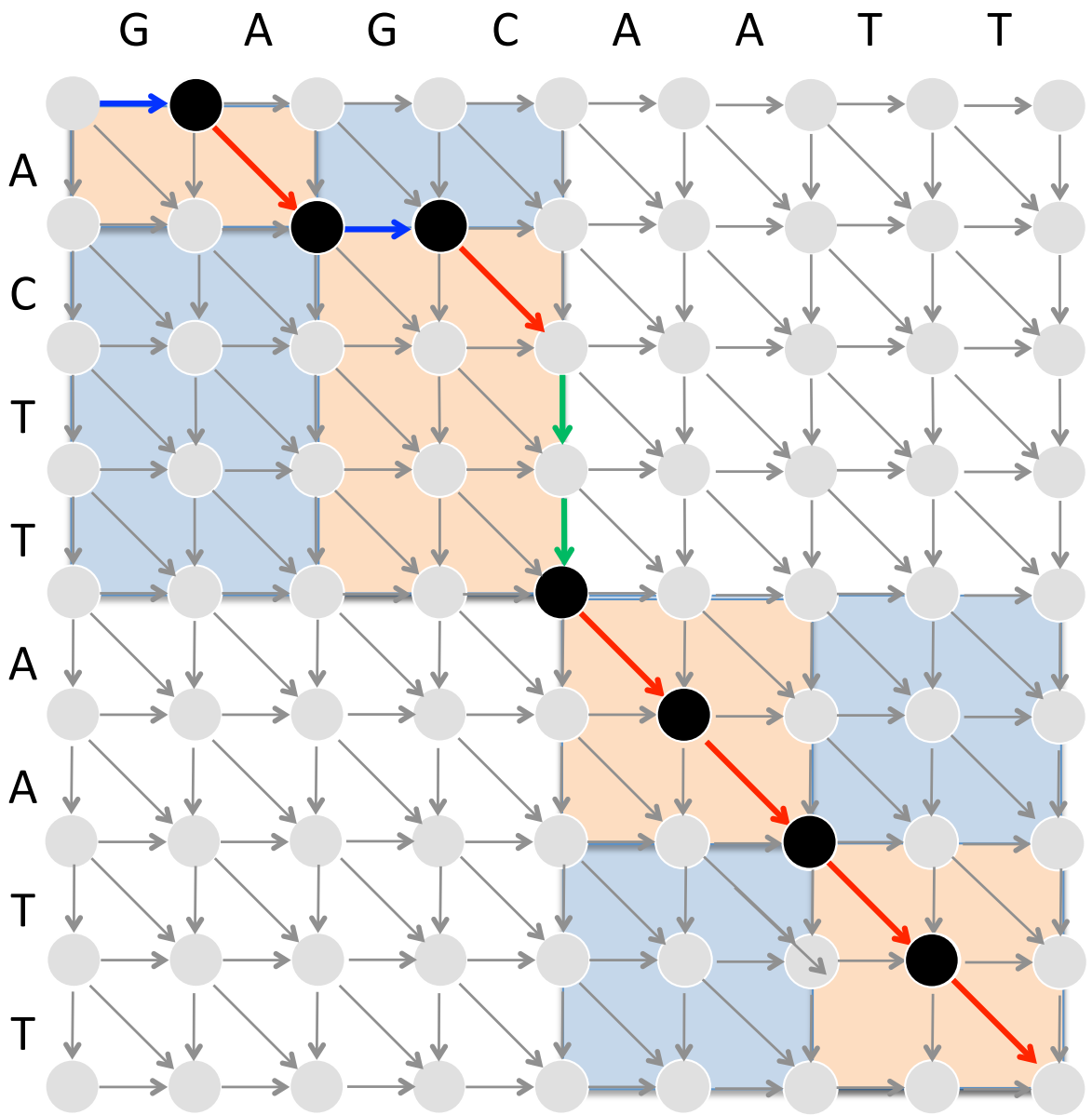


Each subproblem can be conquered in time proportional to its area:

area/4+area/4= ***area/2***

How much time would it take to conquer 2 subproblems?

Laughable Progress: O(*nm+nm/2*) Time to Find **THREE** Nodes!

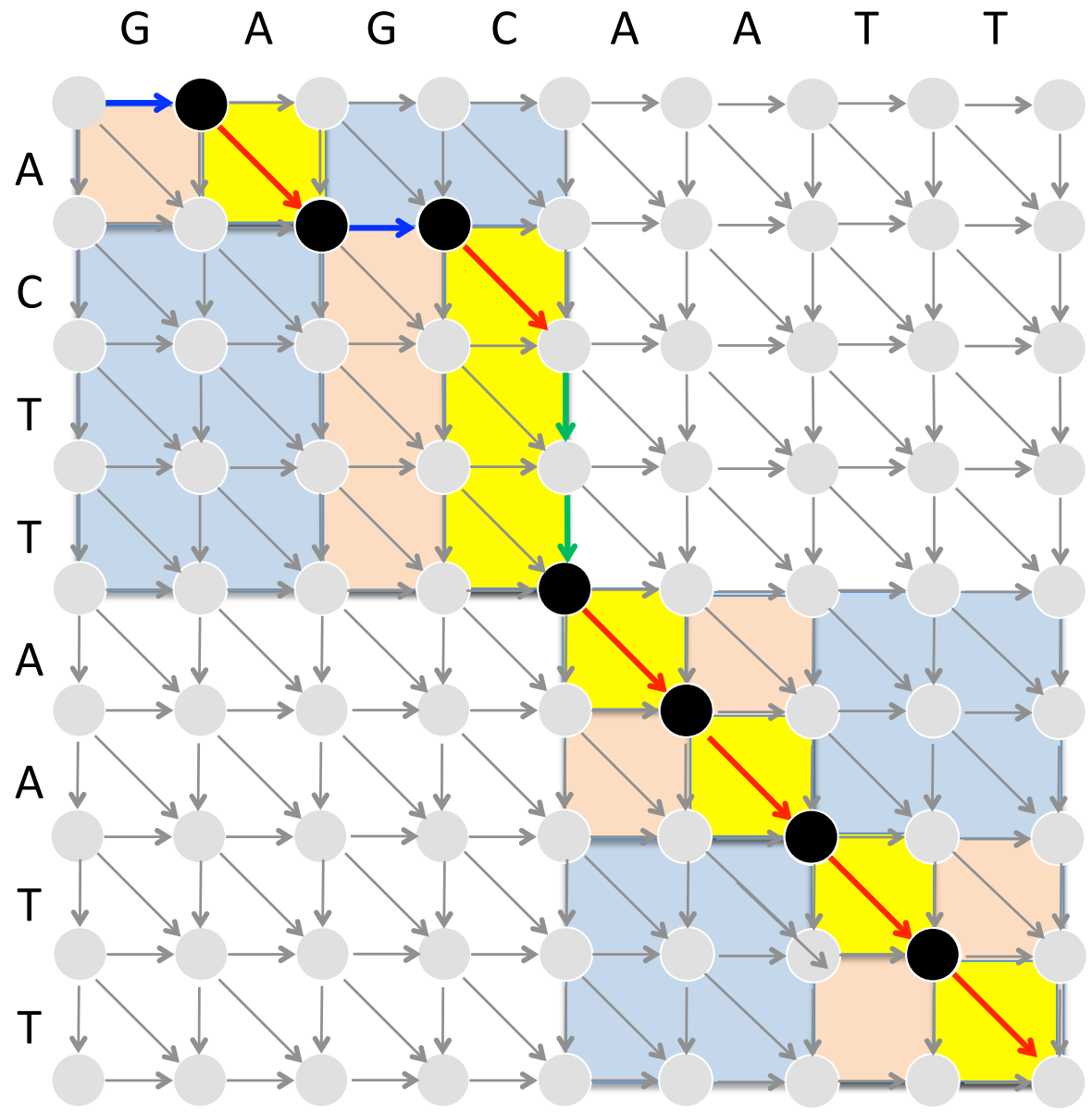


Each subproblem can be conquered in time proportional to its area:

area/8+area/8+ area/8+area/8= ***area/4***

How much time would it take to conquer 4 subproblems?

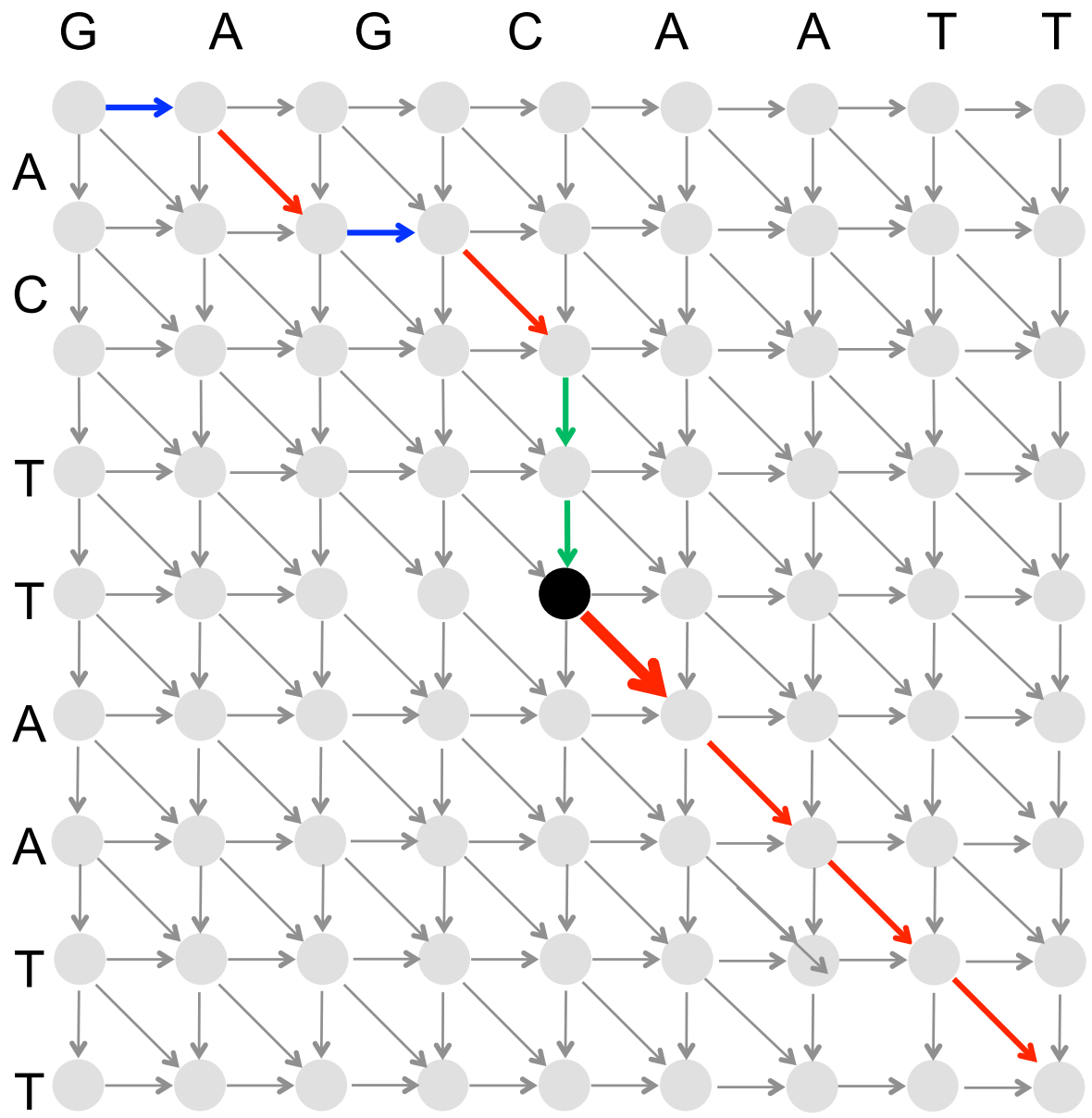
O(*nm+nm/2+nm/4*) Time to Find **NEARLY ALL** Nodes!



area+
area/2
+area/4
+area/8
+area/16
+….+
<
2·area

How much time would it take to conquer ALL subproblems?

The Middle Edge (just to save memory a little bit more)

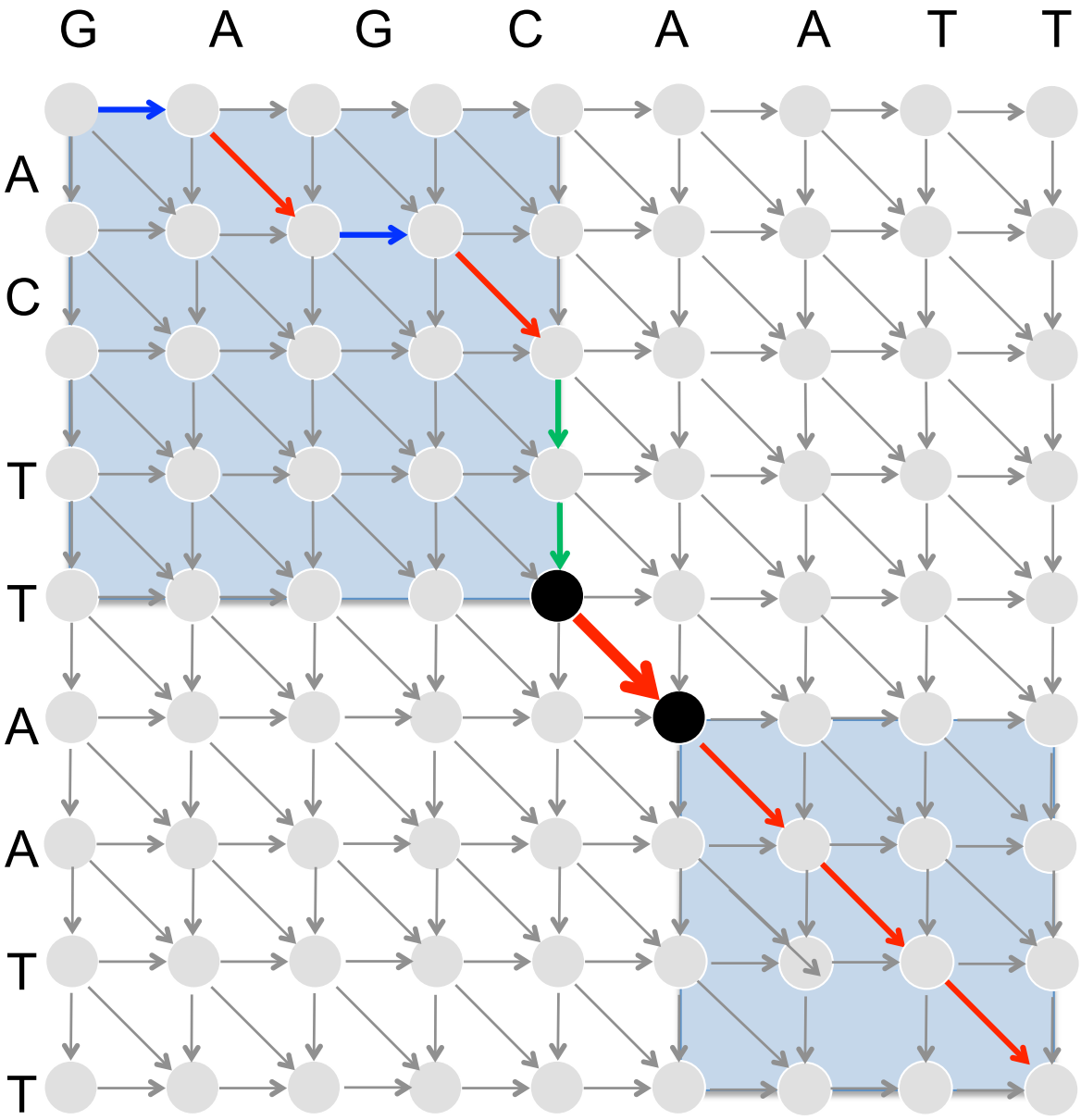


Middle Edge: an edge in an optimal alignment path starting at the middle node

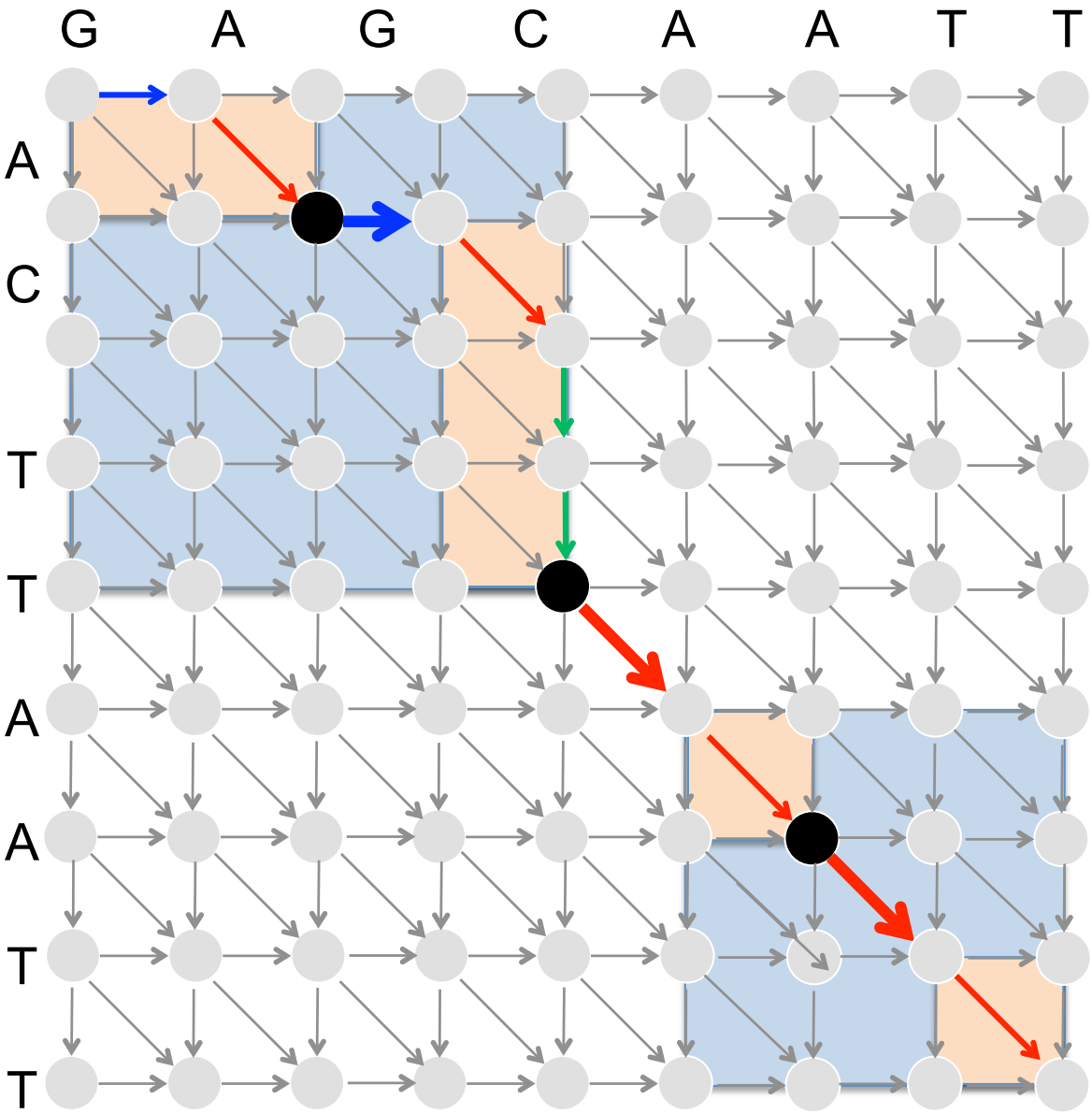
The Middle Edge Problem

Middle Edge in Linear Space Problem. Find a middle edge in the alignment graph in linear space.

- **Input:** Two strings and matrix ***score***.
- **Output:** A middle edge in the alignment graph of these strings (as defined by the matrix ***score***).



A middle edge (shown in bold) starts at the middle node (shown as a black circle). The optimal path travels inside the first highlighted rectangle, passes the middle edge, and travels inside the second highlighted rectangle afterwards.



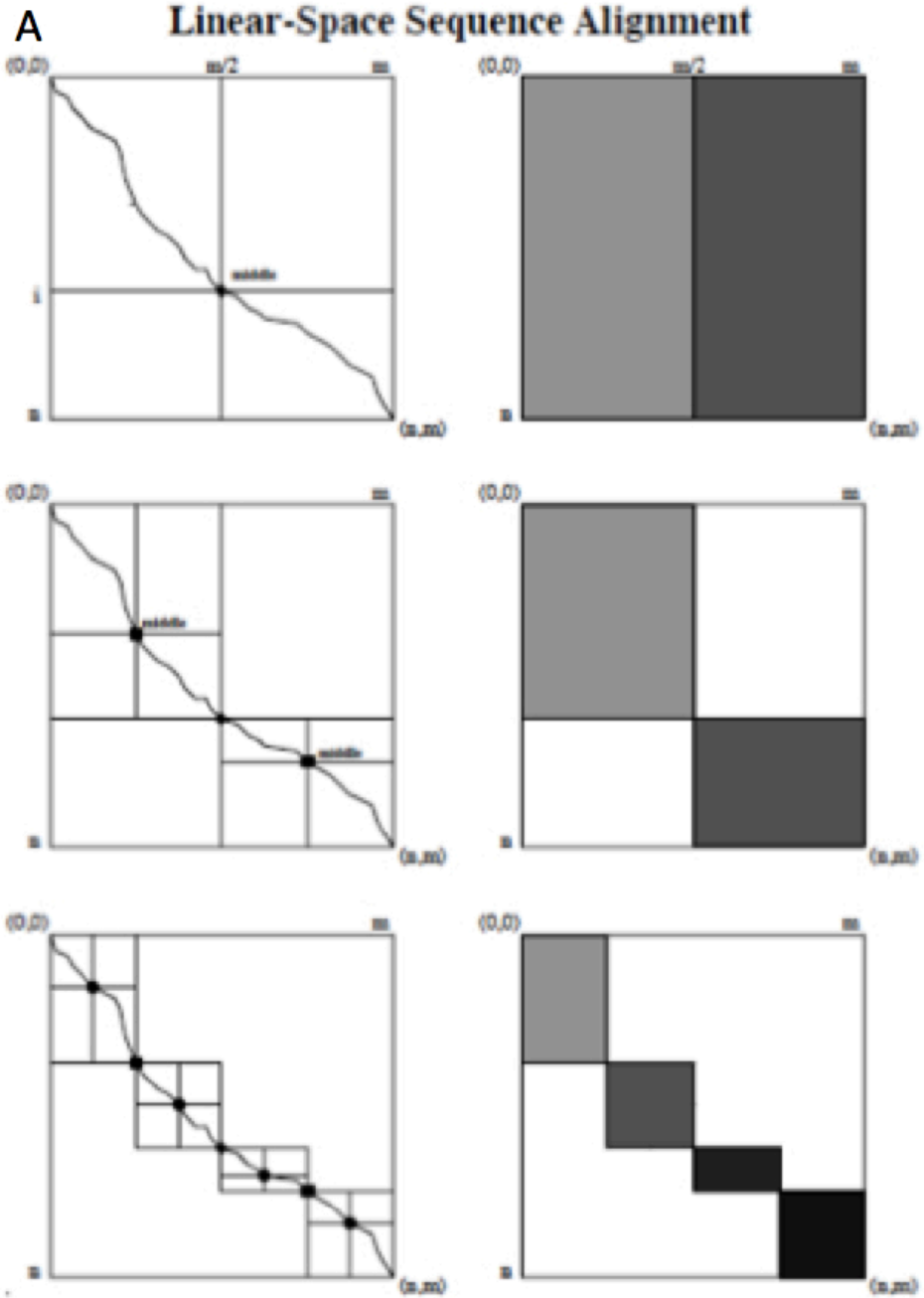
We can eliminate the remaining parts of the alignment graph, which takes up over half of the area formed by the graph, from further consideration.

Finding middle edges (shown in bold) within previously identified rectangles.

Recursive **LinearSpaceAlignment**

```
LinearSpaceAlignment(top,bottom,left,right)
  if left = right
    return alignment formed by bottom-top edges "↓"
  middle ← ⌊(left+right)/2⌋
  midNode ← MiddleNode(top,bottom,left,right)
  midEdge ←  MiddleEdge(top,bottom,left,right)
  LinearSpaceAlignment(top,midNode,left,middle)
  output midEdge
  if midEdge = "→" or midEdge = "↘"
    middle  ← middle+1
  if  midEdge = "↓" or midEdge = "↘"
    midNode  ← midNode+1
  LinearSpaceAlignment(midNode,bottom,middle,right)
```

A

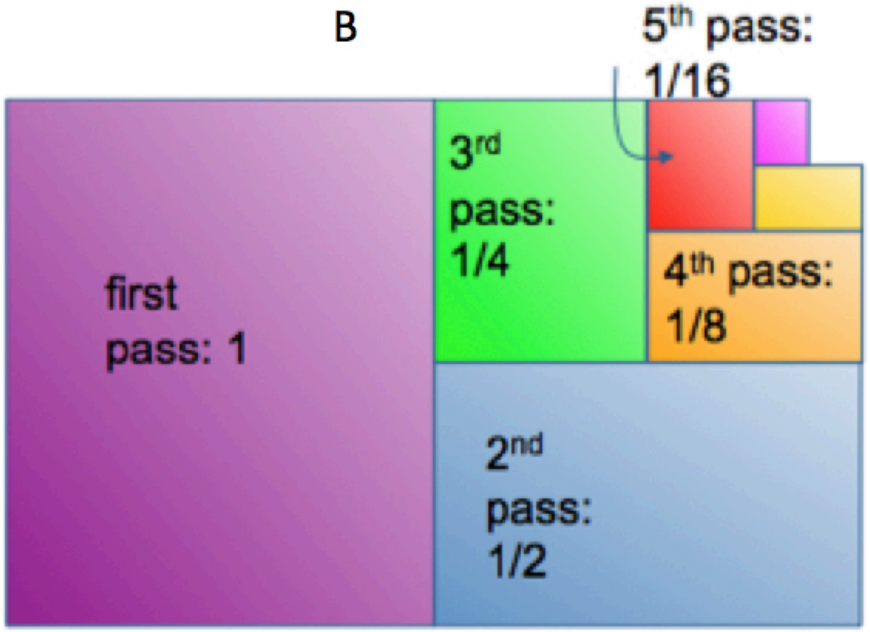


A: space complexity

B: time complexity

Total Time: ***area+area/2+area/4+area/8+area/16+…***

B



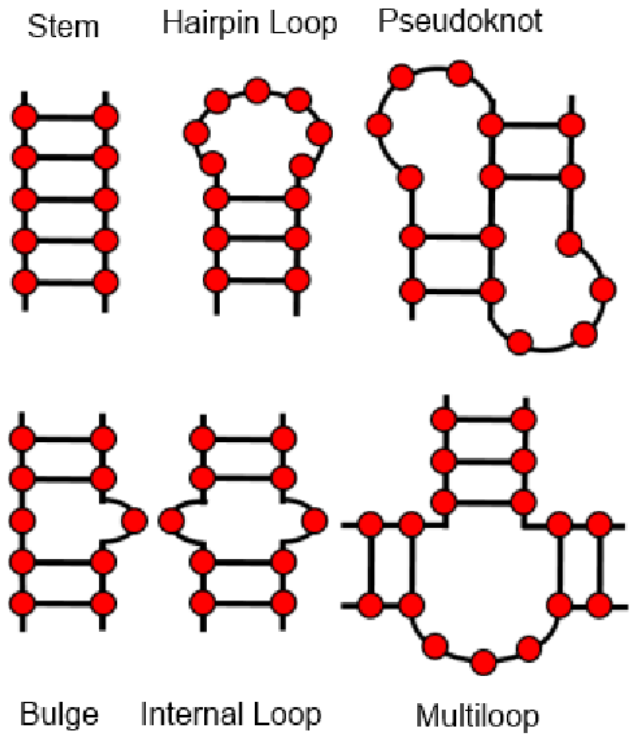
Can we compute the edit distance faster than O(nm)?

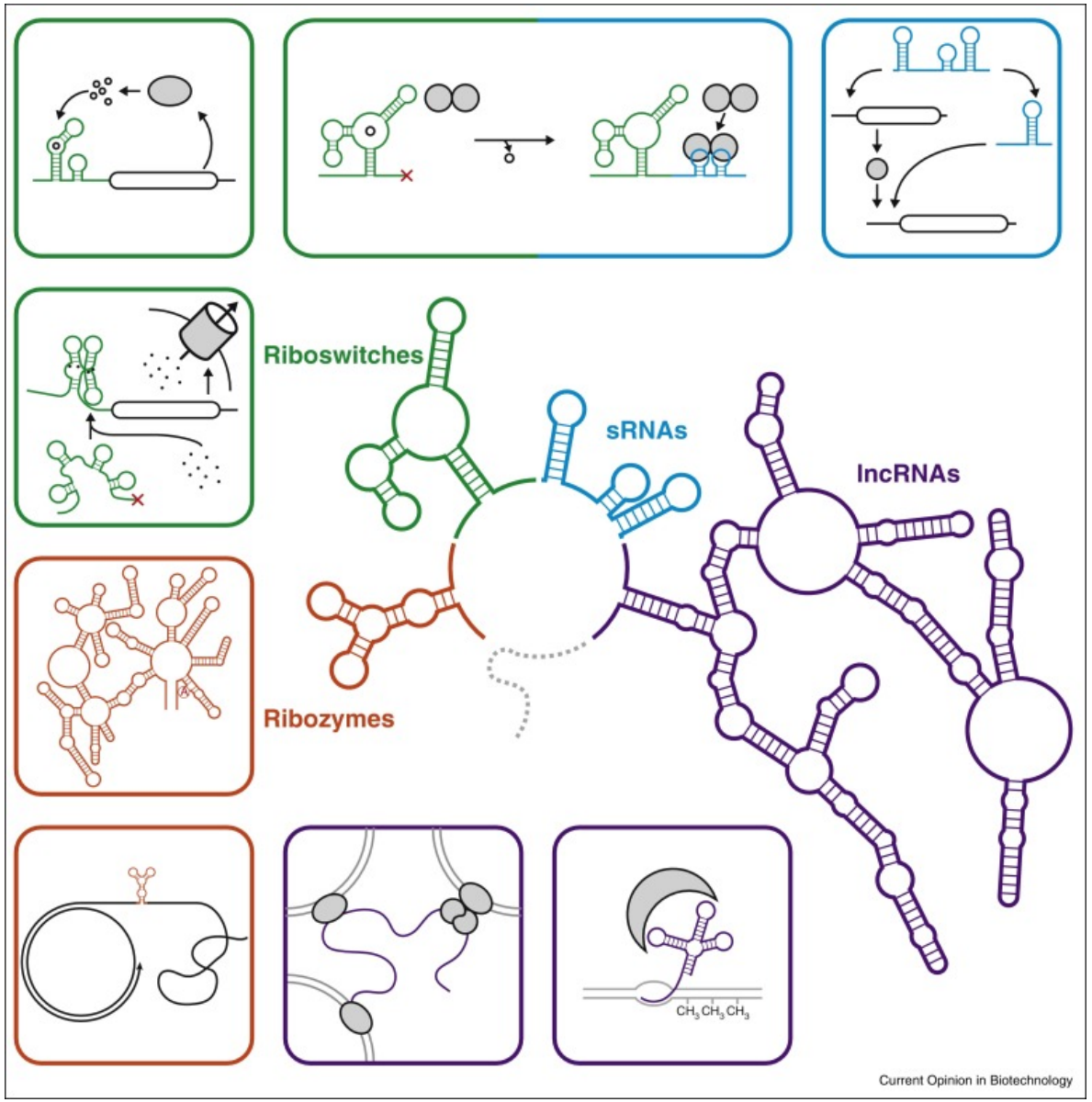
- yes: The Four Russians Technique
- Arlazarov, V.; Dinic, E.; Kronrod, M.; Faradžev, I.
- The basic idea is to precompute parts of the computation involved in filling out the dynamic programming table.
- time O(n^2/logn)
- Assume the block-function b(A, B, C, X[i+1 .. i+t], Y[j+1 .. j+t]) has been precomputed for all possible inputs.
- Article in Russian, easier to look at Aho, Alfred V.; Hopcroft, John E.; Ullman, Jeffrey D. (1974), The design and analysis of computer algorithms, Addison-Wesley

NOT EXAMINABLE

Self Alignment

Pairing rules:
C-G
A-U
(in RNA T is replaced by U)

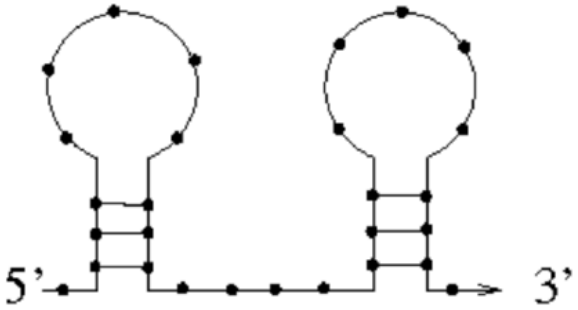


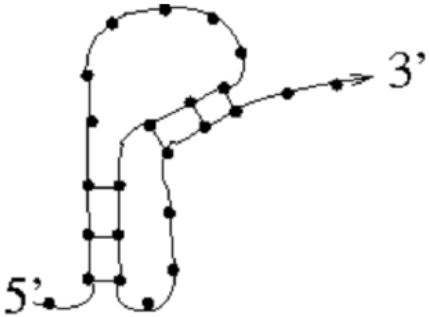


https://www.sciencedirect.com/science/article/pii/S0958166916301082#fig0020

RNA Secondary Structure: The Nussinov Folding Algorithm

Nussinov, R., Pieczenik, G., Griggs, J. R. and Kleitman, D. J. (1978). Algorithms for loop matchings, SIAM J. Appl. Math





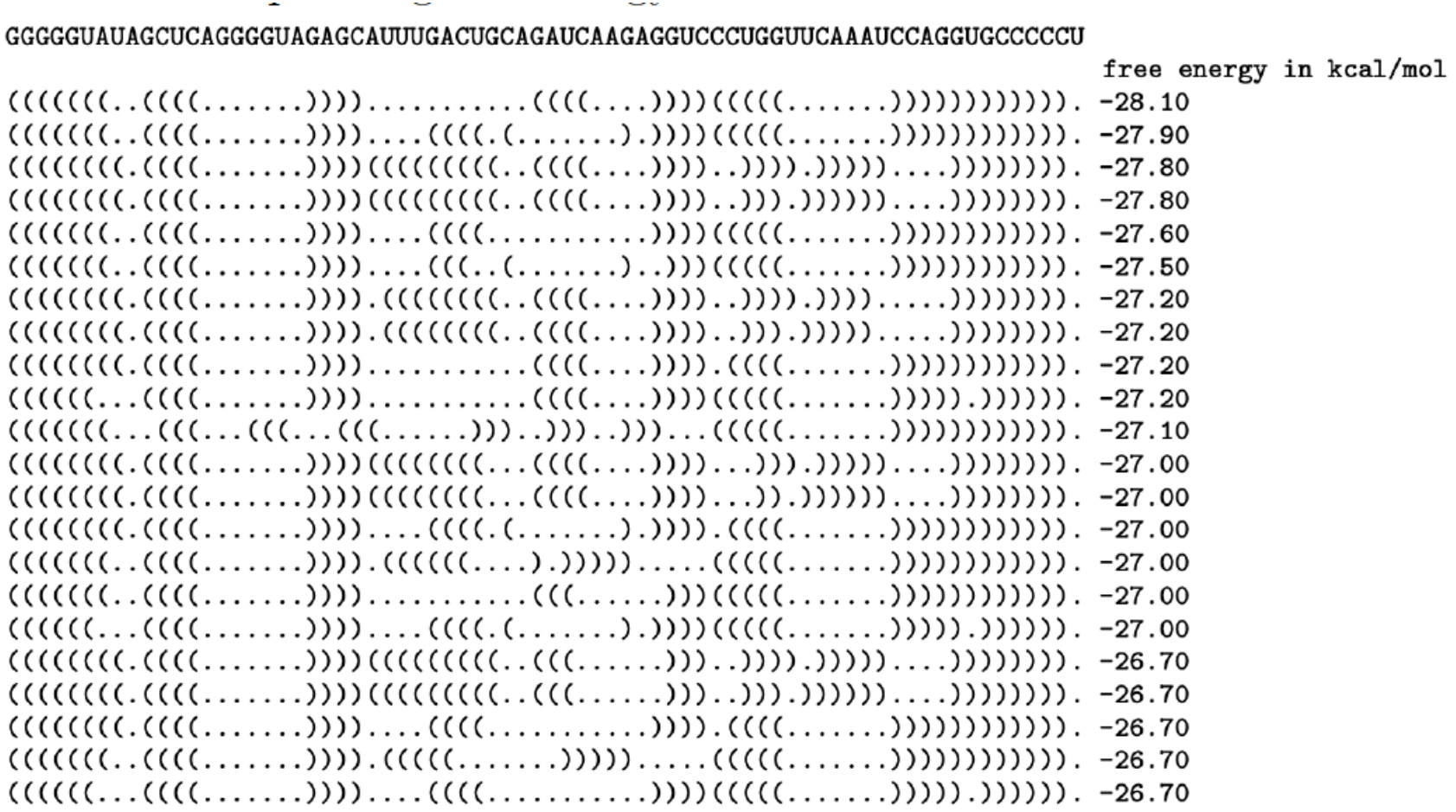
(((.....)))....(((.....))).

(((.....[[[)))...]]]..

Ruth Nussinov

dot-bracket representation for a pseudoknot free structure, as well as the extended pseudoknot representation for a structure containing a pseudoknot.

Link to Image Sour



usually the more the links the more the binding energy. Above: Ensemble of all possible structures for a given RNA sequence, with the corresponding binding energy. The potential energy is negative because you need to give energy to break the links (i.e. the structure), for example by heating.

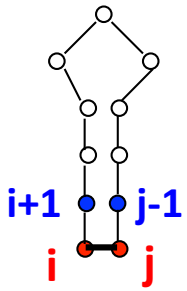
Link to Image Sour

RNA Secondary Structure

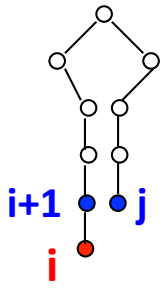
secondary structure=topology of local segments

- Secondary Structure :
  - Set of paired positions on interval $[i,j]$
  - This tells which bases are paired in the subsequence from x_i to x_j
- Every optimal structure can be built by extending optimal substructures.
- Suppose we know all optimal substructures of length less than $j\text{-}i+1$.
  The optimal substructure for $[i,j]$ must be formed in one of four ways:
  1. i,j paired
  2. i unpaired
  3. j unpaired
  4. combining two substructures

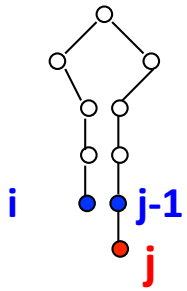
  Note that each of these consists of extending or joining substructures of length less than $j\text{-}i+1$.



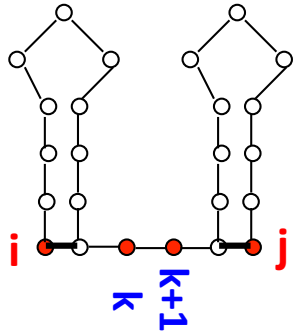
i,j pair



i unpaired



j unpaired



bifurcation

RNA Secondary Structure: The Nussinov Folding Algorithm

Nussinov, R., Pieczenik, G., Griggs, J. R. and Kleitman, D. J. (1978). Algorithms for loop matchings, SIAM J. Appl. Math

Example: GGGAAAUCC

$\gamma(i,j)$ is the maximum number of base pairs in segment $[i,j]$

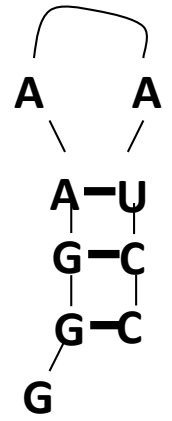
$$Initialisation\ \gamma(i,i-1) = 0\ \&\ \gamma(i,i) = 0$$

Starting with all subsequences of length 2, to length L:

$$\gamma(i,j) = \max \begin{cases} \gamma(i+1,j) \\ \gamma(i,j-1) \\ \gamma(i+1,j-1) + \delta(i,j) \\ \max_{i<k<j}[\gamma(i,k) + \gamma(k+1,j)] \end{cases}$$

Where $d(i,j) = 1$ if x_i and x_j are a complementary base pair, and $d(i,j) = 0$, otherwise.

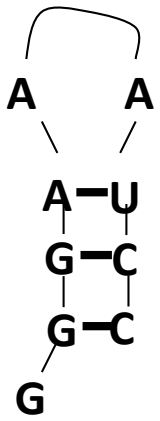
final structure



		G	G	G	A	A	A	U	C	C
		0								
G		0	0							
G			0	0						
G				0	0					
A					0	0				
A						0	0			
U							0	0		
C								0	0	
C									0	0

Nussinov Folding Algorithm: After scores for subsequences of length 2

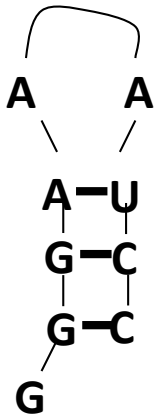
$$\gamma(i,j) = \max \begin{cases} \gamma(i+1,j) \\ \gamma(i,j-1) \\ \gamma(i+1,j-1) + \delta(i,j) \\ \max_{i<k<j}[\gamma(i,k) + \gamma(k+1,j)] \end{cases}$$



		G	G	G	A	A	A	U	C	C
	0	**0**							
G	0	0	**0**						
G		0	0	**0**					
G			0	0	**0**				
A				0	0	**0**			
A					0	0	**1**		
A						0	0	**0**	
U							0	0	**0**
C								0	0
C									

Nussinov Folding Algorithm: After scores for subsequences of length 3

$$\gamma(i,j) = \max \begin{cases} \gamma(i+1,j) \\ \gamma(i,j-1) \\ \gamma(i+1,j-1) + \delta(i,j) \\ \max_{i<k<j}[\gamma(i,k) + \gamma(k+1,j)] \end{cases}$$



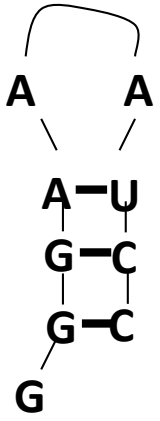
j →

i ↓

	G	G	G	A	A	A	U	C	C
G	0	0	**0**						
G	0	0	0	**0**					
G		0	0	0	**0**				
A			0	0	0	**0**			
A				0	0	0	**1**		
U					0	0	1	**0**	
C						0	0	0	**0**
C							0	0	0
								0	0

Nussinov Folding Algorithm: After scores for subsequences of length 4

$$\gamma(i,j) = \max \begin{cases} \gamma(i+1,j) \\ \gamma(i,j-1) \\ \gamma(i+1,j-1)+\delta(i,j) \\ \max_{i<k<j}[\gamma(i,k)+\gamma(k+1,j)] \end{cases}$$



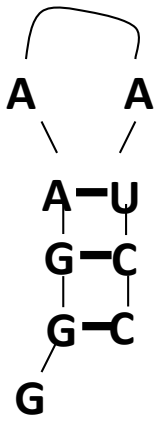
j →

i ↓	G	G	G	A	A	A	U	C	C
G	0	0	0	**0**					
G	0	0	0	0	**0**				
G		0	0	0	0	**0**			
A			0	0	0	0	**1**		
A				0	0	**0**	**1**	**1**	
U					0	0	1	1	**1**
C						0	0	0	0
C							0	0	0
								0	0

Two optimal substructures for same subsequence

Nussinov Folding Algorithm: After scores for subsequences of length 5

$$\gamma(i,j) = \max \begin{cases} \gamma(i+1,j) \\ \gamma(i,j-1) \\ \gamma(i+1,j-1) + \delta(i,j) \\ \max_{i<k<j}[\gamma(i,k) + \gamma(k+1,j)] \end{cases}$$

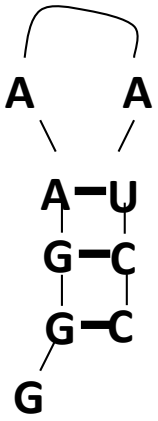


j →

i ↓	G	G	G	A	A	A	U	C	C
G	0	0	0	0	**0**				
G	0	0	0	0	0	**0**			
G		0	0	0	0	0	**1**		
A			0	0	0	0	1	**1**	
A				0	0	0	1	1	**1**
U					0	0	1	1	1
C						0	0	0	0
C							0	0	0
								0	0

Nussinov Folding Algorithm: After scores for subsequences of length 6

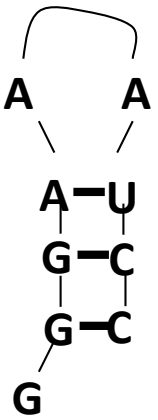
$$\gamma(i,j) = \max \begin{cases} \gamma(i+1,j) \\ \gamma(i,j-1) \\ \gamma(i+1,j-1) + \delta(i,j) \\ \max_{i<k<j}[\gamma(i,k) + \gamma(k+1,j)] \end{cases}$$



i \ j	G	G	G	A	A	A	U	C	C
G	0	0	0	0	0	**0**			
G	0	0	0	0	0	0	**1**		
G		0	0	0	0	0	1	**2**	
A			0	0	0	0	1	1	**1**
A				0	0	0	1	1	1
A					0	0	1	1	1
U						0	0	0	0
C							0	0	0
C								0	0

Nussinov Folding Algorithm
After scores for subsequences of length 7

$$\gamma(i,j) = \max \begin{cases} \gamma(i+1,j) \\ \gamma(i,j-1) \\ \gamma(i+1,j-1) + \delta(i,j) \\ \max_{i<k<j}[\gamma(i,k) + \gamma(k+1,j)] \end{cases}$$



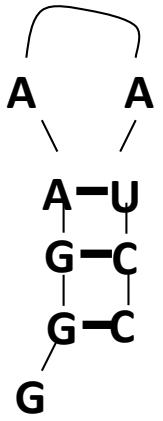
j →

i ↓

	G	G	G	A	A	A	U	C	C
G	0	0	0	0	0	0	**1**		
G	0	0	0	0	0	0	1	**2**	
G		0	0	0	0	0	1	2	**2**
A			0	0	0	0	1	1	1
A				0	0	0	1	1	1
U					0	0	1	1	1
C						0	0	0	0
C							0	0	0
								0	0

Nussinov Folding Algorithm
After scores for subsequences of length 8

$$\gamma(i,j) = \max \begin{cases} \gamma(i+1,j) \\ \gamma(i,j-1) \\ \gamma(i+1,j-1) + \delta(i,j) \\ \max_{i<k<j}[\gamma(i,k) + \gamma(k+1,j)] \end{cases}$$



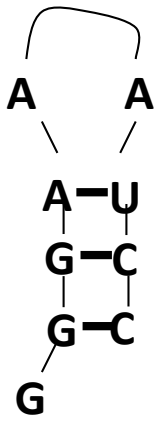
j →

i ↓

	G	G	G	A	A	A	U	C	C
G	0	0	0	0	0	0	1	**2**	
G	0	0	0	0	0	0	1	2	**3**
G		0	0	0	0	0	1	2	2
A			0	0	0	0	1	1	1
A				0	0	0	1	1	1
A					0	0	1	1	1
U						0	0	0	0
C							0	0	0
C								0	0

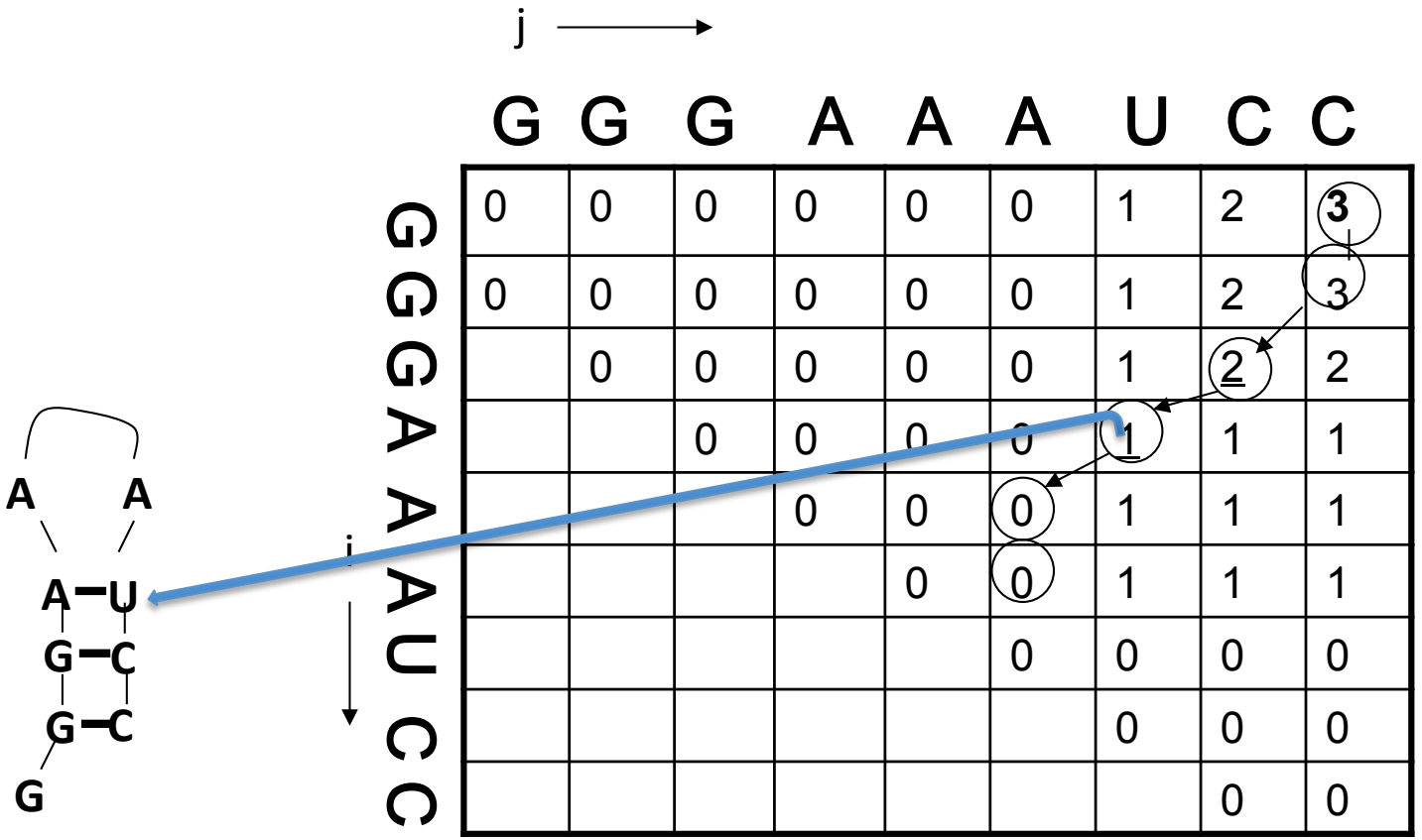
Nussinov Folding Algorithm
After scores for subsequences of length 9

$$\gamma(i,j) = \max \begin{cases} \gamma(i+1,j) \\ \gamma(i,j-1) \\ \gamma(i+1,j-1) + \delta(i,j) \\ \max_{i<k<j}[\gamma(i,k) + \gamma(k+1,j)] \end{cases}$$



i \ j	G	G	G	A	A	A	U	C	C
G	0	0	0	0	0	0	1	2	**3**
G	0	0	0	0	0	0	1	2	3
G		0	0	0	0	0	1	2	2
A			0	0	0	0	1	1	1
A				0	0	0	1	1	1
A					0	0	1	1	1
U						0	0	0	0
C							0	0	0
C								0	0

Nussinov Folding Algorithm Traceback



Nussinov algorithm (a different example): fill-stage

Algorithm: Nussinov RNA folding, fill stage

Initialisation:

$$\gamma(i,i-1) = 0 \qquad \text{for } i = 2 \text{ to } L;$$
$$\gamma(i,i) = 0 \qquad \text{for } i = 1 \text{ to } L.$$

Recursion: starting with all subsequences of length 2, to length L:

$$\gamma(i,j) = \max \begin{cases} \gamma(i+1,j), \\ \gamma(i,j-1), \\ \gamma(i+1,j-1)+\delta(i,j), \\ \max_{i<k<j}\left[\gamma(i,k)+\gamma(k+1,j)\right]. \end{cases}$$

G	G	C	C	A	G	U	U	C
1	2	3	4	5	6	7	8	9

G	1	0	0	1	2	2	2	3	4	4
G	2	0	0	1	1	1	2	2	3	3
C	3		0	0	0	0	1	1	2	2
C	4			0	0	0	1	1	2	2
A	5				0	0	0	1	2	2
G	6					0	0	1	1	1
U	7						0	0	0	0
U	8							0	0	0
C	9								0	0

Scoring system:
$\delta(i,j) = 1$ for all RNA Watson-Crick base-pairs including G-U else $\delta(i,j) = 0$.

Blue: addition of unpaired base 3 or 7

Green: addition of paired bases 1,7

Pink: joining of substructures 1..4 and 5..8

Nussinov algorithm: trace-back

G	G	C	C	A	G	U	U	C
1	2	3	4	5	6	7	8	9

G	1
G	2
C	3
C	4
A	5
G	6
U	7
U	8
C	9

0	0	1	2	2	2	3	4	4
0	0	1	1	1	2	2	3	3
	0	0	0	0	1	1	2	2
		0	0	0	1	1	2	2
			0	0	0	1	2	2
				0	0	1	1	1
					0	0	0	0
						0	0	0
							0	0

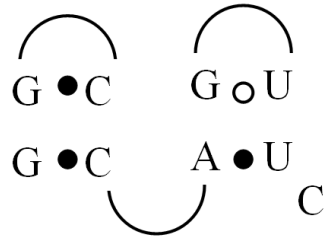
Algorithm: Nussinov RNA folding, traceback stage

Initialisation: Push $(1, L)$ onto stack.

Recursion: Repeat until stack is empty:

- pop (i, j).
- if $i >= j$ continue;
  else if $\gamma(i+1, j) = \gamma(i, j)$ push $(i+1, j)$;
  else if $\gamma(i, j-1) = \gamma(i, j)$ push $(i, j-1)$;
  else if $\gamma(i+1, j-1) + \delta_{i,j} = \gamma(i, j)$:
    - record i, j base pair.
    - push $(i+1, j-1)$.
  else for $k = i+1$ to $j-1$: if $\gamma(i,k) + \gamma(k+1, j) = \gamma(i, j)$:
    - push $(k+1, j)$.
    - push (i, k).
    - break.

current	record	stack
		1,9
1,9		1,8
1,8		1,4 5,8
1,4	1,4	2,3 5,8
2,3	2,3	3,2 5,8
3,2		5,8
5,8	5,8	6,7
6,7	6,7	7,6
7,6		



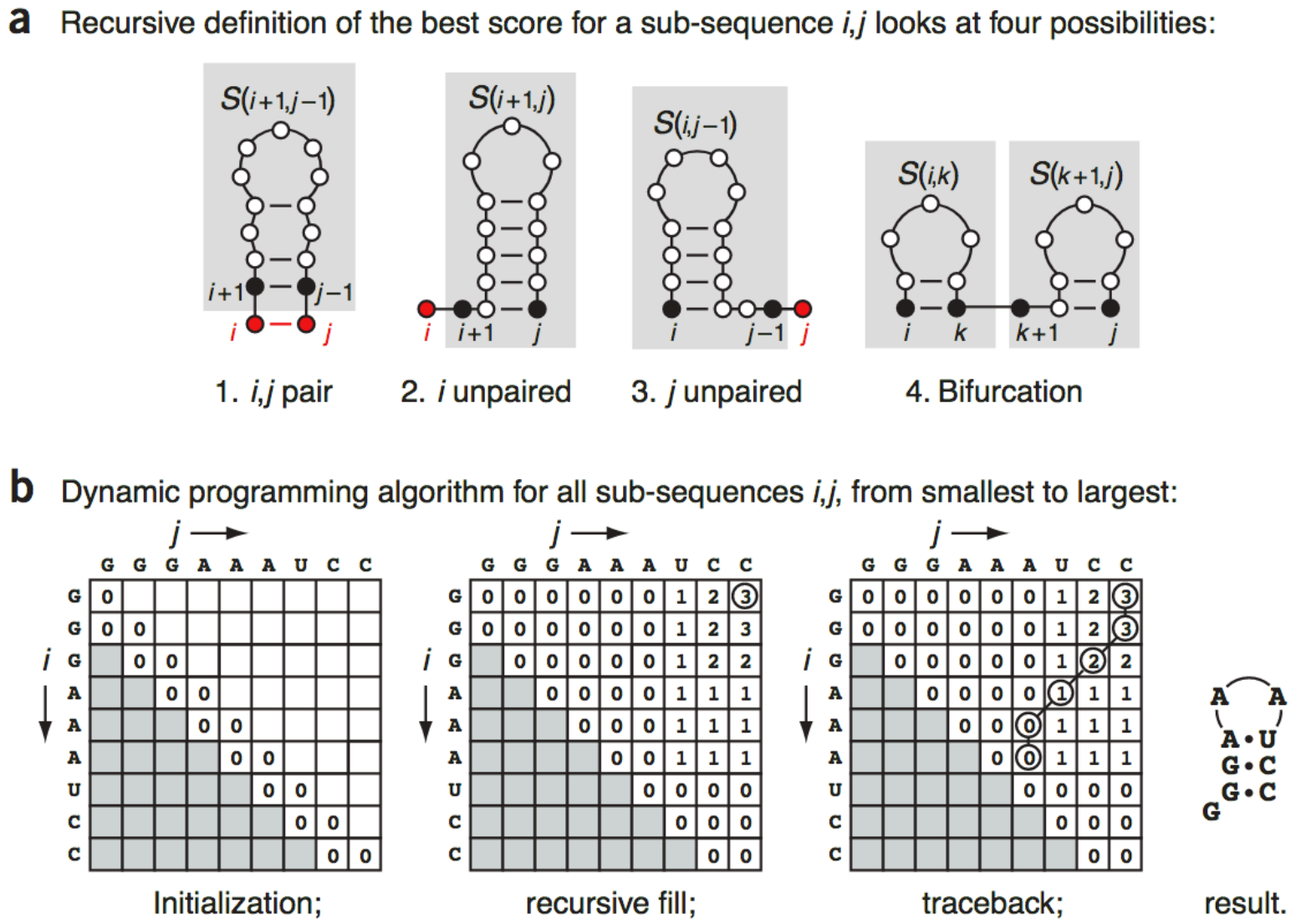


Figure 1 Dynamic programming algorithm for RNA secondary structure prediction. (**a**) The four cases examined by the dynamic programming recursion. Red dots mark the bases being added onto previously calculated optimal sub-structures (*i,j* pair, unpaired *i* or unpaired *j*). Gray boxes are a reminder that the recursion tabulates the *score* of the smaller optimal sub-structures, not the structures themselves. Example sub-structures are shown in the gray boxes solely as examples. (b) The dynamic programming algorithm in operation, showing the matrix $S(i,j)$ for a sequence GGGAAAUCC after initialization, after the recursive fill, and after an optimal structure with three base pairs has been traced back.

RNA Secondary Structure: The Nussinov Folding Algorithm

Nussinov, R., Pieczenik, G., Griggs, J. R. and Kleitman, D. J. (1978). Algorithms for loop matchings, SIAM J. Appl. Math

$$\textit{Initialisation}\ \gamma(i,i-1) = 0\ \&\ \gamma(i,i) = 0$$

$$\gamma(i,j) = \max \begin{cases} \gamma(i+1,j) \\ \gamma(i,j-1) \\ \gamma(i+1,j-1) + \delta(i,j) \\ \max_{i<k<j}[\gamma(i,k) + \gamma(k+1,j)] \end{cases}$$

There are $O(n^2)$ terms to be computed, each requiring calling of O(n) already computed terms for the case of bifurcation. Thus overall complexity is $O(n^3)$ in time and $O(n^2)$ in space.

Summary
(note different notation!)

- Initialise:
  - Sequence: GGGAAAUCC, length (L) $= 9$.
  - $N_{i,i-1} = 0$ for $i = 2 - L$
  - $N_{i,i} = 0$ for $i = 1 - L$

	G	G	G	A	A	A	U	C	C
G	0								
G	0	0							
G		0	0						
A			0	0					
A				0	0				
A					0	0			
U						0	0		
C							0	0	
C								0	0

- Recursion:
  - $\rho(i,j) = 1$ if s_i and s_j are complementary, otherwise $\rho(i,j) = 0$.

	G	G	G	A	A	A	U	C	C
G	0	0	0	0	0	0	1	2	3
G	0	0	0	0	0	0	1	2	3
G		0	0	0	0	0	1	2	2
A			0	0	0	0	1	1	1
A				0	0	0	1	1	1
A					0	0	1	1	1
U						0	0	0	0
C							0	0	0
C								0	0

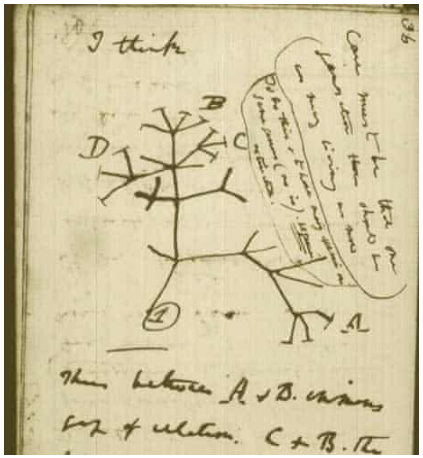
$$N_{i,j} = \max \begin{cases} N_{i+1,j-1} + \rho(i,j), & i,j \text{ pair} \\ N_{i+1,j}, & i \text{ unpaired} \\ N_{i,j-1}, & j \text{ unpaired} \\ \max_{i<k<j}[N_{i,k} + N_{k+1,j}] & \text{bifurcation} \end{cases}$$

- Traceback:

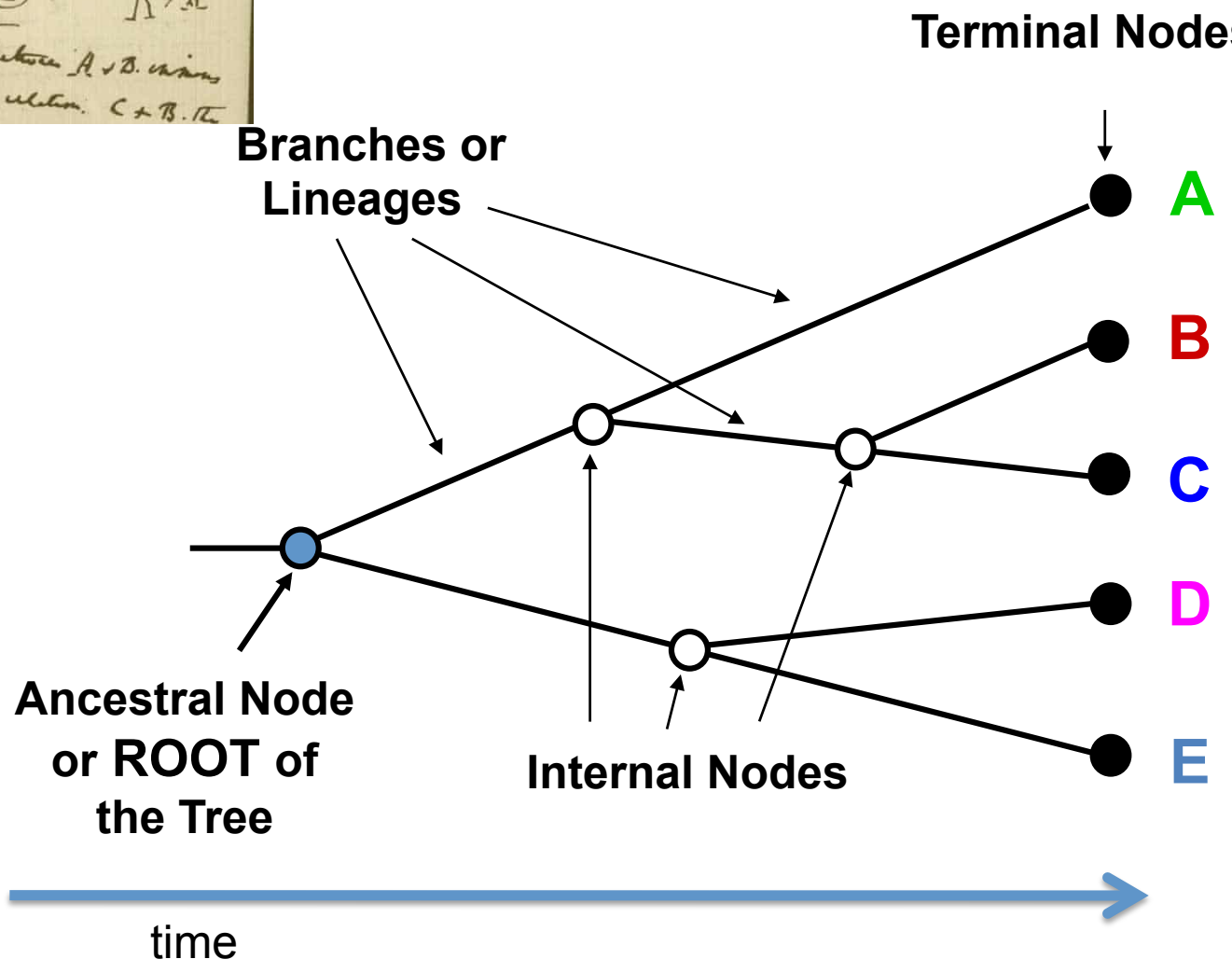
	G	G	G	A	A	A	U	C	C
G	0	0	0	0	0	0	1	2	**3**
G	0	0	0	0	0	0	1	2	**3**
G		0	0	0	0	0	1	**2**	2
A			0	0	0	0	**1**	1	1
A				0	0	**0**	1	1	1
A					0	**0**	1	1	1
U						0	0	0	0
C							0	0	0
C								0	0

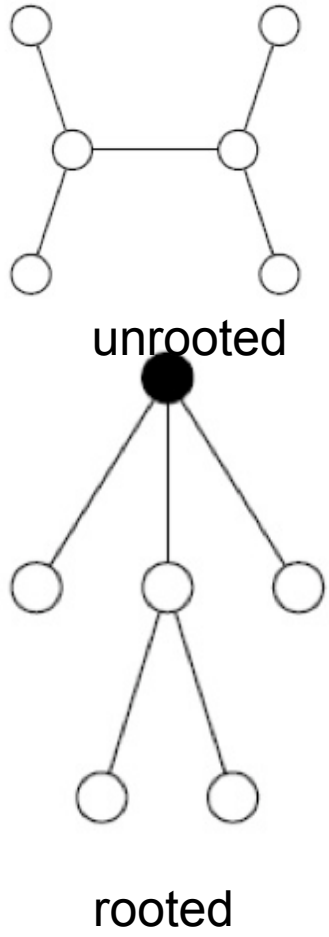
Example lifted from: Durbin *et al.* (1998) Biological sequence analysis. *Cambridge University Press.*

Phylogeny



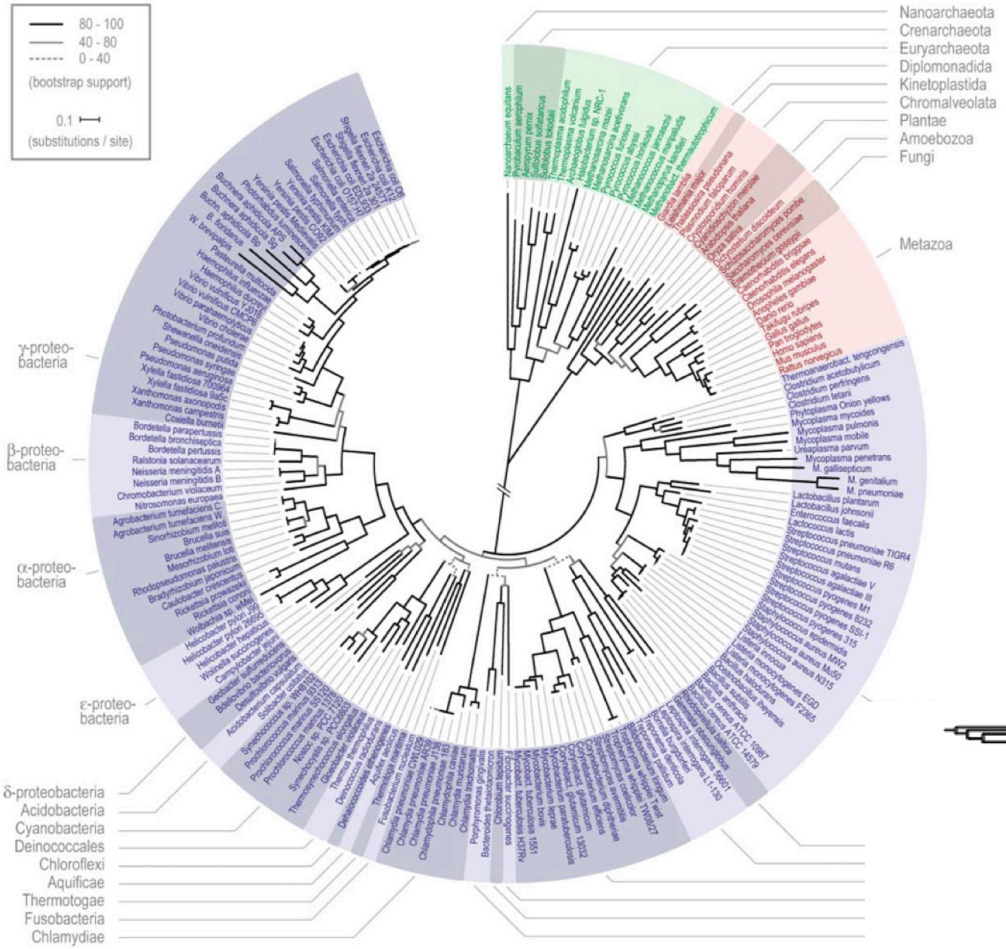
species tree by Darwin



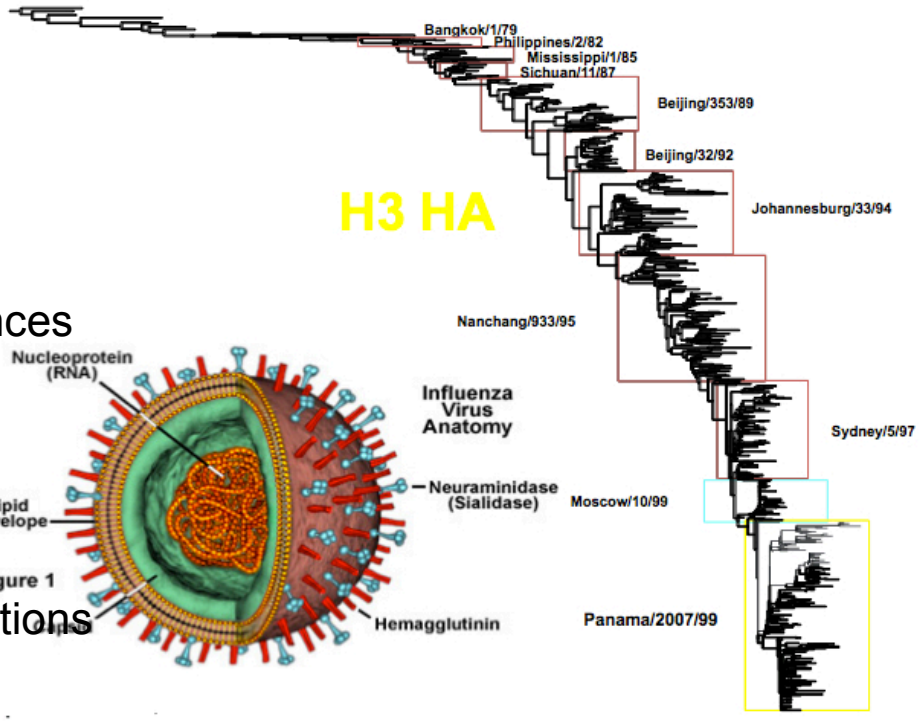


((A,(B,C)),(D,E)) = The above phylogeny as nested parentheses

Phylogenetic tree applications



tree of life based on mitochondrial sequences

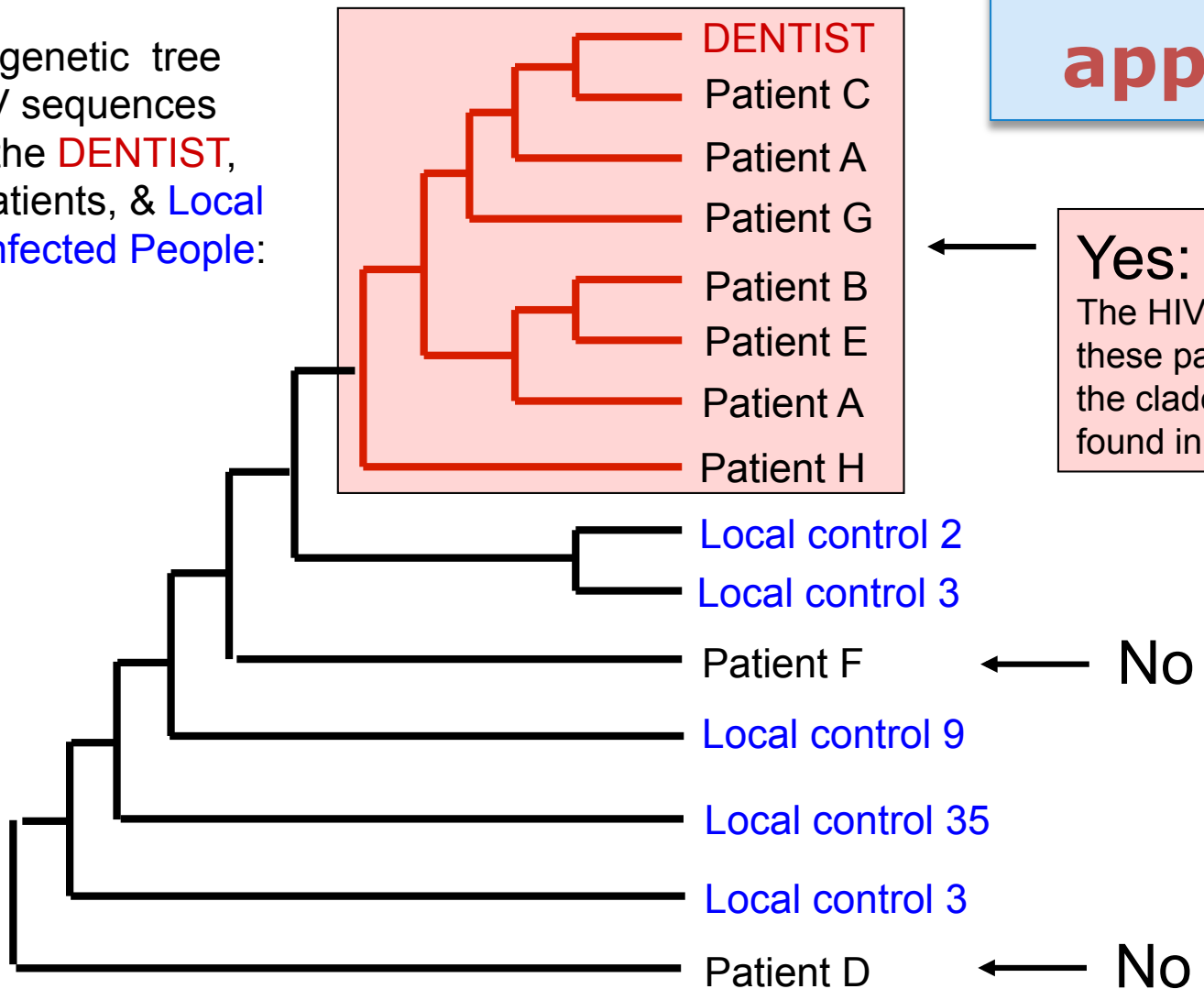


tracing influenza strain variations

Did the *Florida Dentist* infect his pati

Phylogenetic tree applications

Phylogenetic tree of HIV sequences from the DENTIST, his Patients, & Local HIV-infected People:



Yes:
The HIV sequences from these patients fall within the clade of HIV sequences found in the dentist.

No

No

From Ou et *al*. (1992) and Page & Holmes (1998)

EXAMPLE: Phylogenetic-inspired techniques for reverse engineering and detection of malware families

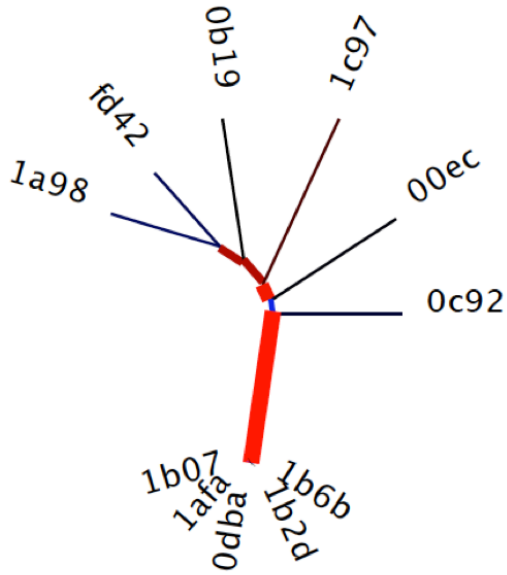
For example, given an execution trace of instructions,

```
push ebp
mov ebp, esp
mov eax, dword ptr [ebp-0x4]
jmp +0x14
```

it is abstracted as a sequence of mnemonics, i.e.

```
push, mov, mov, jmp
```

ignoring the operands. Each mnemonic is then mapped to a unique alphabet-pair, e.g. `mov` = MO, `push` = PH, `jmp` = JM. The resulting sequence is thus `PHMOMOJM`.




```
dbg PHMGSVPHPHPHLEMGMGRPMGMGADCMHLMGADCMHLMGADCMHYMGIMMGIMADMGIMCMHZMGGRMGHMMGCMHZMGCMHZMGCMHYMGGRMGMGPPPPPPMGPPRE
def PHMGPHMGMGADCMHLMGADCMHLMGADCMHYMGIMMGIMADMGIMCMHZMGGRMGHMMGCMHZMGCMHZMGCMHYMGGRMGMGMGPPRE--------------------
spd PHMGPHMGPHMGLECMHLLECMHLLECMHLMGMGIMIMMGPHIMLECMHZMGCMPPHZCMHZCMHZPPPPGRPPRE--------------------------------

(b)
dbg PHMGSVPHPHPHLEMGMGRPMGMGADCMHLMGADCMHLMGADCMHYMGIMMGIMADMG----IMCMHZMGGRMGHMMGCMHZMGCMHZMGCMHYMGGRMGMGPPPPPPMGPPRE
def PHMG--PH------------MGMGADCMHLMGADCMHLMGADCMHYMGIMMGIMADMG----IMCMHZMGGRMGHMMGCMHZMGCMHZMGCMHYMGGRMGMG------MGPPRE
spd PHMG--PHMGP----------HMGLECMHLL--ECMHLL--ECMHLMG--MGIMIMMGPHIMLECMHZMG--------CMPPHZCMHZ--CMHZ----------PPPPGRPPRE

(c)
dbg PHMGSVPHPHPHLEMGMGRPMGMGADCMHLMGADCMHLMGADCMHYMGIMMGIMADMGIMCMHZMGGRMGHMMGCMHZMGCMHZMGCMHYMGGRMGMGPPPPPPMGPPRE
def --------------PHMGPHMGMGADCMHLMGADCMHLMGADCMHYMGIMMGIMADMGIMCMHZMGGRMGHMMGCMHZMGCMHZMGCMHYMGGRMGMGMGPPRE------
spd --------------PHMGPHMGPHMGLECMHLLECMHLLECMHLMGMGIMIMMGPHIMLECMH--------------ZMGCMPPHZCMHZCMHZPPPPGRPPRE------
```

Sequence alignment (dbg: with debugging symbols, def: default settings, spd: optimised for speed). (a) Before alignment. (b) After alignment using an identity substitution matrix. (c) After alignment using a substitution matrix

Trees and Phylogeny
Outline

- Transforming Distance Matrices into Evolutionary Trees
- Toward an Algorithm for Distance-Based Phylogeny Construction
- Additive Phylogeny
- Using Least-Squares to Construct Distance-Based Phylogenies
- Ultrametric Evolutionary Trees
- The Neighbor-Joining Algorithm
- Character-Based Tree Reconstruction
- The Small Parsimony Problem
- The Large Parsimony Problem
- Back to the alignment: progressive alignment

Constructing a Distance Matrix

$D_{i,j}$ = number of differing symbols between i-th and j-th rows of a "multiple alignment".

Species	Alignment	Distance Matrix			
		Chimp	**Human**	**Seal**	**Whale**
Chimp	ACGTAGGCCT	0	3	6	4
Human	ATGTAAGACT	3	0	7	5
Seal	TCGAGAGCAC	6	7	0	2
Whale	TCGAAAGCAT	4	5	2	0

Constructing a Distance Matrix

$D_{i,j}$ = number of differing symbols between i-th and j-th rows of a "multiple alignment".

Species	Alignment
Chimp	ACGTAGGCCT
Human	ATGTAAGACT
Seal	TCGAGAGCAC
Whale	TCGAAAGCAT

Distance Matrix

	Chimp	**Human**	**Seal**	**Whale**
Chimp	0	**3**	6	4
Human	**3**	0	7	5
Seal	6	7	0	2
Whale	4	5	2	0

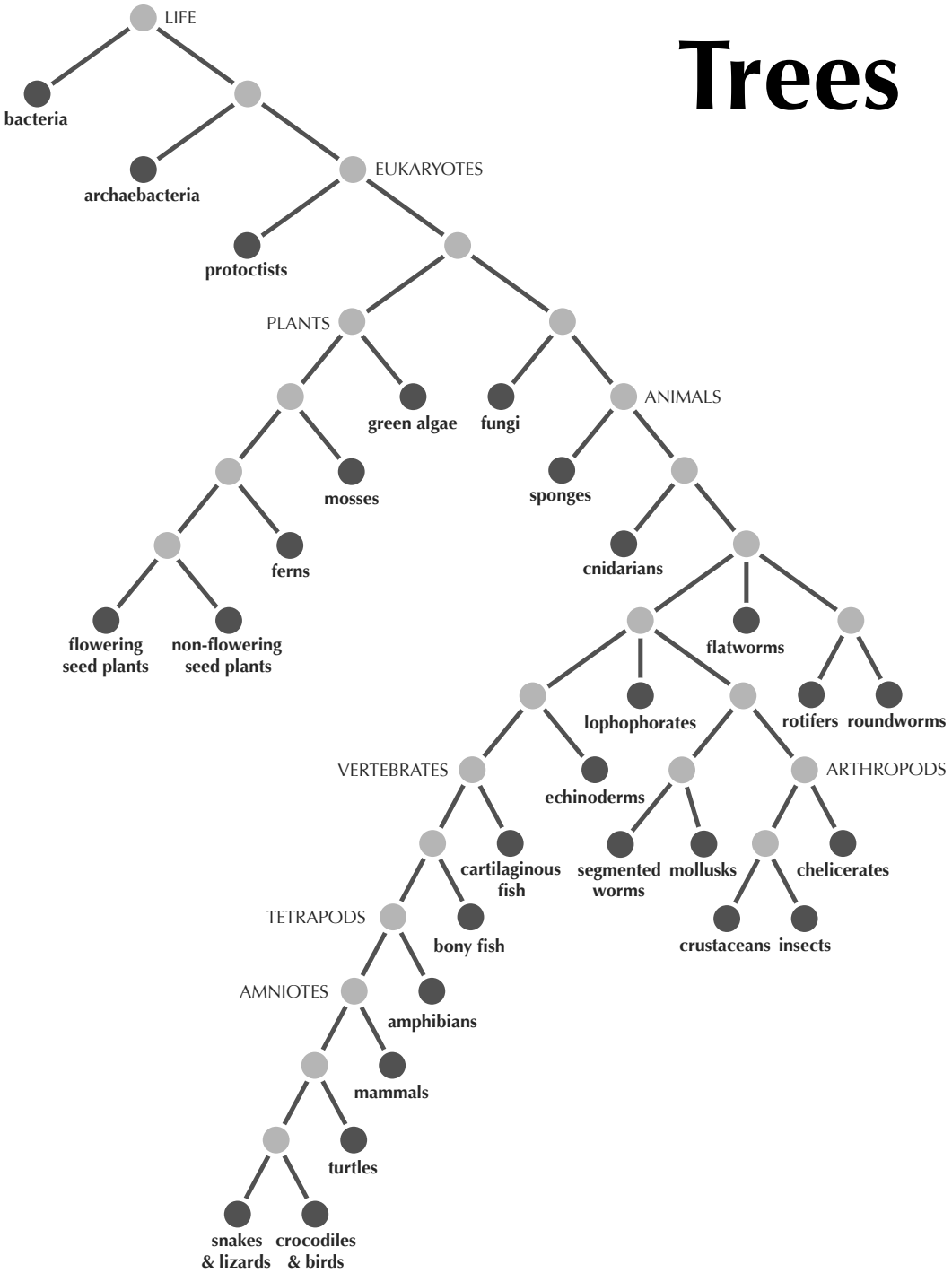
Constructing a Distance Matrix

$D_{i,j}$ = number of differing symbols between i-th and j-th rows of a multiple alignment.

Species	Alignment	Distance Matrix			
		Chimp	**Human**	**Seal**	**Whale**
Chimp	ACGTAGGCCT	0	3	6	4
Human	ATGTAAGACT	3	0	7	5
Seal	TCGAGAGCAC	6	7	0	2
Whale	TCGAAAGCAT	4	5	2	0

How else could we form a distance matrix?

Trees

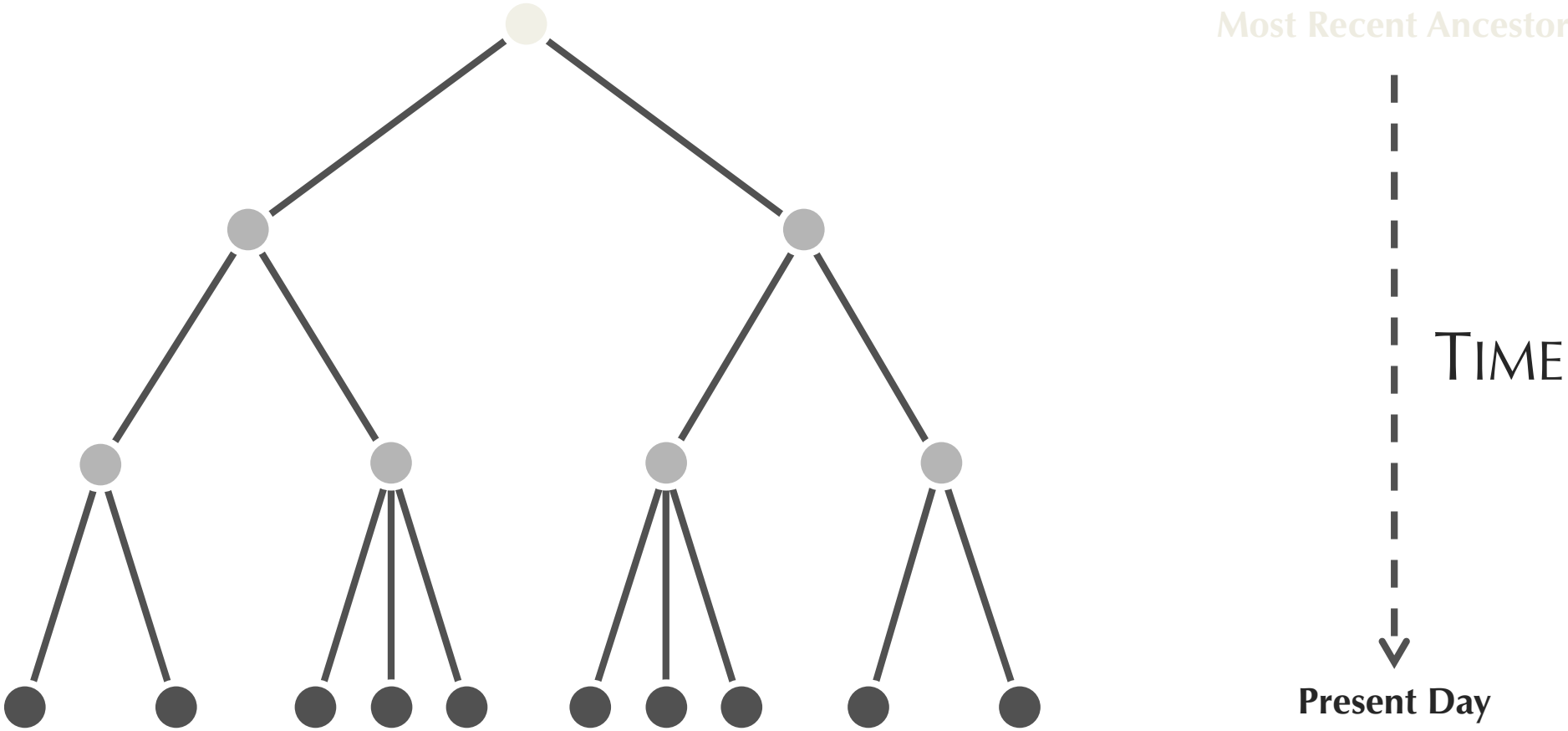


Tree: Connected graph containing no cycles.

Leaves (degree = 1): present-day species

Internal nodes (degree ≥ 1): ancestral species

Trees



Rooted tree: one node is designated as the root (most recent common ancestor)

Distance-Based Phylogeny

Distance-Based Phylogeny Problem: *Construct an evolutionary tree from a distance matrix.*

- **Input:** A distance matrix.
- **Output:** The unrooted tree "fitting" this distance matrix.

Constructing a Distance Matrix

$D_{i,j}$ = number of differing symbols between i-th and j-th rows of a "multiple alignment".

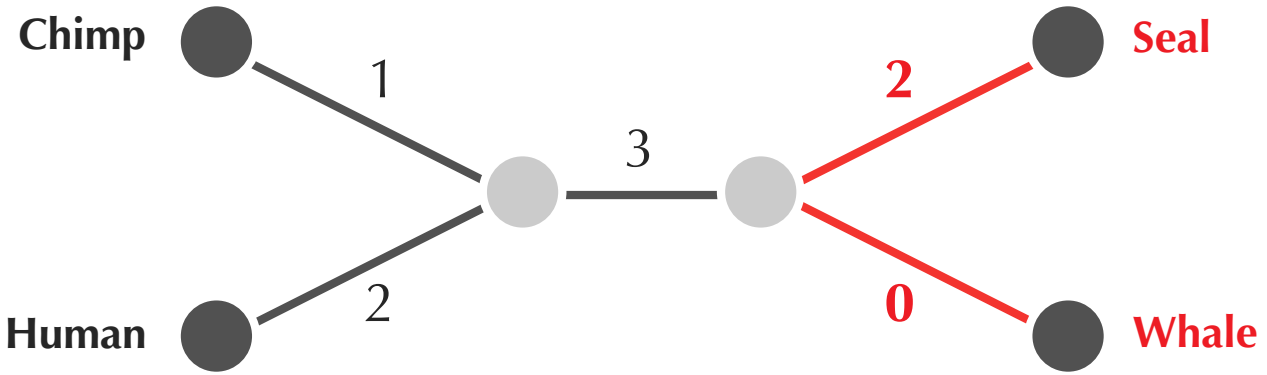
Species	Alignment
Chimp	ACGTAGGCCT
Human	ATGTAAGACT
Seal	TCGAGAGCAC
Whale	TCGAAAGCAT

Distance Matrix

	Chimp	**Human**	**Seal**	**Whale**
Chimp	0	**3**	6	4
Human	**3**	0	7	5
Seal	6	7	0	2
Whale	4	5	2	0

Fitting a Tree to a Matrix

	Chimp	Human	Seal	Whale
Chimp	0	3	6	4
Human	3	0	7	5
Seal	6	7	0	**2**
Whale	4	5	2	0



Return to Distance-Based Phylogeny

Distance-Based Phylogeny Problem: *Construct an evolutionary tree from a distance matrix.*

- **Input:** A distance matrix.
- **Output:** The unrooted tree fitting this distance matrix.

Now is this problem well-defined?

Return to Distance-Based Phylogeny

Exercise Break: Try fitting a tree to the following matrix.

	i	*j*	*k*	*l*
i	0	3	4	3
j	3	0	4	5
k	4	4	0	2
l	3	5	2	0

No Tree Fits a Matrix

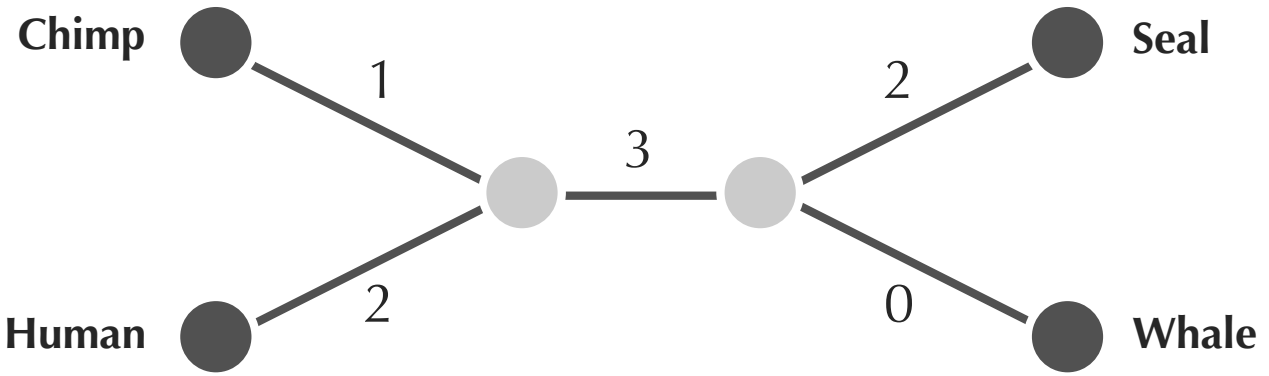
Exercise Break: Try fitting a tree to the following matrix.

	i	*j*	*k*	*l*
i	0	3	4	3
j	3	0	4	5
k	4	4	0	2
l	3	5	2	0

Additive matrix: distance matrix such that there exists an unrooted tree fitting it.

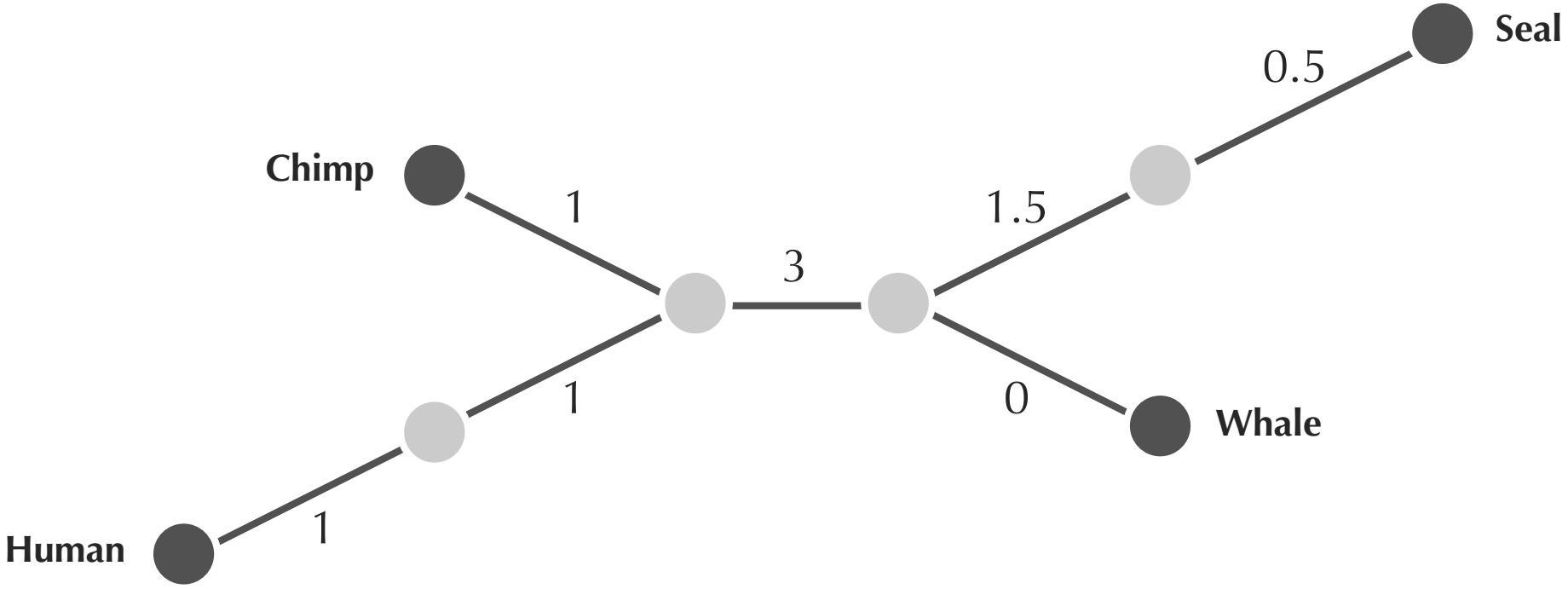
More Than One Tree Fits a Matrix

	Chimp	Human	Seal	Whale
Chimp	0	3	6	4
Human	3	0	7	5
Seal	6	7	0	2
Whale	4	5	2	0

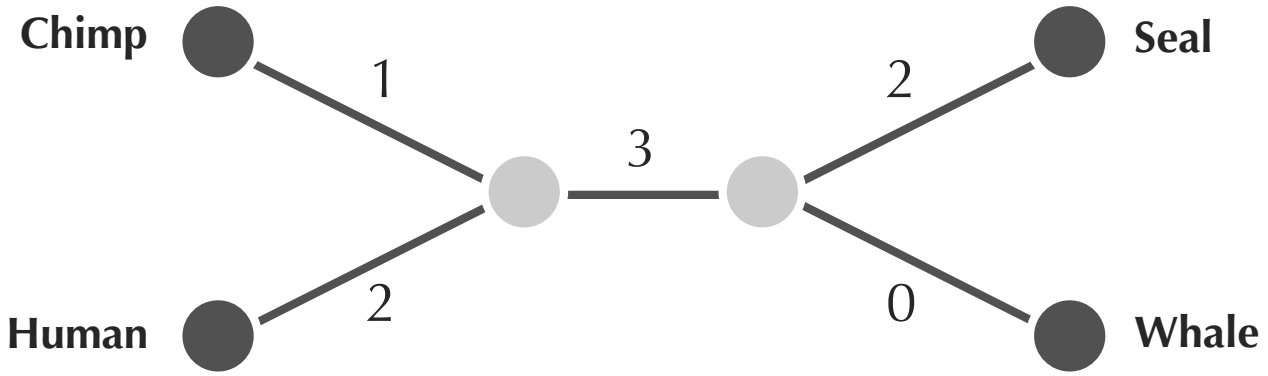


More Than One Tree Fits a Matrix

	Chimp	Human	Seal	Whale
Chimp	0	3	6	4
Human	3	0	7	5
Seal	6	7	0	2
Whale	4	5	2	0



Which Tree is "Better"?



Simple tree: tree with no nodes of degree 2.

Theorem: There is a unique *simple* tree fitting an *additive* matrix.

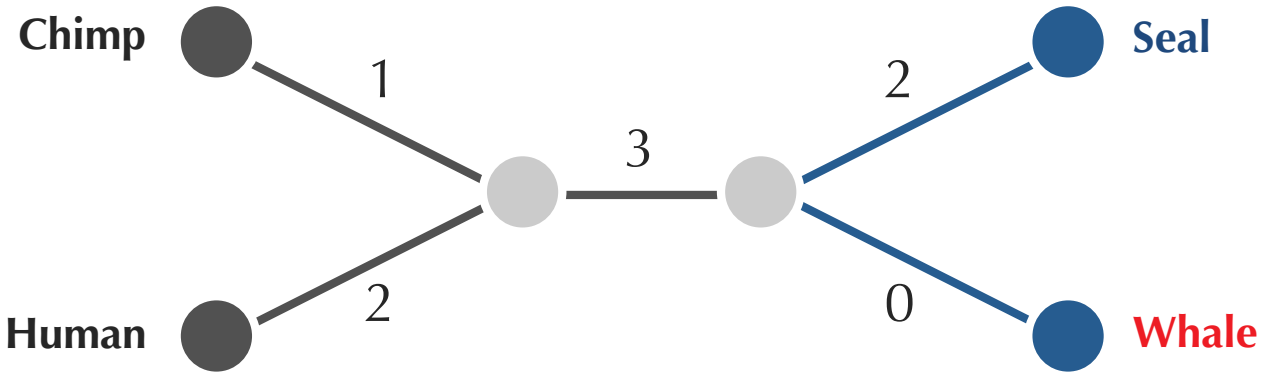
Reformulating Distance-Based Phylogeny

Distance-Based Phylogeny Problem: *Construct an evolutionary tree from a distance matrix.*

- **Input:** A distance matrix.
- **Output:** The simple tree fitting this distance matrix (if this matrix is additive).

An Idea for Distance-Based Phylogeny

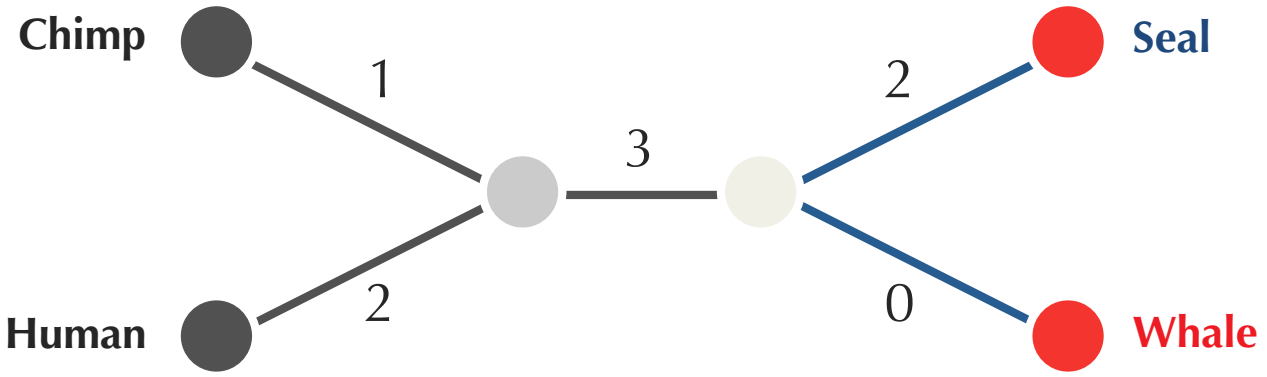
	Chimp	Human	Seal	Whale
Chimp	0	3	6	4
Human	3	0	7	5
Seal	6	7	0	**2**
Whale	4	5	2	0



An Idea for Distance-Based Phylogeny

Seal and whale are **neighbors** (meaning they share the same **parent**).

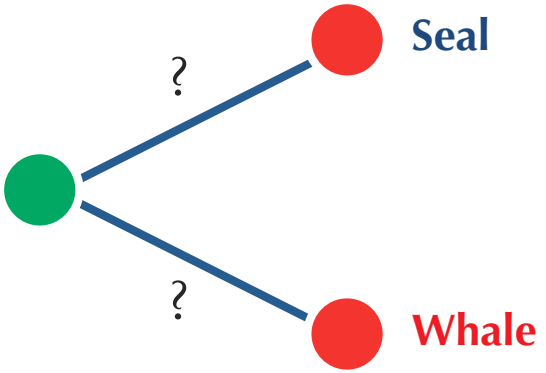
Theorem: Every simple tree with at least two nodes has at least one pair of neighboring leaves.



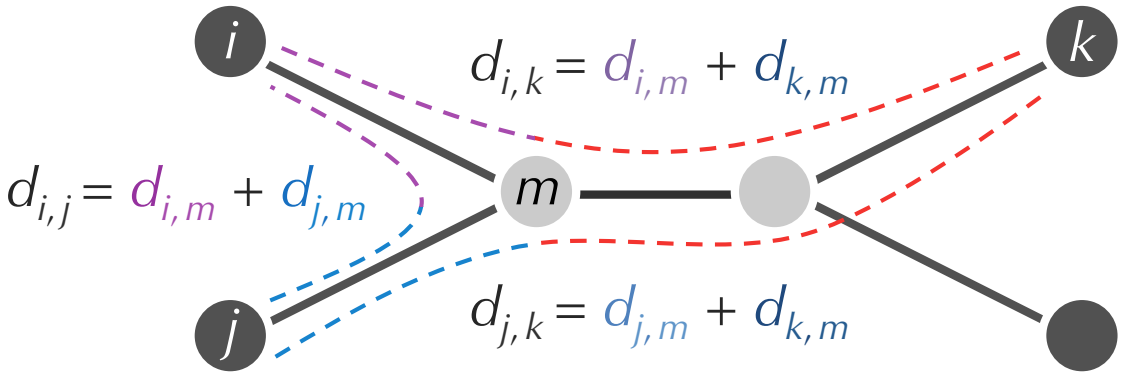
An Idea for Distance-Based Phylogeny

	Chimp	Human	Seal	Whale
Chimp	0	3	6	4
Human	3	0	7	5
Seal	6	7	0	**2**
Whale	4	5	2	0

How do we compute the unknown distances?

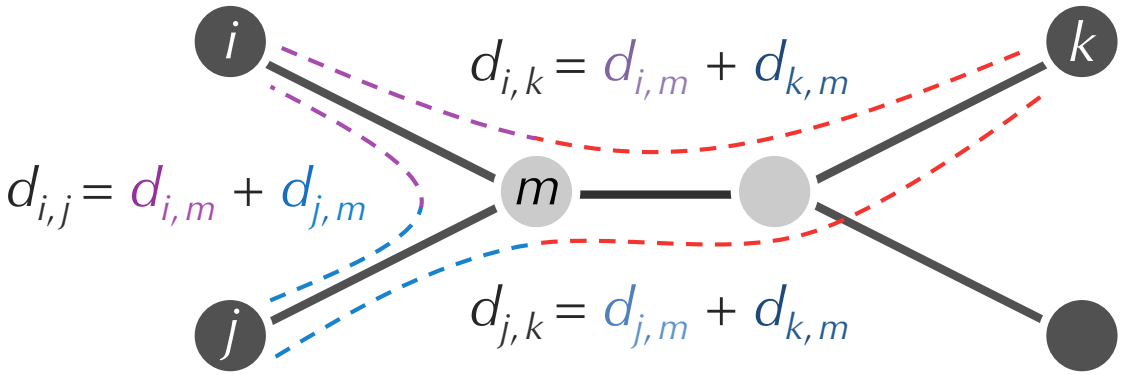


Toward a Recursive Algorithm



$$d_{k,m} = [(d_{i,m} + d_{k,m}) + (d_{j,m} + d_{k,m}) - (d_{i,m} + d_{j,m})] / 2$$

Toward a Recursive Algorithm



$$d_{k,m} = [(d_{i,m} + d_{k,m}) + (d_{j,m} + d_{k,m}) - (d_{i,m} + d_{j,m})] / 2$$

$$d_{k,m} = (d_{i,k} + d_{j,k} - d_{i,j}) / 2$$

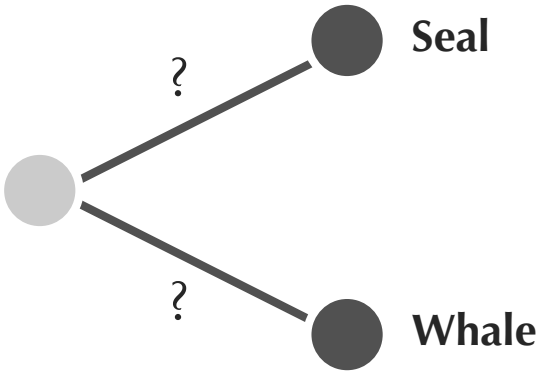
$$d_{k,m} = (D_{i,k} + D_{j,k} - D_{i,j}) / 2$$

$$\therefore \; d_{i,m} = D_{i,k} - (D_{i,k} + D_{j,k} - D_{i,j}) / 2$$

$$d_{i,m} = (D_{i,k} + D_{i,j} - D_{j,k}) / 2$$

An Idea for Distance-Based Phylogeny

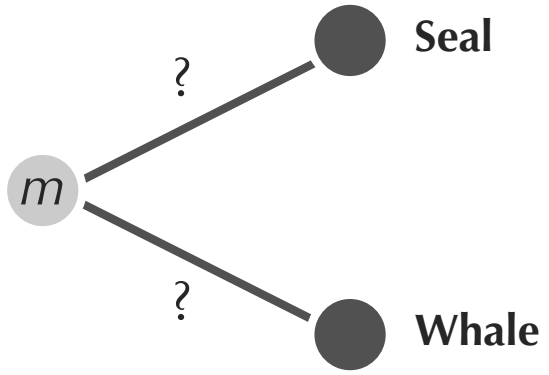
	Chimp	Human	Seal	Whale
Chimp	0	3	6	4
Human	3	0	7	5
Seal	6	7	0	2
Whale	4	5	2	0



$$d_{i,m} = (D_{i,k} + D_{i,j} - D_{j,k}) / 2$$

An Idea for Distance-Based Phylogeny

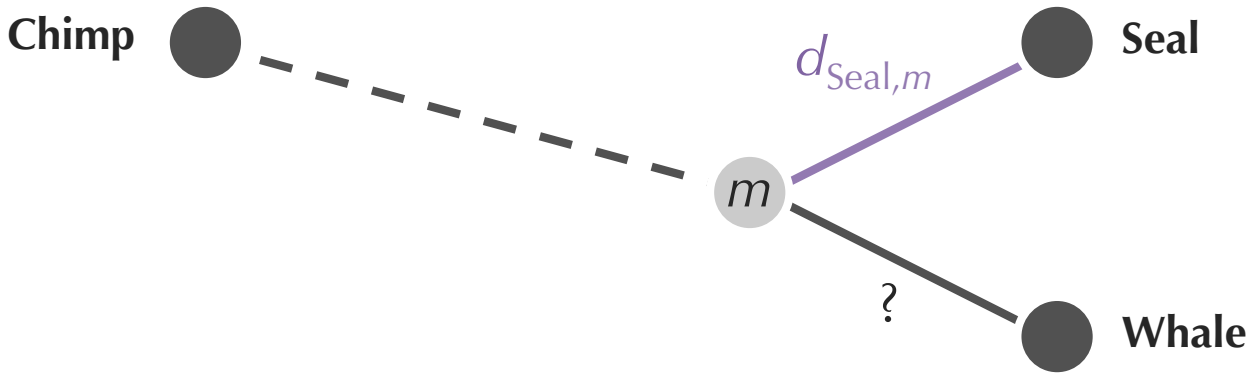
	Chimp	Human	Seal	Whale
Chimp	0	3	6	4
Human	3	0	7	5
Seal	6	7	0	2
Whale	4	5	2	0



$$d_{i,m} = (D_{i,k} + D_{i,j} - D_{j,k}) / 2$$

An Idea for Distance-Based Phylogeny

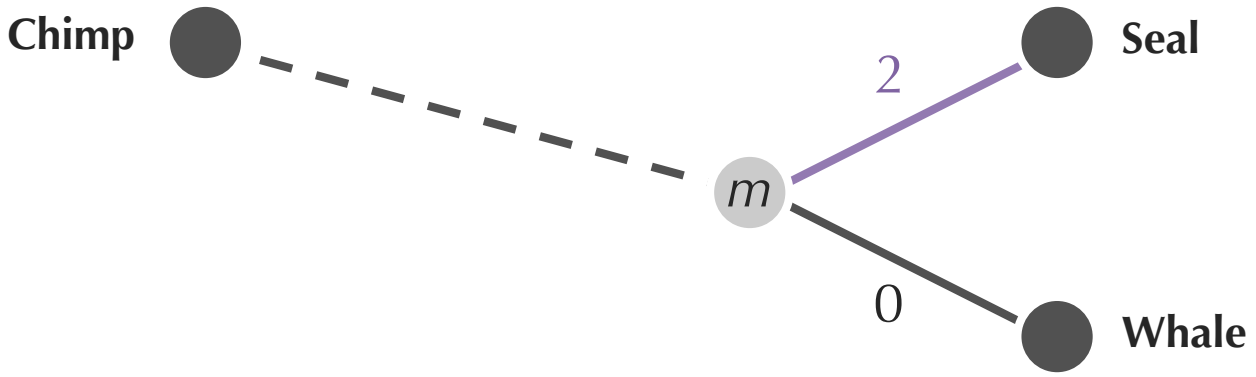
	Chimp	Human	Seal	Whale
Chimp	0	3	6	4
Human	3	0	7	5
Seal	6	7	0	2
Whale	4	5	2	0



$$d_{Seal,m} = (D_{Seal,Chimp} + D_{Seal,Whale} - D_{Whale,Chimp}) / 2$$

An Idea for Distance-Based Phylogeny

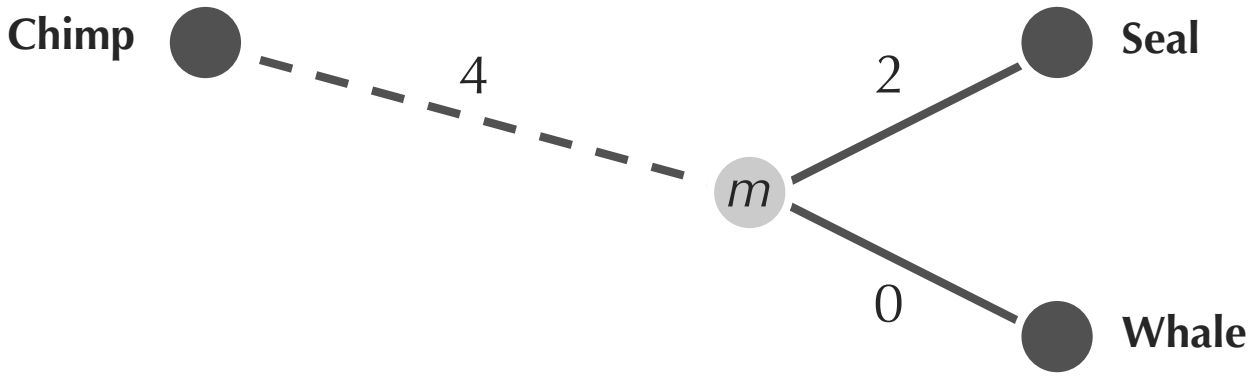
	Chimp	Human	Seal	Whale
Chimp	0	3	6	4
Human	3	0	7	5
Seal	6	7	0	2
Whale	4	5	2	0



$d_{\text{Seal},m} = 2$

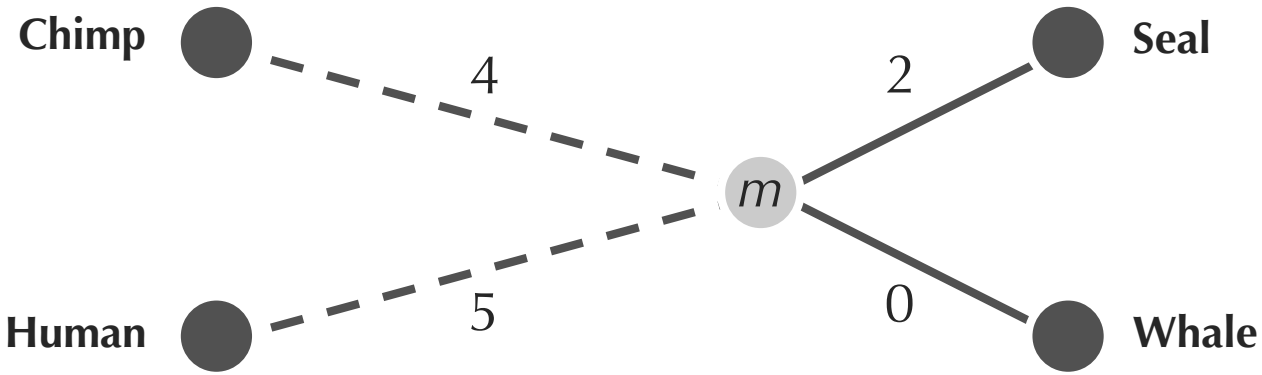
An Idea for Distance-Based Phylogeny

	Chimp	Human	Seal	Whale
Chimp	0	3	6	4
Human	3	0	7	5
Seal	6	7	0	2
Whale	4	5	2	0



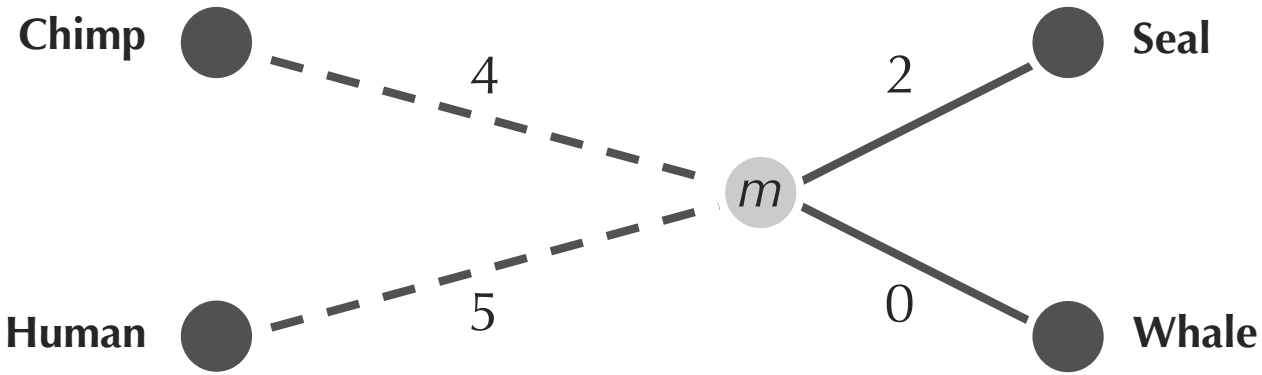
An Idea for Distance-Based Phylogeny

	Chimp	Human	Seal	Whale	*m*
Chimp	0	3	6	4	4
Human	3	0	7	5	5
Seal	6	7	0	2	2
Whale	4	5	2	0	0
m	4	5	2	0	0



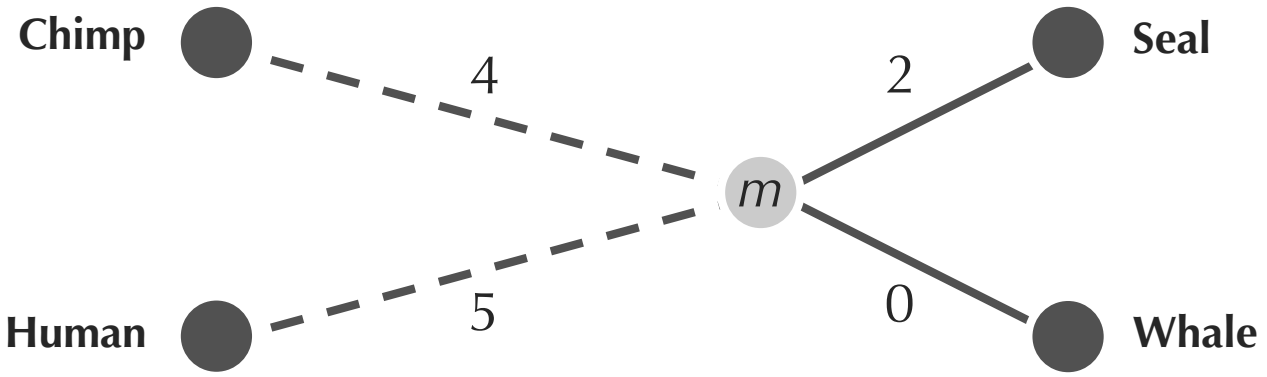
An Idea for Distance-Based Phylogeny

	Chimp	Human	Seal	Whale	*m*
Chimp	0	3	6	4	4
Human	3	0	7	5	5
Seal	6	7	0	2	2
Whale	4	5	2	0	0
m	4	5	2	0	0



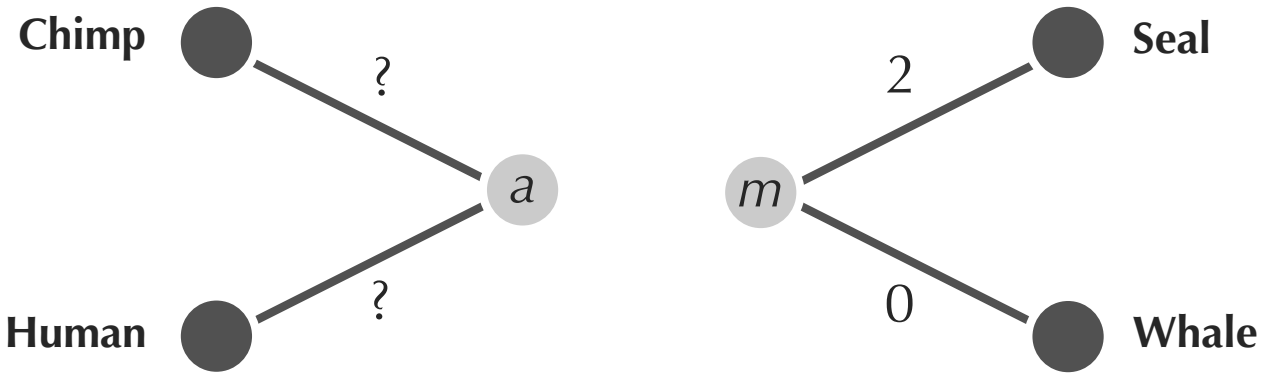
An Idea for Distance-Based Phylogeny

	Chimp	Human	*m*
Chimp	0	3	4
Human	3	0	5
m	4	5	0



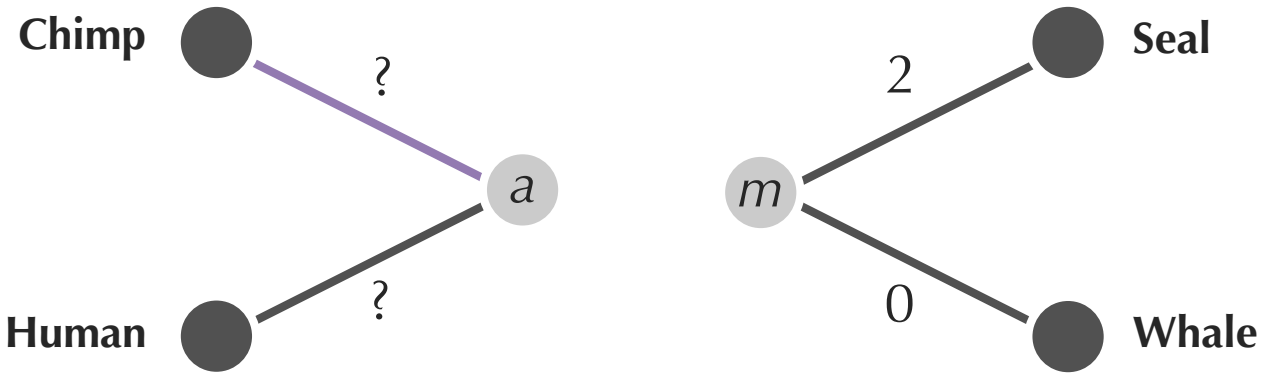
An Idea for Distance-Based Phylogeny

	Chimp	Human	*m*
Chimp	0	**3**	4
Human	**3**	0	5
m	4	5	0



An Idea for Distance-Based Phylogeny

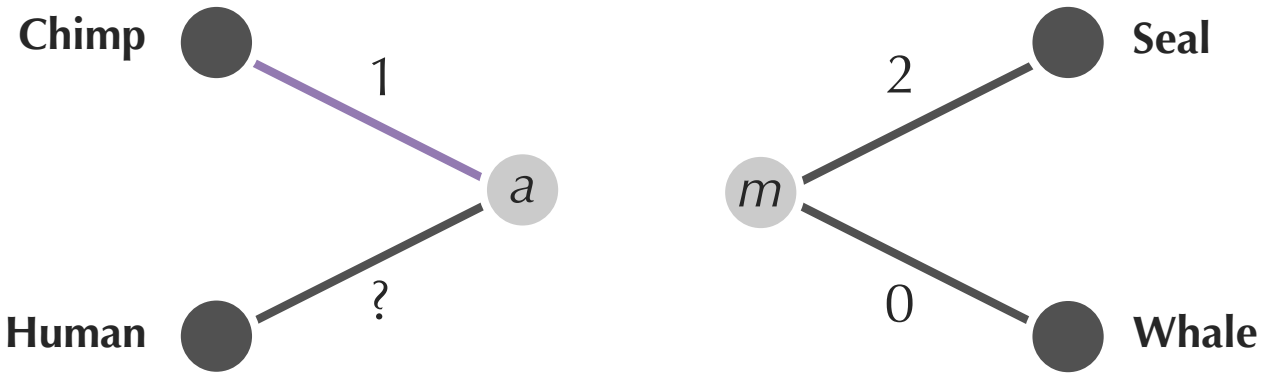
	Chimp	Human	*m*
Chimp	0	3	4
Human	3	0	5
m	4	5	0



$$d_{\text{Chimp},a} = (D_{\text{Chimp},m} + D_{\text{Chimp,Human}} - D_{\text{Human},m}) / 2$$

An Idea for Distance-Based Phylogeny

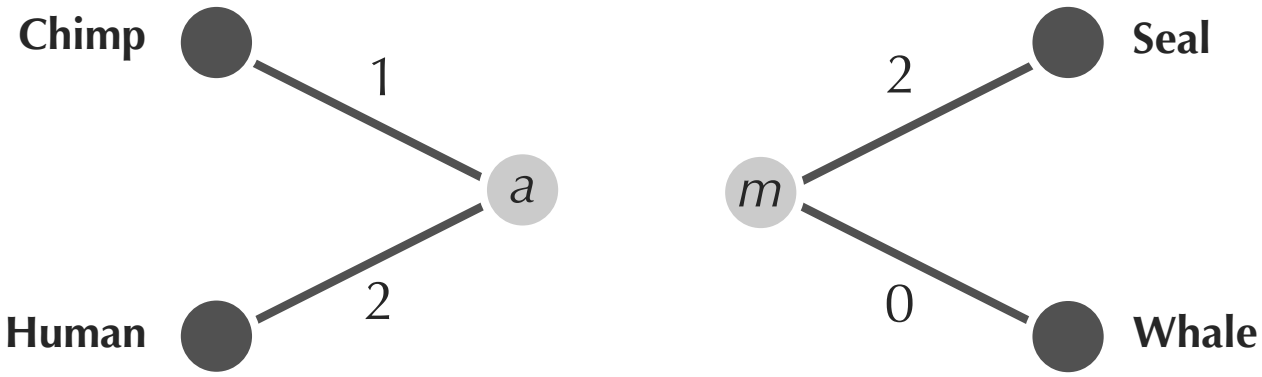
	Chimp	Human	*m*
Chimp	0	3	4
Human	3	0	5
m	4	5	0



$d_{\text{Chimp},a} = 1$

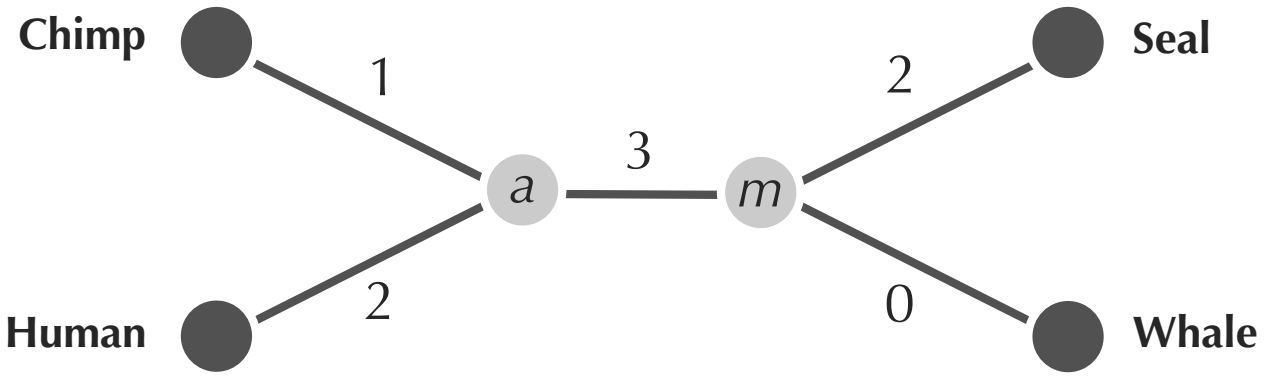
An Idea for Distance-Based Phylogeny

	Chimp	Human	*m*
Chimp	0	3	4
Human	3	0	5
m	4	5	0



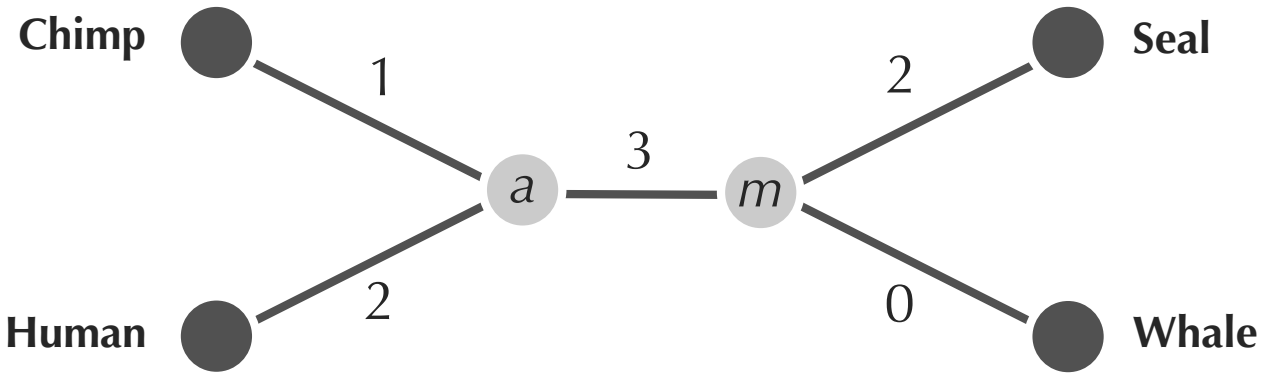
An Idea for Distance-Based Phylogeny

	Chimp	Human	*m*
Chimp	0	3	4
Human	3	0	5
m	4	5	0



An Idea for Distance-Based Phylogeny

	Chimp	Human	Seal	Whale
Chimp	0	3	6	4
Human	3	0	7	5
Seal	6	7	0	2
Whale	4	5	2	0



An Idea for Distance-Based Phylogeny

Exercise Break: Apply this recursive approach to the distance matrix below.

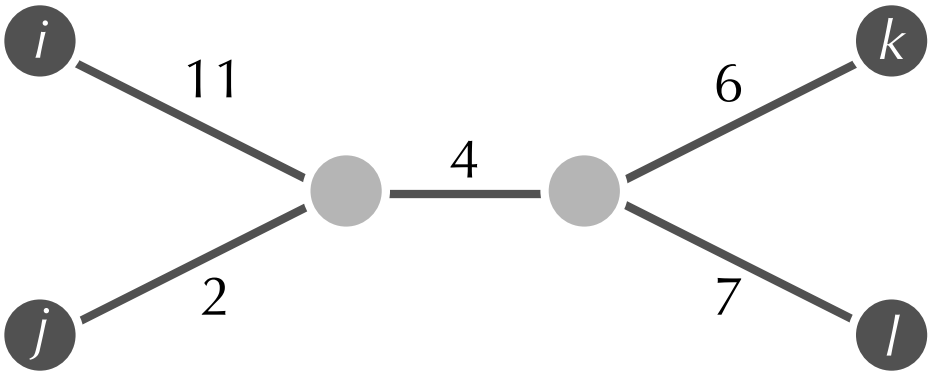
	i	*j*	*k*	*l*
i	0	13	21	22
j	13	0	12	13
k	21	12	0	13
l	22	13	13	0

What Was Wrong With Our Algorithm?

	i	*j*	*k*	*l*
i	0	13	21	22
j	13	0	12	13
k	21	12	0	13
l	22	13	13	0

What Was Wrong With Our Algorithm?

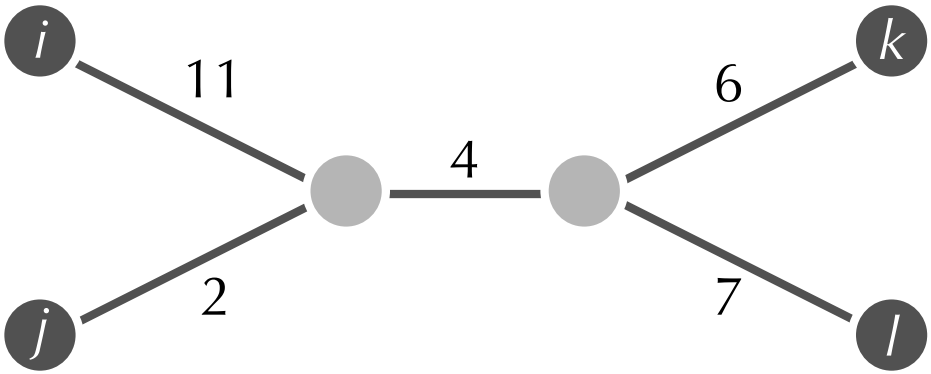
	i	*j*	*k*	*l*
i	0	13	21	22
j	13	0	12	13
k	21	12	0	13
l	22	13	13	0



What Was Wrong With Our Algorithm?

	i	*j*	*k*	*l*
i	0	13	21	22
j	13	0	**12**	13
k	21	12	0	13
l	22	13	13	0

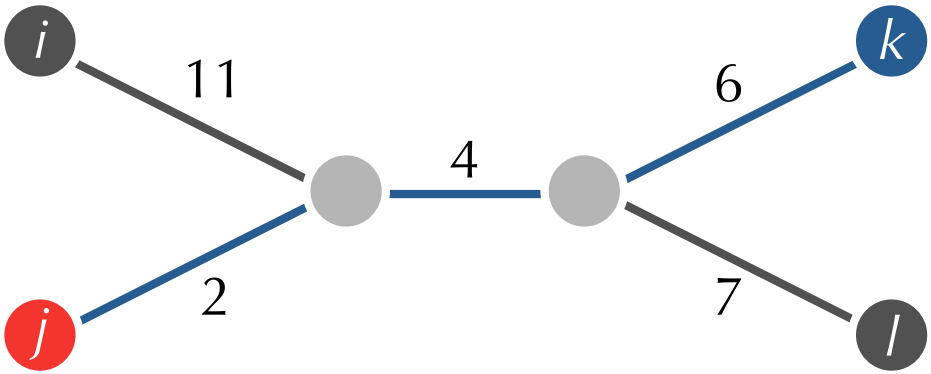
minimum element is $D_{j,k}$



What Was Wrong With Our Algorithm?

	i	*j*	*k*	*l*
i	0	13	21	22
j	13	0	**12**	13
k	21	12	0	13
l	22	13	13	0

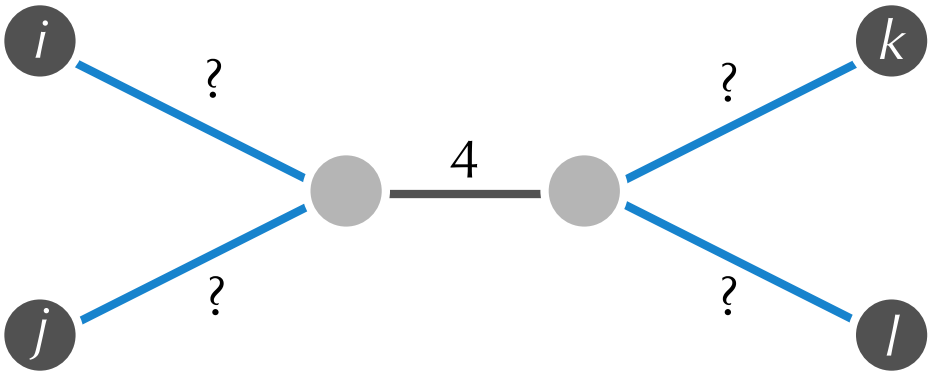
minimum element is $D_{j,k}$



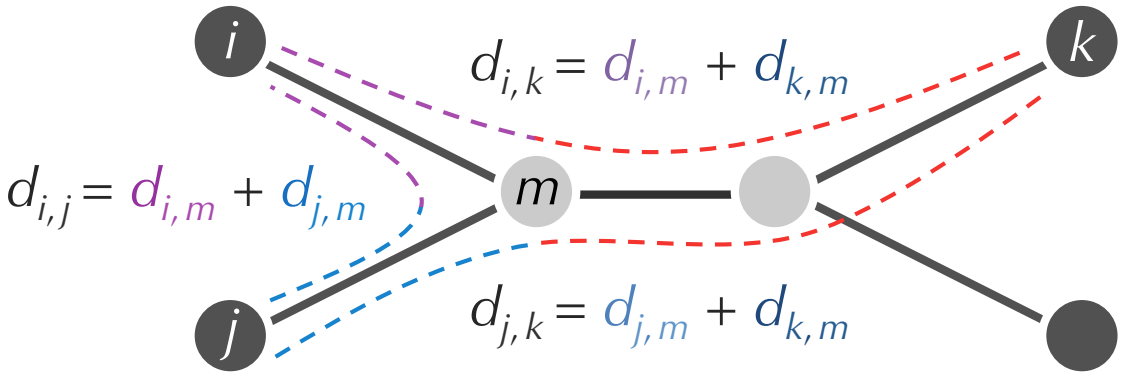
j and ***k*** are **not** neighbors!

From Neighbors to Limbs

Rather than trying to find **neighbors**, let's instead try to compute the length of **limbs**, the edges attached to leaves.



From Neighbors to Limbs



$$d_{k,m} = [(d_{i,m} + d_{k,m}) + (d_{j,m} + d_{k,m}) - (d_{i,m} + d_{j,m})] / 2$$

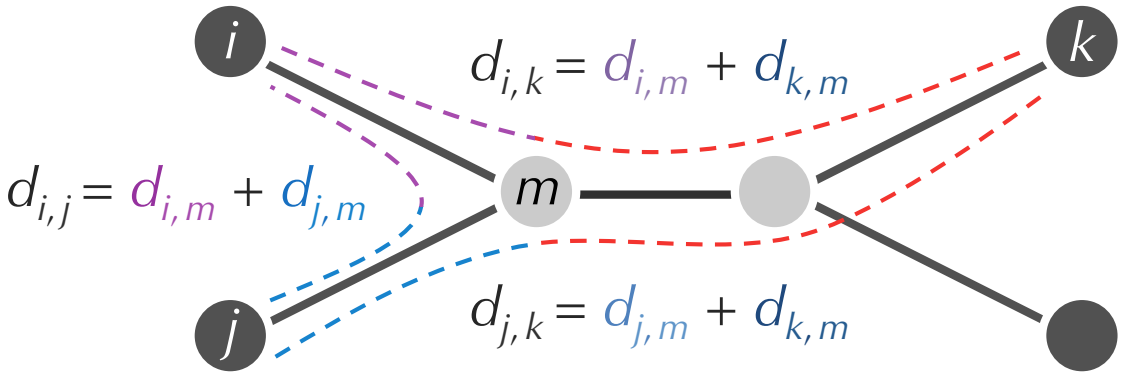
$$d_{k,m} = (d_{i,k} + d_{j,k} - d_{i,j}) / 2$$

$$d_{k,m} = (D_{i,k} + D_{j,k} - D_{i,j}) / 2$$

$$\therefore \; d_{i,m} = D_{i,k} - (D_{i,k} + D_{j,k} - D_{i,j}) / 2$$

$$d_{i,m} = (D_{i,k} + D_{i,j} - D_{j,k}) / 2$$

From Neighbors to Limbs



$$d_{k,m} = [(d_{i,m} + d_{k,m}) + (d_{j,m} + d_{k,m}) - (d_{i,m} + d_{j,m})] / 2$$

$$d_{k,m} = (d_{i,k} + d_{j,k} - d_{i,j}) / 2$$

$$d_{k,m} = (D_{i,k} + D_{j,k} - D_{i,j}) / 2$$

$$\therefore\ d_{i,m} = D_{i,k} - (D_{i,k} + D_{j,k} - D_{i,j}) / 2$$

$$d_{i,m} = (D_{i,k} + D_{i,j} - D_{j,k}) / 2$$

Assumes that *i* and *j* are *neighbors*...

Computing Limb Lengths

Limb Length Theorem: *LimbLength*(*i*) is equal to the minimum value of $(D_{i,k} + D_{i,j} - D_{j,k})/2$ over all leaves *j* and *k*.

Limb Length Problem: *Compute the length of a limb in the simple tree fitting an additive distance matrix.*

- **Input:** An additive distance matrix *D* and an integer *j*.
- **Output:** The length of the limb connecting leaf *j* to its parent, *LimbLength*(*j*).

Code Challenge: Solve the Limb Length Problem.

Computing Limb Lengths

Limb Length Theorem: *LimbLength*(chimp) is equal to the minimum value of $(D_{\text{chimp},k} + D_{\text{chimp},j} - D_{\text{j},k})/2$ over all leaves j and k.

	Chimp	Human	Seal	Whale
Chimp	0	3	6	4
Human	3	0	7	5
Seal	6	7	0	2
Whale	4	5	2	0

$$(D_{\text{chimp, human}} + D_{\text{chimp, seal}} - D_{\text{human, seal}}) / 2 \qquad = (3 + 6 - 7) / 2 = 1$$

Computing Limb Lengths

Limb Length Theorem: *LimbLength*(chimp) is equal to the minimum value of $(D_{\text{chimp},k} + D_{\text{chimp},j} - D_{\text{j},k})/2$ over all leaves j and k.

	Chimp	Human	Seal	Whale
Chimp	0	3	6	4
Human	3	0	7	5
Seal	6	7	0	2
Whale	4	5	2	0

$$(D_{\text{chimp, human}} + D_{\text{chimp, seal}} - D_{\text{human, seal}}) / 2 = (3 + 6 - 7) / 2 = 1$$

$$(D_{\text{chimp, human}} + D_{\text{chimp, whale}} - D_{\text{human, whale}}) / 2 = (3 + 4 - 5) / 2 = 1$$

Computing Limb Lengths

Limb Length Theorem: *LimbLength*(chimp) is equal to the minimum value of $(D_{\text{chimp},k} + D_{\text{chimp},j} - D_{\text{j},k})/2$ over all leaves j and k.

	Chimp	Human	Seal	Whale
Chimp	0	3	6	4
Human	3	0	7	5
Seal	6	7	0	2
Whale	4	5	2	0

$$(D_{\text{chimp, human}} + D_{\text{chimp, seal}} - D_{\text{human, seal}}) / 2 = (3 + 6 - 7) / 2 = 1$$

$$(D_{\text{chimp, human}} + D_{\text{chimp, whale}} - D_{\text{human, whale}}) / 2 = (3 + 4 - 5) / 2 = 1$$

$$(D_{\text{chimp, whale}} + D_{\text{chimp, seal}} - D_{\text{whale, seal}}) / 2 = (6 + 4 - 2) / 2 = 4$$

Computing Limb Lengths

Limb Length Theorem: *LimbLength*(chimp) is equal to the minimum value of $(D_{\text{chimp},k} + D_{\text{chimp},j} - D_{\text{j},k})/2$ over all leaves j and k.

	Chimp	Human	Seal	Whale
Chimp	0	3	6	4
Human	3	0	7	5
Seal	6	7	0	2
Whale	4	5	2	0

$(D_{\text{human, chimp}} + D_{\text{chimp, seal}} - D_{\text{human, seal}}) / 2 = (3 + 6 - 7) / 2 = 1$

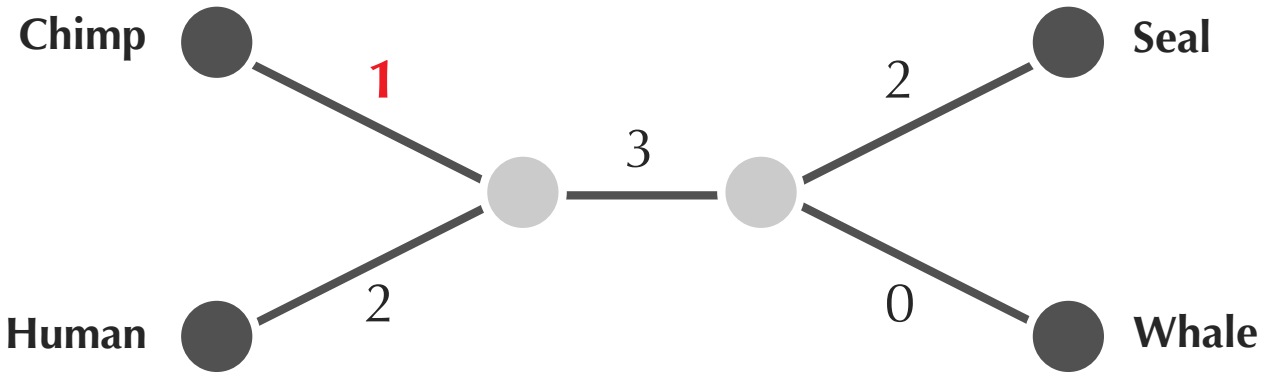
$(D_{\text{human, chimp}} + D_{\text{chimp, whale}} - D_{\text{human, whale}}) / 2 = (3 + 4 - 5) / 2 = 1$

$(D_{\text{whale, chimp}} + D_{\text{chimp, seal}} - D_{\text{whale, seal}}) / 2 = (6 + 4 - 2) / 2 = 4$

Computing Limb Lengths

Limb Length Theorem: *LimbLength*(chimp) is equal to the minimum value of $(D_{\text{chimp},k} + D_{\text{chimp},j} - D_{\text{j},k})/2$ over all leaves j and k.

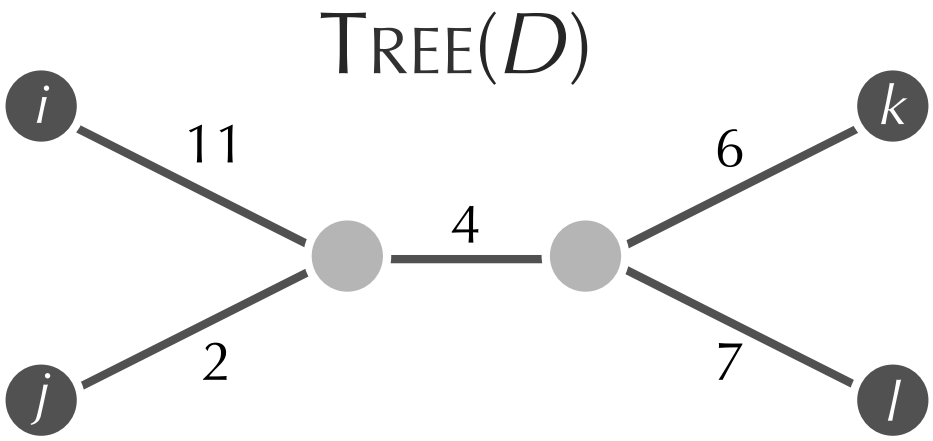
	Chimp	Human	Seal	Whale
Chimp	0	3	6	4
Human	3	0	7	5
Seal	6	7	0	2
Whale	4	5	2	0



AdditivePhylogeny In Action

D

	i	*j*	*k*	*l*
i	0	13	21	22
j	13	0	12	13
k	21	12	0	13
l	22	13	13	0



AdditivePhylogeny In Action

D

	i	*j*	*k*	*l*
i	0	13	21	22
j	13	0	12	13
k	21	12	0	13
l	22	13	13	0

1. Pick an arbitrary leaf *j*.

AdditivePhylogeny In Action

D

	i	*j*	*k*	*l*
i	0	13	21	22
j	13	0	12	13
k	21	12	0	13
l	22	13	13	0

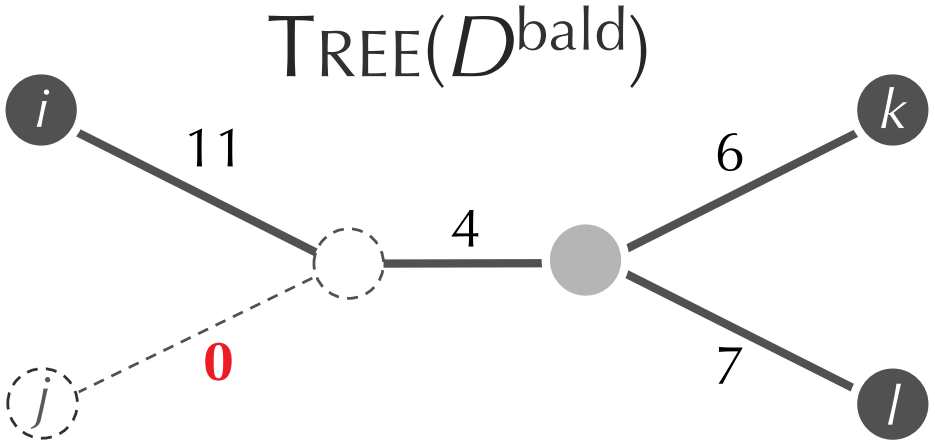
$LimbLength(j) = 2$

2. Compute its limb length, *LimbLength*(*j*).

AdditivePhylogeny In Action

D^{bald}

	i	*j*	*k*	*l*
i	0	11	21	22
j	11	0	10	11
k	21	10	0	13
l	22	11	13	0



3. Subtract *LimbLength*(*j*) from each row and column to produce D^{bald} in which *j* is a **bald limb** (length 0).

AdditivePhylogeny In Action

D^{trim}

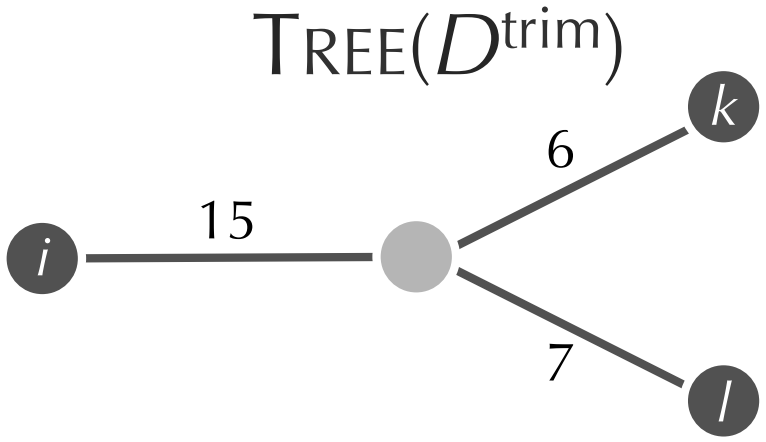
	i	*j*	***k***	***l***
i	0	11	21	22
j	11	0	10	11
k	21	10	0	13
l	22	11	13	0

4. Remove the *j*-th row and column of the matrix to form the $(n - 1) \times (n - 1)$ matrix D^{trim}.

AdditivePhylogeny In Action

D^{trim}

	i	*j*	*k*	*l*
i	0	11	21	22
j	11	0	10	11
k	21	10	0	13
l	22	11	13	0

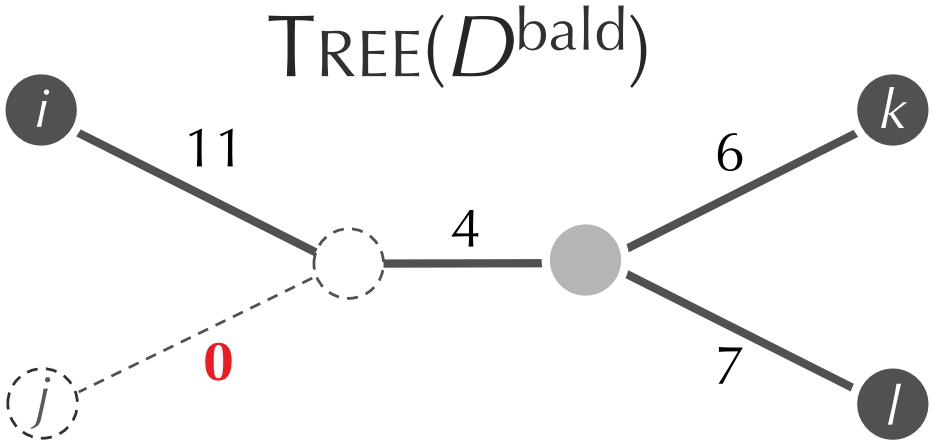


5. Construct *Tree*(D^{trim}).

AdditivePhylogeny In Action

D^{bald}

	i	*j*	*k*	*l*
i	0	11	21	22
j	11	0	10	11
k	21	10	0	13
l	22	11	13	0

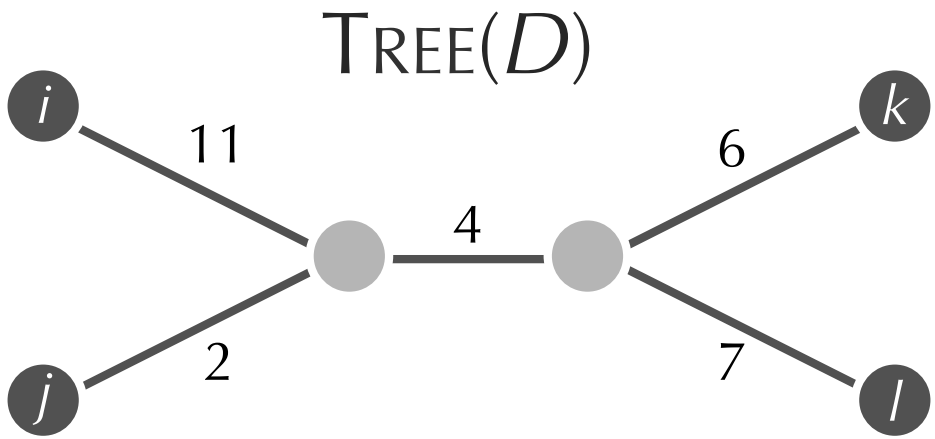


6. Identify the point in *Tree*(D^{trim}) where leaf *j* should be attached.

AdditivePhylogeny In Action

D

	i	*j*	*k*	*l*
i	0	13	21	22
j	13	0	12	13
k	21	12	0	13
l	22	13	13	0



LimbLength(*j*) = 2

7. Attach *j* by an edge of length *LimbLength*(*j*) in order to form *Tree*(*D*).

AdditivePhylogeny

AdditivePhylogeny(*D*):

1. Pick an arbitrary leaf *j*.
2. Compute its limb length, *LimbLength*(*j*).
3. Subtract *LimbLength*(*j*) from each row and column to produce D^{bald} in which *j* is a bald limb (length 0).
4. Remove the *j*-th row and column of the matrix to form the (*n* – 1) x (*n* – 1) matrix D^{trim}.
5. Construct *Tree*(D^{trim}).
6. Identify the point in *Tree*(D^{trim}) where leaf *j* should be attached.
7. Attach *j* by an edge of length *LimbLength*(*j*) in order to form *Tree*(*D*).

AdditivePhylogeny

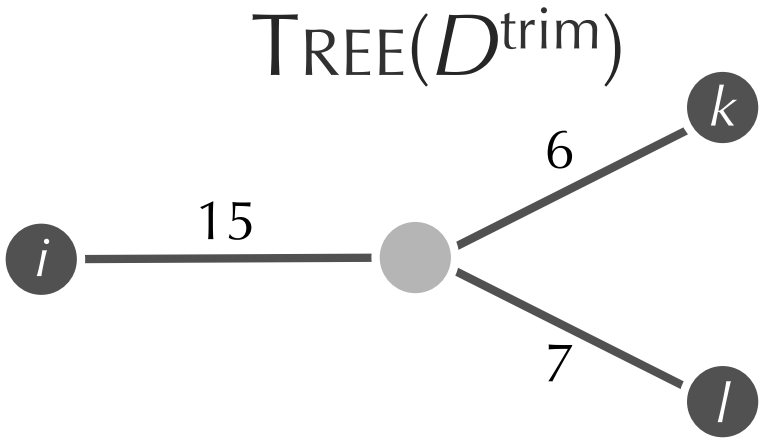
AdditivePhylogeny(*D*):

1. Pick an arbitrary leaf *j*.
2. Compute its limb length, *LimbLength*(*j*).
3. Subtract *LimbLength*(*j*) from each row and column to produce D^{bald} in which *j* is a bald limb (length 0).
4. Remove the *j*-th row and column of the matrix to form the $(n - 1) \times (n - 1)$ matrix D^{trim}.
5. Construct *Tree*(D^{trim}).
6. **Identify the point in *Tree*(D^{trim}) where leaf *j* should be attached.**
7. Attach *j* by an edge of length *LimbLength*(*j*) in order to form *Tree*(*D*).

Attaching a Limb

D^{bald}

	i	*j*	*k*	*l*
i	0	11	21	22
j	11	0	10	11
k	21	10	0	13
l	22	11	13	0

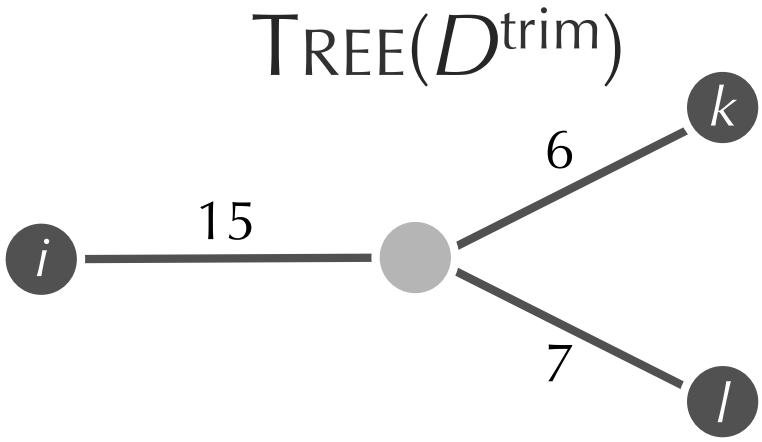


Limb Length Theorem: the length of the limb of j is equal to the minimum value of $(D^{bald}_{i,j} + D^{bald}_{j,k} - D^{bald}_{i,k})/2$ over all leaves i and k.

Attaching a Limb

D^{bald}

	i	*j*	*k*	*l*
i	0	**11**	**21**	22
j	11	0	**10**	11
k	21	10	0	13
l	22	11	13	0



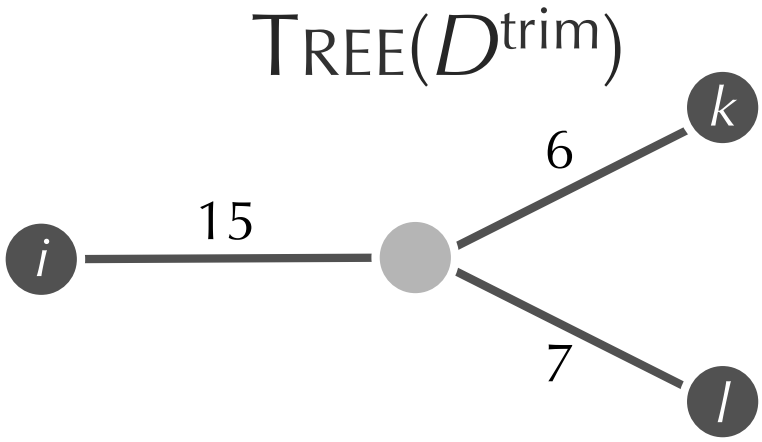
Limb Length Theorem: the length of the limb of j is equal to the minimum value of $(D^{bald}_{i,j} + D^{bald}_{j,k} - D^{bald}_{i,k})/2$ over all leaves i and k.

$$(D^{bald}_{i,j} + D^{bald}_{j,k} - D^{bald}_{i,k})/2 = \mathbf{0}$$

Attaching a Limb

D^{bald}

	i	*j*	*k*	*l*
i	0	**11**	**21**	22
j	11	0	**10**	11
k	21	10	0	13
l	22	11	13	0



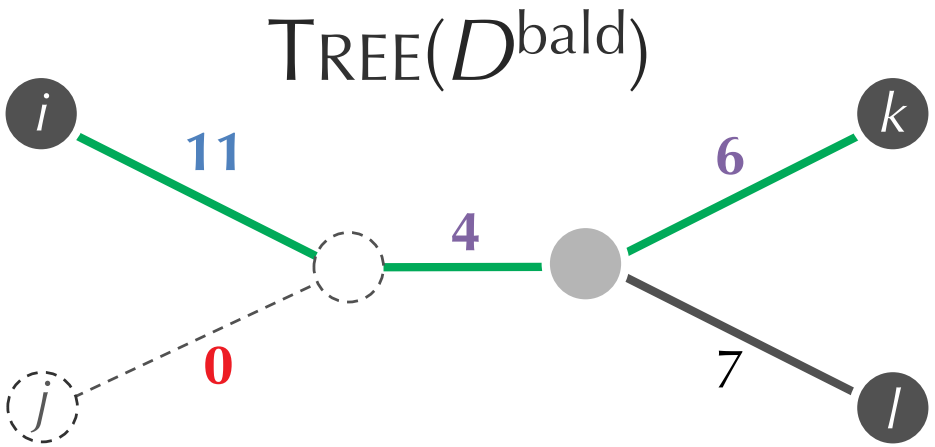
$$(D^{bald}_{i,j} + D^{bald}_{j,k} - D^{bald}_{i,k})/2 = \mathbf{0}$$

$$D^{bald}_{i,j} + D^{bald}_{j,k} = D^{bald}_{i,k}$$

Attaching a Limb

D^{bald}

	i	*j*	*k*	*l*
i	0	**11**	**21**	22
j	11	0	**10**	11
k	21	10	0	13
l	22	11	13	0



The attachment point for *j* is found on the path between leaves *i* and *k* at distance $D^{bald}_{i,j}$ from *i*.

$$D^{bald}_{i,j} + D^{bald}_{j,k} = D^{bald}_{i,k}$$

AdditivePhylogeny

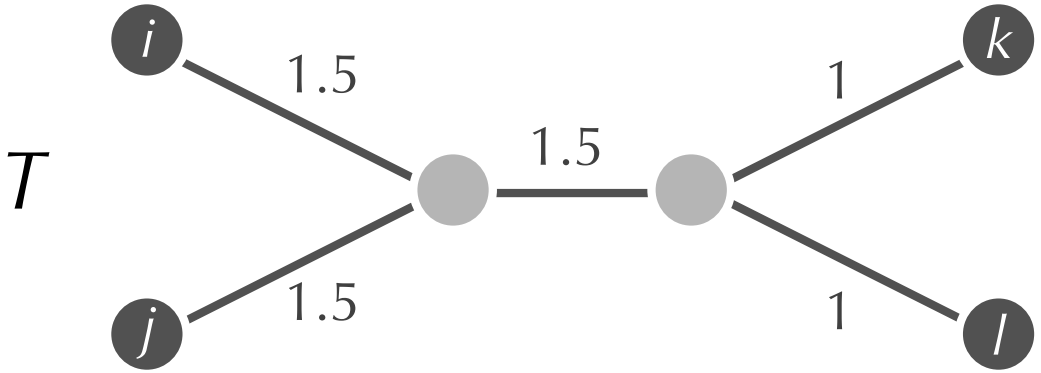
AdditivePhylogeny(*D*):

1. Pick an arbitrary leaf *j*.
2. Compute its limb length, *LimbLength*(*j*).
3. Subtract *LimbLength*(*j*) from each row and column to produce D^{bald} in which *j* is a bald limb (length 0).
4. Remove the *j*-th row and column of the matrix to form the $(n - 1) \times (n - 1)$ matrix D^{trim}.
5. Construct *Tree*(D^{trim}).
6. Identify the point in *Tree*(D^{trim}) where leaf *j* should be attached.
7. Attach *j* by an edge of length *LimbLength*(*j*) in order to form *Tree*(*D*).

Code Challenge: Implement **AdditivePhylogeny**.

Sum of Squared Errors

$$Discrepancy(T, D) = \Sigma_{1 \le i < j \le n} (d_{i,j}(T) - D_{i,j})^2$$
$$= 1^2 + 1^2 = 2$$



D

	i	*j*	*k*	*l*
i	0	3	4	**3**
j	3	0	4	**5**
k	4	4	0	2
l	3	5	2	0

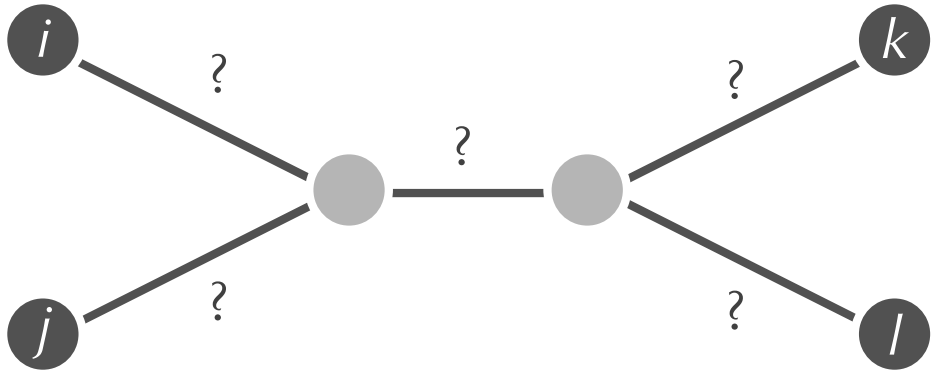
d

	i	*j*	*k*	*l*
i	0	3	4	**4**
j	3	0	4	**4**
k	4	4	0	2
l	4	4	2	0

Sum of Squared Errors

Exercise Break: Assign lengths to edges in T in order to minimize *Discrepancy*(T, D).

T



D

	i	*j*	*k*	*l*
i	0	3	4	3
j	3	0	4	5
k	4	4	0	2
l	3	5	2	0

d

	i	*j*	*k*	*l*
i	0	?	?	?
j	?	0	?	?
k	?	?	0	?
l	?	?	?	0

Least-Squares Phylogeny

Least-Squares Distance-Based Phylogeny Problem:
Given a distance matrix, find the tree that minimizes the sum of squared errors.

- **Input:** An n x n distance matrix D.
- **Output:** A weighted tree T with n leaves minimizing *Discrepancy*(T, D) over all weighted trees with n leaves.

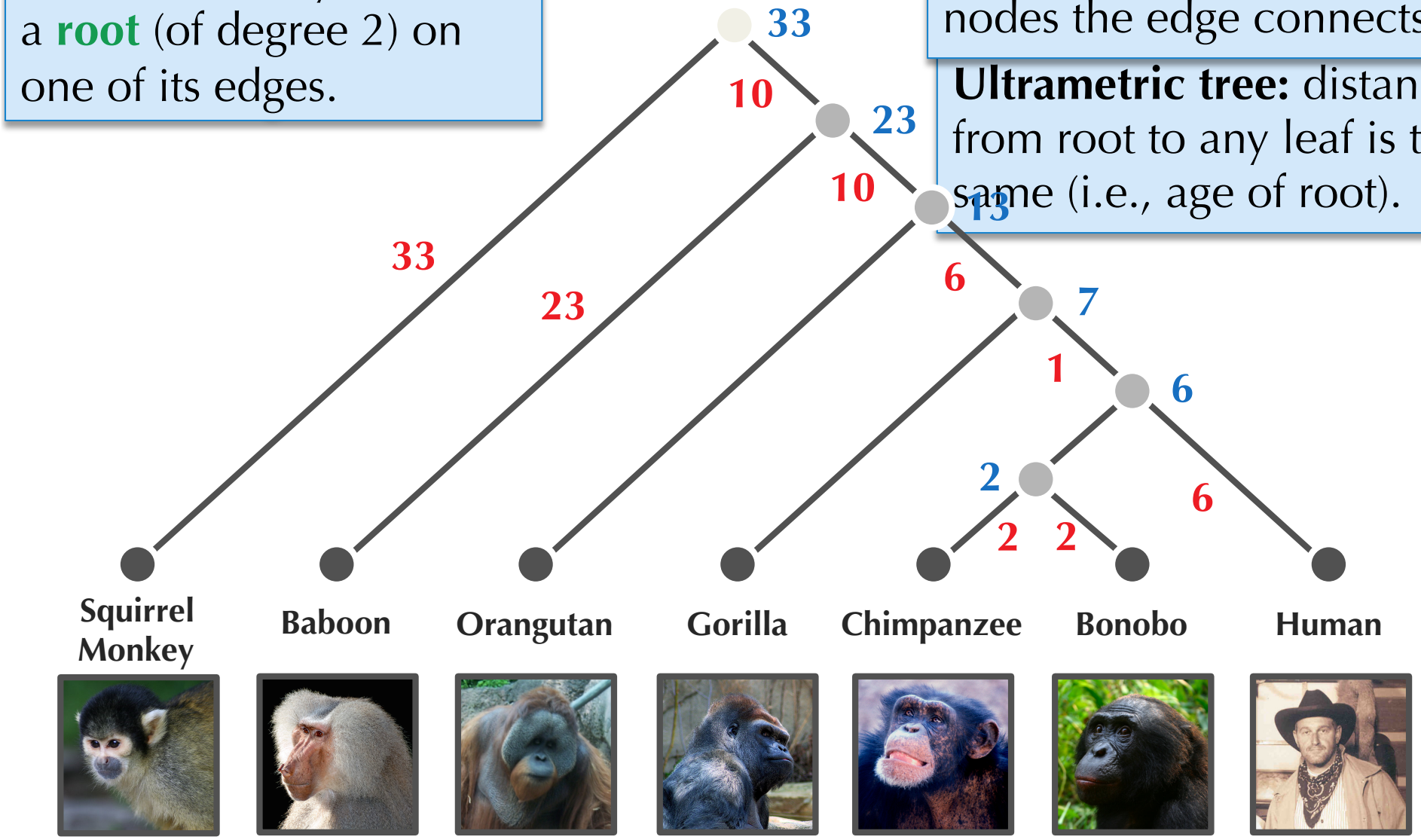
Unfortunately, this problem is *NP*-Complete...

Ultrametric Trees

Rooted binary tree: an unrooted binary tree with a **root** (of degree 2) on one of its edges.

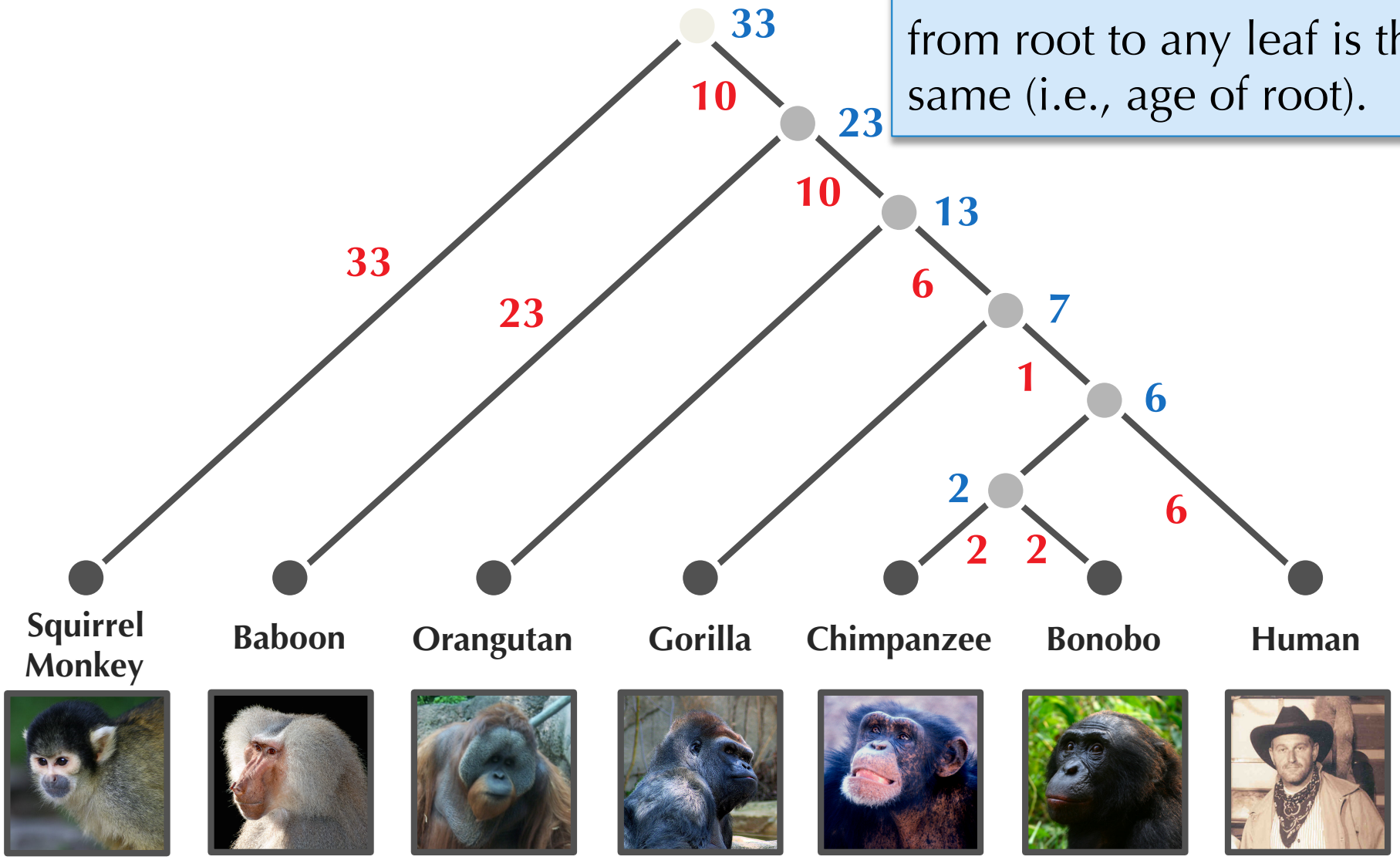
edge weights: correspond to difference in ages on the nodes the edge connects.

Ultrametric tree: distance from root to any leaf is the same (i.e., age of root).



Ultrametric Trees

Ultrametric tree: distance from root to any leaf is the same (i.e., age of root).



UPGMA: A Clustering Heuristic

1. Form a cluster for each present-day species, each containing a single leaf.

	i	***j***	***k***	***l***
i	0	3	4	3
j	3	0	4	5
k	4	4	0	2
l	3	5	2	0

UPGMA: A Clustering Heuristic

2. Find the two closest clusters C_1 and C_2 according to the average distance

$$D_{avg}(C_1, C_2) = \Sigma_{i \text{ in } C1,\, j \text{ in } C2}\, D_{i,j} / |C_1| \bullet |C_2|$$

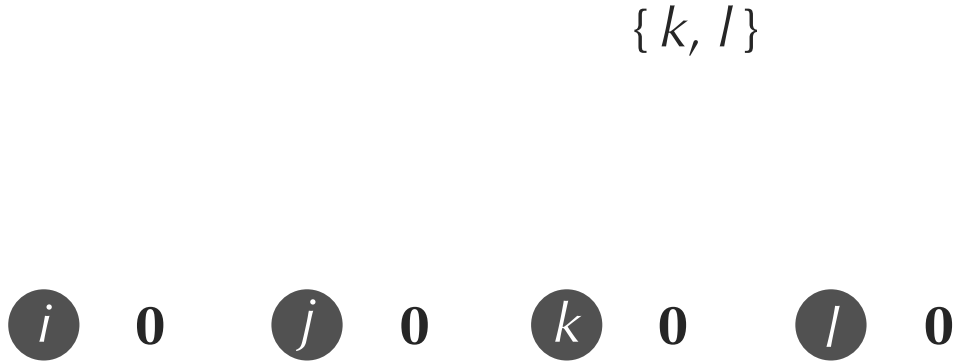
where $|C|$ denotes the number of elements in C.

	i	***j***	***k***	***l***
i	0	3	4	3
j	3	0	4	5
k	4	4	0	**2**
l	3	5	**2**	0

UPGMA: A Clustering Heuristic

3. Merge C_1 and C_2 into a single cluster C.

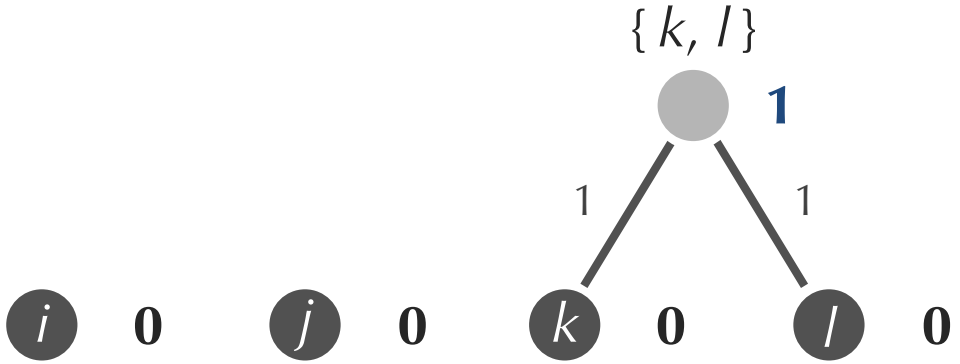
	i	*j*	*k*	*l*
i	0	3	4	3
j	3	0	4	5
k	4	4	0	**2**
l	3	5	**2**	0



UPGMA: A Clustering Heuristic

4. Form a new node for C and connect to C_1 and C_2 by an edge. Set age of C as $D_{avg}(C_1, C_2)/2$.

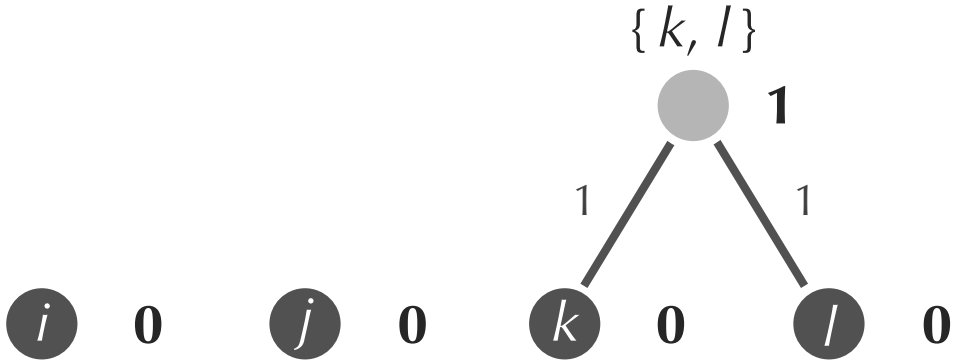
	i	*j*	*k*	*l*
i	0	3	4	3
j	3	0	4	5
k	4	4	0	**2**
l	3	5	**2**	0



UPGMA: A Clustering Heuristic

5. Update the distance matrix by computing the average distance between each pair of clusters.

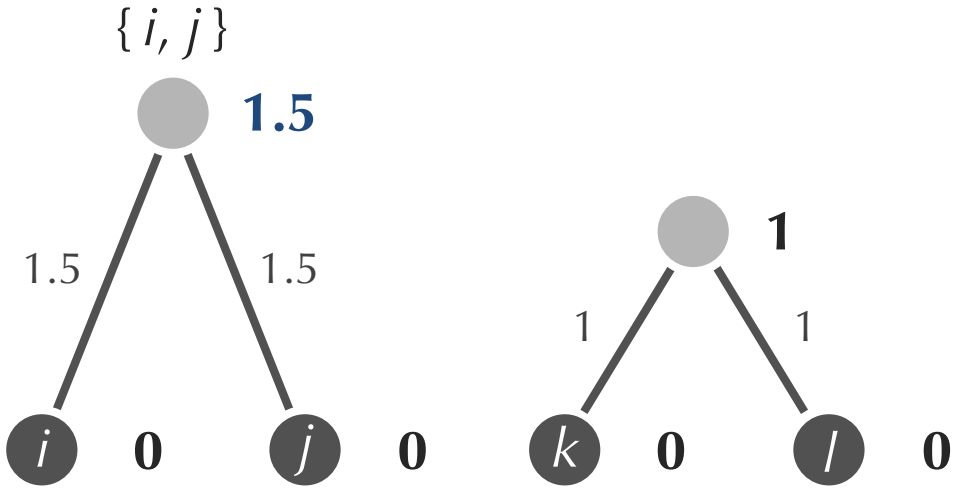
	i	***j***	**{*k*, *l*}**
i	0	3	3.5
j	3	0	4.5
{*k*, *l*}	3.5	4.5	0



UPGMA: A Clustering Heuristic

6. Iterate until a single cluster contains all species.

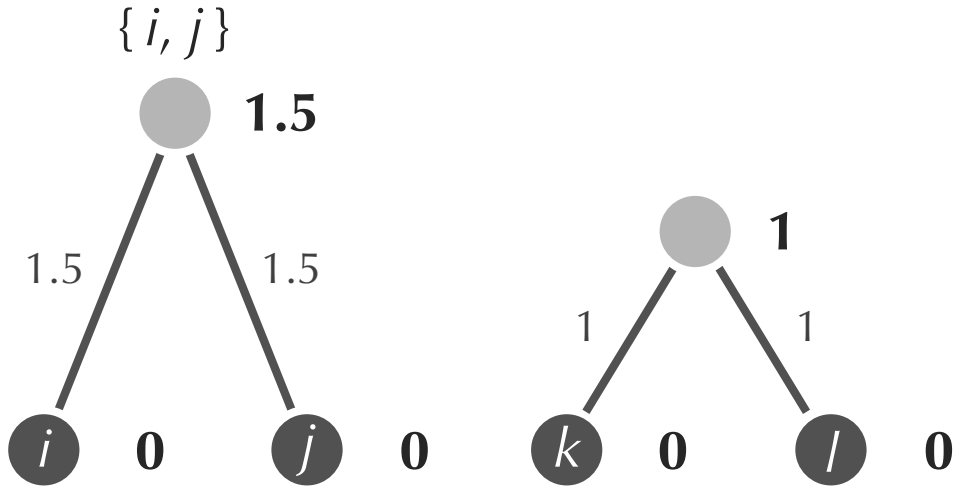
	i	j	$\{k, l\}$
i	0	**3**	3.5
j	**3**	0	4.5
$\{k, l\}$	3.5	4.5	0



UPGMA: A Clustering Heuristic

6. Iterate until a single cluster contains all species.

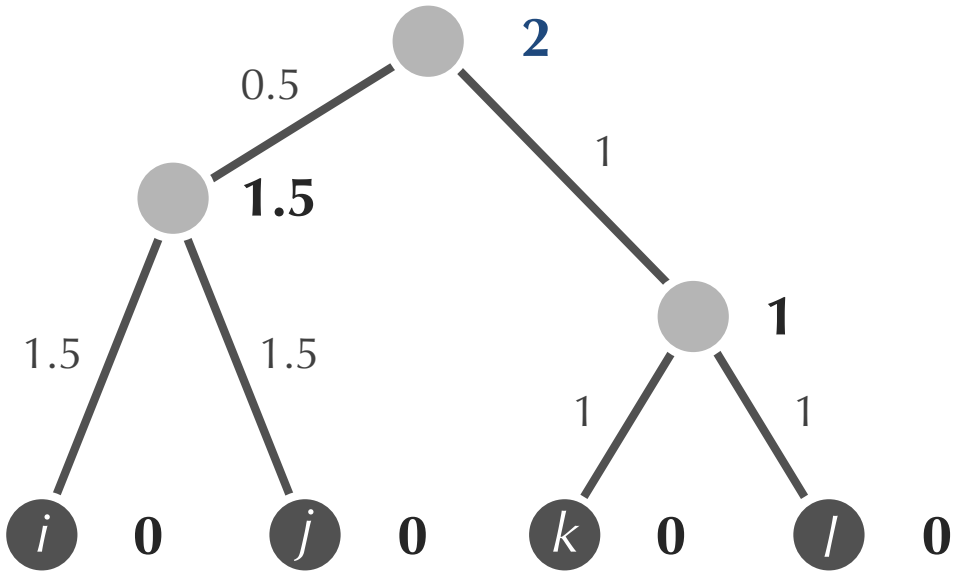
	$\{i, j\}$	$\{k, l\}$
$\{i, j\}$	0	4
$\{k, l\}$	4	0



UPGMA: A Clustering Heuristic

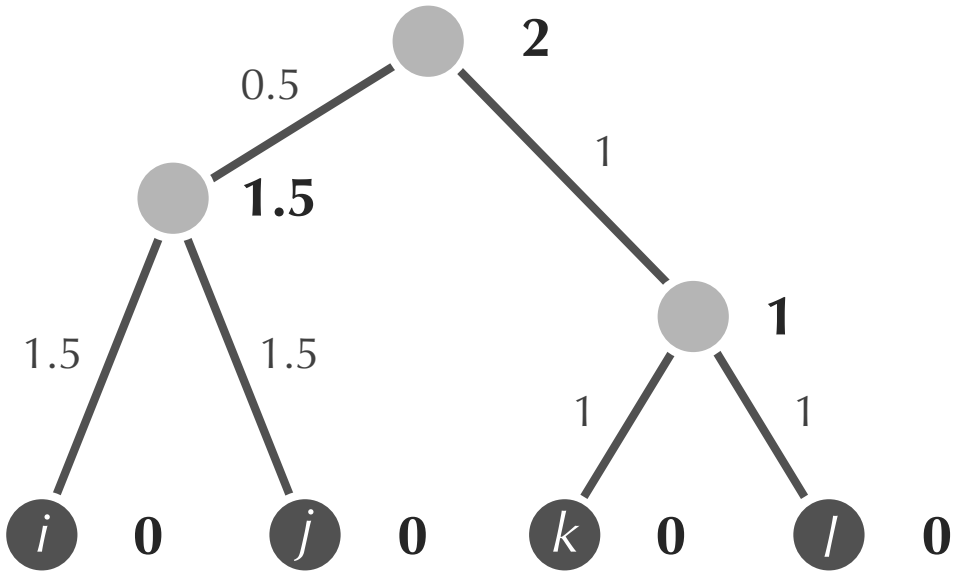
6. Iterate until a single cluster contains all species.

	$\{i, j\}$	$\{k, l\}$
$\{i, j\}$	0	**4**
$\{k, l\}$	**4**	0



UPGMA: A Clustering Heuristic

6. Iterate until a single cluster contains all species.



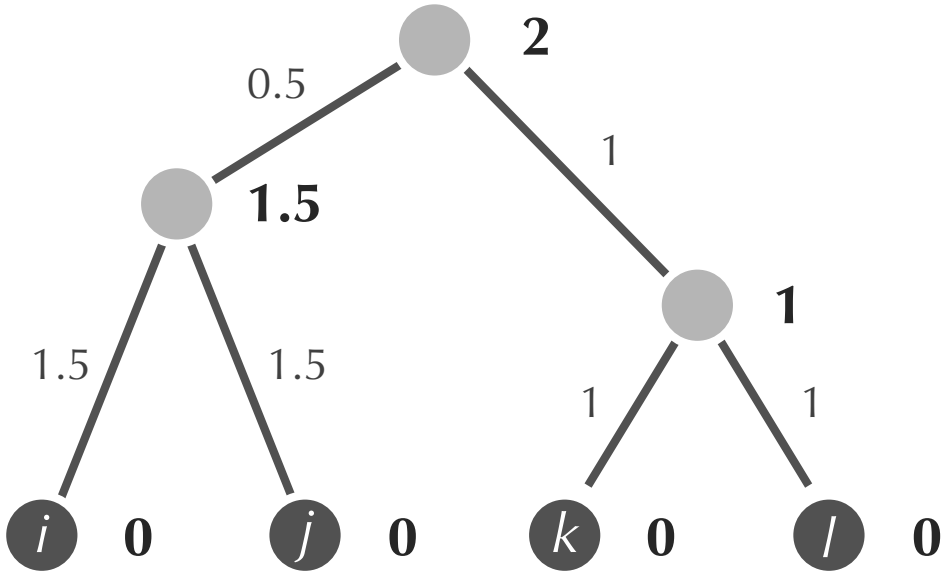
UPGMA: A Clustering Heuristic

UPGMA(*D*):

1. Form a cluster for each present-day species, each containing a single leaf.
2. Find the two closest clusters C_1 and C_2 according to the average distance
   $$D_{avg}(C_1, C_2) = \Sigma_{i \text{ in } C1,\, j \text{ in } C2}\, D_{i,j} / |C_1| \bullet |C_2|$$
   where $|C|$ denotes the number of elements in C
3. Merge C_1 and C_2 into a single cluster C.
4. Form a new node for C and connect to C_1 and C_2 by an edge. Set age of C as $D_{avg}(C_1, C_2)/2$.
5. Update the distance matrix by computing the average distance between each pair of clusters.
6. Iterate steps 2-5 until a single cluster contains all species.

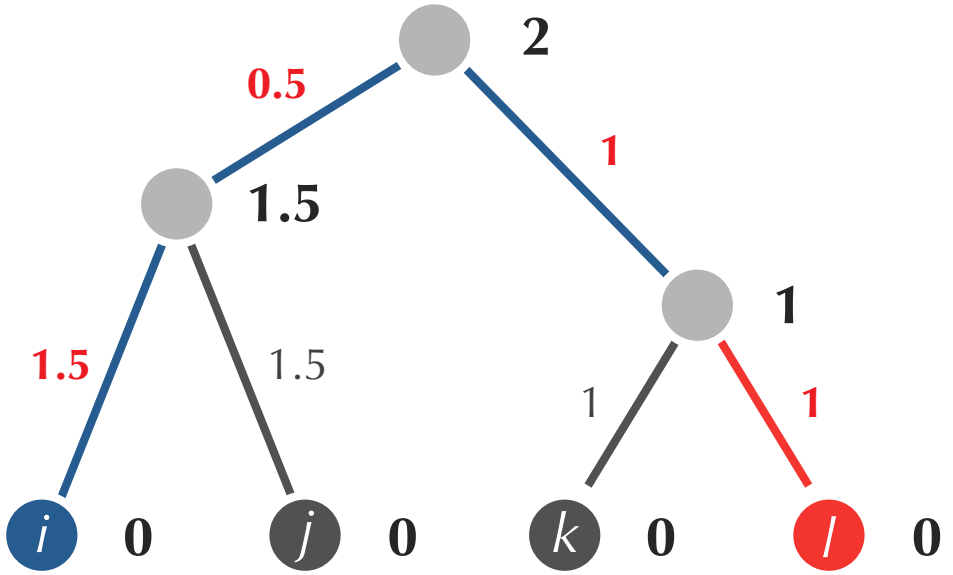
UPGMA Doesn't "Fit" a Tree to a Matrix

	i	*j*	*k*	*l*
i	0	3	4	3
j	3	0	4	5
k	4	4	0	2
l	3	5	2	0



UPGMA Doesn't "Fit" a Tree to a Matrix

	i	*j*	*k*	*l*
i	0	3	4	**3**
j	3	0	4	5
k	4	4	0	2
l	3	5	2	0



In Summary...

- **AdditivePhylogeny:**
  - good: produces the tree fitting an *additive* matrix
  - bad: fails completely on a *non-additive* matrix
- **UPGMA:**
  - good: produces a tree for any matrix
  - bad: tree doesn't necessarily fit an additive matrix
- **?????:**
  - good: produces the tree fitting an additive matrix
  - good: provides heuristic for a non-additive matrix

Neighbor-Joining Theorem

Given an n x n distance matrix D, its **neighbor-joining matrix** is the matrix D^* defined as

$$D^*_{i,j} = (n - 2) \bullet D_{i,j} - TotalDistance_D(i) - TotalDistance_D(j)$$

where $TotalDistance_D(i)$ is the sum of distances from i to all other leaves.

D

	i	*j*	*k*	*l*	$TotalDistance_D$
i	0	13	21	22	56
j	13	0	12	13	38
k	21	12	0	13	46
l	22	13	13	0	48

D^*

	i	*j*	*k*	*l*
i	0	-68	-60	-60
j	-68	0	-60	-60
k	-60	-60	0	-68
l	-60	-60	-68	0

Neighbor-Joining Theorem

Neighbor-Joining Theorem: If D is additive, then the smallest element of D^* corresponds to neighboring leaves in $Tree(D)$.

D

	i	*j*	*k*	*l*	$TotalDistance_D$
i	0	13	21	22	56
j	13	0	**12**	13	38
k	21	**12**	0	13	46
l	22	13	13	0	48

D^*

	i	*j*	*k*	*l*
i	0	**-68**	-60	-60
j	**-68**	0	-60	-60
k	-60	-60	0	**-68**
l	-60	-60	**-68**	0

Neighbor-Joining in Action

D^*

	i	*j*	*k*	*l*	$TotalDistance_D$
i	0	-68	-60	-60	56
j	-68	0	-60	-60	38
k	-60	-60	0	-68	46
l	-60	-60	-68	0	48

1. Construct neighbor-joining matrix D^* from D.

Neighbor-Joining in Action

D^*		*i*	*j*	*k*	*l*	$TotalDistance_D$
	i	0	**-68**	-60	-60	56
	j	**-68**	0	-60	-60	38
	k	-60	-60	0	**-68**	46
	l	-60	-60	**-68**	0	48

2. Find a minimum element $D^*_{i,j}$ of D^*.

Neighbor-Joining in Action

D^*

	i	*j*	*k*	*l*	$TotalDistance_D$
i	0	**-68**	-60	-60	56
j	**-68**	0	-60	-60	38
k	-60	-60	0	-68	46
l	-60	-60	-68	0	48

2. Find a minimum element $D^*_{i,j}$ of D^*.

Neighbor-Joining in Action

D^*

	i	*j*	*k*	*l*	$TotalDistance_D$
i	0	**-68**	-60	-60	**56**
j	**-68**	0	-60	-60	**38**
k	-60	-60	0	-68	46
l	-60	-60	-68	0	48

$\Delta_{i,j} = (\mathbf{56} - \mathbf{38}) / (4 - 2)$
$= 9$

3. Compute $\Delta_{i,j} = (TotalDistance_D(i) - TotalDistance_D(j)) / (n - 2)$.

Neighbor-Joining in Action

D

	i	*j*	*k*	*l*	$TotalDistance_D$
i	0	**13**	21	22	56
j	**13**	0	12	13	38
k	21	12	0	13	46
l	22	13	13	0	48

$\Delta_{i,j} = (56 - 38) / (4 - 2)$
$= 9$

$LimbLength(i) = \frac{1}{2}(13 + 9) = 11$
$LimbLength(i) = \frac{1}{2}(13 - 9) = 2$

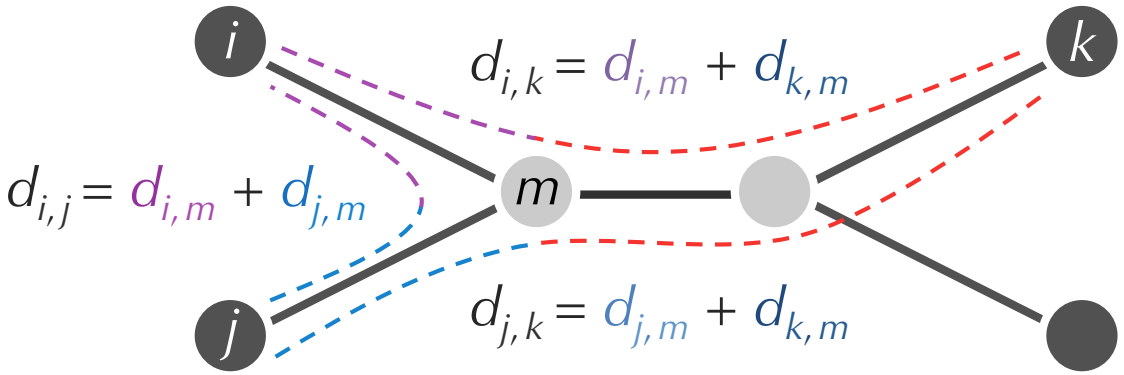
4. Set $LimbLength(i)$ equal to $\frac{1}{2}(D_{i,j} + \Delta_{i,j})$ and $LimbLength(j)$ equal to $\frac{1}{2}(D_{i,j} - \Delta_{j,i})$.

Neighbor-Joining in Action

		m	*k*	*l*	$TotalDistance_D$
	m	0	10	11	21
D'	***k***	10	0	13	23
	l	11	13	0	24

5. Form a matrix D' by removing i-th and j-th row/column from D and adding an m-th row/column such that for any k, $D_{k,m} = (D_{i,k} + D_{j,k} - D_{i,j}) / 2$.

Flashback: Computation of $d_{k,m}$



$$d_{k,m} = [(d_{i,m} + d_{k,m}) + (d_{j,m} + d_{k,m}) - (d_{i,m} + d_{j,m})] / 2$$

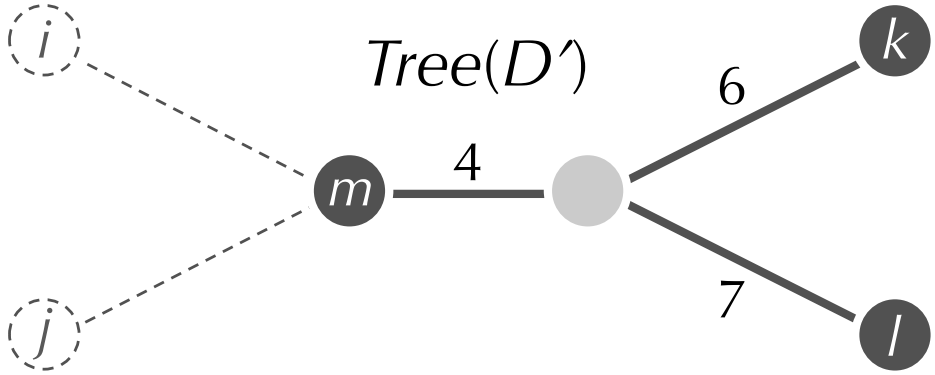
$$d_{k,m} = (d_{i,k} + d_{j,k} - d_{i,j}) / 2$$

$$d_{k,m} = (D_{i,k} + D_{j,k} - D_{i,j}) / 2$$

Neighbor-Joining in Action

D'

	m	***k***	***l***
m	0	10	11
k	10	0	13
l	11	13	0

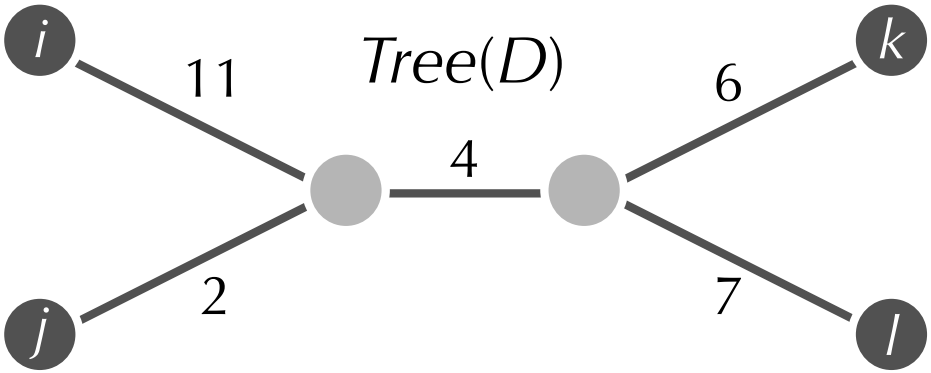


6. Apply **NeighborJoining** to D' to obtain *Tree*(D').

Neighbor-Joining in Action

D'

	m	***k***	***l***
m	0	10	11
k	10	0	13
l	11	13	0



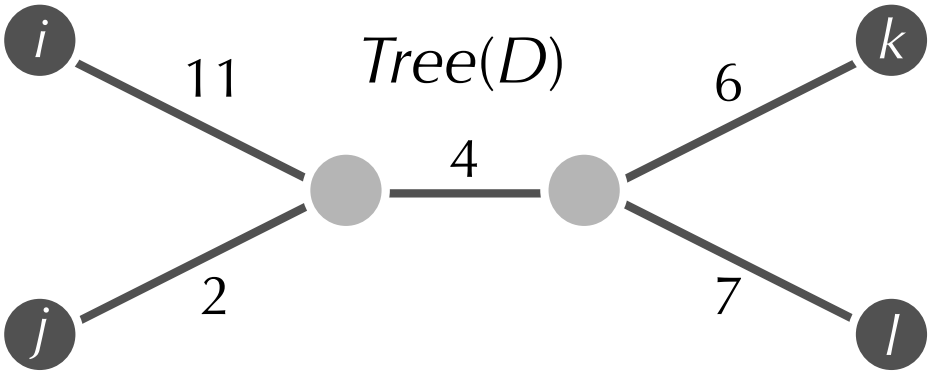
LimbLength(*i*) = ½(13 + 9) = 11
LimbLength(*i*) = ½(13 − 9) = 2

7. Reattach limbs of *i* and *j* to obtain *Tree*(*D*).

Neighbor-Joining in Action

D'

	m	***k***	***l***
m	0	10	11
k	10	0	13
l	11	13	0



7. Reattach limbs of *i* and *j* to obtain *Tree*(*D*).

Neighbor-Joining

NeighborJoining(*D*):

1. Construct neighbor-joining matrix D^* from D.
2. Find a minimum element $D^*_{i,j}$ of D^*.
3. Compute $\Delta_{i,j} = (TotalDistance_D(i) - TotalDistance_D(j)) / (n - 2)$.
4. Set $LimbLength(i)$ equal to $\frac{1}{2}(D_{i,j} + \Delta_{i,j})$ and $LimbLength(j)$ equal to $\frac{1}{2}(D_{i,j} - \Delta_{j,i})$.
5. Form a matrix D' by removing i-th and j-th row/column from D and adding an m-th row/column such that for any k, $D_{k,m} = (D_{k,i} + D_{k,j} - D_{i,j}) / 2$.
6. Apply **NeighborJoining** to D' to obtain $Tree(D')$.
7. Reattach limbs of i and j to obtain $Tree(D)$.

Code Challenge: Implement **NeighborJoining**.

Neighbor-Joining

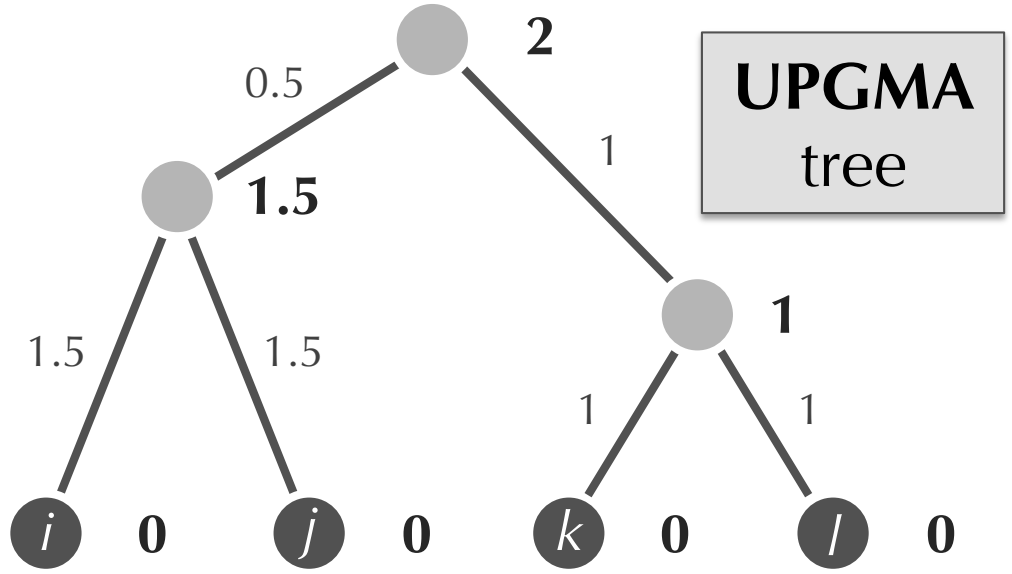
Exercise Break, check the following: Neighbor joining on a set of r taxa requires r-3 iterations. At each step one has to build and search a *D** matrix. Initially the *D** matrix is size r^2, then the next step it is $(r-1)^2$, etc. This leads to a time complexity of $O(r^3)$.

Neighbor-Joining

Exercise Break: Find the tree returned by **NeighborJoining** on the following non-additive matrix. How does the result compare with the tree produced by **UPGMA**?

D

	i	*j*	*k*	*l*
i	0	3	4	3
j	3	0	4	5
k	4	4	0	2
l	3	5	2	0



Example (different notation)

Distance matrix

	A	B	C	D	E
B	5				
C	4	7			
D	7	10	7		
E	6	9	6	5	
F	8	11	8	9	8

	U_1	C	D	E
C	3			
D	6	7		
E	5	6	5	
F	7	8	9	8

	U_1	C	U_2
C	3		
U_2	3	4	
F	7	8	6

	U_2	U_3
U_3	2	
F	6	6

	U_4
F	5

Step 1

S calculations

S_x = (sum all D_x)/($N-2$), where N is the # of OTUs in the set.

$S_A = (5+4+7+6+8)/4 = 7.5$
$S_B = (5+7+10+9+11)/4 = 10.5$
$S_C = (4+7+7+6+8)/4 = 8$
$S_D = (7+10+7+5+9)/4 = 9.5$
$S_E = (6+9+6+5+8)/4 = 8.5$
$S_F = (8+11+8+9+8)/4 = 11$

$S_{U_1} = (3+6+5+7)/3 = 7$
$S_C = (3+7+6=8)/3 = 8$
$S_D = (6+7+5+9)/3 = 9$
$S_E = (5+6+5+8)/3 = 8$
$S_F = (7+8+9+8)/3 = 10.6$

$S_{U_1} = (3+3+7)/2 = 6.5$
$S_C = (3+4+8)/2 = 7.5$
$S_{U_2} = (3+4+6)/2 = 6.5$
$S_F = (7+8+6)/2 = 10.5$

$S_{U_2} = (2+6)/1 = 8$
$S_{U_3} = (2+6)/1 = 8$
$S_F = (6+6)/1 = 12$

Because $N-2 = 0$, we cannot do this calculation.

Step 2

Calculate pair with smallest (M), where $M_{ij} = D_{ij} - S_i - S_j$.

Smallest are
$M_{AB} = 5 - 7.5 - 10.5 = -13$
$M_{DE} = 5 - 9.5 - 8.5 = -13$
Choose one of these (AB here).

Smallest is
$M_{CU_1} = 3 - 7 - 8 = -12$
$M_{DE} = 5 - 9 - 8 = -12$
Choose one of these (DE here).

Smallest is
$M_{CU_1} = 3 - 6.5 - 7.5 = -11$

Smallest is
$M_{U_2F} = 6 - 8 - 12 = -14$
$M_{U_3F} = 6 - 8 - 12 = -14$
$M_{U_2U_3} = 2 - 8 - 8 = -14$
Choose one of these ($M_{U_2U_3}$ here).

Step 3

Create a node (U) that joins pair with lowest M_{ij} such that $S_{iU} = D_{ij}/2 + (S_i - S_j)/2$.

U_1 joins A and B:
$S_{AU_1} = D_{AB}/2 + (S_A - S_B)/2 = 1$
$S_{BU_1} = D_{AB}/2 + (S_B - S_A)/2 = 4$

U_2 joins D and E:
$S_{DU_2} = D_{DE}/2 + (S_D - S_E)2 = 3$
$S_{EU_2} = D_{DE}/2 + (S_E - S_D)/2 = 2$

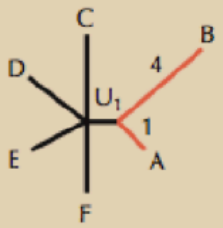
U_3 joins C and U_1:
$S_{CU_3} = D_{CU_1}/2 + (S_C - S_{U_1})/2 = 2$
$S_{U_1U_3} = D_{CU_1}/2 + (S_{U_1} - S_C)/2 = 1$

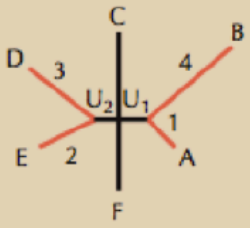
U_4 joins U_2 and U_3:
$S_{U_2U_4} = D_{U_2U_3}/2 + (S_{U_2} - S_{U_3})/2 = 1$
$S_{U_3U_4} = D_{U_2U_3}/2 + (S_{U_3} - S_{U_2})/2 = 1$.

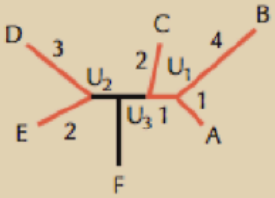
For last pair, connect U_4 and F with branch length = 5.

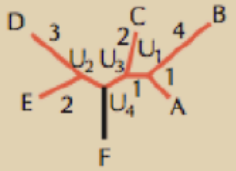
Step 4

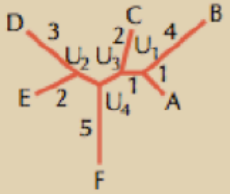
Join *i* and *j* according to *S* above and make all other taxa in form of a star. Branches in black are of unknown length. Branches in red are of known length.











Step 5

Calculate new distance matrix of all other taxa to U with $D_{xU} = D_{ix} + D_{jx} - D_{ij}$, where *i* and *j* are those selected from above.

Comments

Note this is the same tree we started with (drawn in unrooted form here).

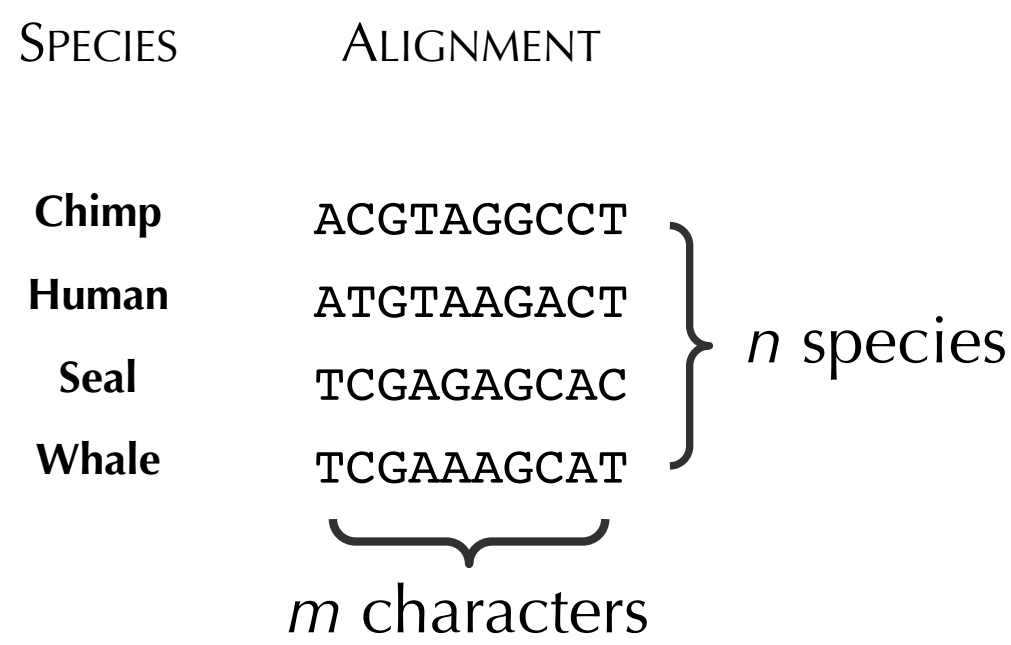
Weakness of Distance-Based Methods

Distance-based algorithms for evolutionary tree reconstruction say nothing about ancestral states at internal nodes.

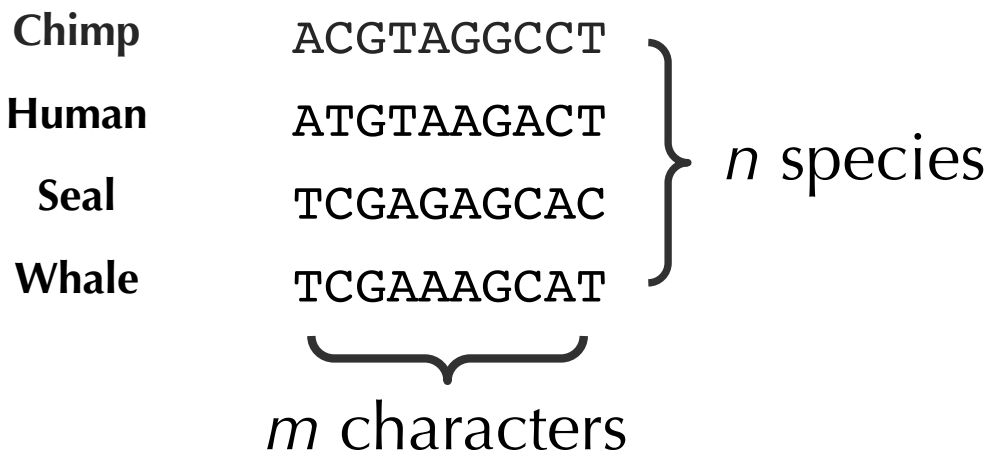
We *lost* information when we converted a multiple alignment to a distance matrix...

Species	Alignment	Distance Matrix			
		Chimp	**Human**	**Seal**	**Whale**
Chimp	`ACGTAGGCCT`	0	3	6	4
Human	`ATGTAAGACT`	3	0	7	5
Seal	`TCGAGAGCAC`	6	7	0	2
Whale	`TCGAAAGCAT`	4	5	2	0

An Alignment As a Character Table

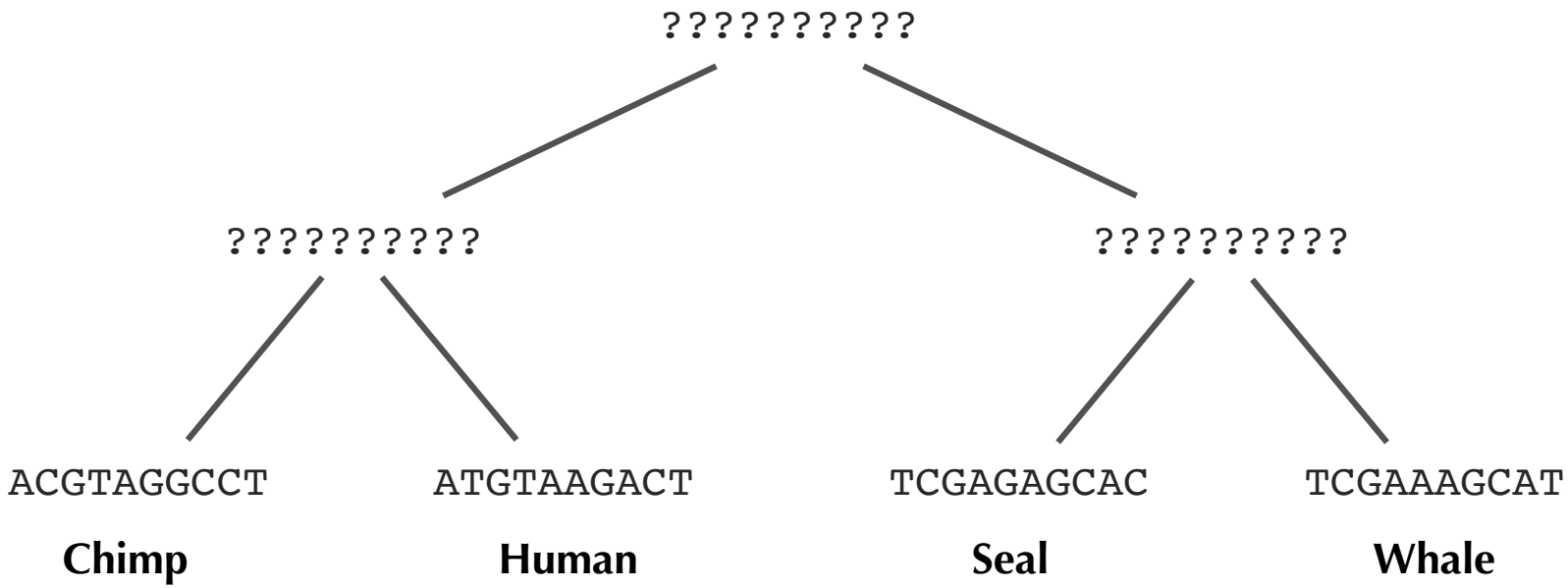


Toward a Computational Problem

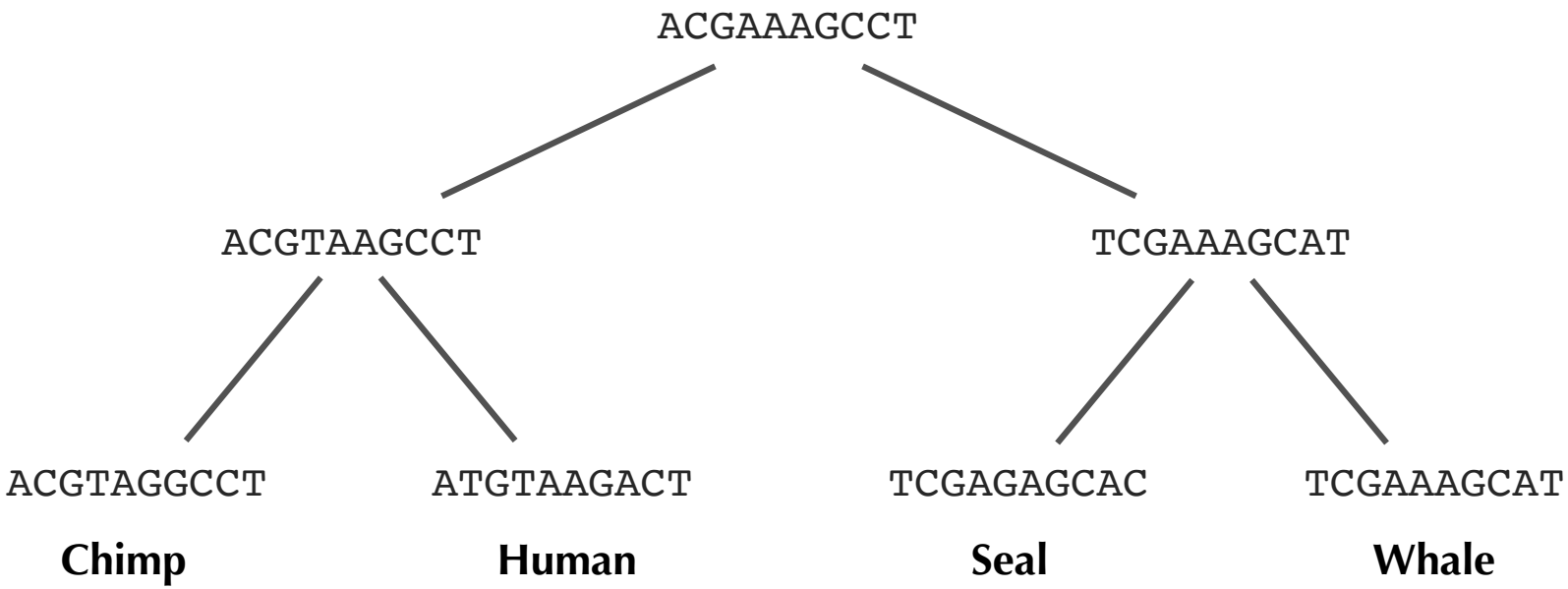


Toward a Computational Problem

Chimp	ACGTAGGCCT
Human	ATGTAAGACT
Seal	TCGAGAGCAC
Whale	TCGAAAGCAT

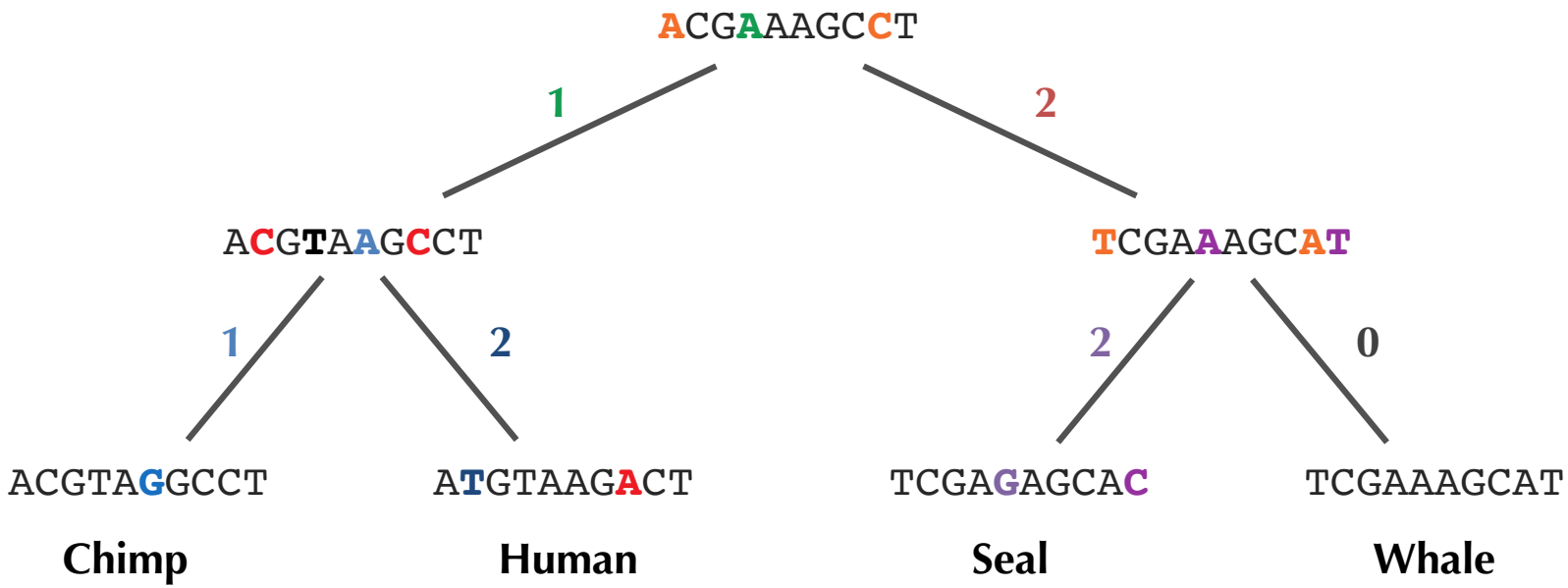


Toward a Computational Problem



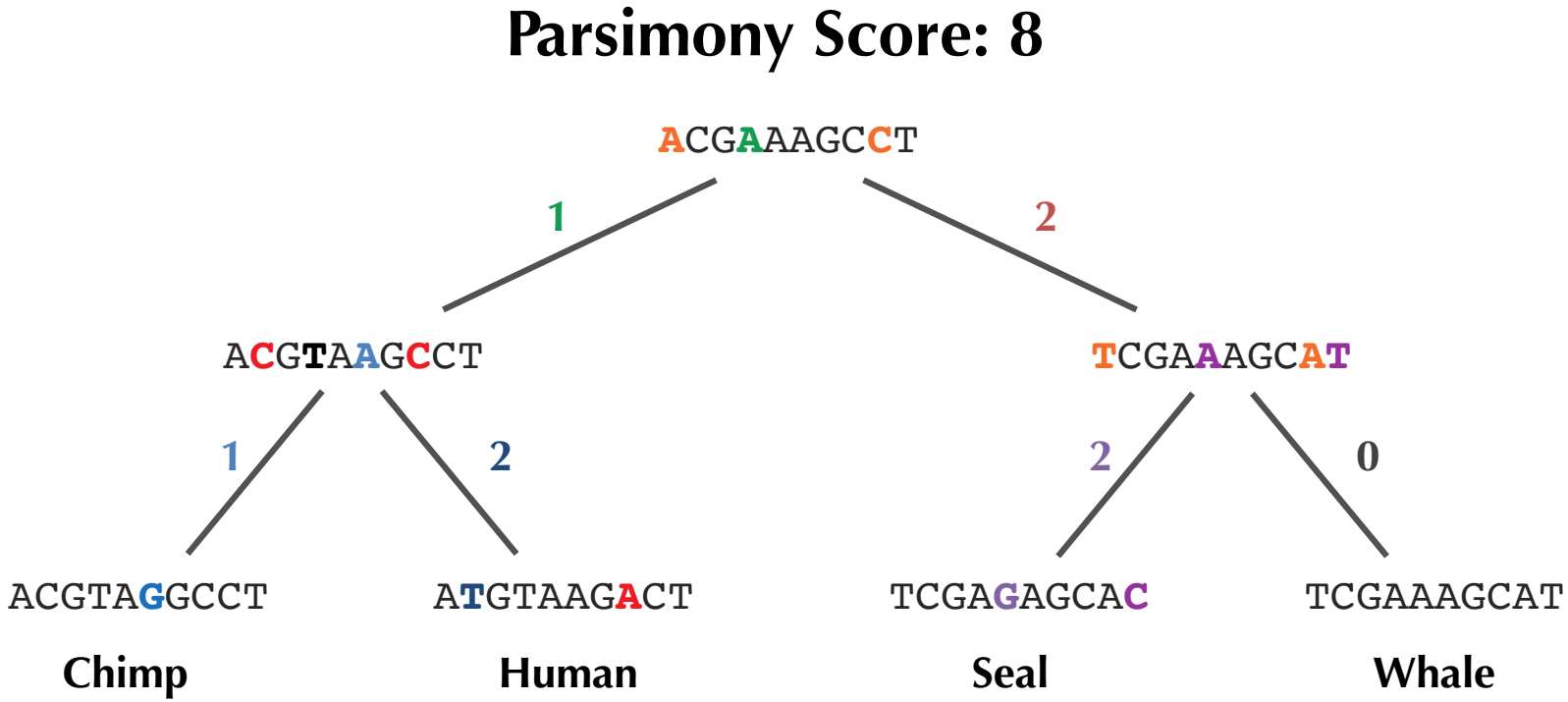
Toward a Computational Problem

Parsimony score: sum of Hamming distances along each edge.



Toward a Computational Problem

Parsimony score: sum of Hamming distances along each edge.



Toward a Computational Problem

Small Parsimony Problem: *Find the most parsimonious labeling of the internal nodes of a rooted tree.*

- **Input:** A rooted binary tree with each leaf labeled by a string of length *m*.
- **Output:** A labeling of all other nodes of the tree by strings of length *m* that minimizes the tree's parsimony score.

Toward a Computational Problem

Small Parsimony Problem: *Find the most parsimonious labeling of the internal nodes of a rooted tree.*

- **Input:** A rooted binary tree with each leaf labeled by a string of length *m*.
- **Output:** A labeling of all other nodes of the tree by strings of length *m* that minimizes the tree's parsimony score.

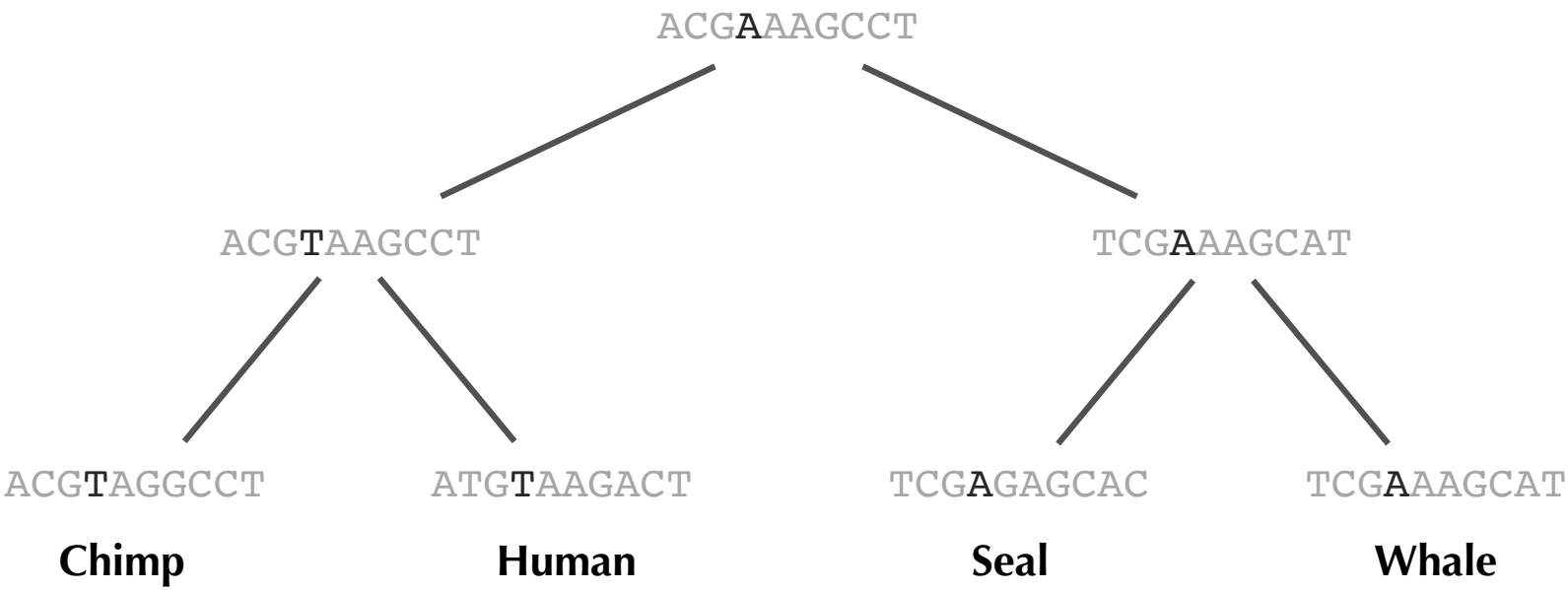
Is there any way we can simplify this problem statement?

Toward a Computational Problem

Small Parsimony Problem: *Find the most parsimonious labeling of the internal nodes of a rooted tree.*

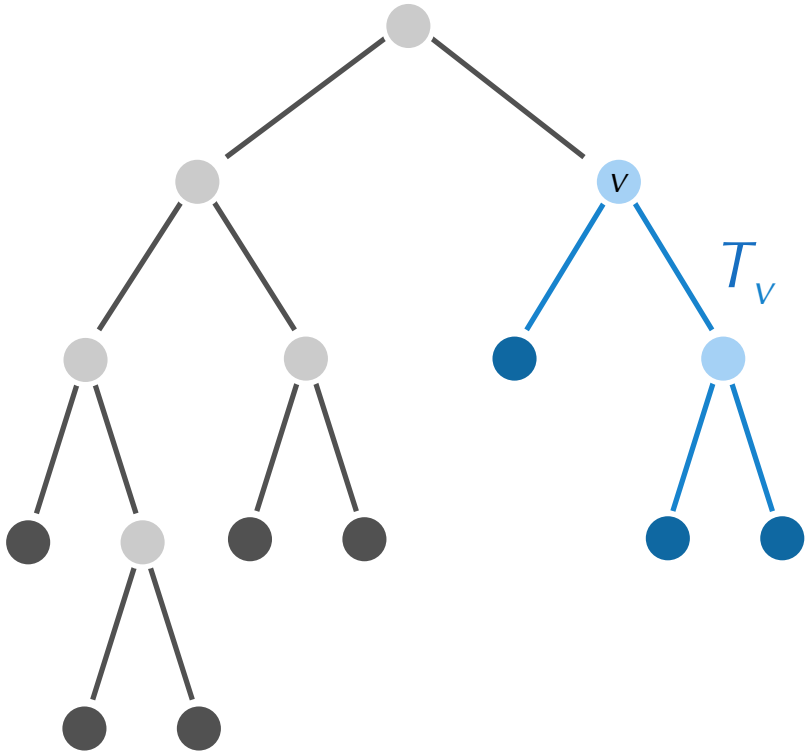
- **Input:** A rooted binary tree with each leaf labeled by a **single symbol**.
- **Output:** A labeling of all other nodes of the tree by **single symbols** that minimizes the tree's parsimony score.

Toward a Computational Problem



A Dynamic Programming Algorithm

Let T_v denote the subtree of T whose root is v.



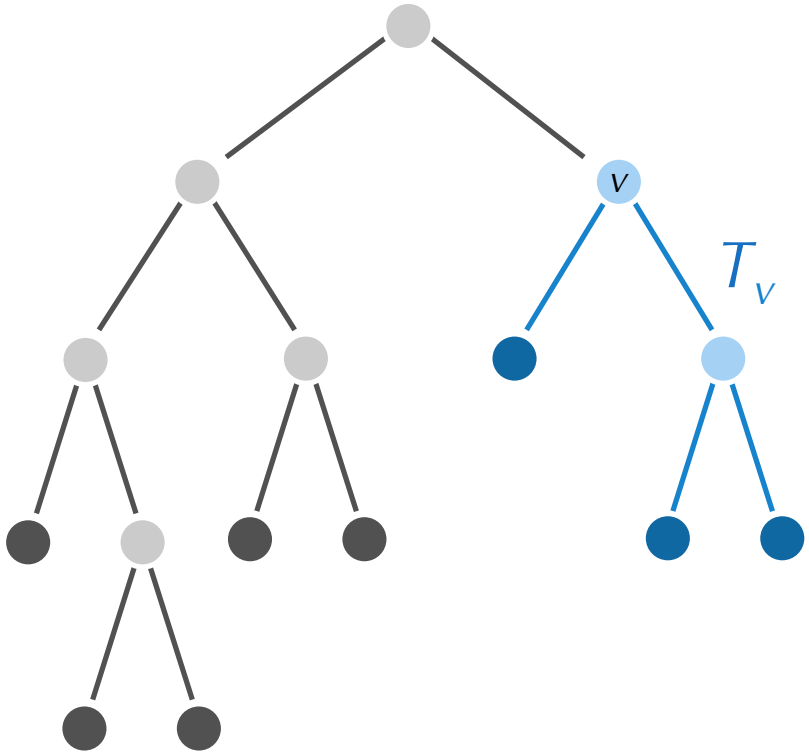
Define $s_k(v)$ as the minimum parsimony score of T_v over all labelings of T_v, assuming that v is labeled by k.

The minimum parsimony score for the tree is equal to the minimum value of $s_k(root)$ over all symbols k.

A Dynamic Programming Algorithm

For symbols i and j, define

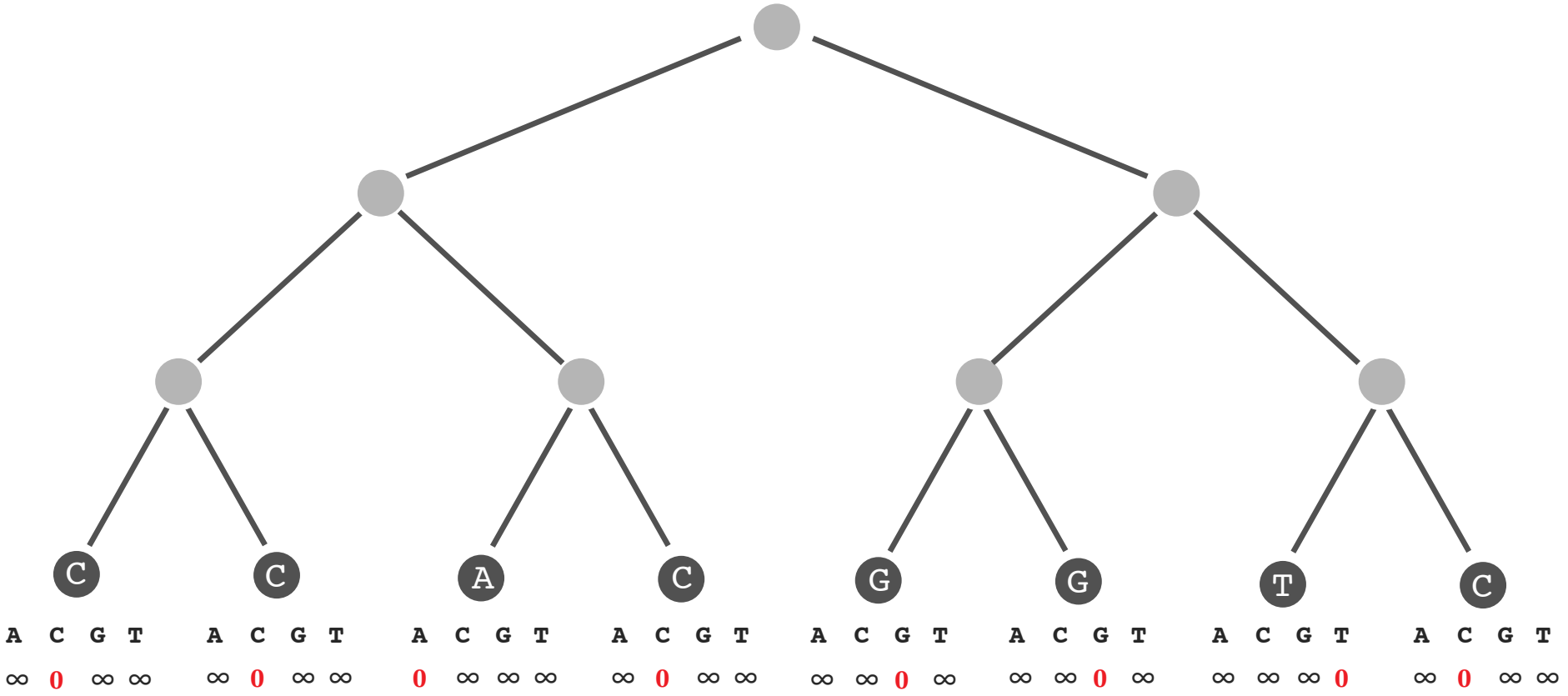
- $\delta_{i,j} = 0$ if $i = j$
- $\delta_{i,j} = 1$ otherwise.



Exercise Break: Prove the following recurrence relation:

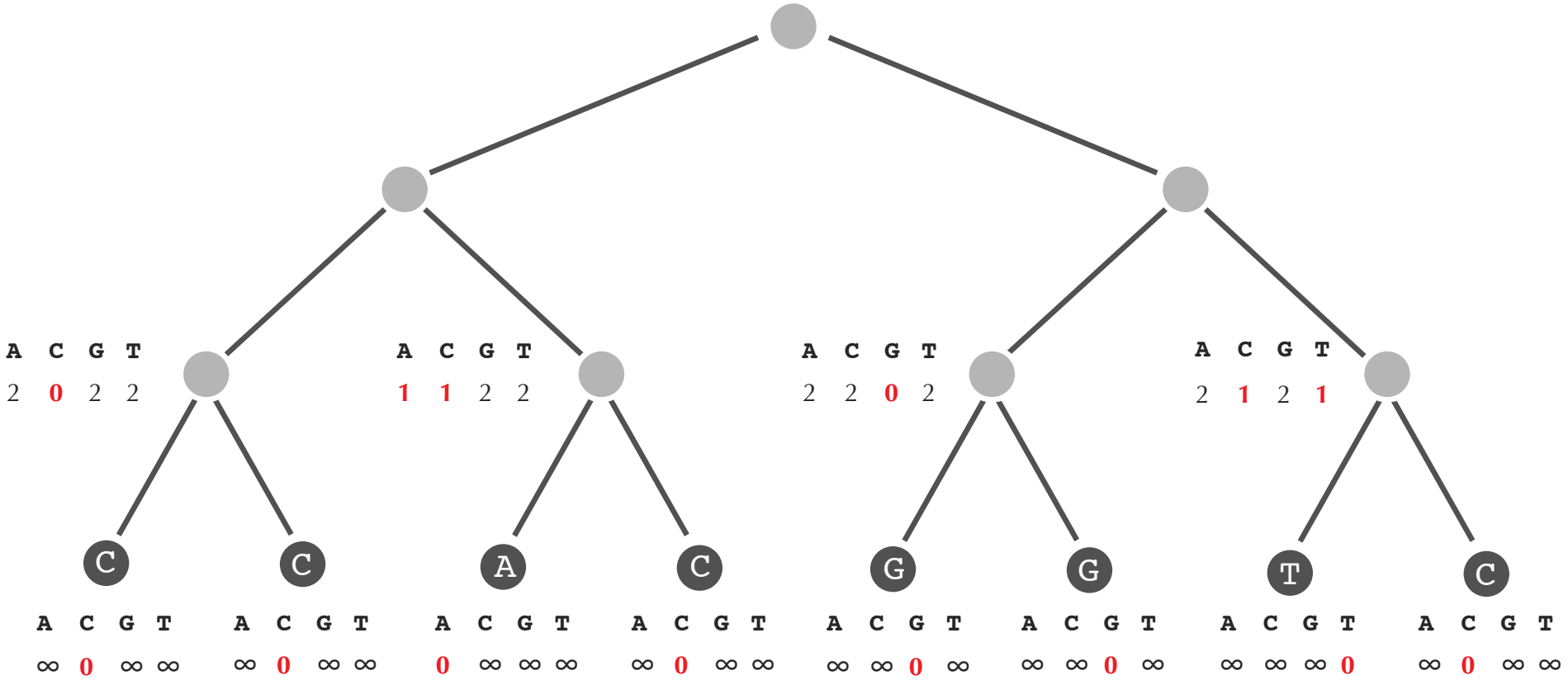
$$s_k(v) = \min_{\text{all symbols } i} \{s_i(Daughter(v)) + \delta_{i,k}\} + \min_{\text{all symbols } i} \{s_i(Son(v)) + \delta_{j,k}\}$$

A Dynamic Programming Algorithm



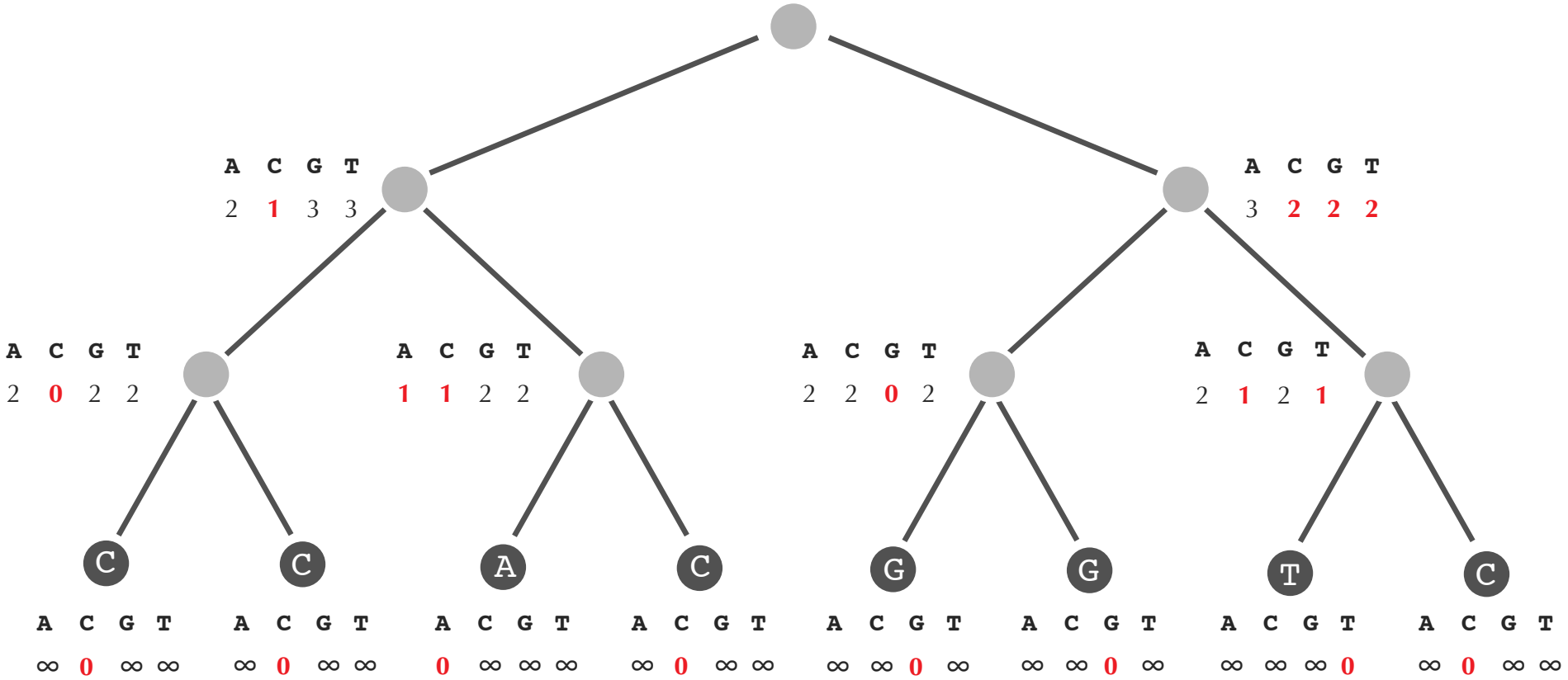
$$s_k(v) = \min_{\text{all symbols } i} \{s_i(Daughter(v)) + \delta_{i,k}\} + \min_{\text{all symbols } i} \{s_i(Son(v)) + \delta_{j,k}\}$$

A Dynamic Programming Algorithm



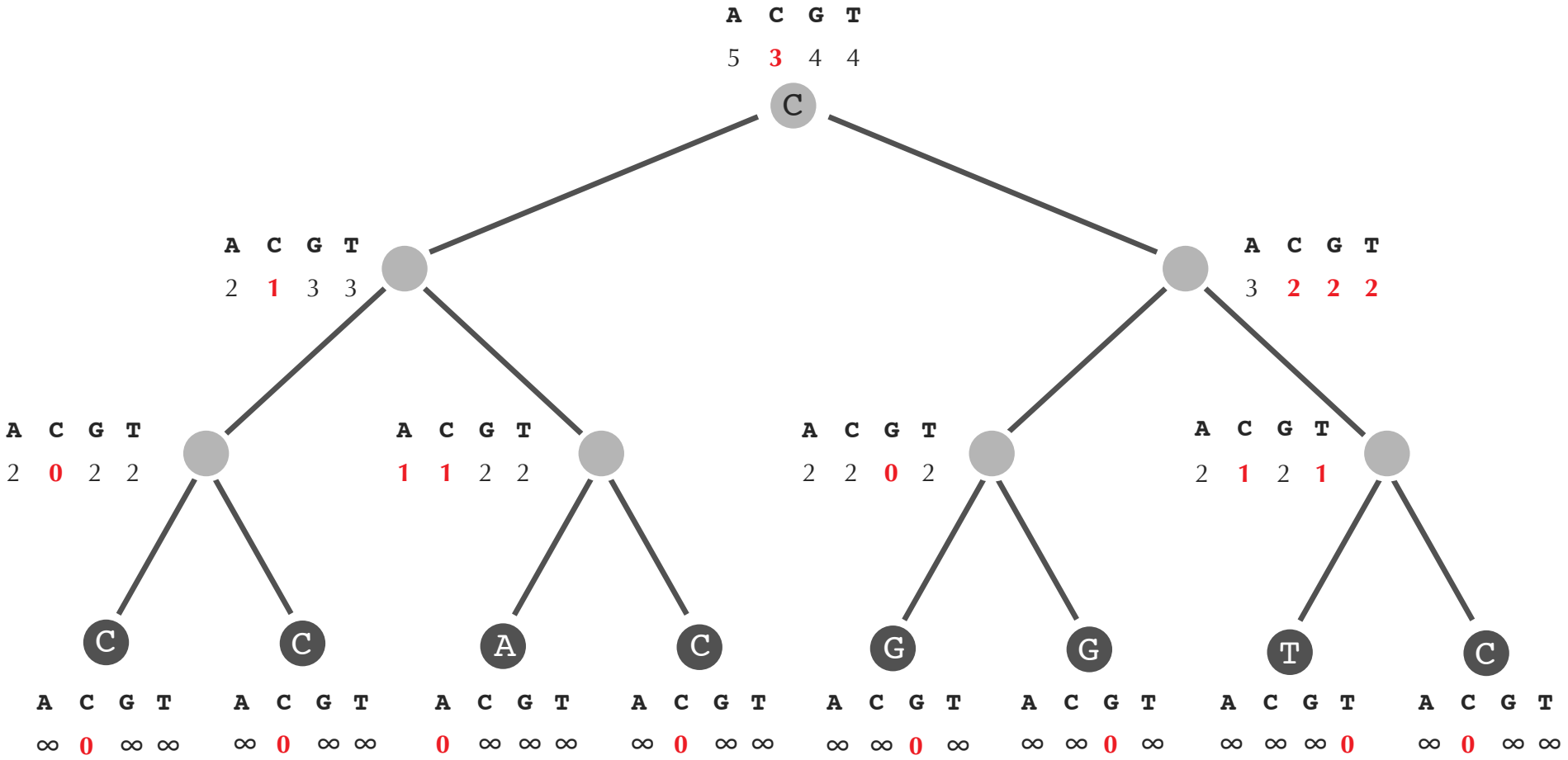
$$s_k(v) = \min_{\text{all symbols } i} \{s_i(Daughter(v)) + \delta_{i,k}\} + \min_{\text{all symbols } i} \{s_i(Son(v)) + \delta_{j,k}\}$$

A Dynamic Programming Algorithm



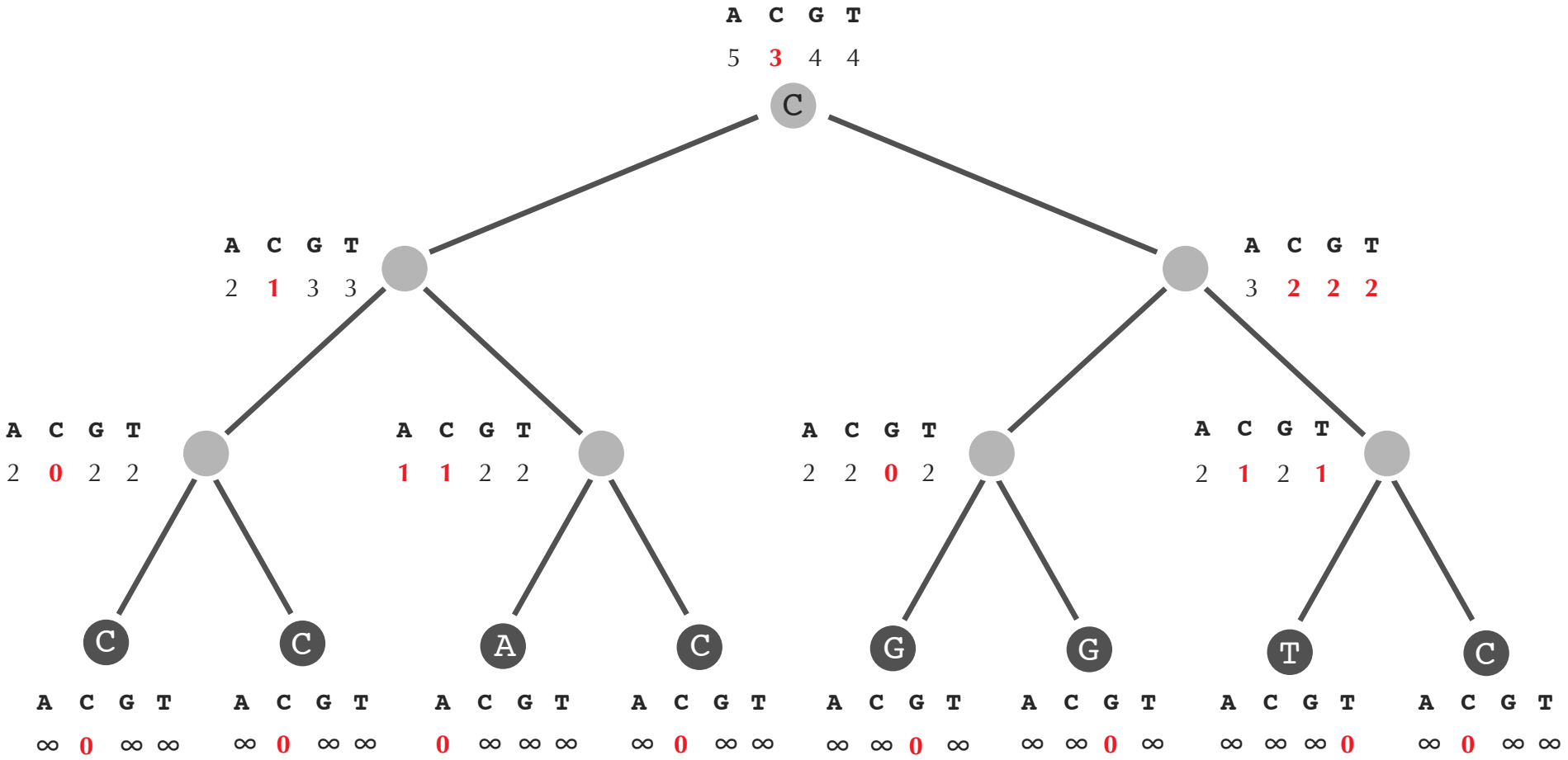
$$s_k(v) = \min_{\text{all symbols } i} \{s_i(Daughter(v)) + \delta_{i,k}\} + \min_{\text{all symbols } i} \{s_i(Son(v)) + \delta_{j,k}\}$$

A Dynamic Programming Algorithm



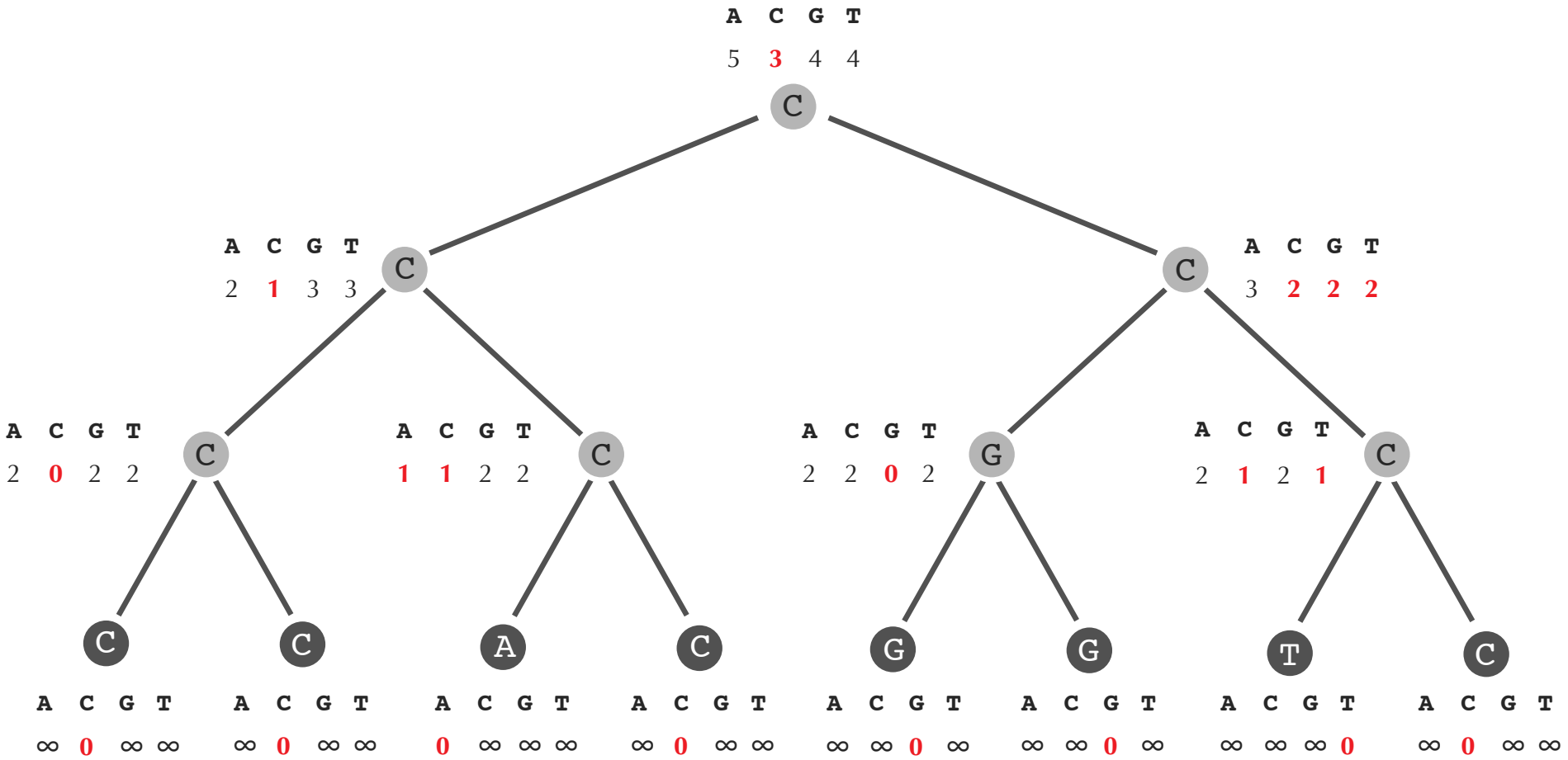
$$s_k(v) = \min_{\text{all symbols } i} \{s_i(Daughter(v)) + \delta_{i,k}\} + \min_{\text{all symbols } i} \{s_i(Son(v)) + \delta_{j,k}\}$$

A Dynamic Programming Algorithm



Exercise Break: "Backtrack" to fill in the remaining nodes of the tree.

A Dynamic Programming Algorithm



Code Challenge: Solve the Small Parsimony Problem.

Parsimony

Exercise Break, check the following: Complexity: if we want to calculate the overall length (cost) of a tree with m species, n characters, and k states, the Parsimony algorithm is of complexity $O(mnk^2)$.

David Sankoff

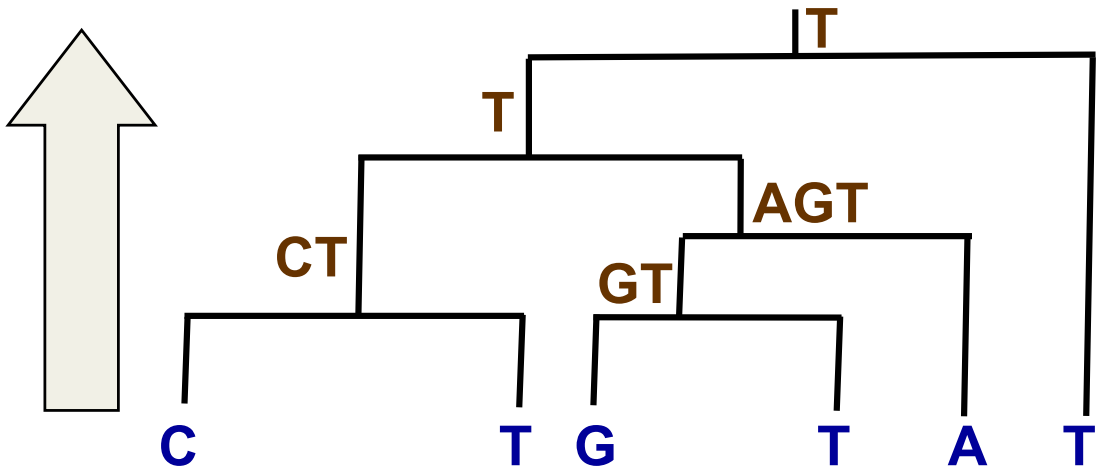
Parsimony

Exercise Break, check the following: Complexity: if we want to calculate the overall length (cost) of a tree with m species, n characters, and k states, the Parsimony algorithm is of complexity $O(mnk^2)$.

COMMENT: if each mutation costs the same then a simplified, earlier version of this algorithm from Walter Fitch gives a run time complexity of O(mnk). If **Each mutation *a*↔*b* costs differently** you have a weighted edit distance (particularly for amino acid sequences) then your complexity is likely to be $O(mnk^2)$

Simple example

$$R_i = \begin{cases} R_j \cap R_k \ \ if \ \ R_j \cap R_k \neq \phi \\ R_j \cup R_k \ \ otherwise \end{cases}$$



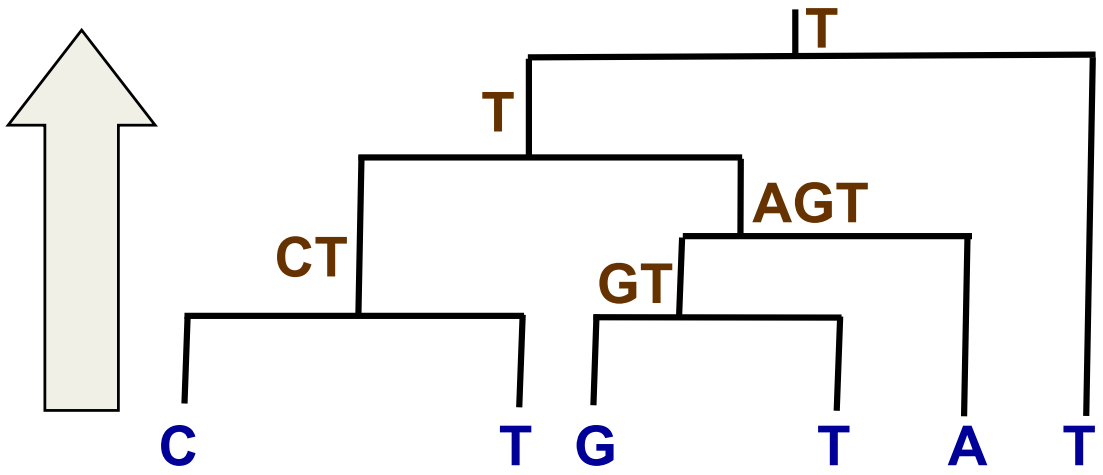
simple case Sankoff equivalent to computing this

using this scoring matrix

	A	T	G	C
A	0	1	1	1
T	1	0	1	1
G	1	1	0	1
C	1	1	1	0

Bottom-UP phase

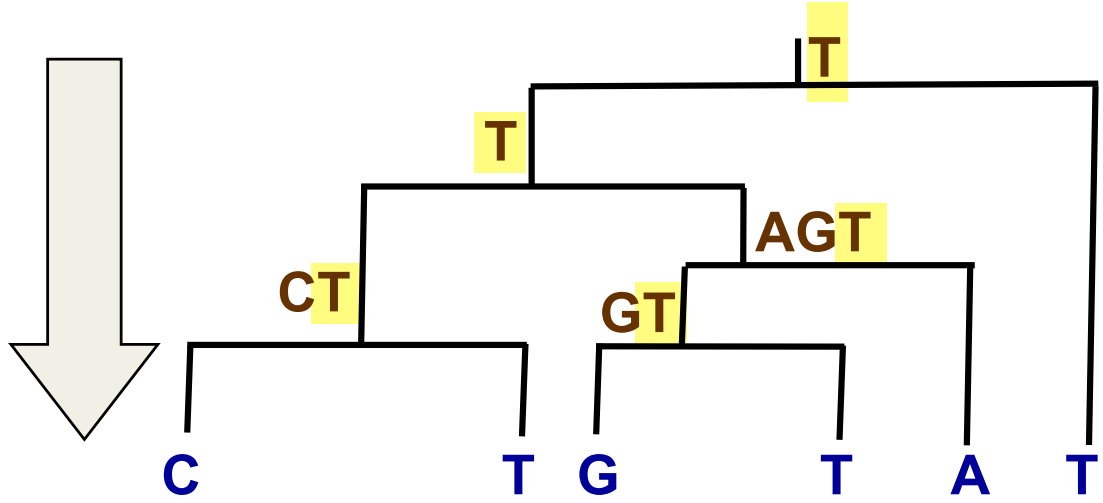
$$R_i = \begin{cases} R_j \cap R_k \ \ if \ \ R_j \cap R_k \neq \phi \\ R_j \cup R_k \ \ otherwise \end{cases}$$



Top-down phase

$$s_j = \begin{cases} s_i \ \ if \ \ s_i \in R_j \\ arbitrary \ \ state \in R_j \ \ otherwise \end{cases}$$

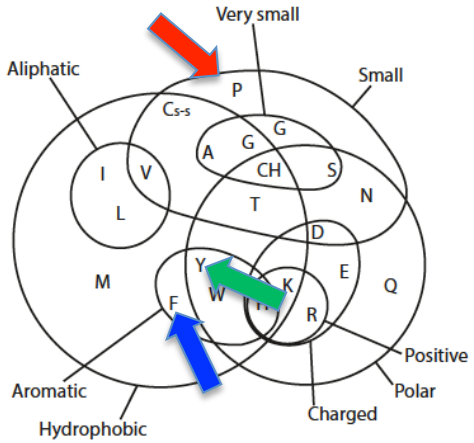
Complexity: ***O(mnk)***



score = 3

How to compare amino acids: scoring matrices

		C	S	T	P	A	G	N	D	E	Q	H	R	K	M	I	L	V	F	Y	W
C	Cys	12																			
S	Ser	0	2																		
T	Thr	-2	1	3																	
P	Pro	-3	1	0	6																
A	Ala	-2	1	1	1	2															
G	Gly	-3	1	0	-1	1	5														
N	Asn	-4	1	0	-1	0	0	2													
D	Asp	-5	0	0	-1	0	1	2	4												
E	Glu	-5	0	0	-1	0	0	1	3	4											
Q	Gln	-5	-1	-1	0	0	-1	1	2	2	4										
H	His	-3	-1	-1	0	-1	-2	2	1	1	3	6									
R	Arg	-4	0	-1	0	-2	-3	0	-1	-1	1	2	6								
K	Lys	-5	0	0	-1	-1	-2	1	0	0	1	0	3	5							
M	Met	-5	-2	-1	-2	-1	-3	-2	-3	-2	-1	-2	0	0	6						
I	Ile	-2	-1	0	-2	-1	-3	-2	-2	-2	-2	-2	-2	-2	2	5					
L	Leu	-6	-3	-2	-3	-2	-4	-3	-4	-3	-2	-2	-3	-3	4	2	6				
V	Val	-2	-1	0	-1	0	-1	-2	-2	-2	-2	-2	-2	-2	2	4	2	4			
F	Phe	-4	-3	-3	-5	-5	-5	-4	-6	-5	-5	-2	-4	-5	0	1	2	-1	9		
Y	Tyr	0	-3	-3	**-5**	-3	-5	-2	-4	-4	-4	0	-4	-4	-2	-1	-1	-2	**7**	10	
W	Trp	-8	-2	-5	-6	-6	-7	-4	-7	-7	-5	-3	2	-3	-4	-5	-2	-6	0	0	17



Margaret Dayhoff

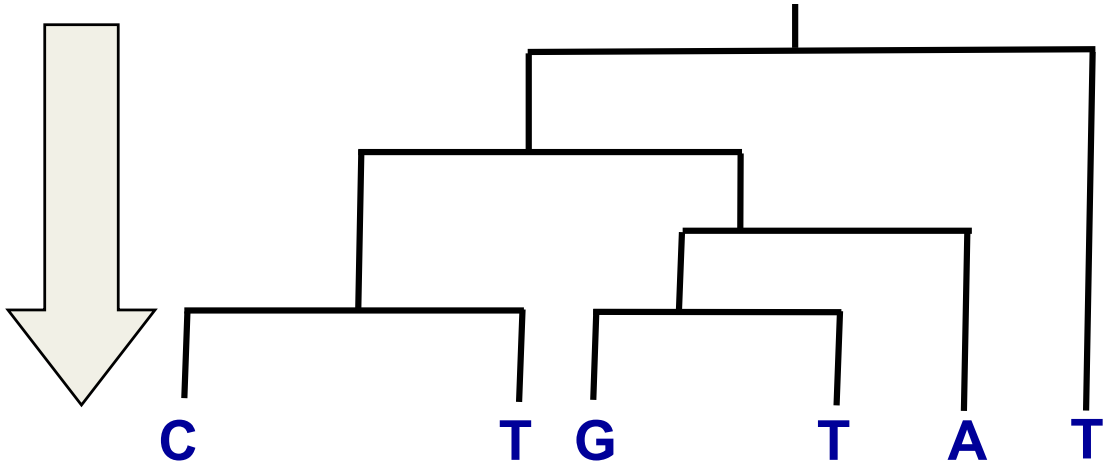
example: Y (Tyr) often mutates into F (score +7) but rarely mutates into P (score -5)

Top-down phase

Sankoff's Algorithm

Pick states for each internal node

- Select minimal cost character for root (s minimizing $R_{root}(s)$)
- Do pre-order (from root to leaves) traversal of tree:
  - For internal node j, with parent i, select state that produced minimal cost at i (use pointers kept in 1st stage)



$$R_i(s) = \min_{s'}\{R_j(s') + S(s',s)\} + \min_{s'}\{R_k(s') + S(s',s)\}$$

Complexity: $O(mnk^2)$

JOURNAL OF COMPUTATIONAL BIOLOGY
Volume 21, Number 11, 2014

Pp. 799–808
DOI: 10.1089/cmb.2014.0128

The Worst Case Complexity of Maximum Parsimony

AMIR CARMEL, NOA MUSA-LEMPEL, DEKEL TSUR, and MICHAL ZIV-UKELSON

ABSTRACT

One of the core classical problems in computational biology is that of constructing the most parsimonious phylogenetic tree interpreting an input set of sequences from the genomes of evolutionarily related organisms. We reexamine the classical maximum parsimony (MP) optimization problem for the general (asymmetric) scoring matrix case, where rooted phylogenies are implied, and analyze the worst case bounds of three approaches to MP: The approach of Cavalli-Sforza and Edwards, the approach of Hendy and Penny, and a new agglomerative, "bottom-up" approach we present in this article. We show that the second and third approaches are faster than the first one by a factor of $\Theta(\sqrt{n})$ and $\Theta(n)$, respectively, where n is the number of species.

Key words: maximum parsimony, large parsimony, phylogeny, phylogenetic reconstruction, asymmetric scoring matrix, dendograms.

simple versus more general case

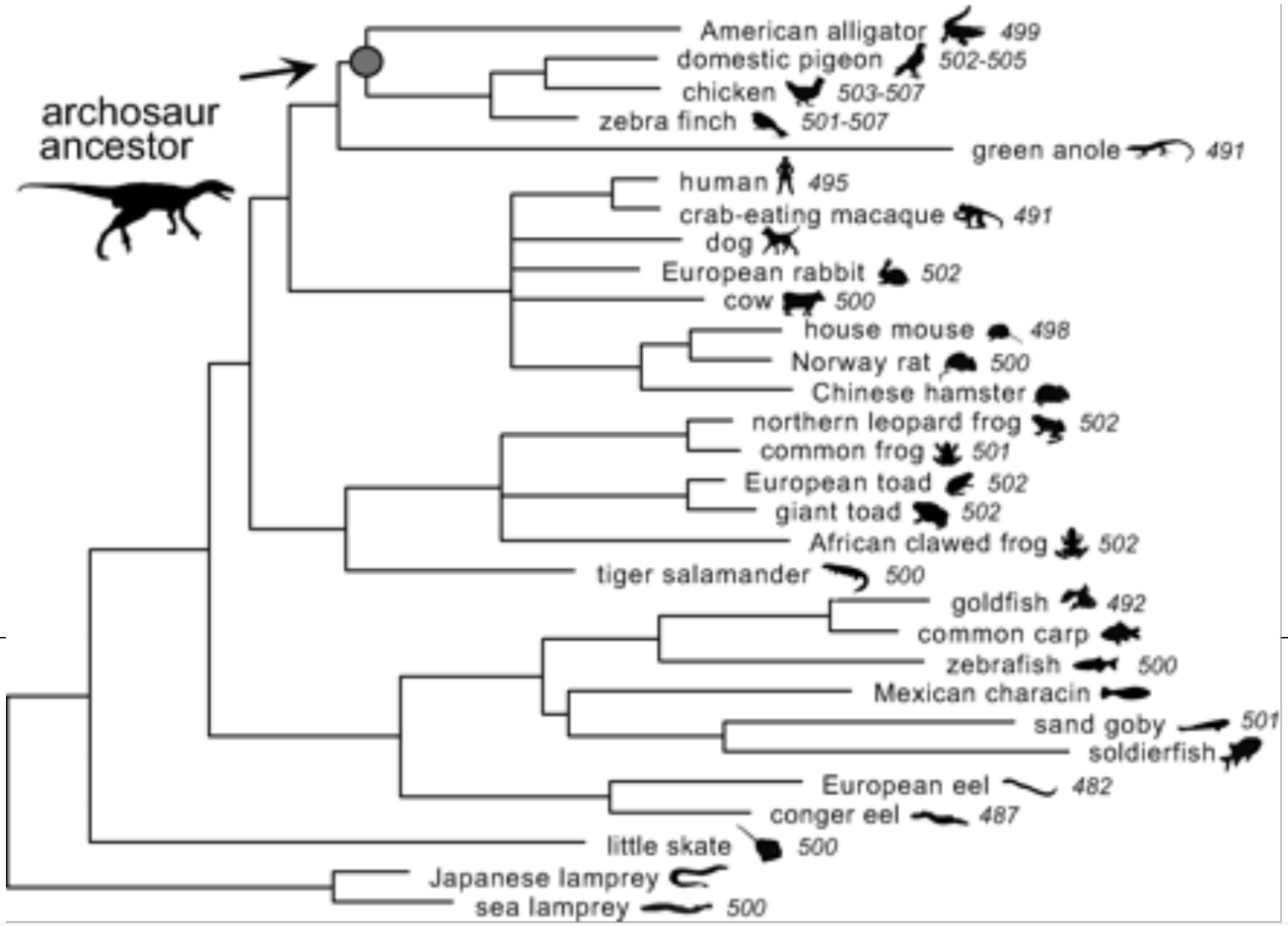
Measuring SP and MP complexity in terms of basic operations. SP and MP algorithms work by computing some information for every internal vertex of the input phylogeny. This information, as well as the complexity of its computation, depend on the scoring scheme employed by the parsimony algorithm. Thus, in what follows, we will use the term *basic operation* to denote the work invested in the computation of the information of a single vertex of a considered phylogeny for a specific scoring scheme. For example, in the Fitch SP algorithm (Fitch, 1971), which computes a minimal Hamming distance SP score, an $O(m)$-time basic operation is applied, while in the Sankoff algorithm (Sankoff, 1975), which optimizes an SP score of minimal weighted edit distance, an $O(m\Sigma^2)$-time basic operation is applied, where Σ denotes the size of the alphabet spelling the input sequences.

Recreating a Functional Ancestral Archosaur Visual Pigment FREE

Belinda S. W. Chang, Karolina Jönsson, Manija A. Kazmi, Michael J. Donoghue, Thomas P. Sakmar

Molecular Biology and Evolution, Volume 19, Issue 9, 1 September 2002, Pages 1483–1489, https://doi.org/10.1093/oxfordjournals.molbev.a004211

Why is interesting to know internal node's composition?



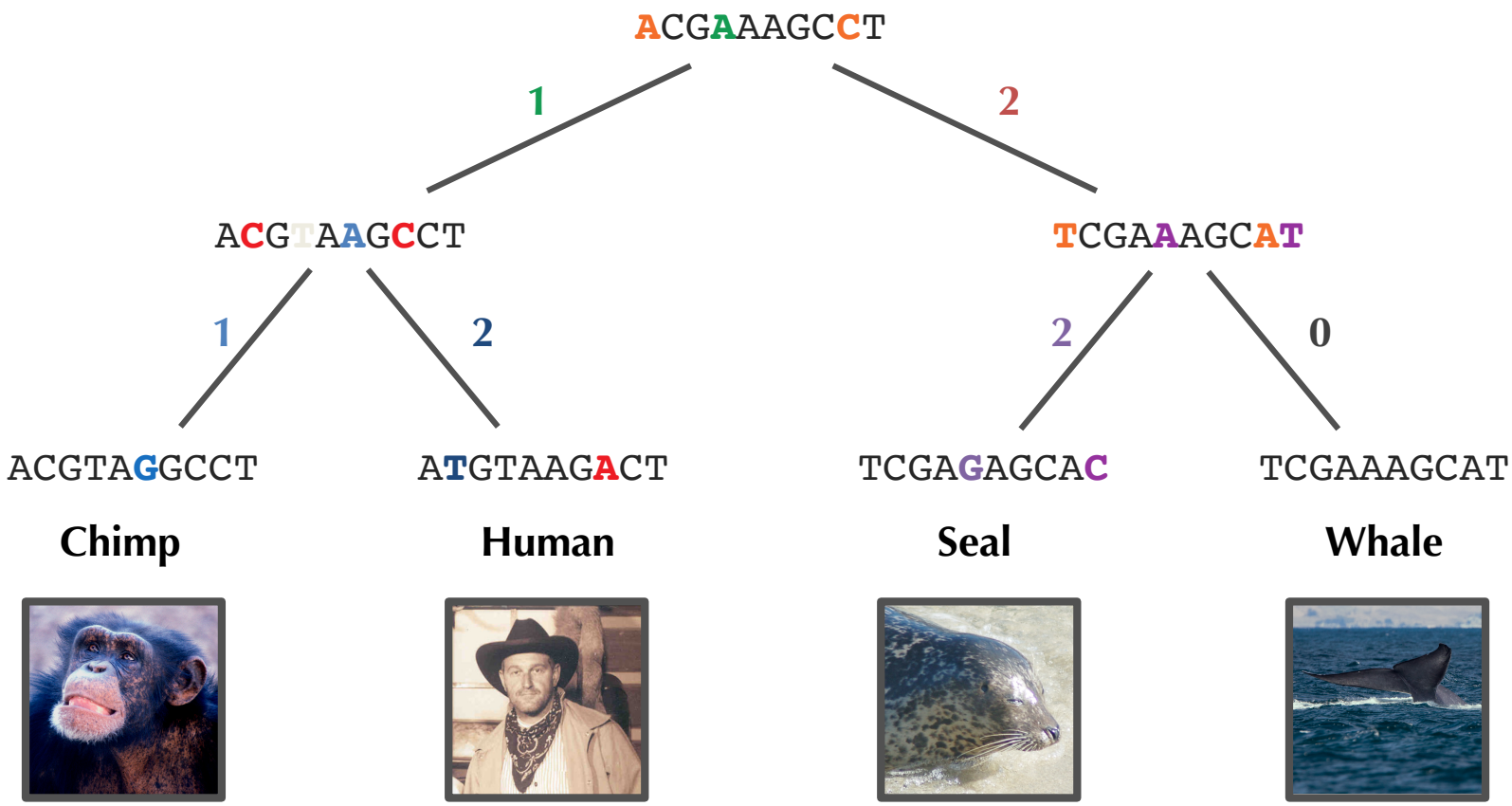
Small Parsimony for Unrooted Trees

Small Parsimony in an Unrooted Tree Problem: *Find the most parsimonious labeling of the internal nodes of an unrooted tree.*

- **Input:** An unrooted binary tree with each leaf labeled by a string of length *m*.
- **Output:** A position of the root and a labeling of all other nodes of the tree by strings of length *m* that minimizes the tree's parsimony score.

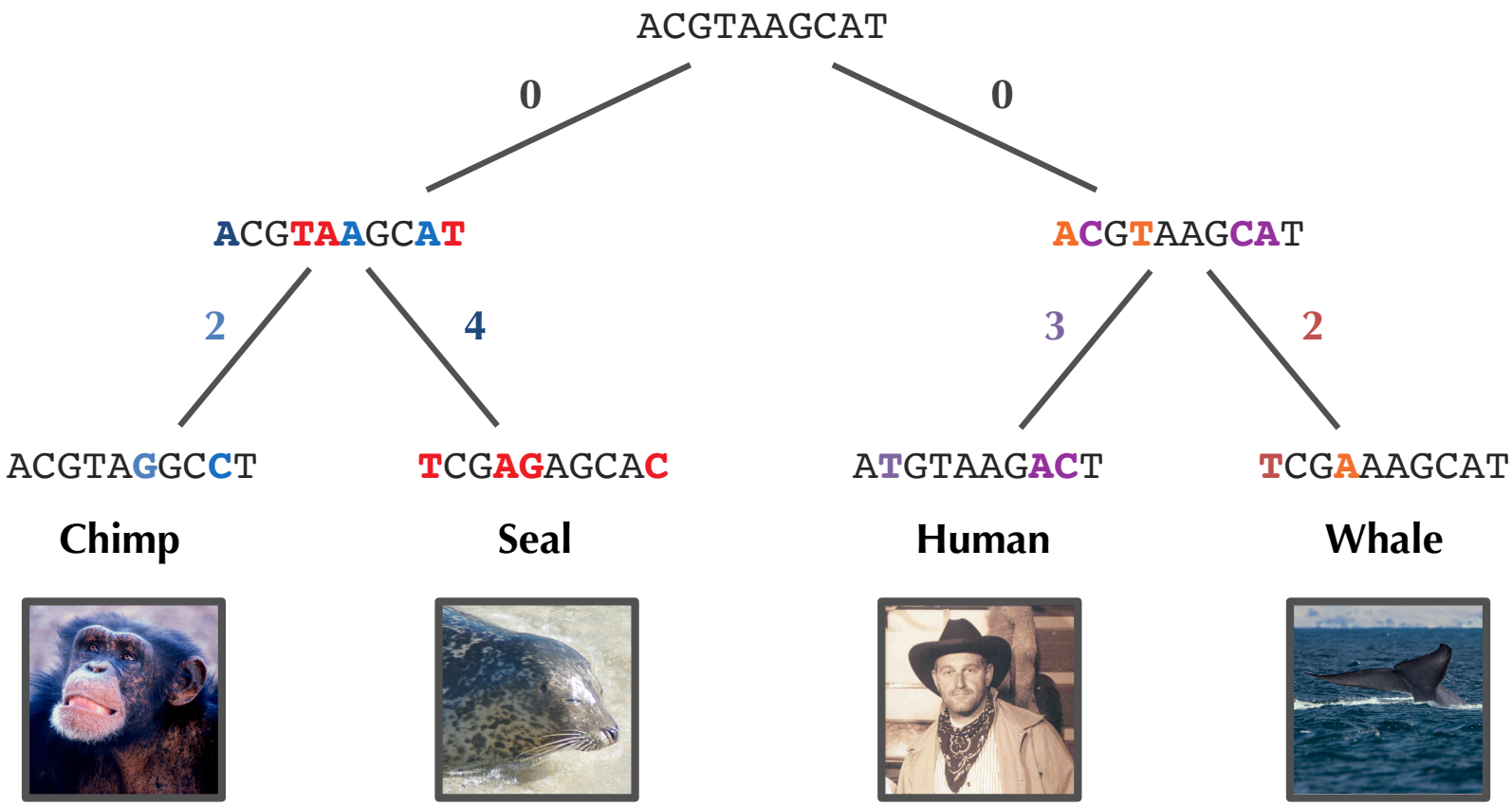
Code Challenge: Solve this problem.

Finding the Most Parsimonious Tree



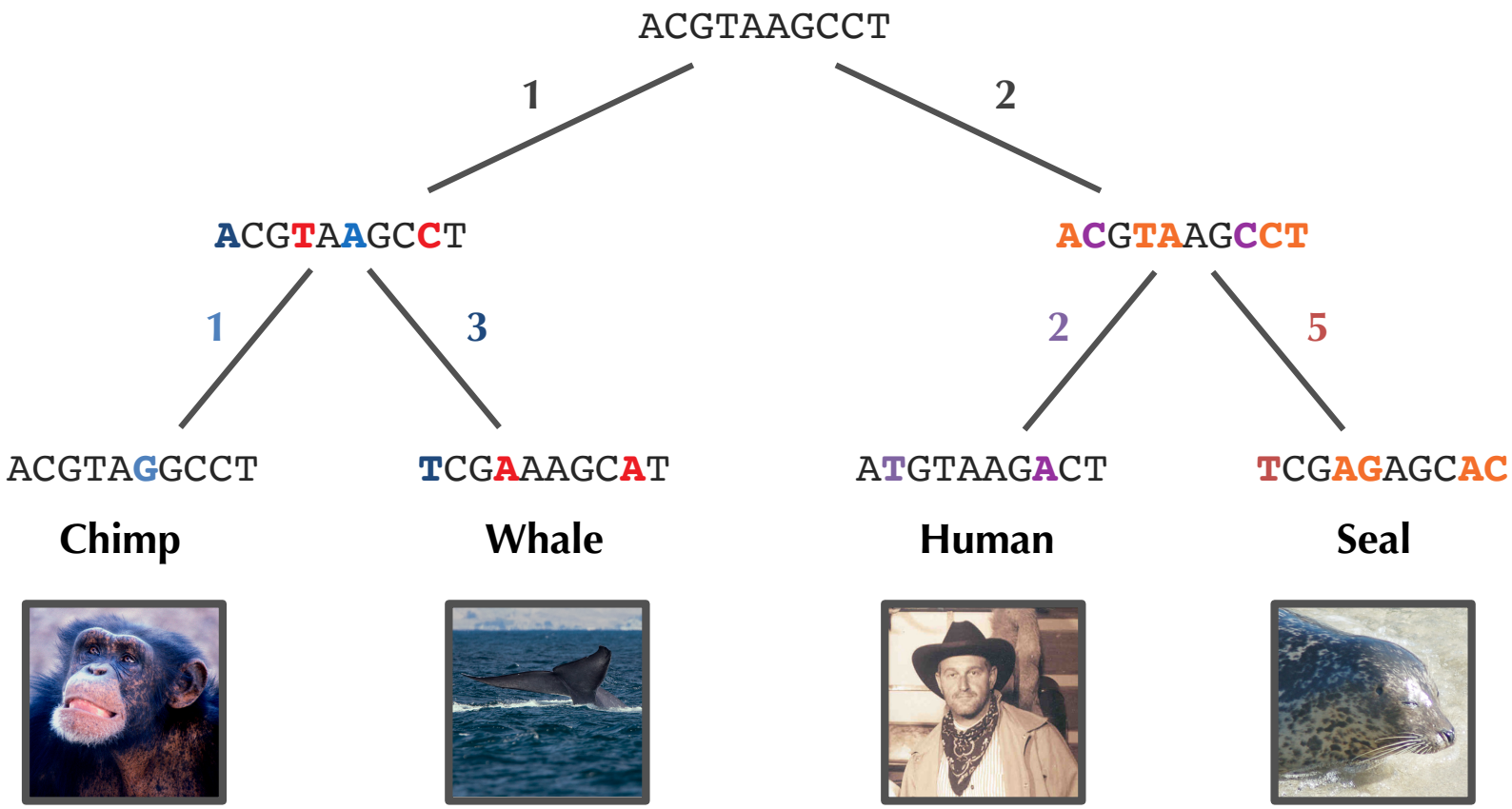
Parsimony Score: 8

Finding the Most Parsimonious Tree



Parsimony Score: 11

Finding the Most Parsimonious Tree



Parsimony Score: 14

Finding the Most Parsimonious Tree

Large Parsimony Problem: *Given a set of strings, find a tree* (*with leaves labeled by all these strings*) *having minimum parsimony score*.

- **Input:** A collection of strings of equal length.
- **Output:** A rooted binary tree *T* that minimizes the parsimony score among all possible rooted binary trees with leaves labeled by these strings.

Finding the Most Parsimonious Tree

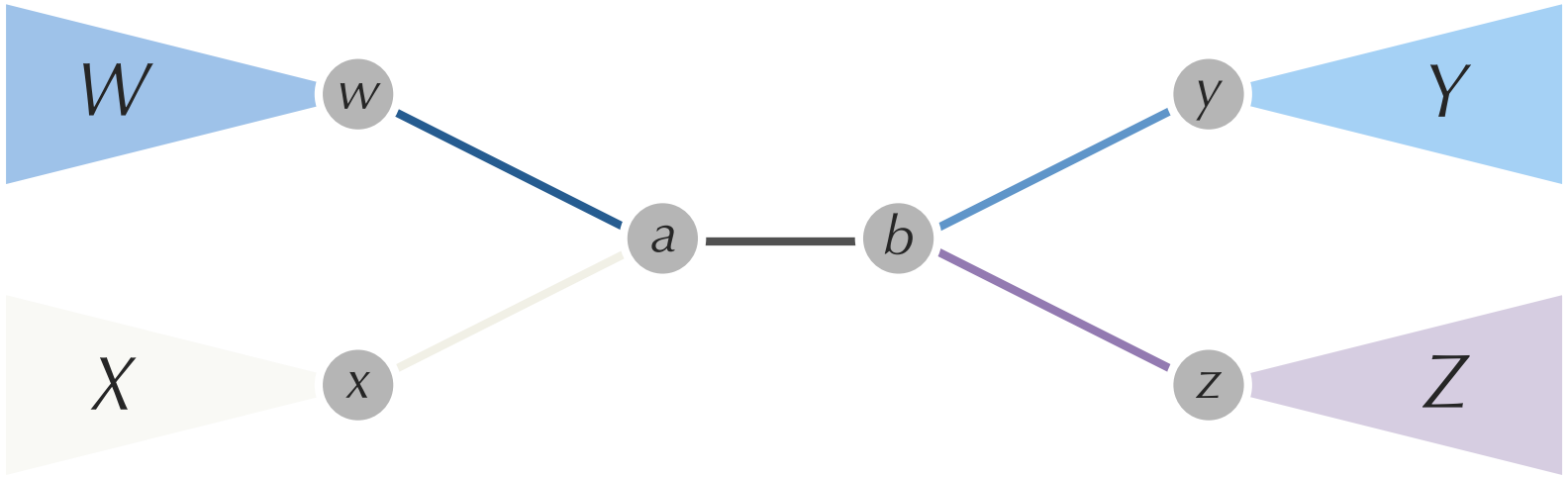
Large Parsimony Problem: *Given a set of strings, find a tree* (*with leaves labeled by all these strings*) *having minimum parsimony score*.

- **Input:** A collection of strings of equal length.
- **Output:** A rooted binary tree *T* that minimizes the parsimony score among all possible rooted binary trees with leaves labeled by these strings.

Unfortunately, this problem is *NP*-Complete...

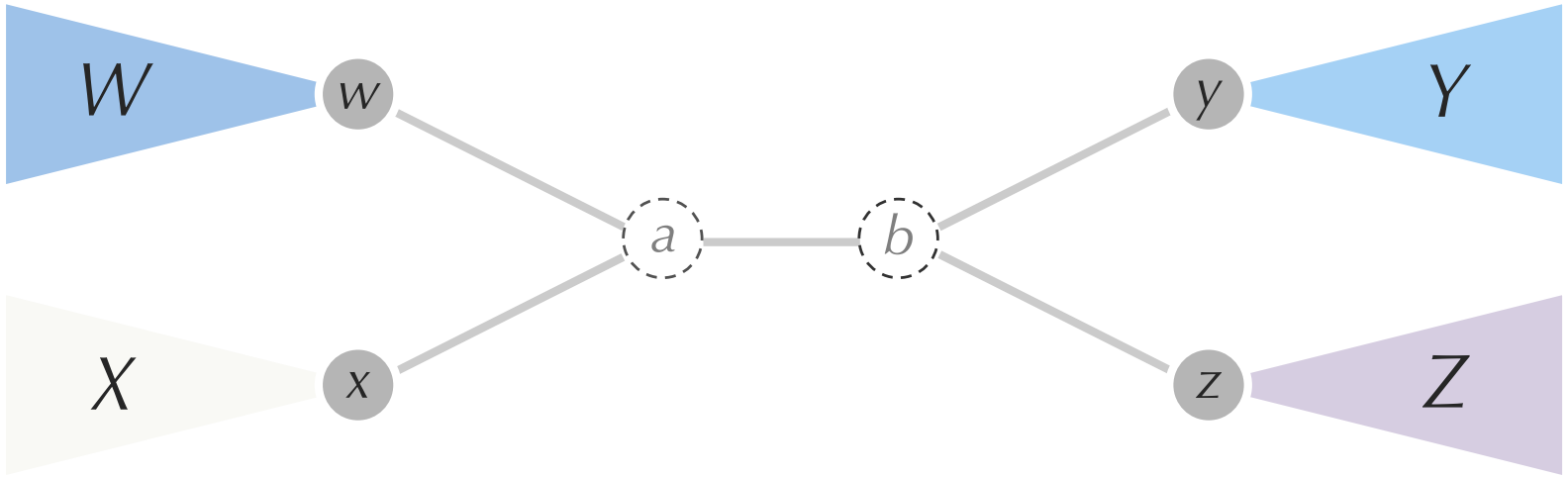
A Greedy Heuristic for Large Parsimony

Note that removing an **internal edge**, an edge connecting two internal nodes (along with the nodes), produces four subtrees (*W*, *X*, *Y*, *Z*).



A Greedy Heuristic for Large Parsimony

Note that removing an **internal edge**, an edge connecting two internal nodes (along with the nodes), produces four subtrees (*W*, *X*, *Y*, *Z*).

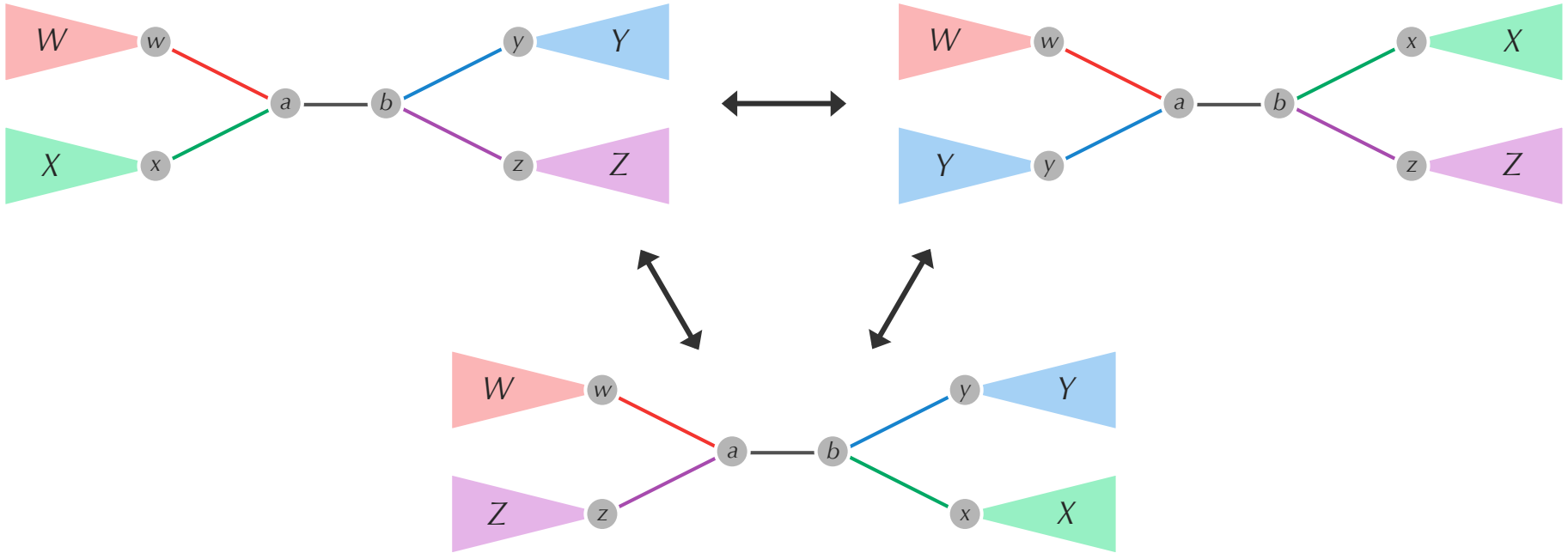


A Greedy Heuristic for Large Parsimony

Note that removing an **internal edge**, an edge connecting two internal nodes (along with the nodes), produces four subtrees (*W*, *X*, *Y*, *Z*).

A Greedy Heuristic for Large Parsimony

Rearranging these subtrees is called a **nearest neighbor interchange.**



A Greedy Heuristic for Large Parsimony

Nearest Neighbors of a Tree Problem: *Given an edge in a binary tree, generate the two neighbors of this tree.*

- **Input:** An internal edge in a binary tree.
- **Output:** The two nearest neighbors of this tree (for the given internal edge).

Code Challenge: Solve this problem.

A Greedy Heuristic for Large Parsimony

Nearest Neighbor Interchange Heuristic:

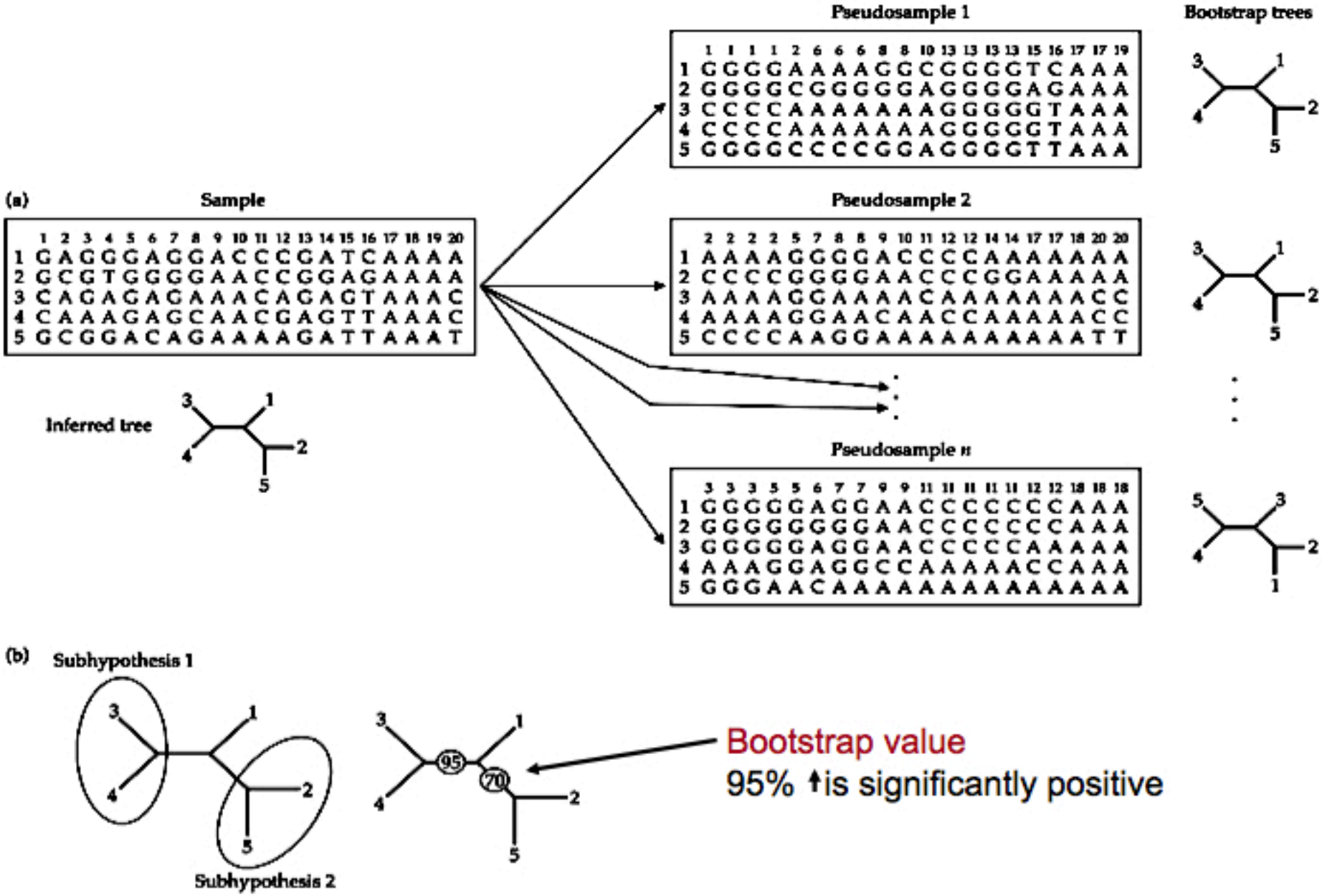
1. Set current tree equal to arbitrary binary rooted tree structure.
2. Go through all internal edges and perform all possible nearest neighbor interchanges.
3. Solve Small Parsimony Problem on each tree.
4. If any tree has parsimony score improving over optimal tree, set it equal to the current tree. Otherwise, return current tree.

Code Challenge: Implement the nearest-neighbor interchange heuristic.

Tree validation: the bootstrap algorithm

- If there are m sequences, each with n nucleotides, a phylogenetic tree can be reconstructed using some tree building methods.
- From each sequence, n nucleotides are randomly chosen with replacements, giving rise to m rows of n columns each. These now constitute a new set of sequences.
- A tree is then reconstructed with these new sequences using the same tree building method as before.
- Next the topology of this tree is compared to that of the original tree. Each interior branch of the original tree that is different from the bootstrap tree is given a score of 0; all other interior branches are given the value 1.
- This procedure of resampling the sites and tree reconstruction is repeated several hundred times, and the percentage of times each interior branch is given a value of 1 is noted. This is known as the bootstrap value. As a general rule, if the bootstrap value for a given interior branch is 95% or higher, then the topology at that branch is considered "correct".

Tree validation: the bootstrap algorithm



EXAMPLE: Phylogenetic-inspired techniques for reverse engineering and detection of malware families

For example, given an execution trace of instructions,

```
push ebp
mov ebp, esp
mov eax, dword ptr [ebp-0x4]
jmp +0x14
```

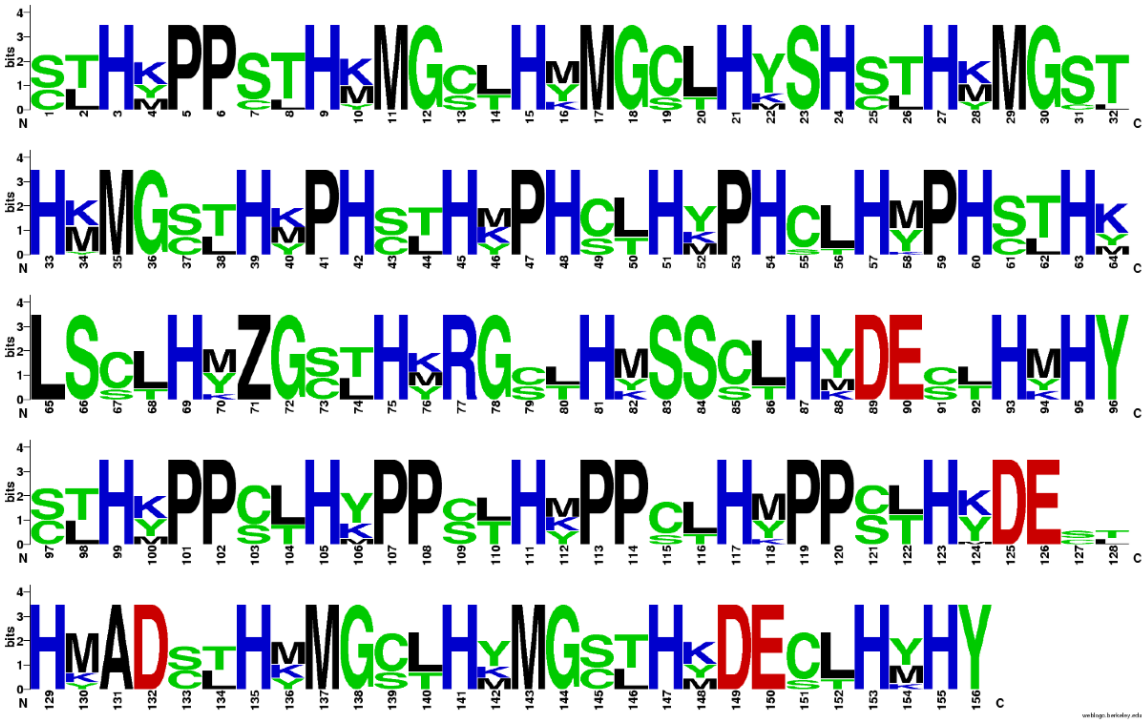
it is abstracted as a sequence of mnemonics, i.e.

```
push, mov, mov, jmp
```

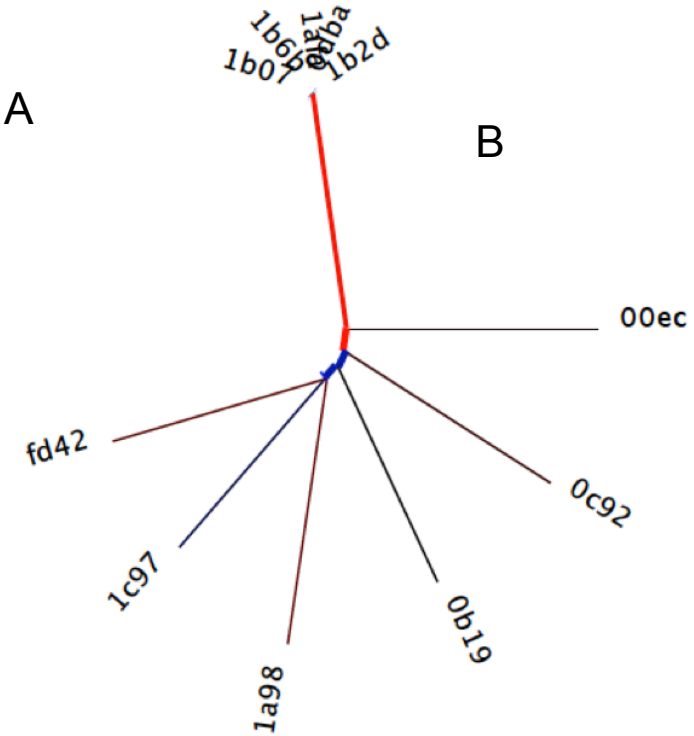
ignoring the operands. Each mnemonic is then mapped to a unique alphabet-pair, e.g. `mov` = MO, `push` = PH, `jmp` = JM. The resulting sequence is thus `PHMOMOJM`.

Phylogenetic tree applications in computer science

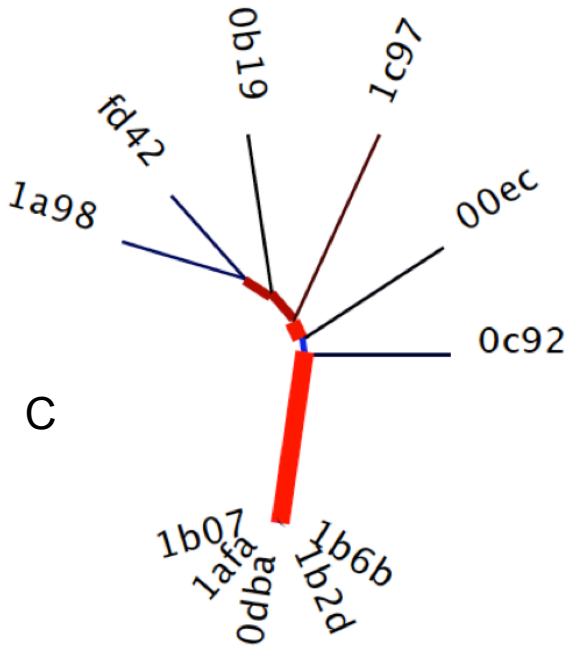
Sequence alignment (dbg: with debugging symbols, def: default settings, spd: optimised for speed). (a) Before alignment. (b) After alignment using an identity substitution matrix. (c) After alignment using a substitution matrix



A



B



C

Distance algorithm in computer science

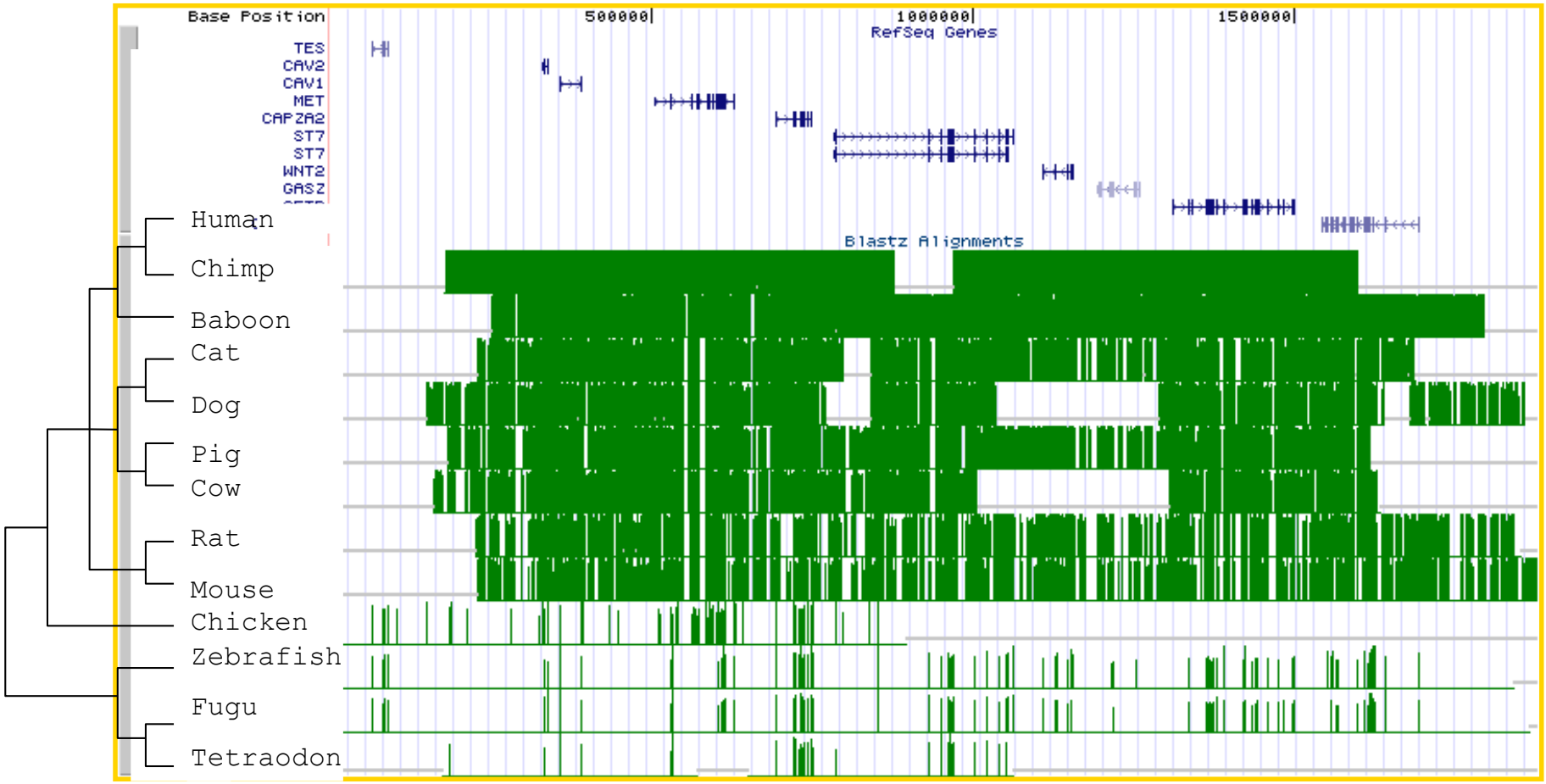
A) A sequence logo for the FakeAV-DO function “ F1 ”. Positions with large characters indicate invariant parts of the function; positions with small characters vary due to code metamorphism

B) A neighbour joining tree of FakeAV-DO set of procedures F1.

C) Neighbor joining tree of FakeAV-DO set of procedures F2 from
the same samples of B.

(W.M. Khoo and P. Lio’ Unity in diversity: Phylogenetic-inspired techniques for reverse engineering and detection of malware families. 2011 First SysSec Workshop)

More species increases power to detect conserved sequence elements: the phylogeny becomes a weight



Data from Eric Green at NGHRI, alignments by Webb Miller

Generalizing Pairwise to Multiple Alignment

- Alignment of 2 sequences is a 2-row matrix.
- Alignment of 3 sequences is a 3-row matrix

```
A T - G C G -
A - C G T - A
A T C A C - A
```

- Our scoring function should score alignments with conserved columns higher.

Alignments = Paths in 3-D

- Alignment of ATGC, AATC, and ATGC

0	1	1	2	3	4
	A	--	T	G	C

0	1	2	3	3	4
	A	A	T	--	C

	--	A	T	G	C

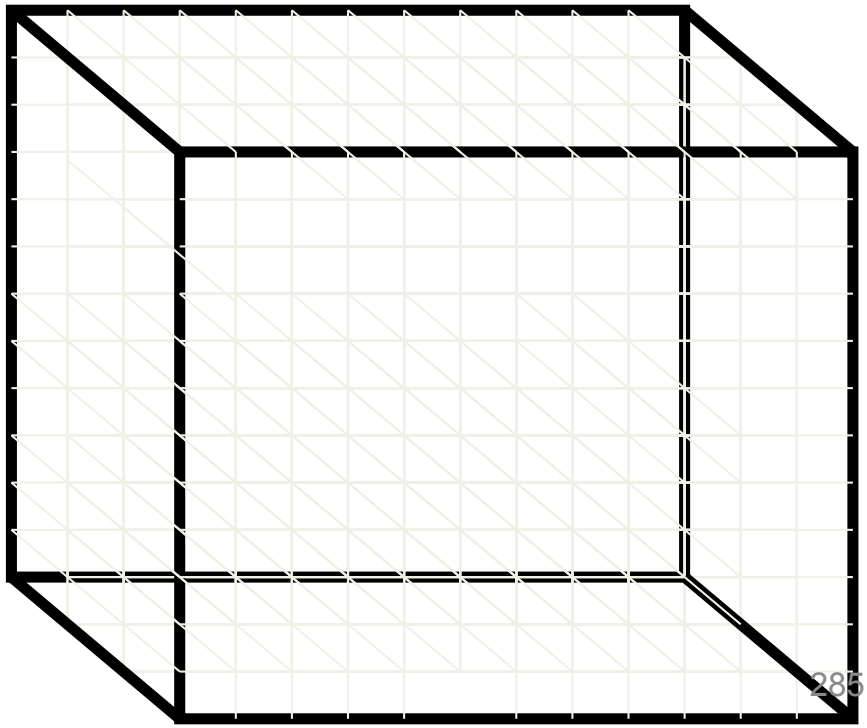
#symbols up to a given position

Alignments = Paths in 3-D

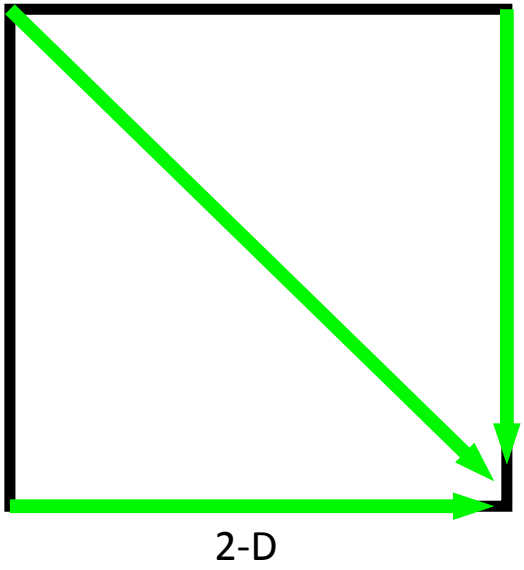
- Alignment of ATGC, AATC, and ATGC

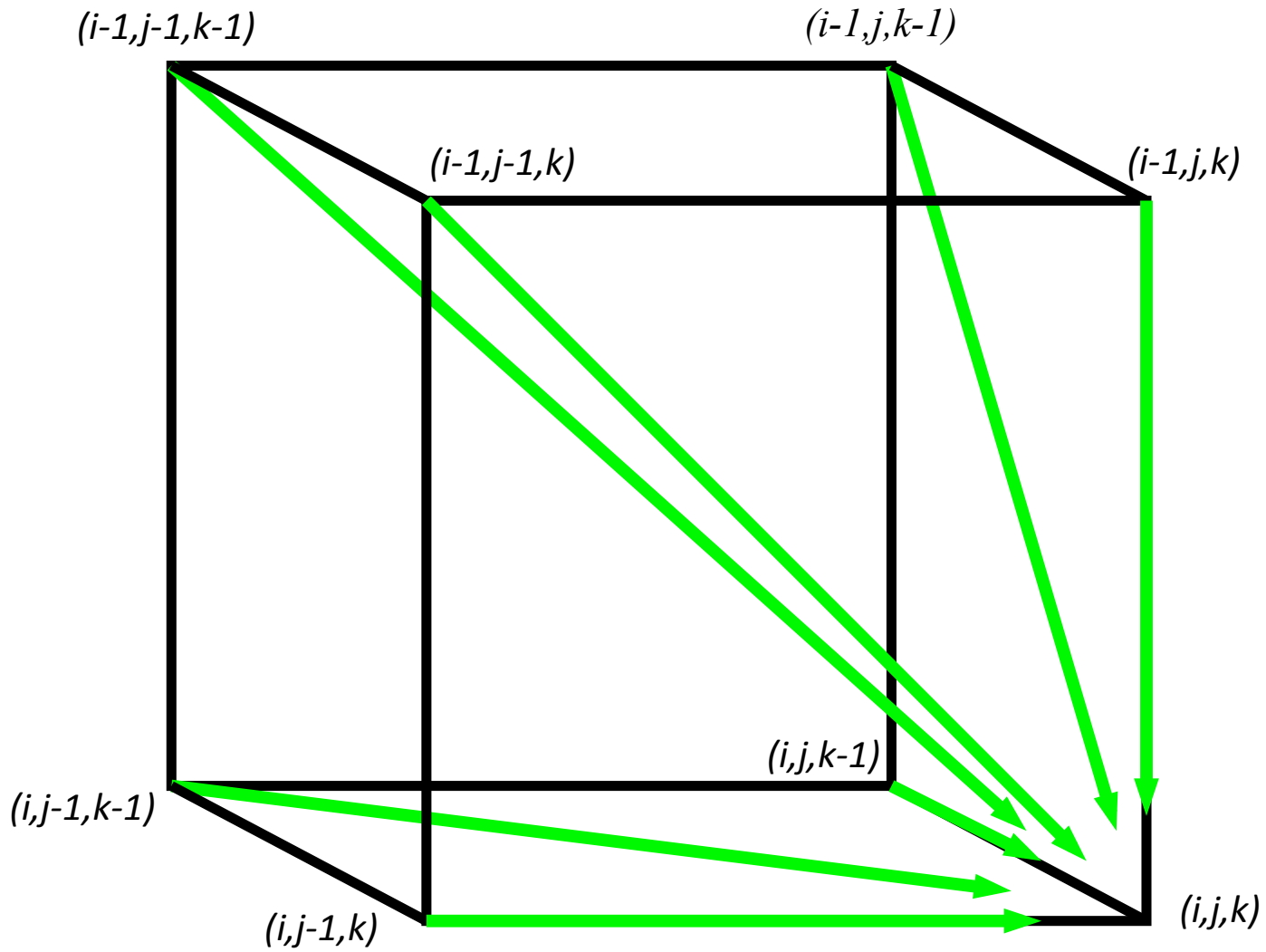
(0,0,0)→(1,1,0)→(1,2,1) →(2,3,2) →(3,3,3) →(4,4,4)

0	1	1	2	3	4
	A	--	T	G	C
0	1	2	3	3	4
	A	A	T	--	C
0	0	1	2	3	4
	--	A	T	G	C



2-D Alignment Cell versus 3-D Alignment Cell





Multiple Alignment: Dynamic Programming

$$s_{i,j,k} = \max\begin{cases} s_{i-1,j-1,k-1} + \delta(v_i, w_j, u_k) \\ s_{i-1,j-1,k} + \delta(v_i, w_j, -) \\ s_{i-1,j,k-1} + \delta(v_i, -, u_k) \\ s_{i,j-1,k-1} + \delta(-, w_j, u_k) \\ s_{i-1,j,k} + \delta(v_i, -, -) \\ s_{i,j-1,k} + \delta(-, w_j, -) \\ s_{i,j,k-1} + \delta(-, -, u_k) \end{cases}$$

- $\delta(x, y, z)$ is an entry in the 3-D scoring matrix.

Multiple Alignment: Running Time

- For 3 sequences of length n, the run time is proportional to $7n^3$
- For a k-way alignment, build a k-dimensional Manhattan graph with
  - n^k nodes
  - most nodes have $2^k - 1$ incoming edges.
  - Runtime: $O(2^k n^k)$

Multiple Alignment Induces Pairwise Alignments

Every multiple alignment induces pairwise alignments:

```
A C - G C G G - C
A C - G C - G A G
G C C G C - G A G
```

```
ACGCGG-C    AC-GCGG-C    AC-GCGAG
ACGC-GAC    GCCGC-GAG    GCCGCGAG
```

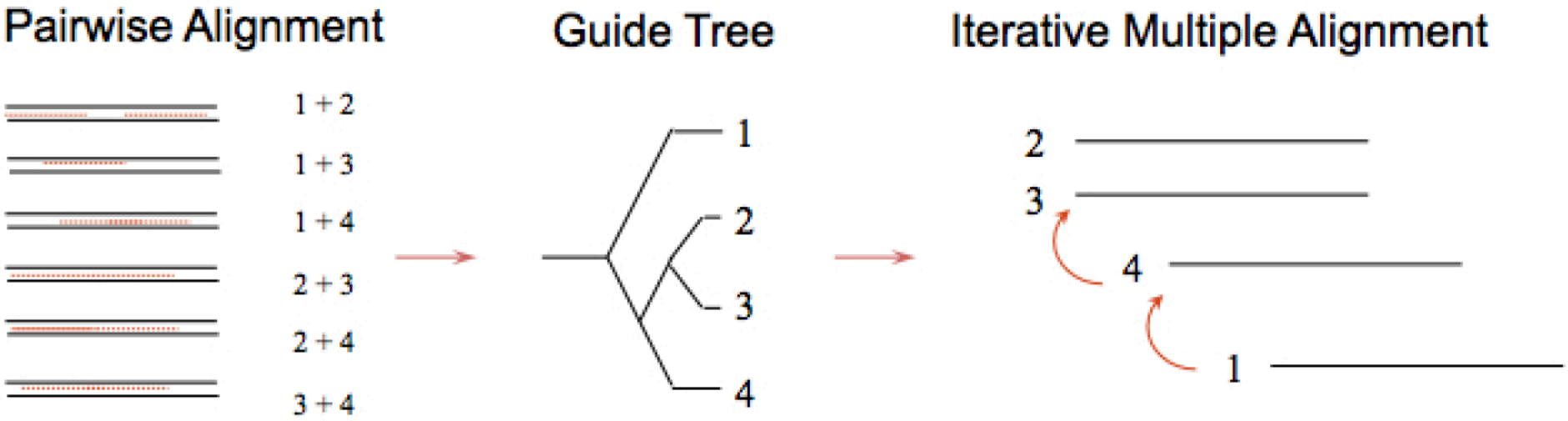
Idea: Construct Multiple from Pairwise Alignments

Given a set of **arbitrary** pairwise alignments, can we construct a multiple alignment that induces them?

```
AAAATTTT----        ----AAAATTTT        TTTTGGGG----
----TTTTGGGG        GGGGAAAA----        ----GGGGAAAA
```

Progressive alignment

Progressive alignment methods are heuristic in nature. They produce multiple alignments from a number of pairwise alignments. Perhaps the most widely used algorithm of this type is the software CLUSTAL (https://www.ebi.ac.uk/Tools/msa/clustalo/)



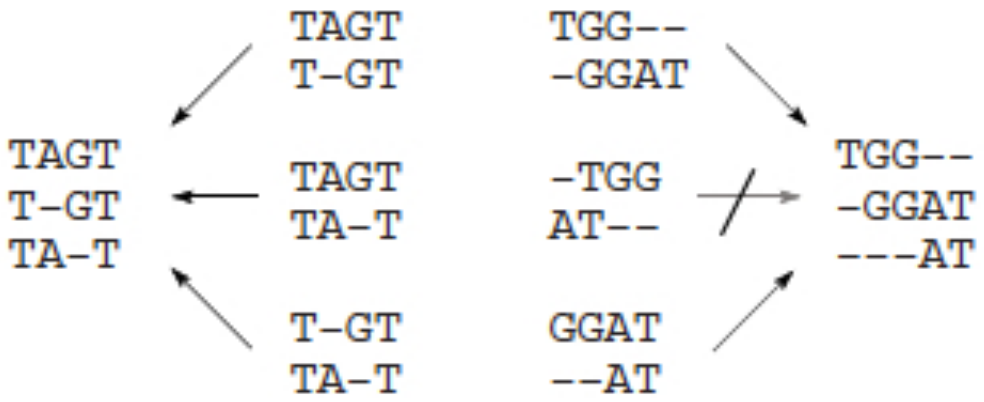
Progressive Alignment

Clustalw:

1. Given N sequences, align each sequence against each other.
2. Use the score of the pairwise alignments to compute a distance matrix.
3. Build a guide tree (tree shows the best order of progressive alignment).
4. Progressive Alignment guided by the tree.

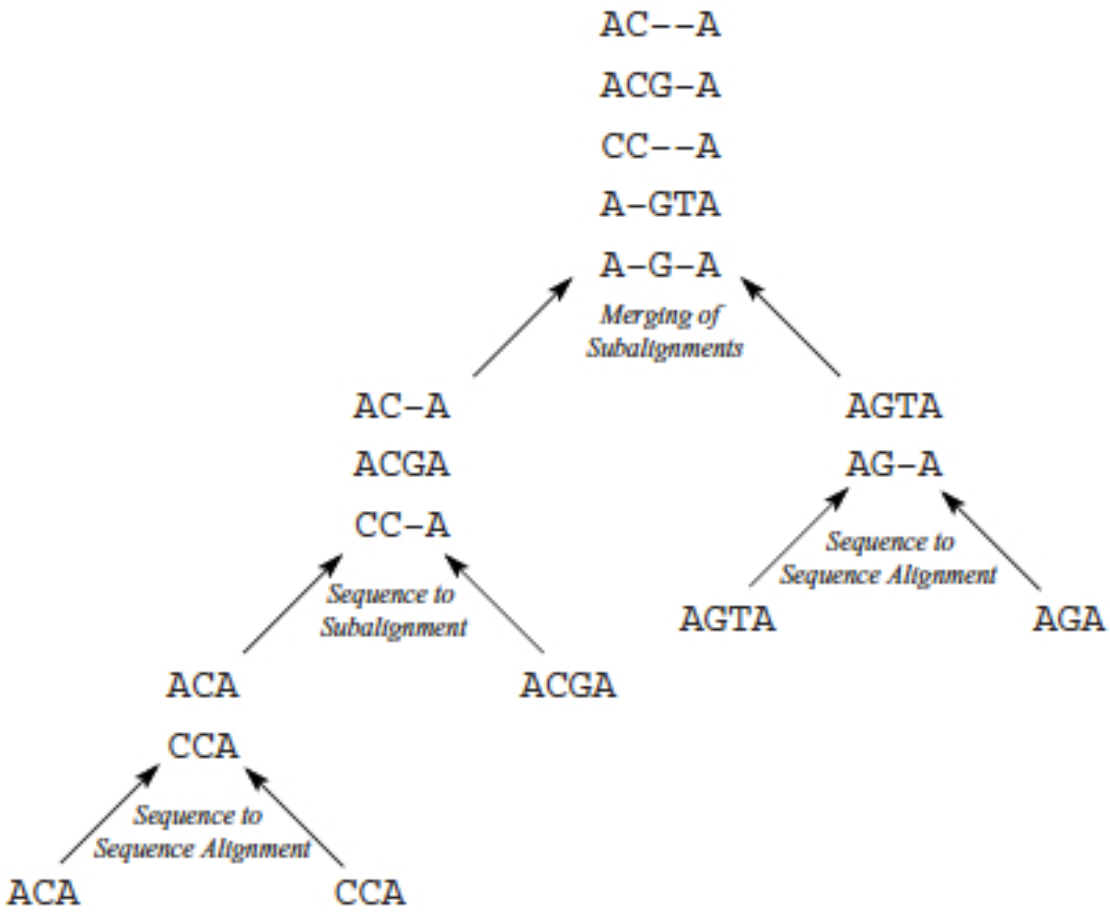
Progressive Alignment

Not all the pairwise alignments build well into a multiple sequence alignment (compare the alignments on the left and right)

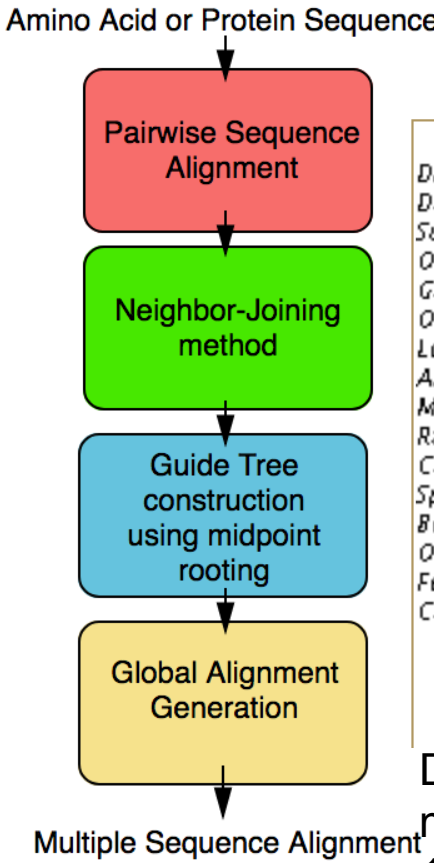


Progressive Alignment

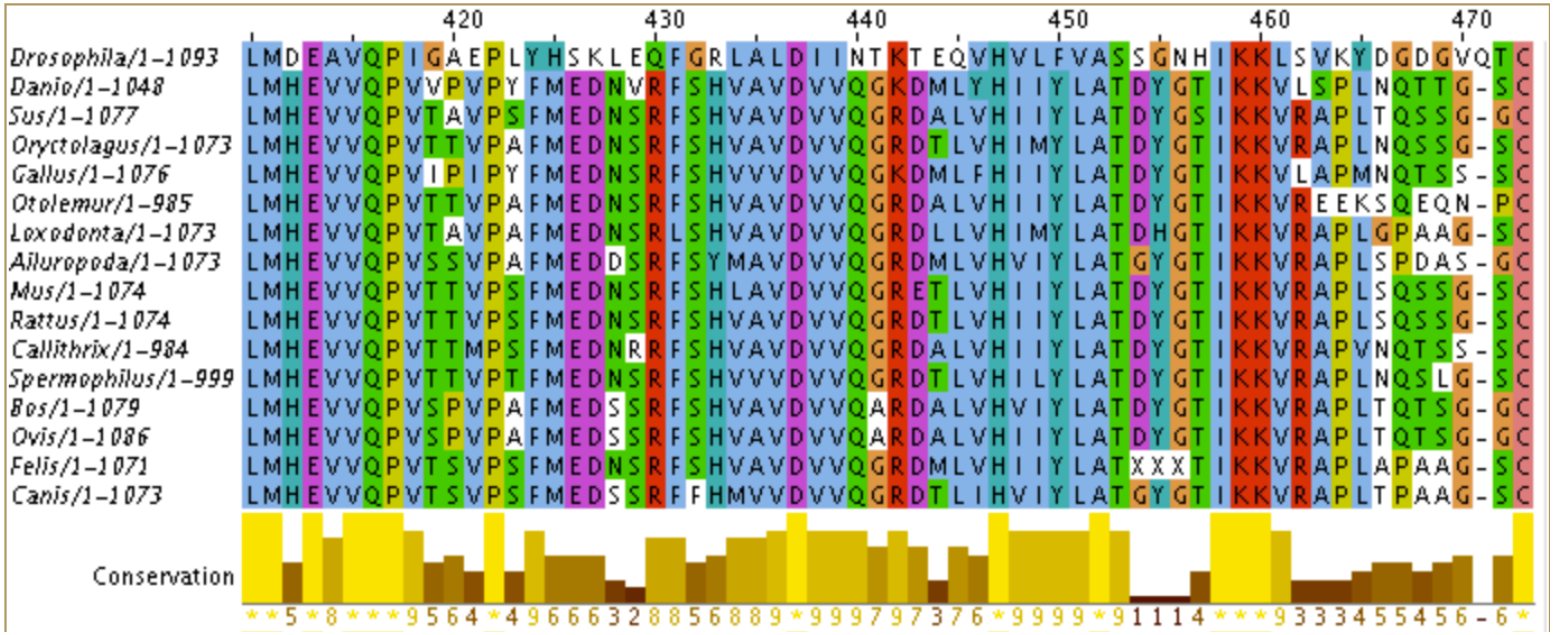
The progressive alignment builds a final alignment by merging sub-alignments (bottom to top) with a guide tree



Progressive alignment (Clustal). Input: a set of sequences in Fasta format (also thousands).
Output: alignment of the set of sequences: multi sequence alignment (MSA). Interest: find conserved patterns (across sequences, i.e. columns retaining similar patterns) may indicate functional constraints. In other words, if the same pattern is conserved in multiple sequences from different species, the substring could have an important functional role.
Main question in this lecture: how similar is this group of sequences?

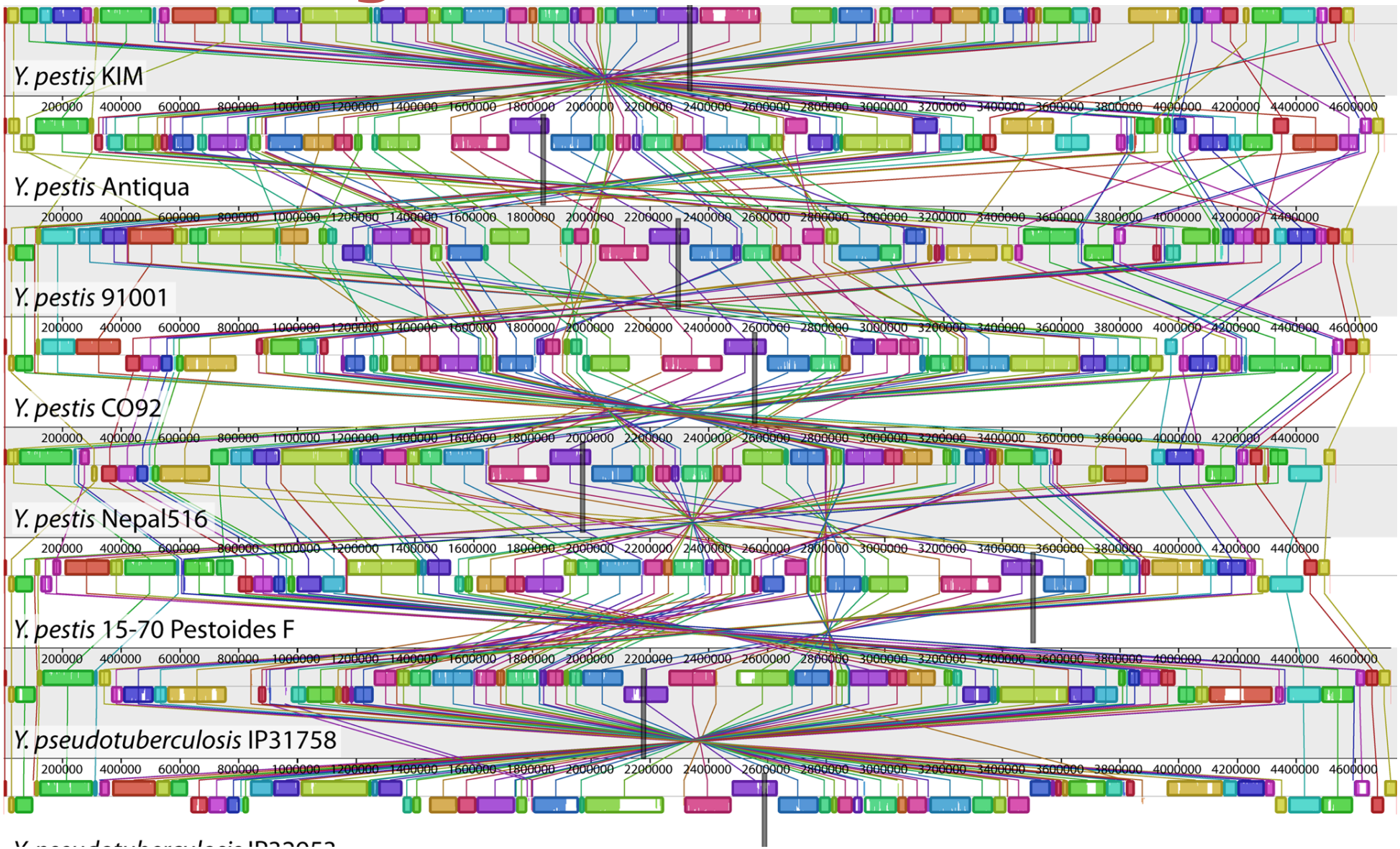


from wikipedia



D.G. Higgins, J.D. Thompson, and T.J. Gibson. Using CLUSTAL for multiple sequence alignments. Methods in Enzymology, 266:383402, 1996.

Example of complexity in alignment: bacterial genomes



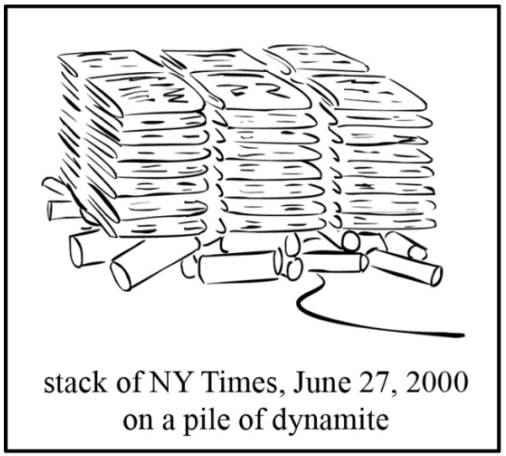
Genome Sequencing

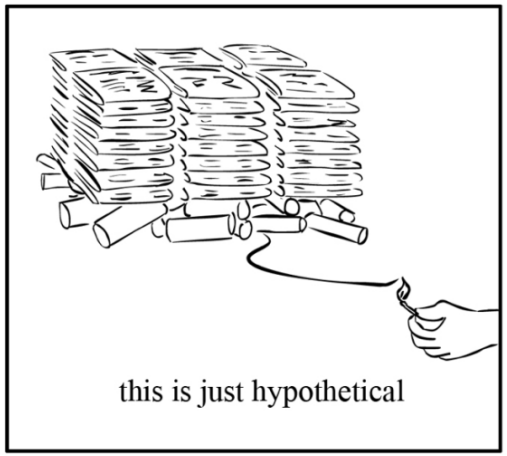
- What Is Genome Sequencing: Exploding Newspapers analogy
- The String Reconstruction Problem
- String Reconstruction as a Hamiltonian Path Problem
- String Reconstruction as an Eulerian Path Problem
- De Bruijn Graphs
- Euler's Theorem
- Assembling Read-Pairs
- De Bruijn Graphs Face Harsh Realities of Assembly

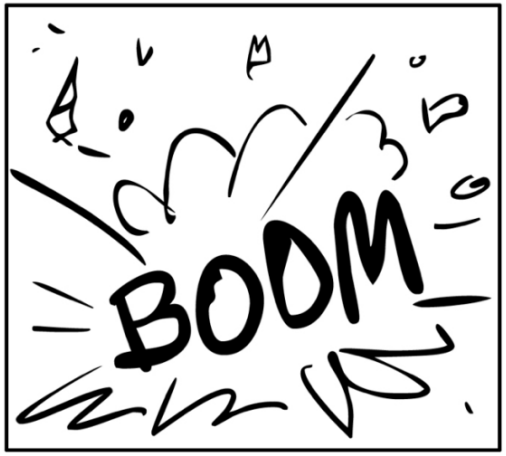
Why Do We Sequence Personal Genomes?

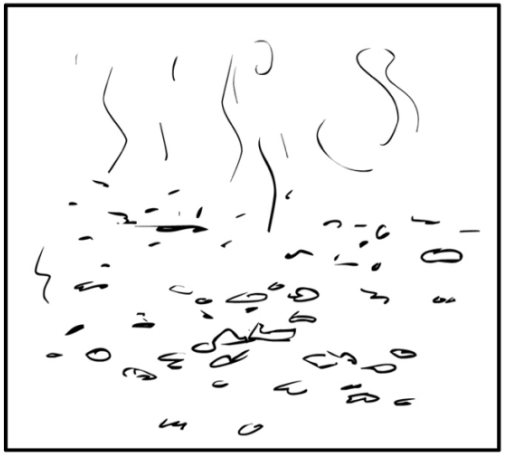
- **2010:** Nicholas Volker became the first human being to be saved by genome sequencing.
  - Doctors could not diagnose his condition; he went through dozens of surgeries.
  - Sequencing revealed a rare mutation in a *XIAP* gene linked to a defect in his immune system.
  - This led doctors to use immunotherapy, which saved the child.
- Different people have slightly different genomes: on average, roughly 1 mutation in 1000 nucleotides.

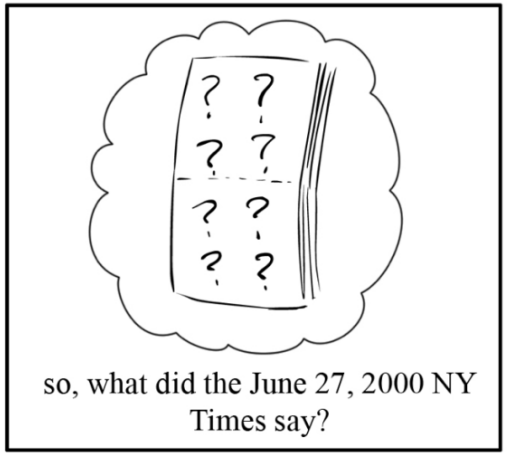
The Newspaper Problem



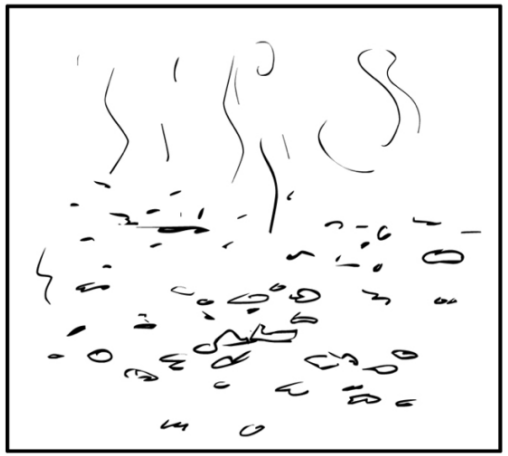


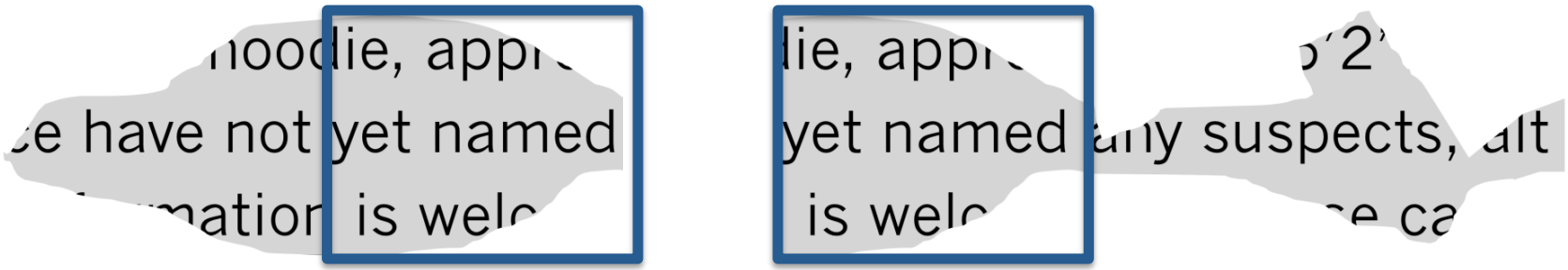






The Newspaper Problem as an Overlapping Puzzle





The Newspaper Problem as an Overlapping Puzzle

Multiple Copies of a Genome (Millions of them)

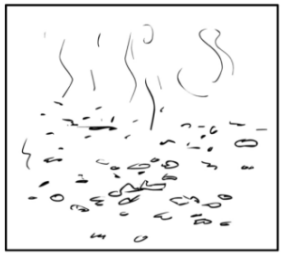
CTGATGATGGACTACGCTACTACTGCTAGCTGTATTACGATCAGCTACCACATCGTAGCTACGATGCATTAGCAAGCTATCGGATCAGCTACCACATCGTAGC
CTGATGATGGACTACGCTACTACTGCTAGCTGTATTACGATCAGCTACCACATCGTAGCTACGATGCATTAGCAAGCTATCGGATCAGCTACCACATCGTAGC
CTGATGATGGACTACGCTACTACTGCTAGCTGTATTACGATCAGCTACCACATCGTAGCTACGATGCATTAGCAAGCTATCGGATCAGCTACCACATCGTAGC
CTGATGATGGACTACGCTACTACTGCTAGCTGTATTACGATCAGCTACCACATCGTAGCTACGATGCATTAGCAAGCTATCGGATCAGCTACCACATCGTAGC

Breaking the Genomes at Random Positions

Generating “Reads”

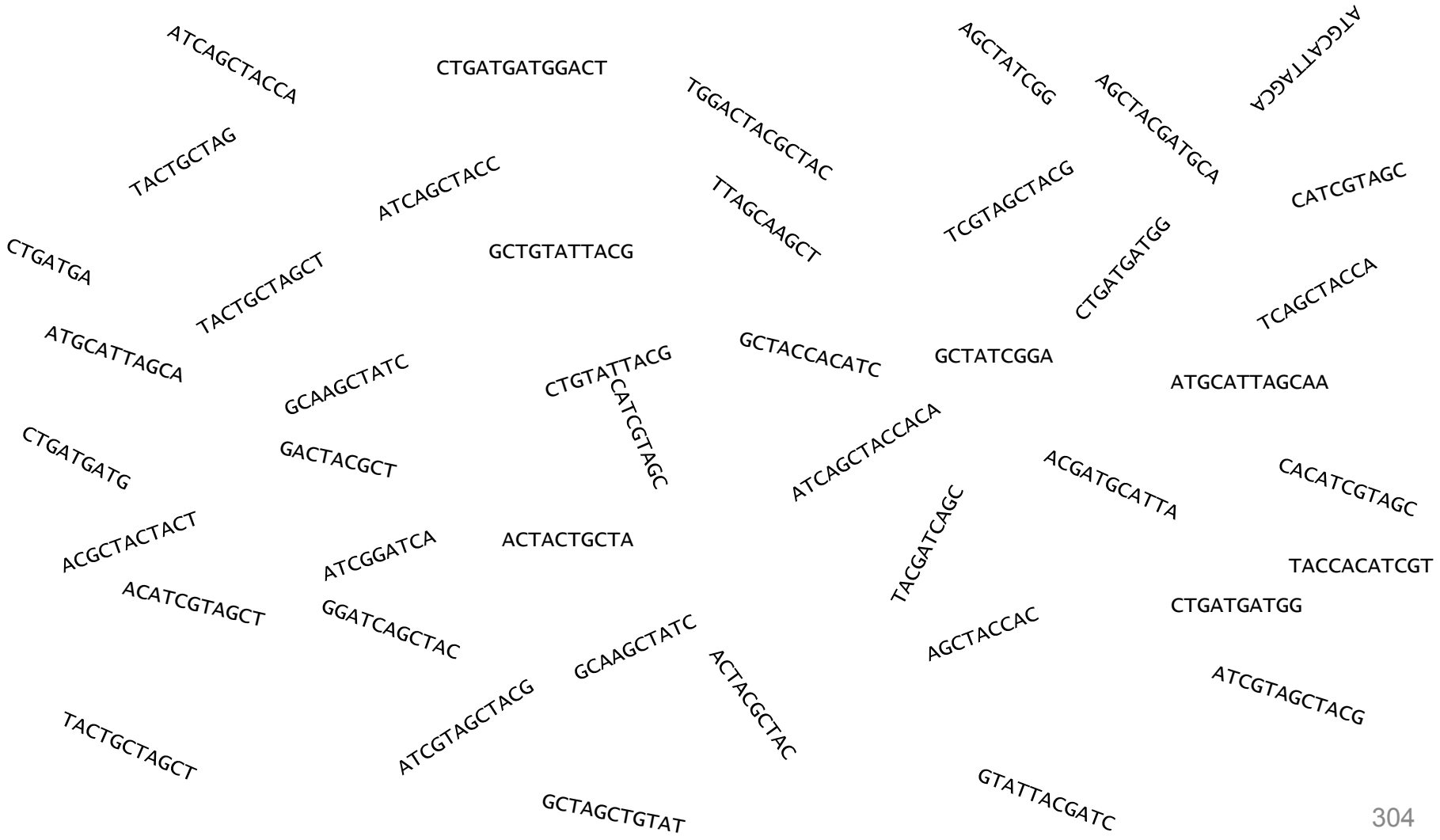
CTGATGA TGGACTACGCTAC TACTGCTAG CTGTATTACG ATCAGCTACCACA TCGTAGCTACG ATGCATTAGCAA GCTATCGGA TCAGCTACCA CATCGTAGC
CTGATGATG GACTACGCT ACTACTGCTA GCTGTATTACG ATCAGCTACC ACATCGTAGCT ACGATGCATTA GCAAGCTATC GGATCAGCTAC CACATCGTAGC
CTGATGATGG ACTACGCTAC TACTGCTAGCT GTATTACGATC AGCTACCAC ATCGTAGCTACG ATGCATTAGCA AGCTATCGG A TCAGCTACCA CATCGTAGC
CTGATGATGGACT ACGCTACTACT GCTAGCTGTAT TACGATCAGC TACCACATCGT AGCTACGATGCA TTAGCAAGCT ATCGGATCA GCTACCACATC GTAGC

“Burning” Some Reads

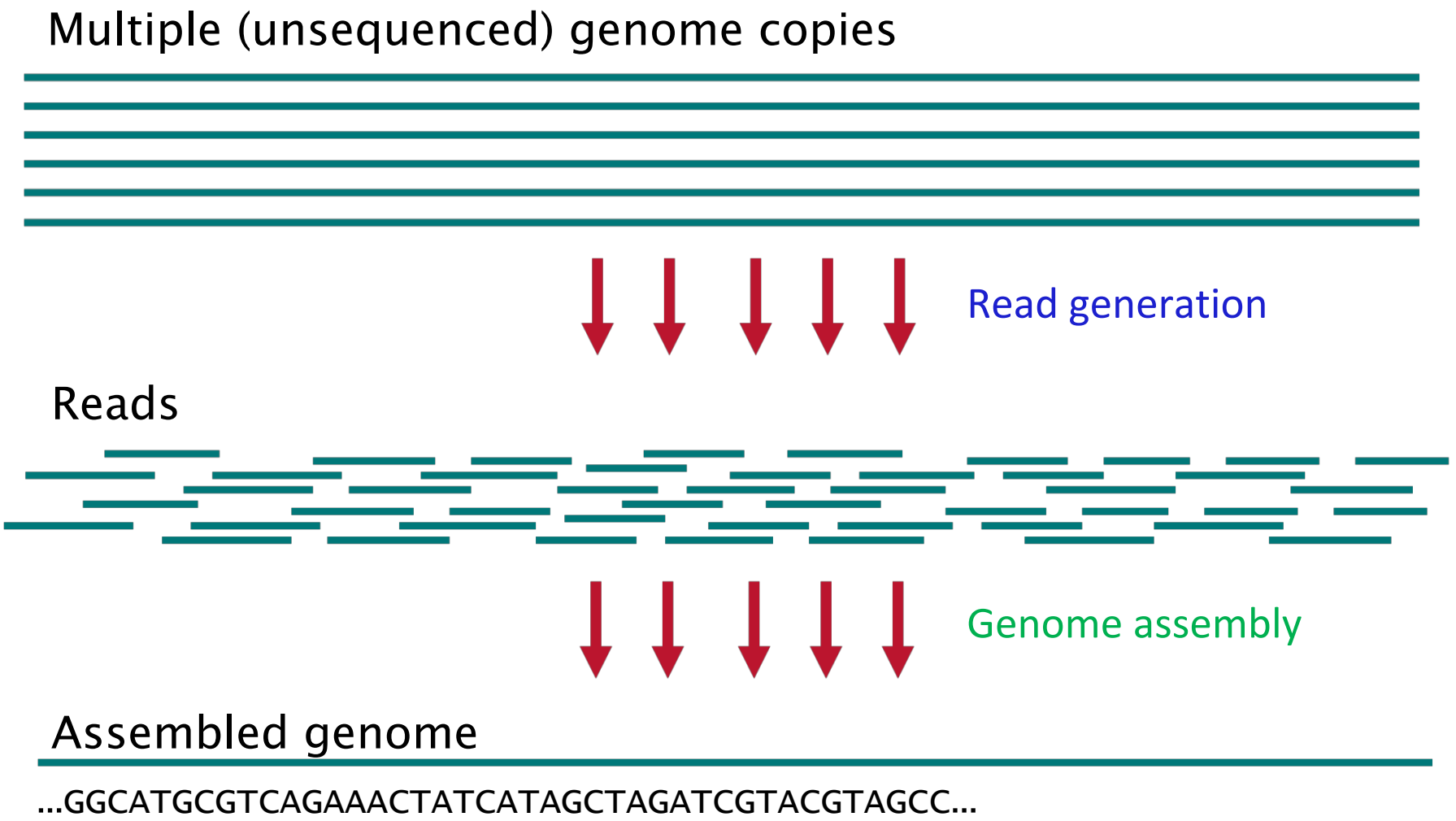


CTGATGA TGGACTACGCTAC TACTGCTAG CTGTATTACG ATCAGCTACCACA TCGTAGCTACG ATGCATTAGCAA GCTATCGGA TCAGCTACCA CATCGTAGC
CTGATGATG GACTACGCT ACTACTGCTA GCTGTATTACG ATCAGCTACC ACATCGTAGCT ACGATGCATTA GCAAGCTATC GGATCAGCTAC CACATCGTAGC
CTGATGATGG ACTACGCTAC TACTGCTAGCT GTATTACGATC AGCTACCAC ATCGTAGCTACG ATGCATTAGCA AGCTATCGG A TCAGCTACCA CATCGTAGC
CTGATGATGGACT ACGCTACTACT GCTAGCTGTAT TACGATCAGC TACCACATCGT AGCTACGATGCA TTAGCAAGCT ATCGGATCA GCTACCACATC GTAGC

No Idea What Position Every Read Comes From



From Experimental to Computational Challenges



What Makes Genome Sequencing Difficult?

- Modern sequencing machines cannot read an entire genome one nucleotide at a time from beginning to end (like we read a book)
- They can only shred the genome and generate short **reads**.
- The genome assembly is not the same as a jigsaw puzzle: we must use *overlapping* reads to reconstruct the genome, a giant **overlap puzzle**!

Genome Sequencing Problem. Reconstruct a genome from reads.

- Input. A collection of strings Reads.
- Output. A string Genome reconstructed from Reads.

What Is k-mer Composition?

***Composition*$_3$(TAATGCCATGGGATGTT)=**

TAA
AAT
ATG
TGC
GCC
CCA
CAT
ATG
TGG
GGG
GGA
GAT
ATG
TGT
GTT

k-mer Composition

$Composition_3$(TAATGCCATGGGATGTT) =

TAA AAT ATG TGC GCC CCA CAT ATG TGG GGG GGA GAT ATG TGT GTT

=

AAT ATG ATG ATG CAT CCA GAT GCC GGA GGG GTT TAA TGC TGG TGT

e.g., lexicographic order (like in a dictionary)

Reconstructing a String from its Composition

String Reconstruction Problem. Reconstruct a string from its k-mer composition.

- Input. A collection of k-mers.
- Output. A Genome such that $\text{Composition}_k(\text{Genome})$ is equal to the collection of k-mers.

A Naive String Reconstruction Approach

ATG ATG ATG CAT CCA GAT GCC GGA GGG GTT TGC TGG TGT

TAA
AAT

ATG ATG CAT CCA GAT GCC GGA GGG TGC TGG

TAA
AAT
ATG
TGT
GTT

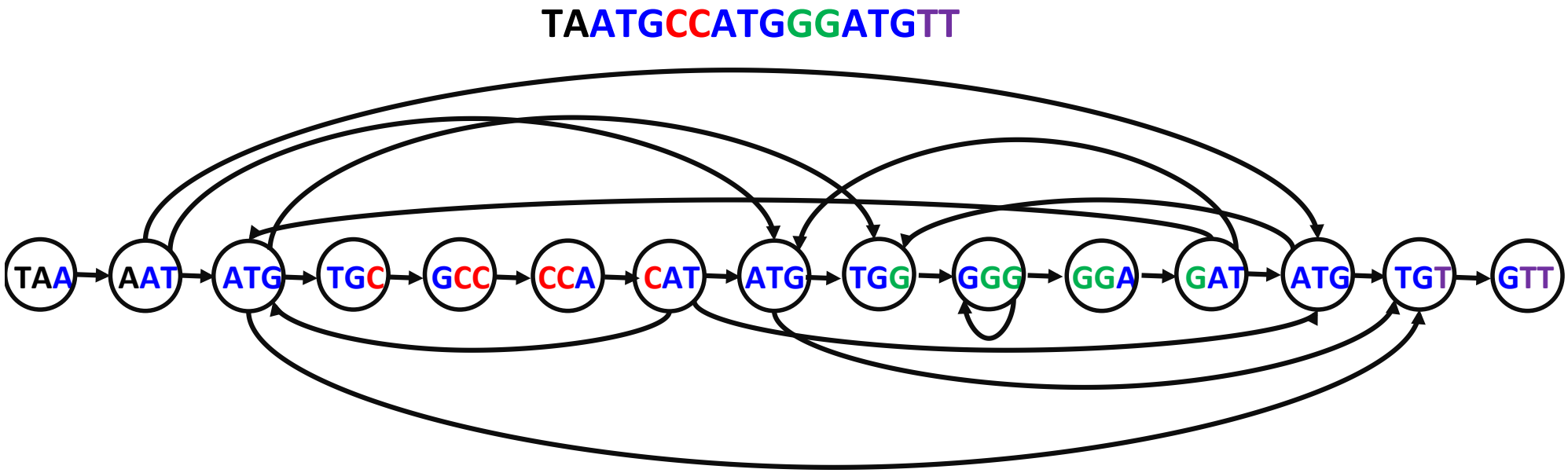
Representing a Genome as a Path

$\text{Composition}_3(\text{TAATGCCATGGGATGTT}) =$

Can we construct this genome path without knowing the genome TAATGCCATGGGATGTT, only from its composition?

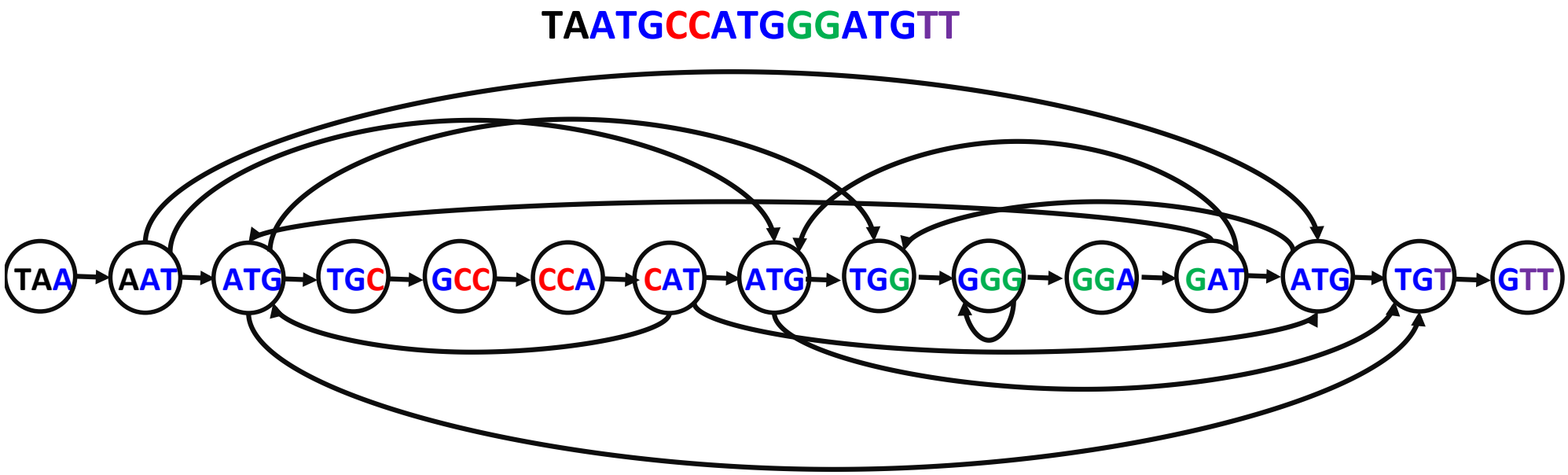
Yes. We simply need to connect k-mer_1 with k-mer_2 if suffix(k-mer_1)=prefix(k-mer_2).
E.g. TAA → AAT

A Path Turns into a Graph



Yes. We simply need to connect k-mer$_1$ with k-mer$_2$ if suffix(k-mer$_1$)=prefix(k-mer$_2$).
E.g. TAA → AAT

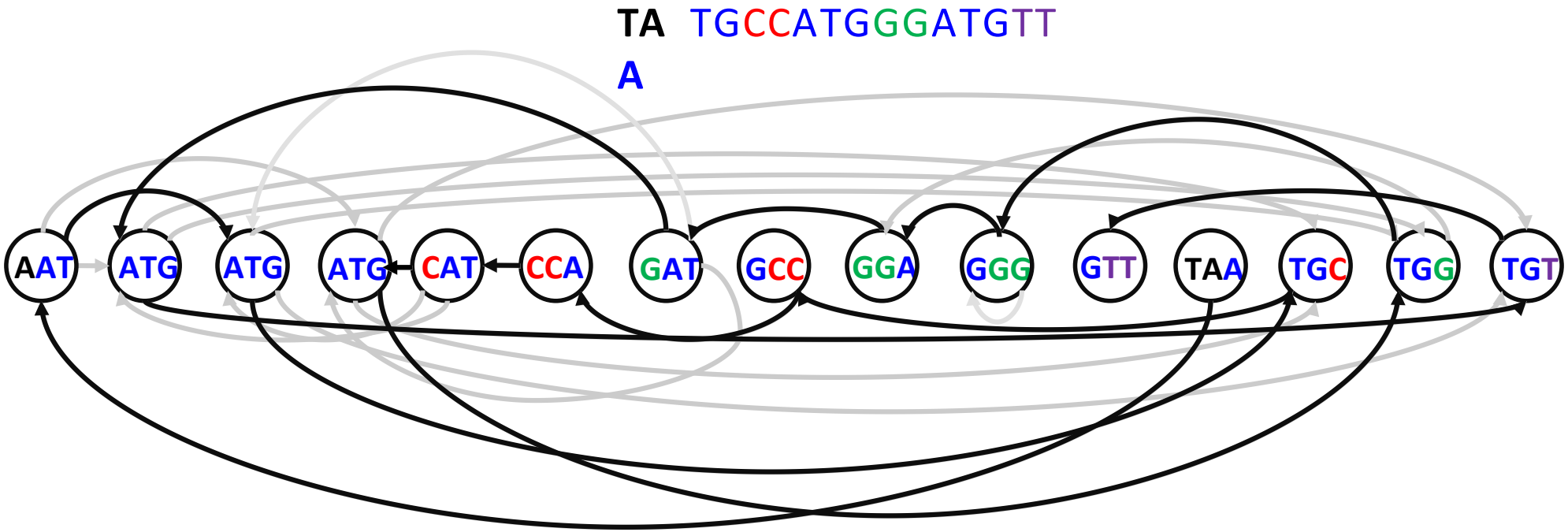
A Path Turns into a Graph



Can we still find the genome path in this graph?

Where Is the Genomic Path?

A **Hamiltonian path:** a path that visits each node in a graph exactly once.



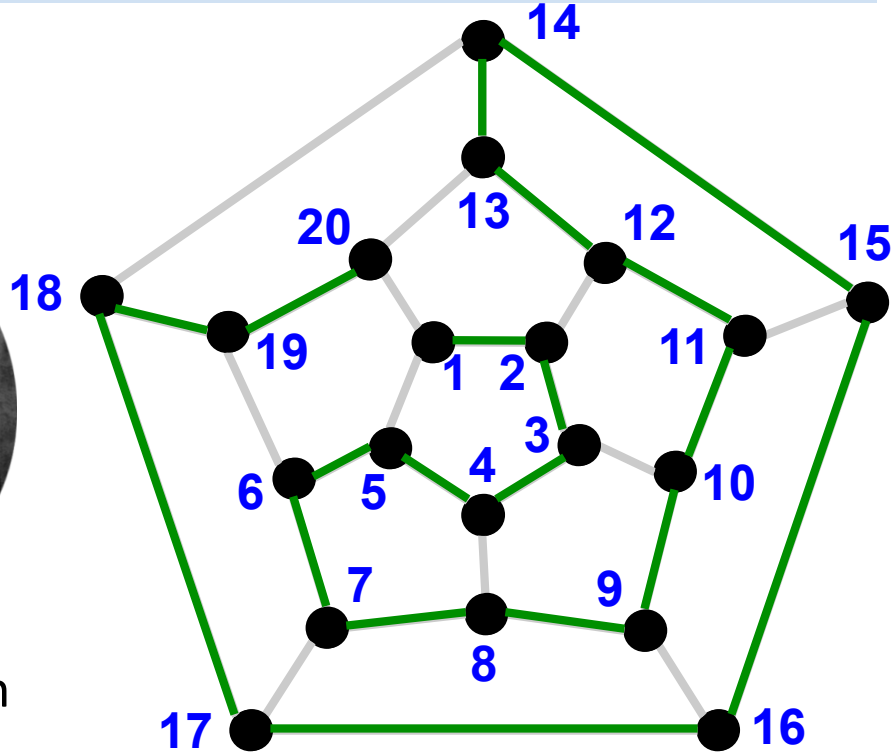
What are we trying to find in this graph?

Does This Graph Have a Hamiltonian Path?

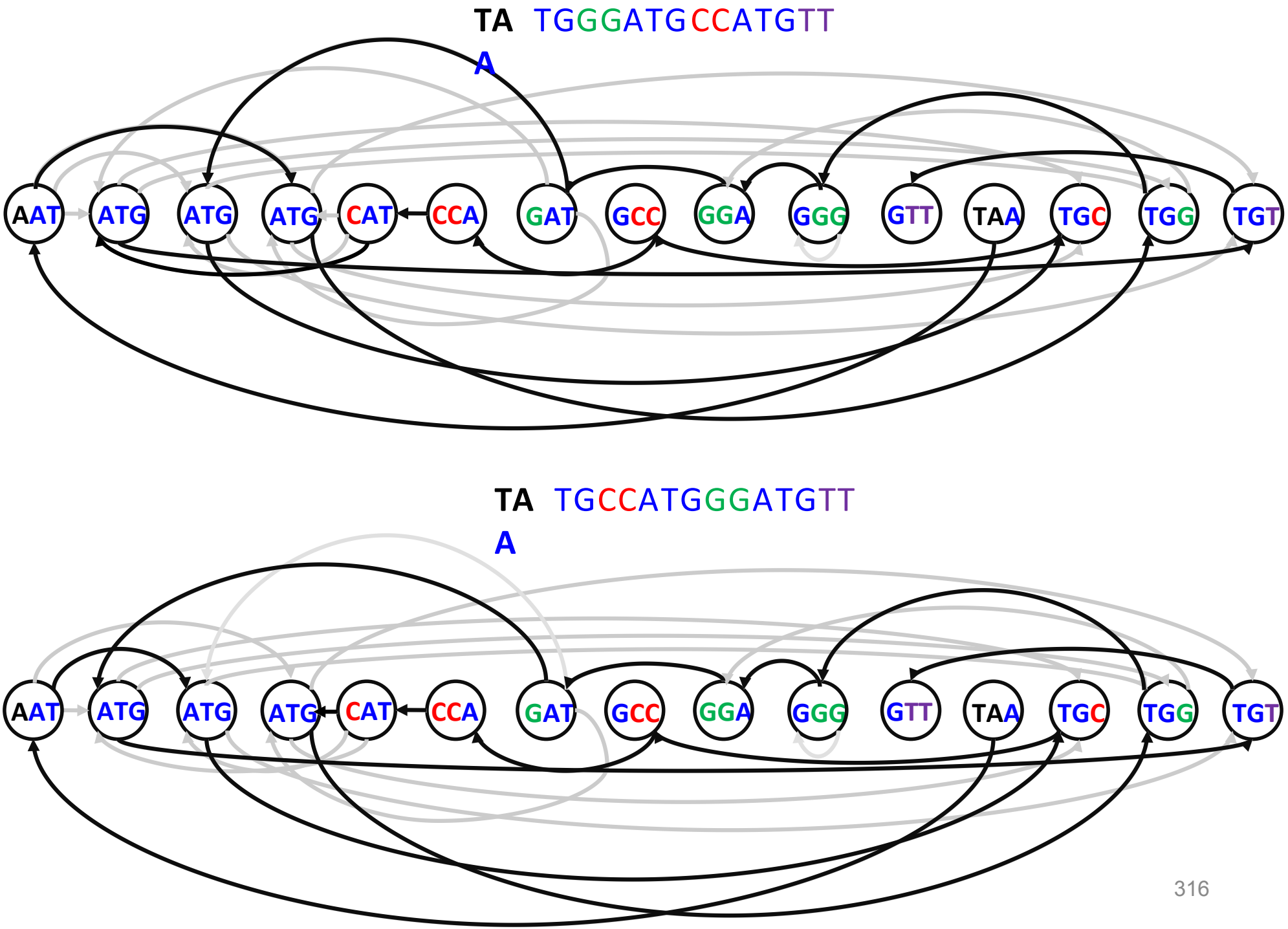
Hamiltonian Path Problem. Find a Hamiltonian path in a graph.
Input. A graph.
Output. A path visiting every node in the graph exactly once.

Icosian game (1857)

William Hamilton



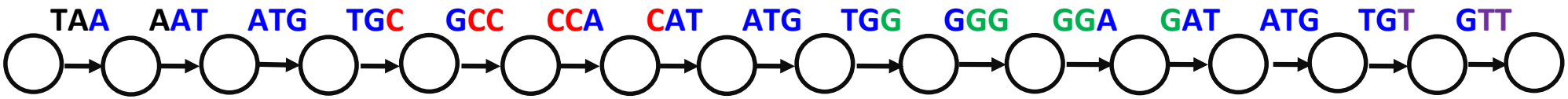
Undirected graph


TA TGGGATGCCATGTT
A
AAT
ATG
ATG
ATG
CAT
CCA
GAT
GCC
GGA
GGG
GTT
TAA
TGC
TGG
TGT
TA TGCCATGGGATGTT
A
AAT
ATG
ATG
ATG
CAT
CCA
GAT
GCC
GGA
GGG
GTT
TAA
TGC
TGG
TGT

A Slightly Different Path

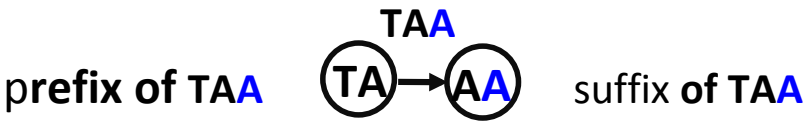
TAATGCCATGGGATGTT

3-mers as nodes



3-mers as edges

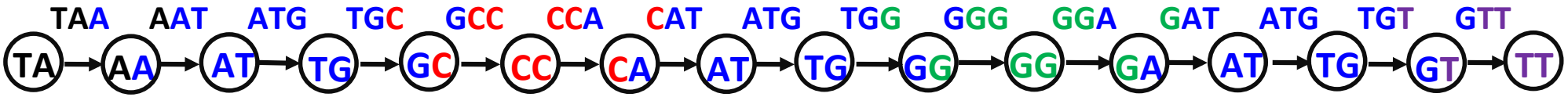
How do we label the starting and ending nodes of an edge?



Labeling Nodes in the New Path

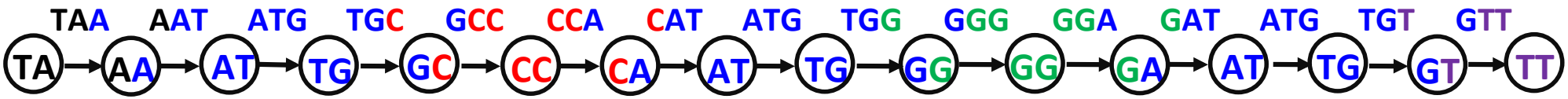
TAATGCCATGGGATGTT

3-mers as nodes



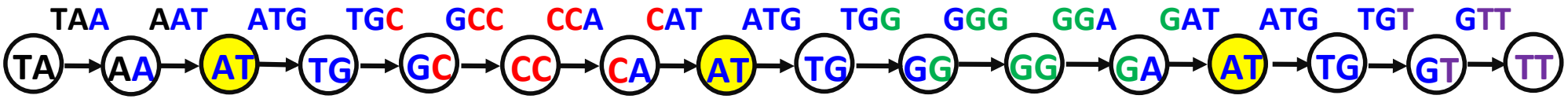
3-mers as edges and 2-mers as nodes

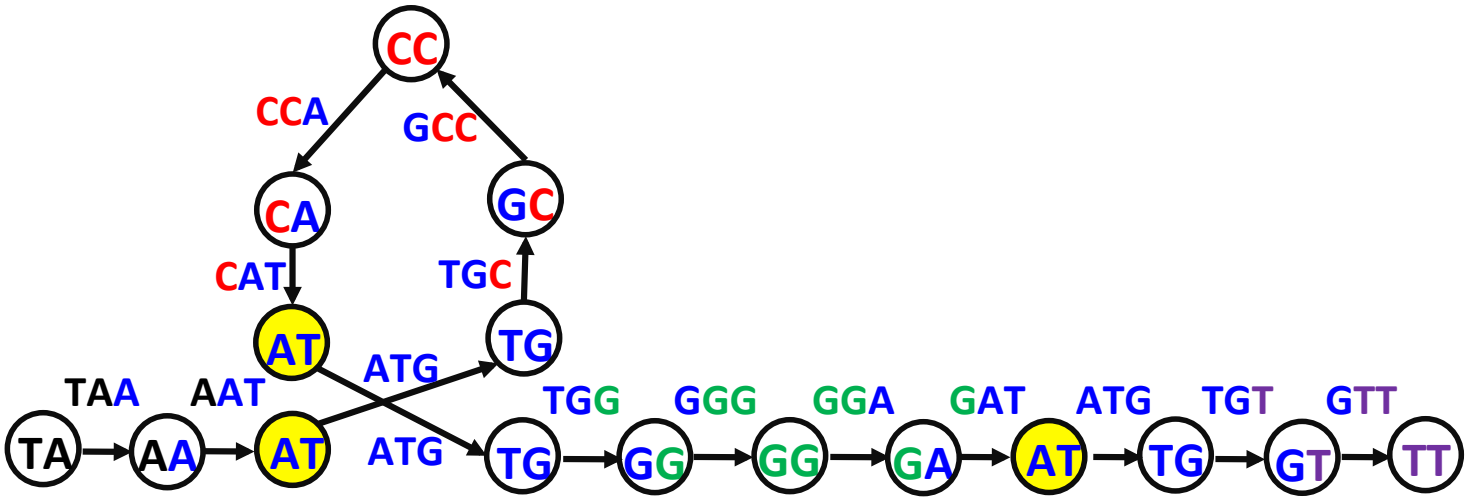
Labeling Nodes in the New Path



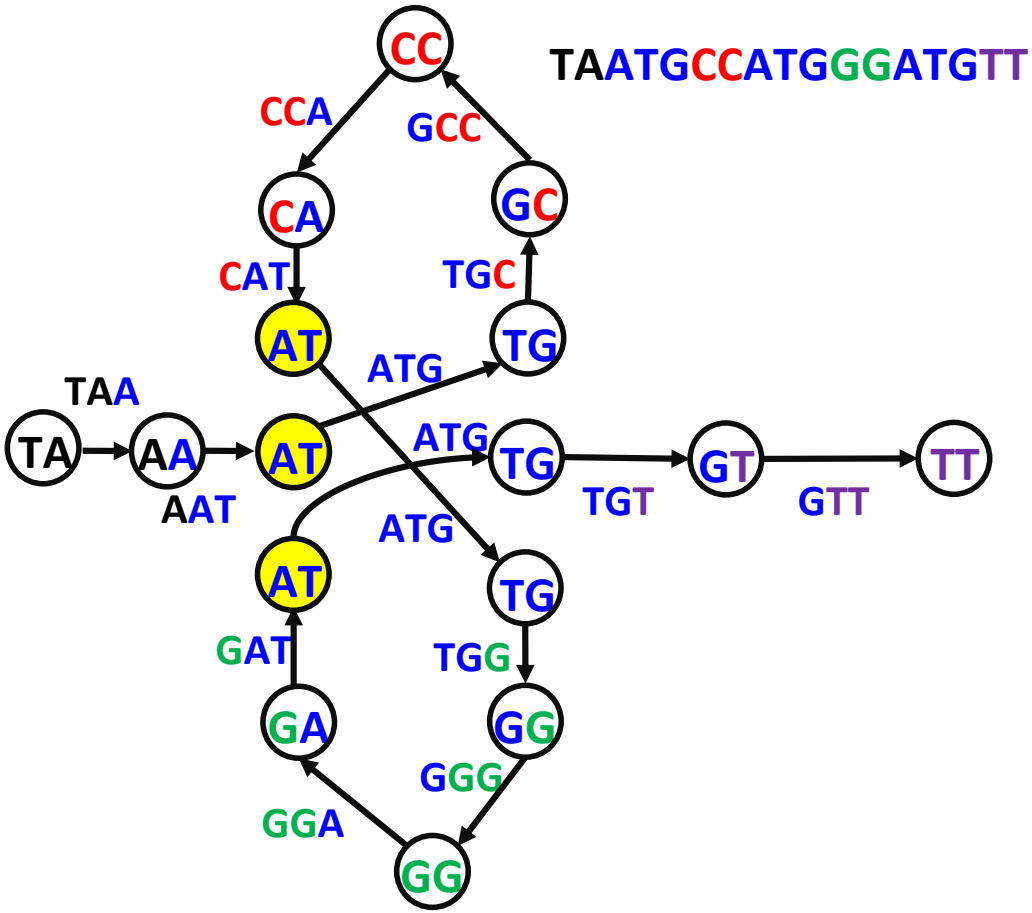
3-mers as edges and 2-mers as nodes

Gluing Identically Labeled Nodes

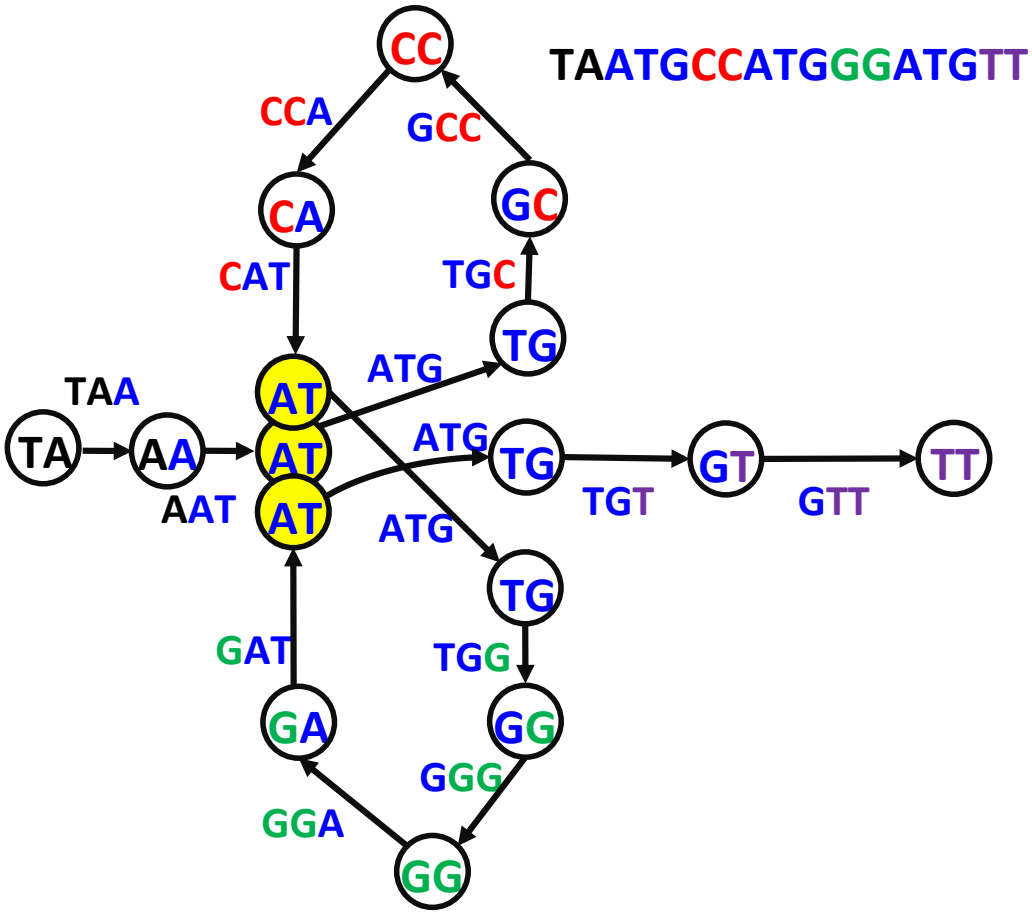




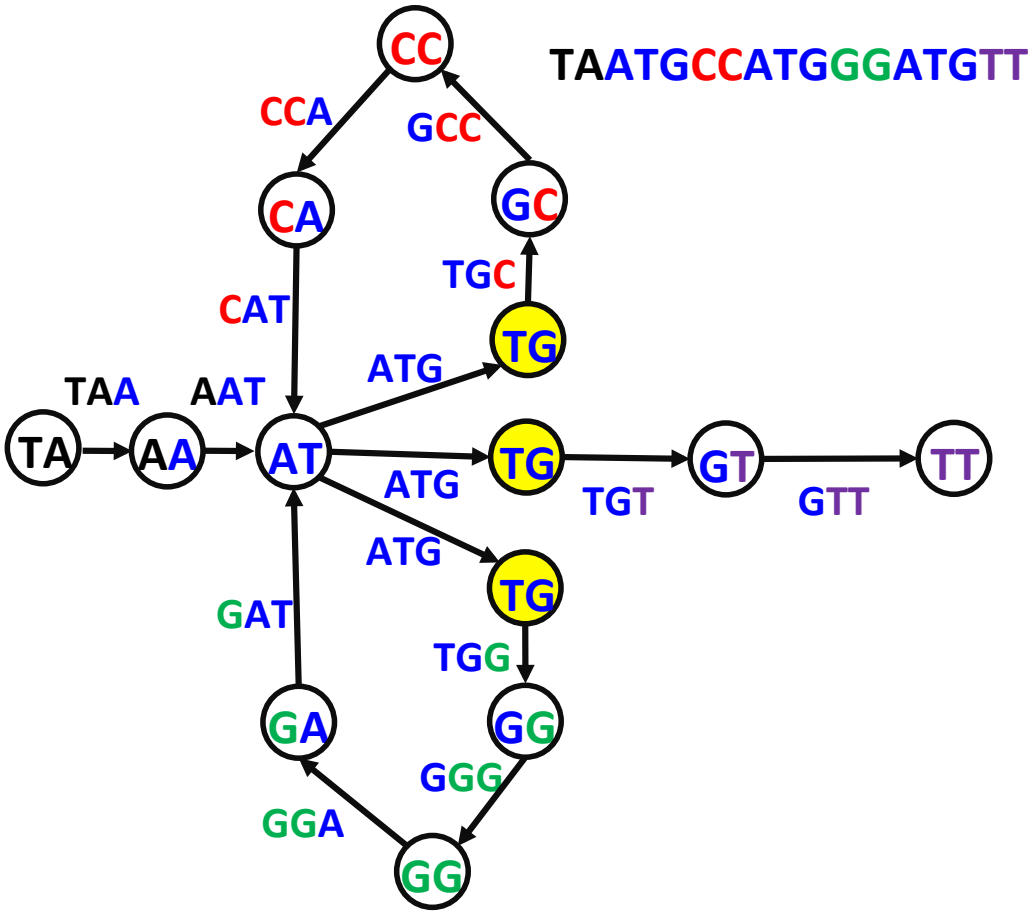
Gluing Identically Labeled Nodes



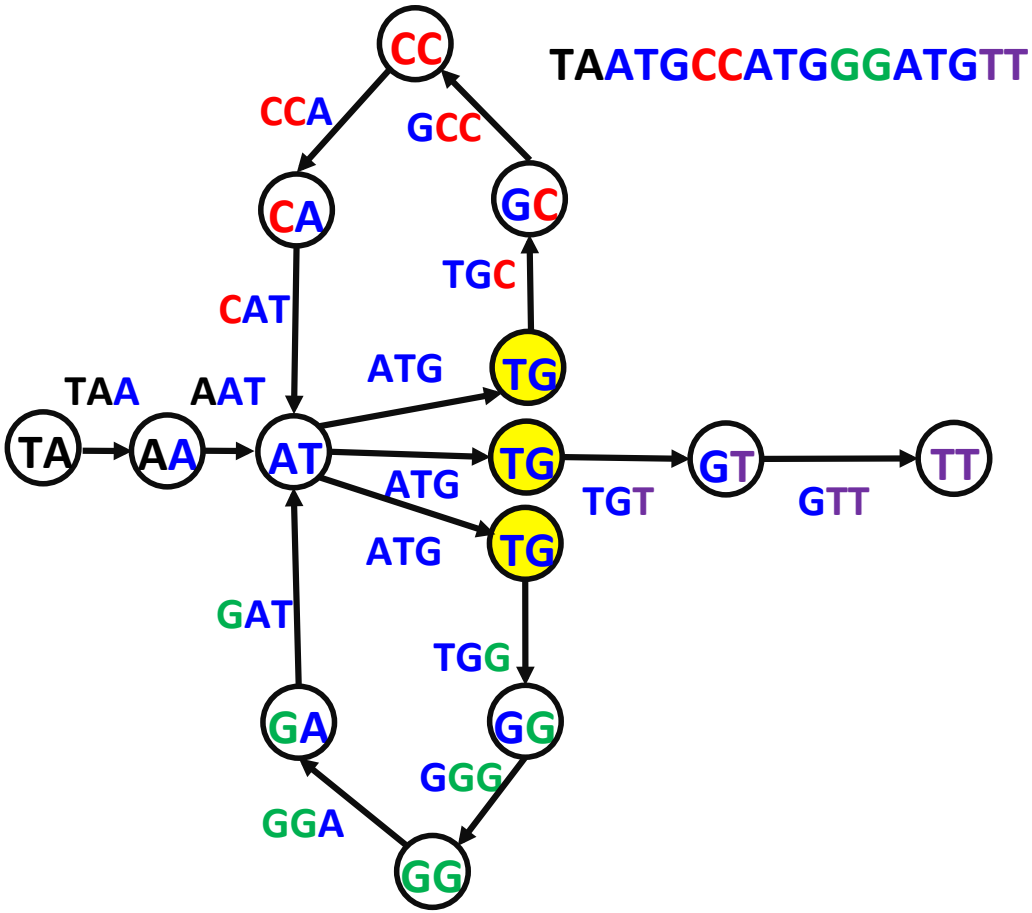
Gluing Identically Labeled Nodes



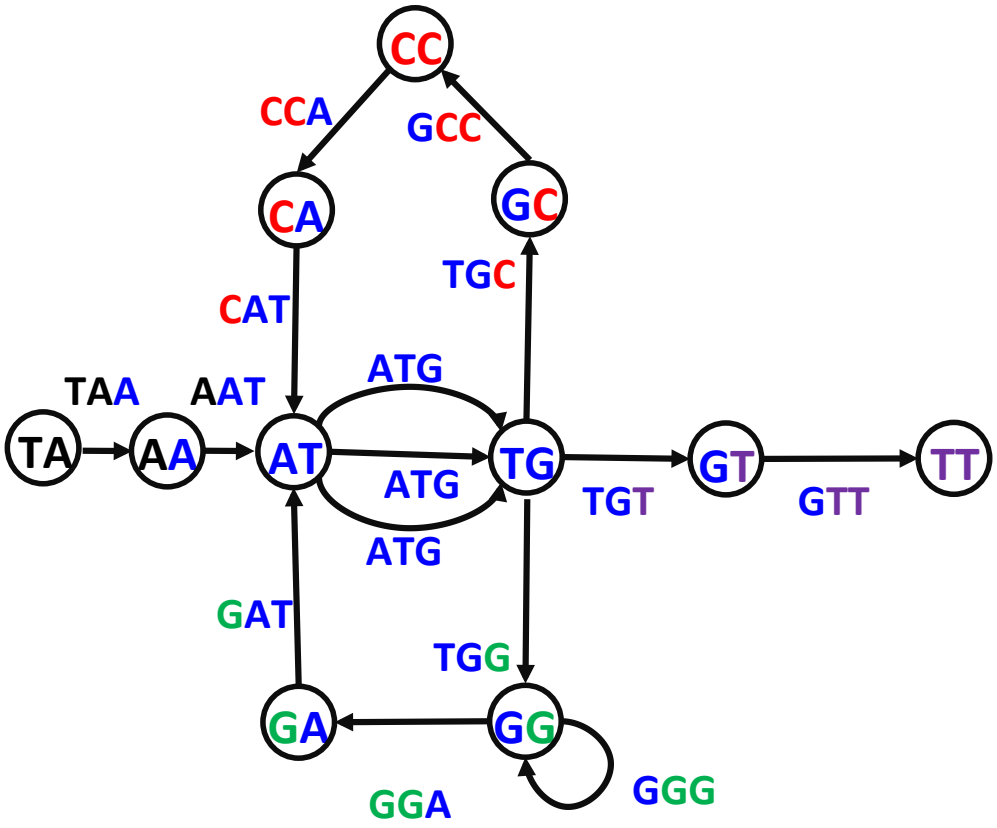
Gluing Identically Labeled Nodes



Gluing Identically Labeled Nodes



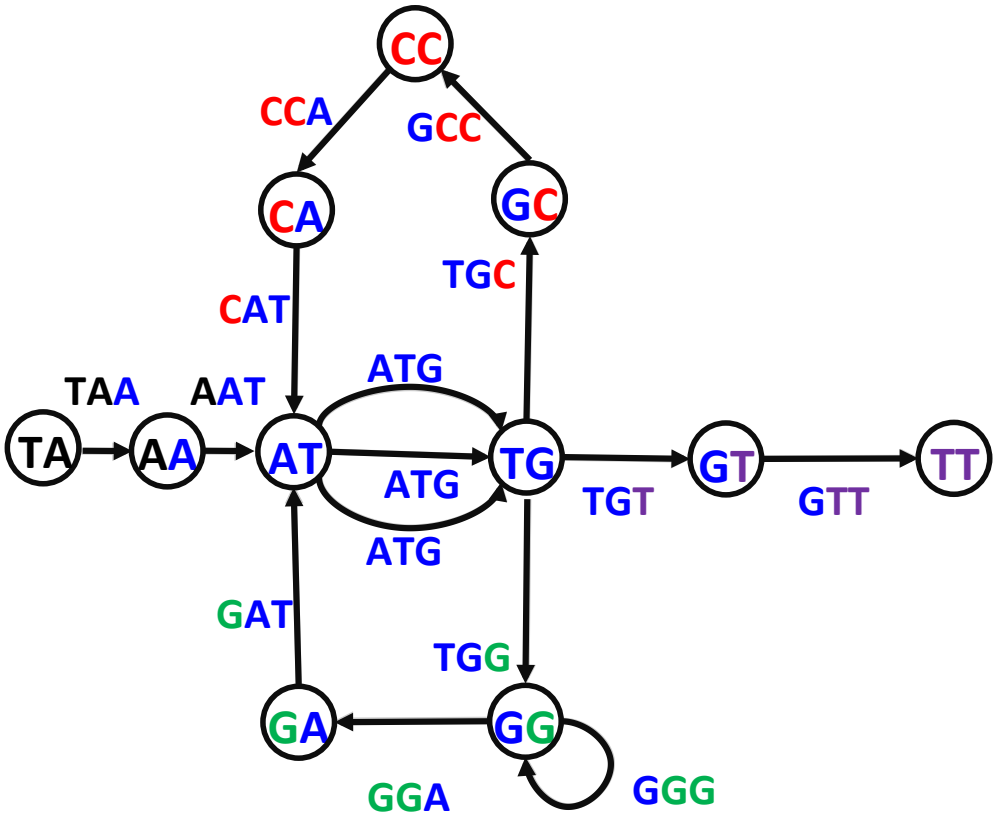
De Bruijn Graph of TAATGCCATGGGATGTT



Where is the Genome hiding in this graph?

It Was Always There!

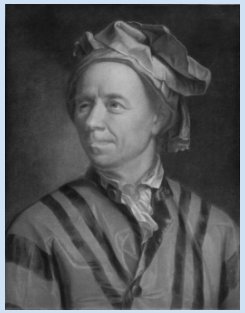
TA TGCCATGGGATGTT
A



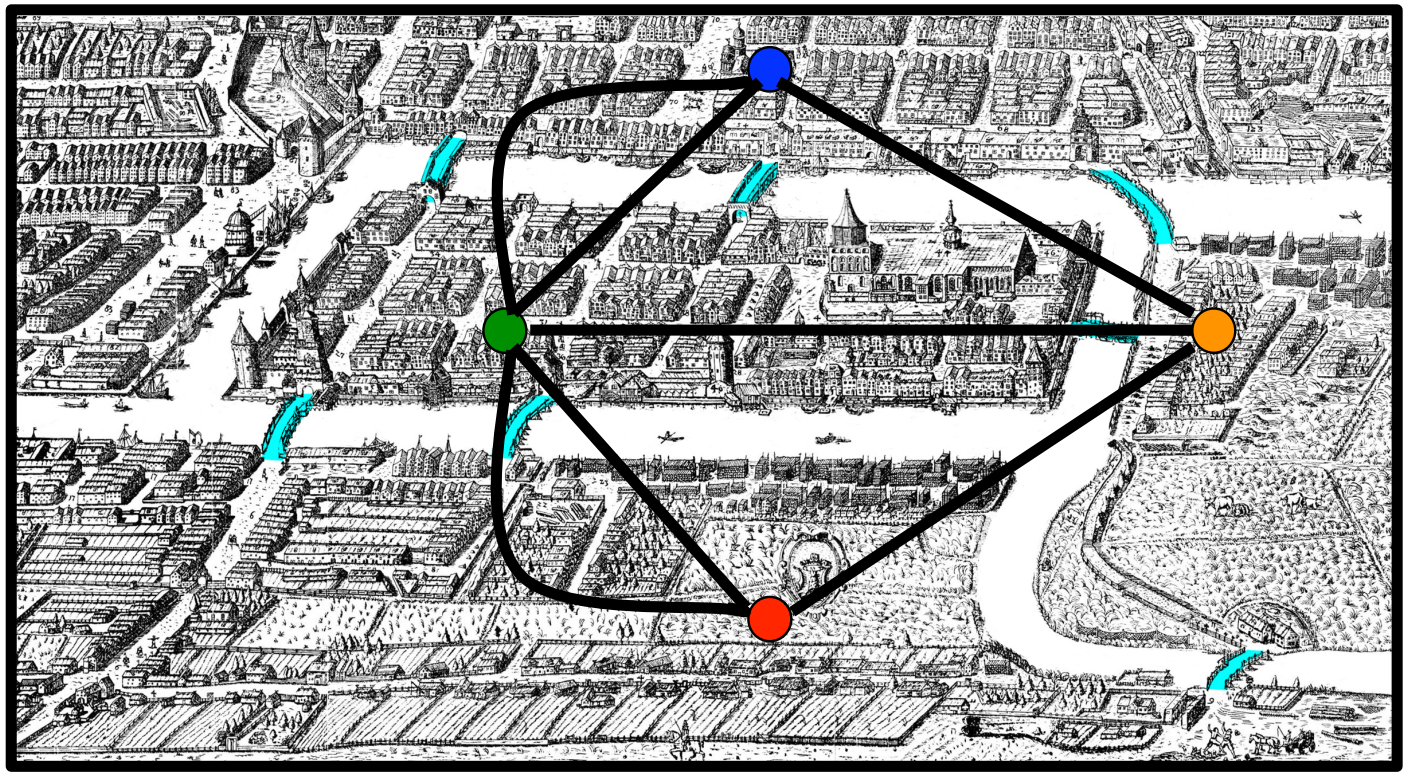
An Eulerian **path** in a graph is a path that visits each edge exactly once.

Eulerian Path Problem

Eulerian Path Problem. Find an Eulerian path in a graph.



- Input. A graph.
- Output. A path visiting every edge in the graph exactly once.



Eulerian Versus Hamiltonian Paths

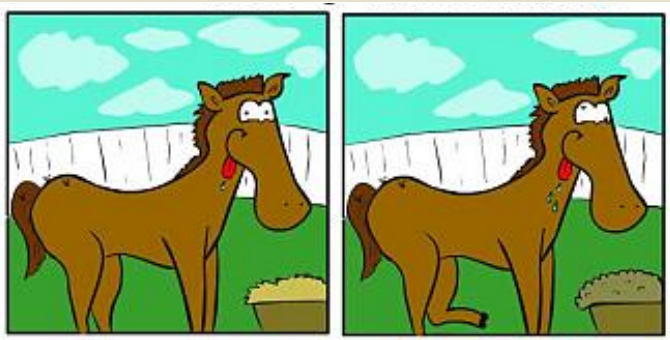
Eulerian Path Problem. Find an Eulerian path in a graph.

- Input. A graph.
- Output. A path visiting every edge in the graph exactly once.

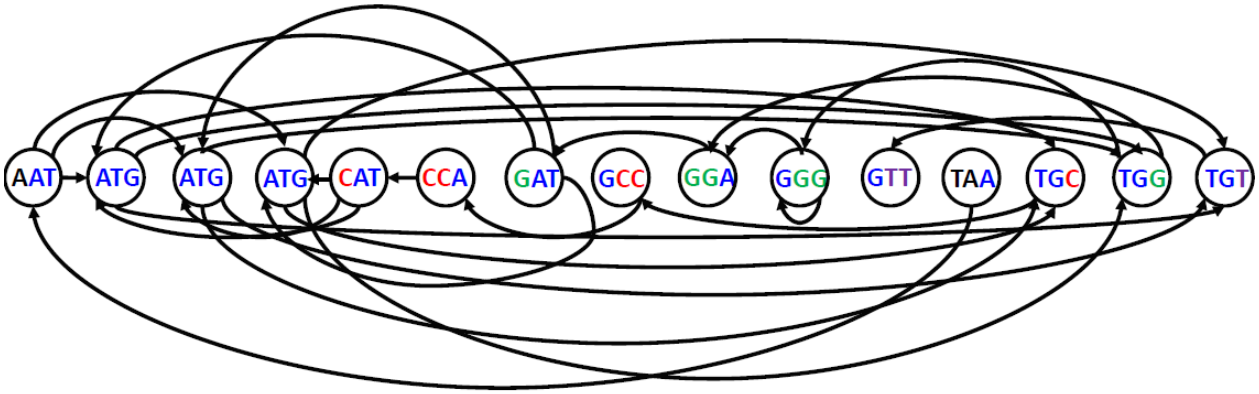
Hamiltonian Path Problem. Find a Hamiltonian path in a graph.

- Input. A graph.
- Output. A path visiting every node in the graph exactly once.

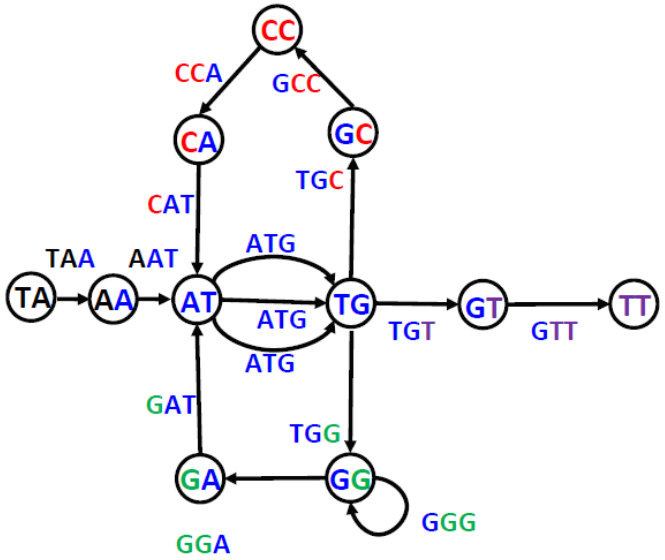
Find a difference!



What Problem Would You Prefer to Solve?

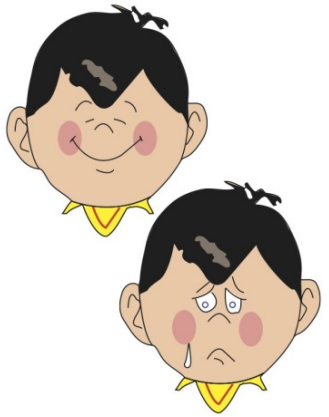


Hamiltonian Path Problem



Eulerian Path Problem

While Euler solved the Eulerian Path Problem (even for a city with a million bridges), nobody has developed a fast algorithm for the Hamiltonian Path Problem yet.



NP-Complete Problems

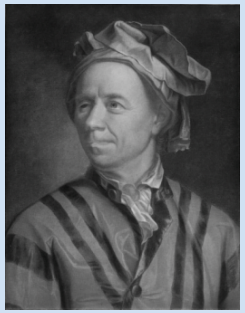
- The Hamiltonian Path Problem belongs to a collection containing thousands of computational problems for which no fast algorithms are known.

That would be an excellent argument, but the question of whether or not NP-Complete problems can be solved efficiently is one of seven **Millennium Problems** in mathematics.

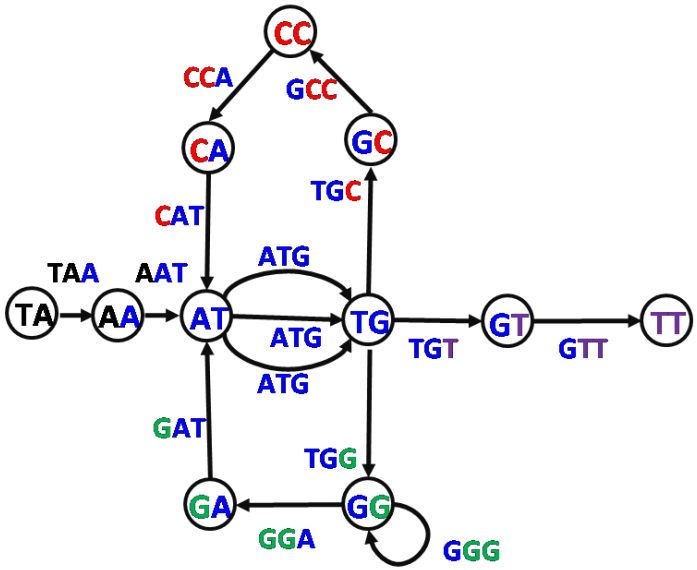
NP-Complete problems are all equivalent: find an efficient solution to one, and you have an efficient solution to them all.

Eulerian Path Problem

Eulerian Path Problem. Find an Eulerian path in a graph.

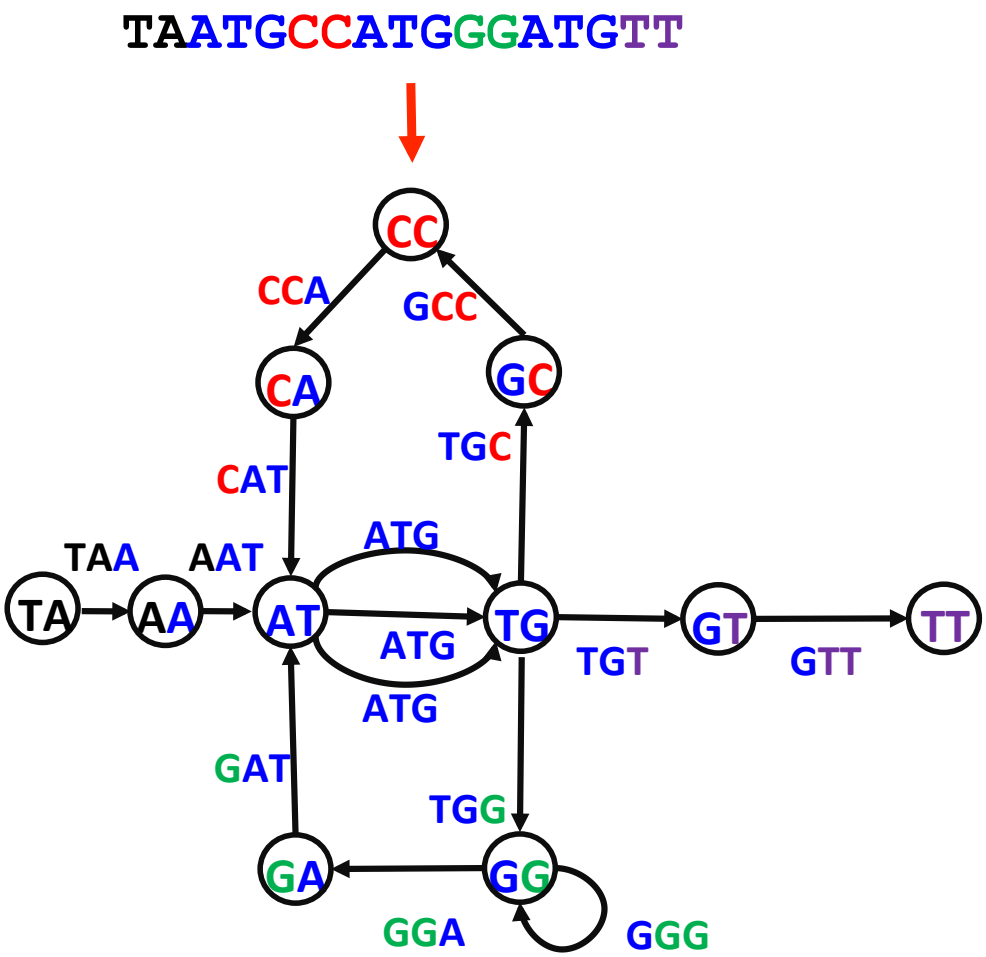


- Input. A graph.
- Output. A path visiting every edge in the graph exactly once.

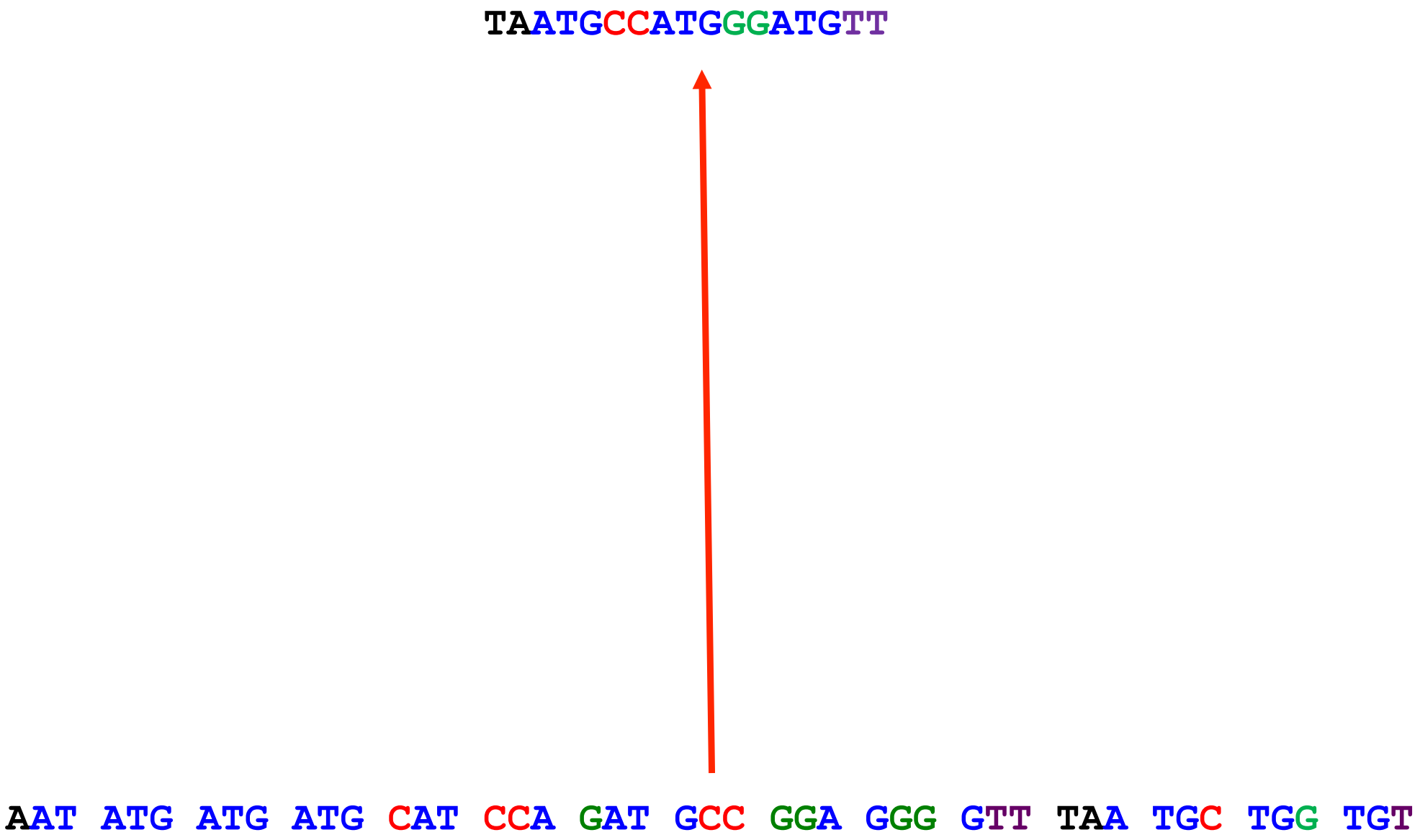


We constructed the de Bruijn graph from Genome, but in reality, Genome is unknown!

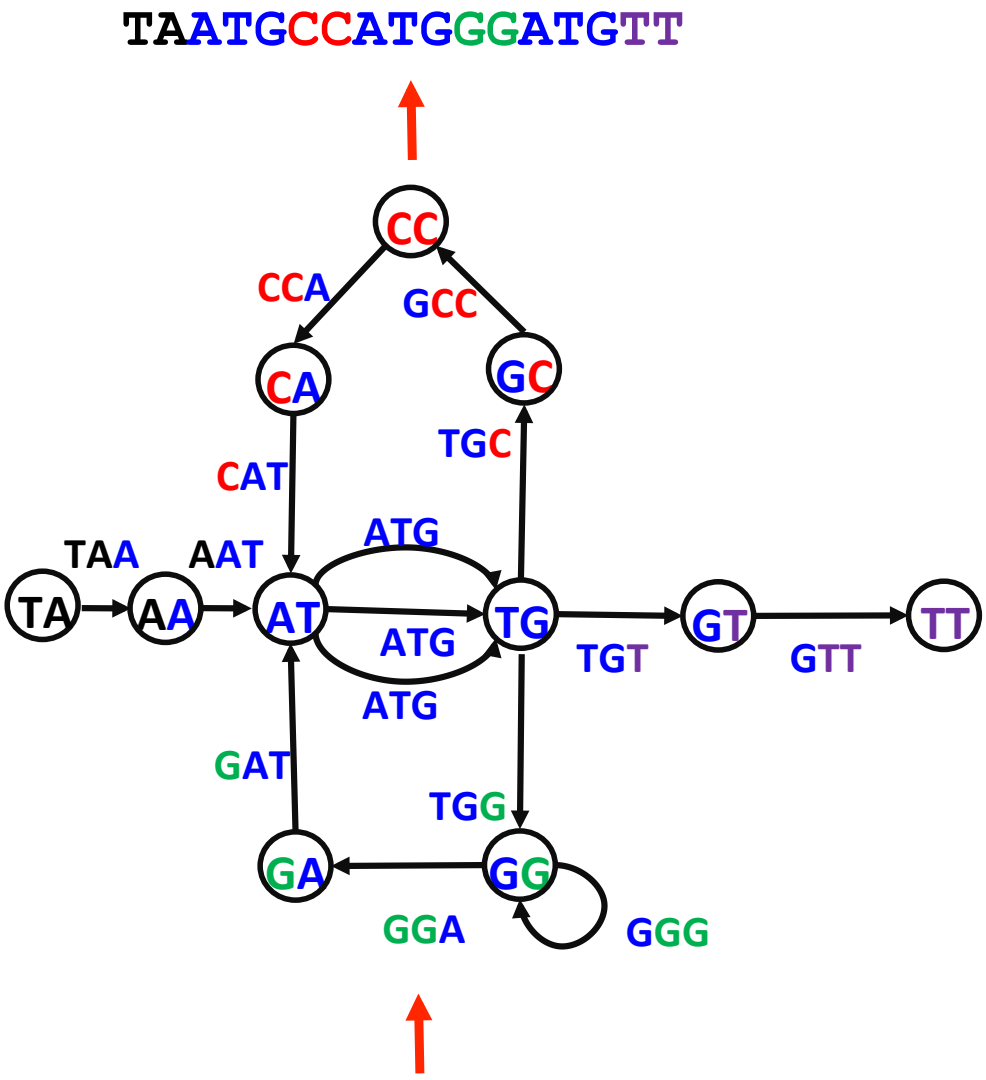
What We Have Done: From Genome to de Bruijn Graph



What We Want: From Reads (k-mers) to Genome



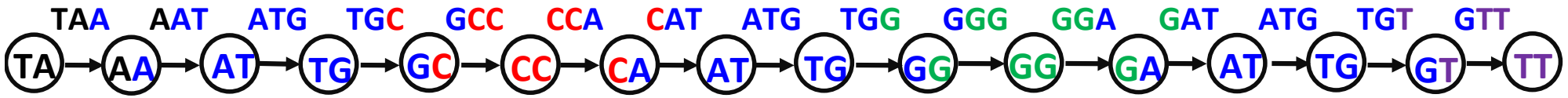
What We will Show: From Reads to de Bruijn Graph to Genome



AAT ATG ATG ATG CAT CCA GAT GCC GGA GGG GTT TAA TGC TGG TGT

Constructing de Bruijn Graph when Genome Is Known

TAATGCCATGGGATGTT



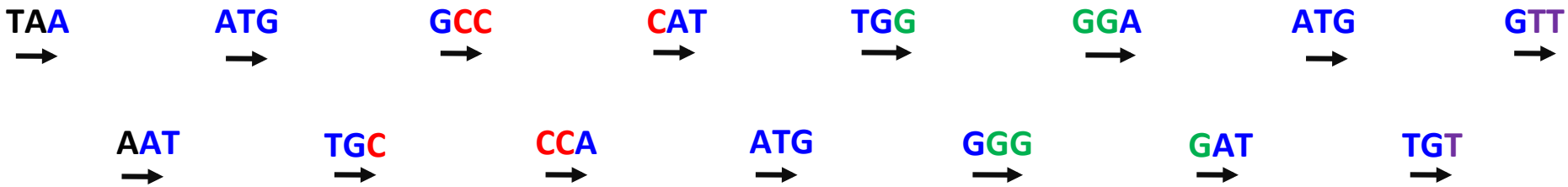
Constructing de Bruijn when Genome Is Unknown

TAA ATG GCC CAT TGG GGA ATG GTT

AAT TGC CCA ATG GGG GAT TGT

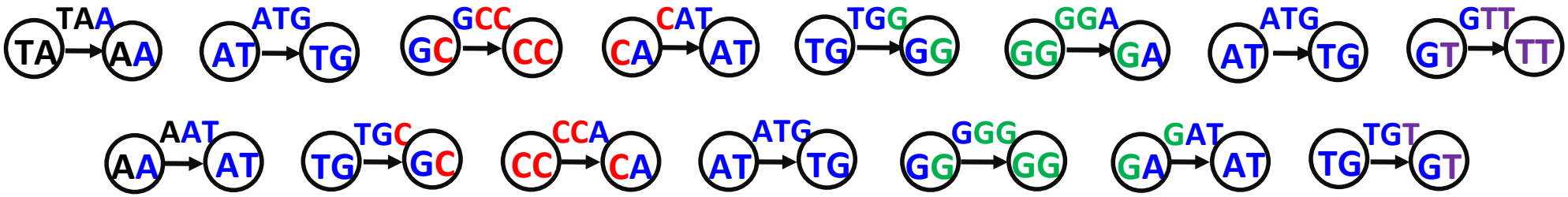
Composition$_3$(TAATGCCATGGGATGTT)

Representing Composition as a Graph Consisting of Isolated Edges



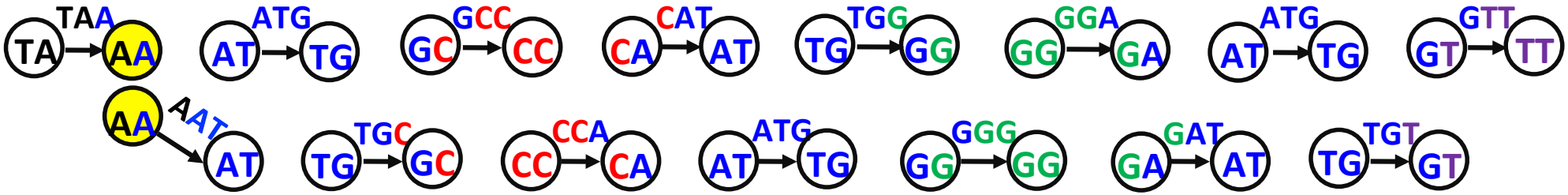
Composition$_3$(**TAATGCCATGGGATGTT**)

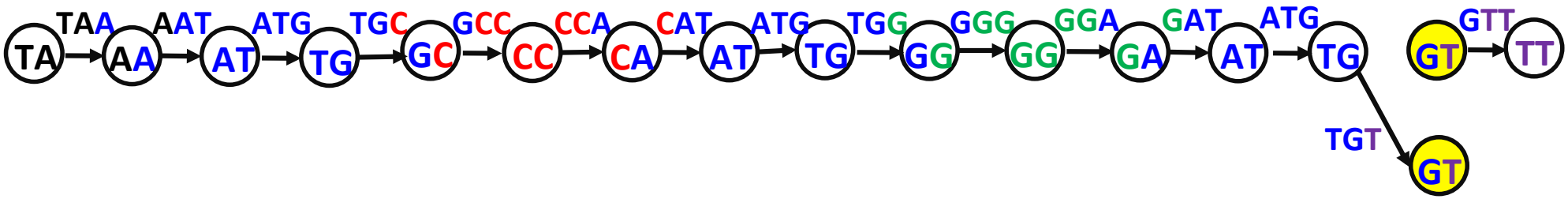
Constructing de Bruijn Graph from k-mer Composition



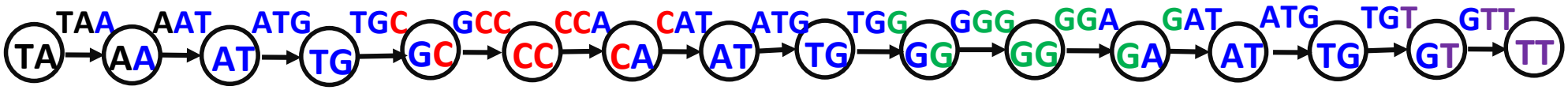
Composition$_3$(**TAATGCCATGGGATGTT**)

Gluing Identically Labeled Nodes

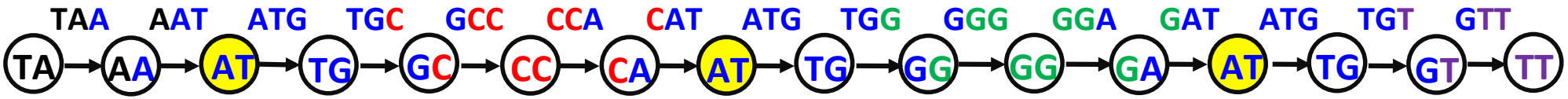


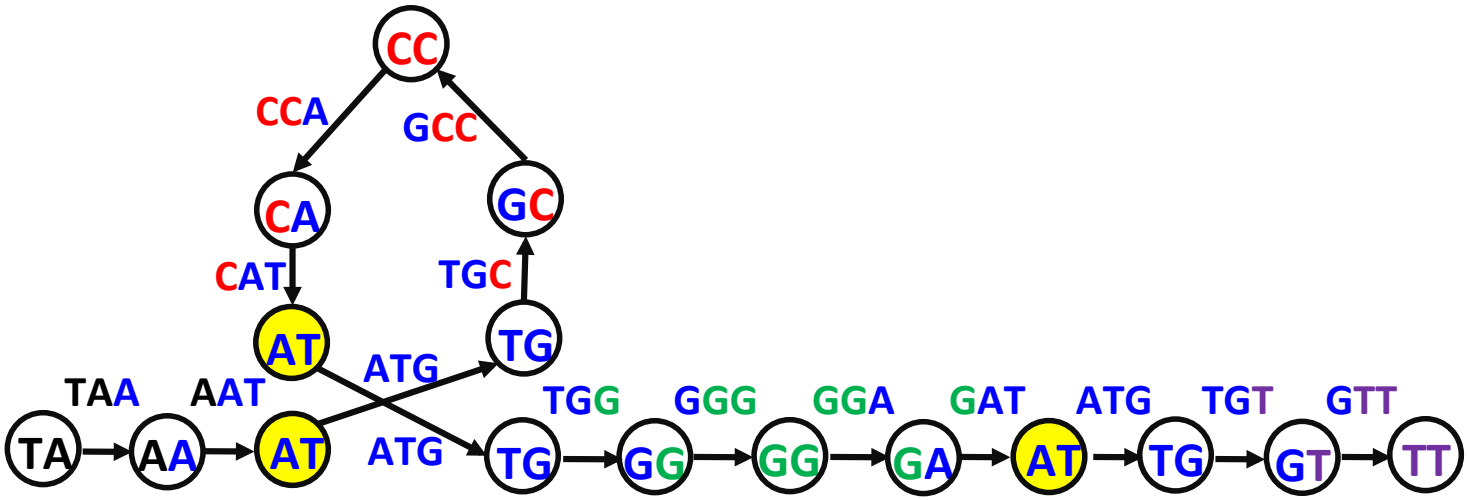

TA
AA
AT
TG
GC
CC
CA
AT
TG
GG
GG
GA
AT
TG
GT
TT
GT
TAA
AAT
ATG
TGC
GCC
CCA
CAT
ATG
TGG
GGG
GGA
GAT
ATG
GTT
TGT

We Are Not Done with Gluing Yet

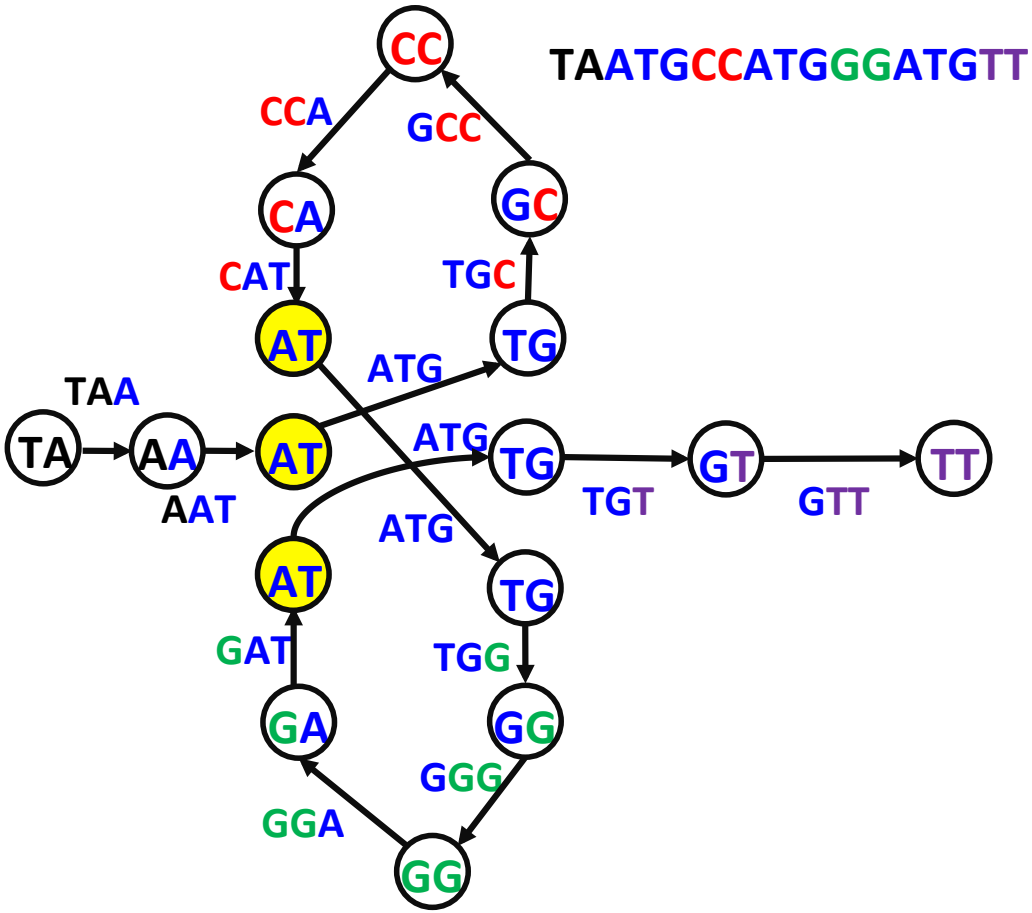


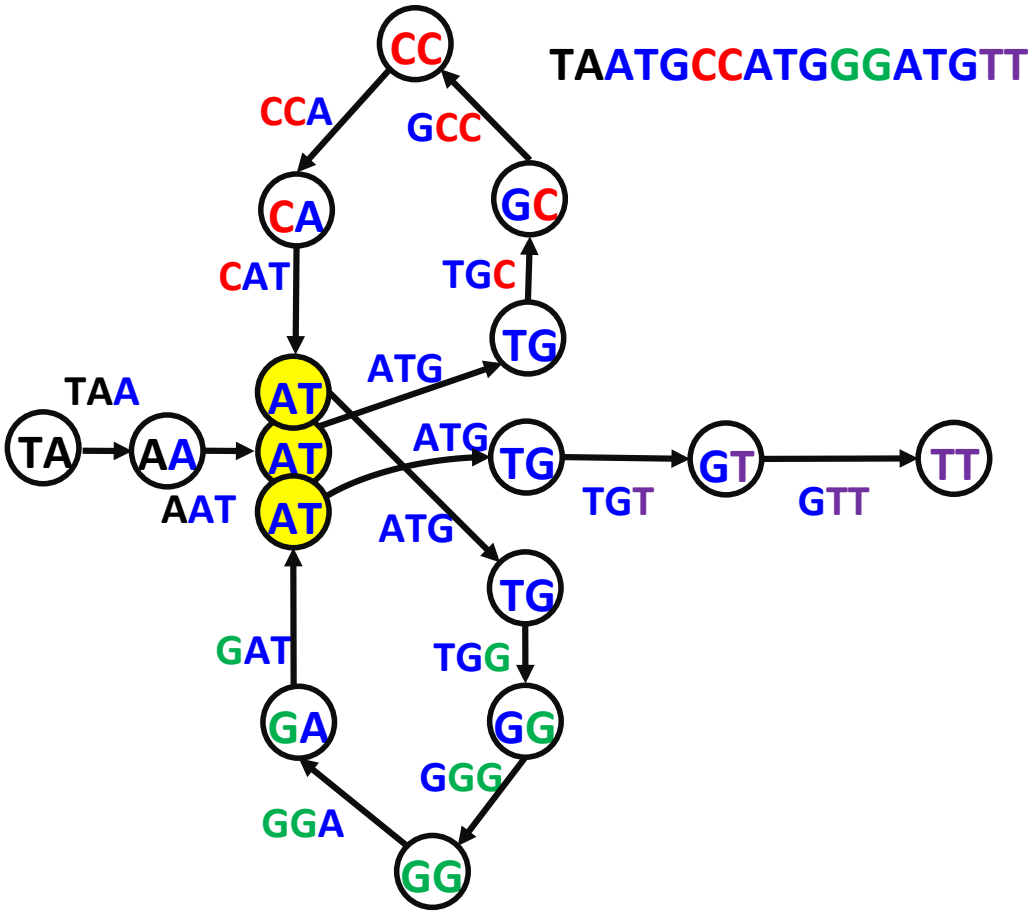
Gluing Identically Labeled Nodes



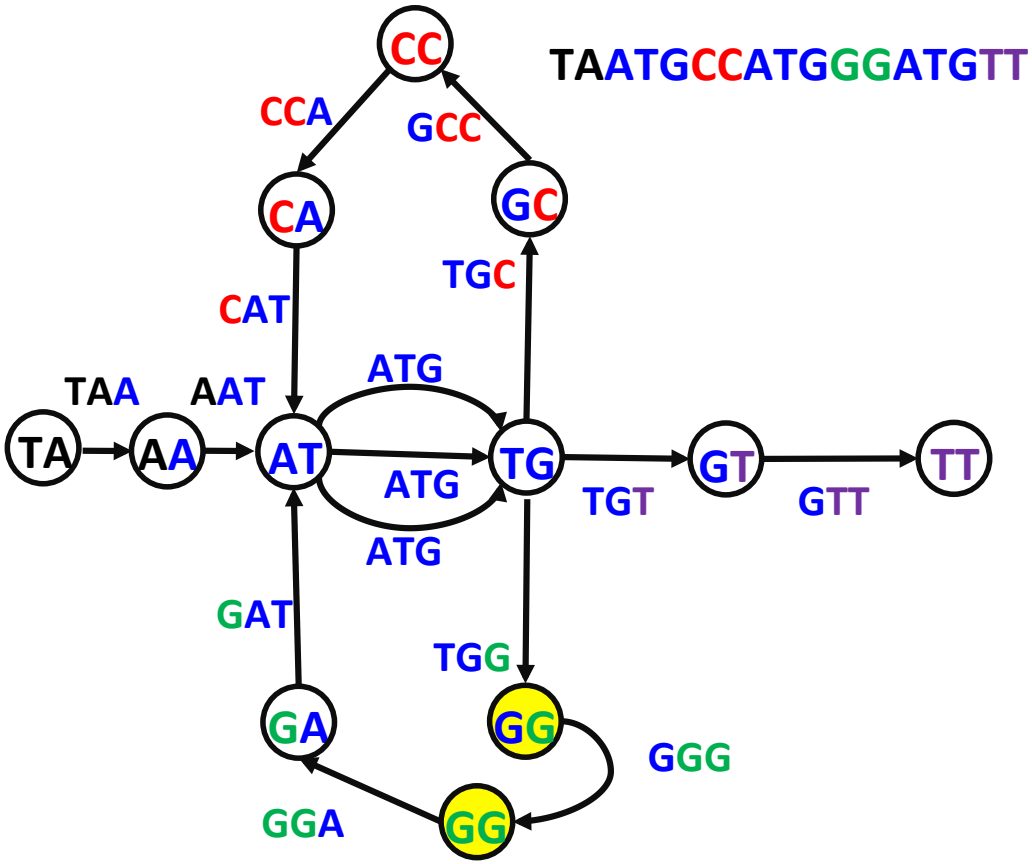


Gluing Identically Labeled Nodes

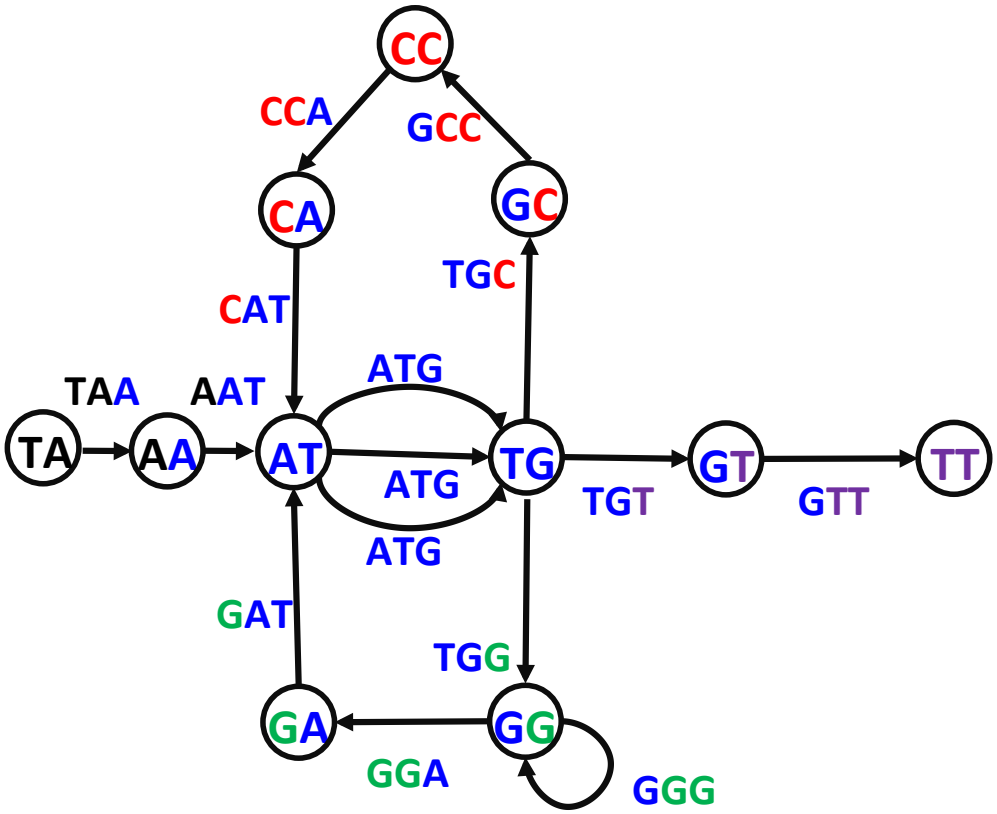



TAATGCCATGGGATGTT
CC
CCA
GCC
CA
GC
CAT
TGC
TG
ATG
TAA
AT
TA
AA
AT
ATG
TG
GT
TT
AAT
AT
TGT
GTT
ATG
TG
GAT
TGG
GA
GG
GGG
GGA
GG

Gluing Identically Labeled Nodes



The Same de Bruijn Graph: DeBruin(Genome)=DeBruin(Genome Composition)

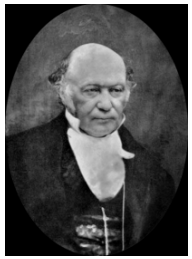
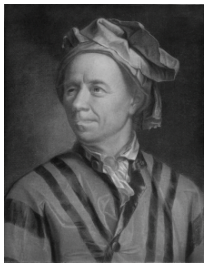


Constructing de Bruijn Graph

De Bruijn graph of a collection of *k*-mers:

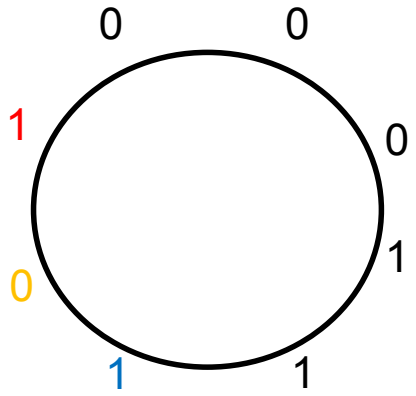
- Represent every *k*-mer as an edge between its prefix and suffix
- Glue **ALL** nodes with identical labels.

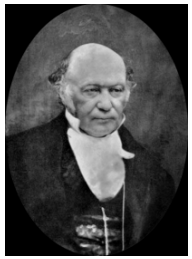
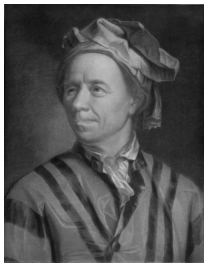
```
DeBruijn(k-mers)
  form a node for each (k-1)-mer from k-mers
  for each k-mer in k-mers
     connect its prefix node with its suffix node by an edge
```

From Hamilton  to Euler  to de Bruijn 

Universal String Problem (Nicolaas de Bruijn, 1946). Find a circular string containing each binary k-mer exactly once.

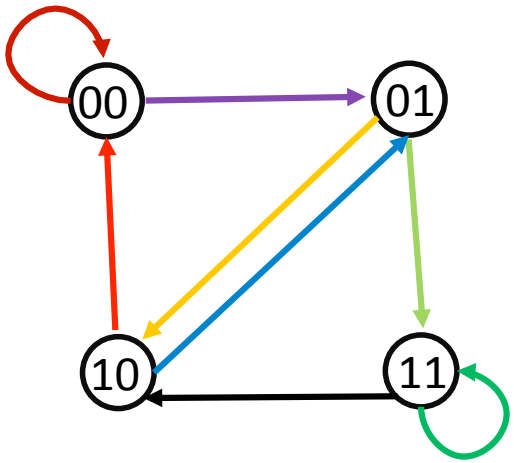
000 001 010 011 100 101 110 111

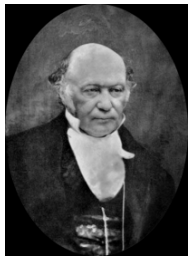
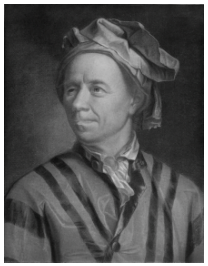


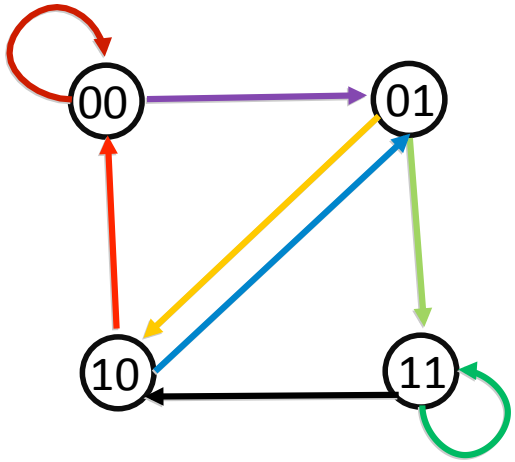
From Hamilton  to Euler  to de Bruijn 

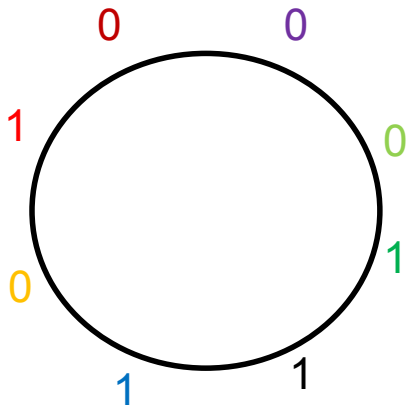
Universal String Problem (Nicolaas de Bruijn, 1946). Find a circular string containing each binary k-mer exactly once.

000 001 010 011 100 101 110 111

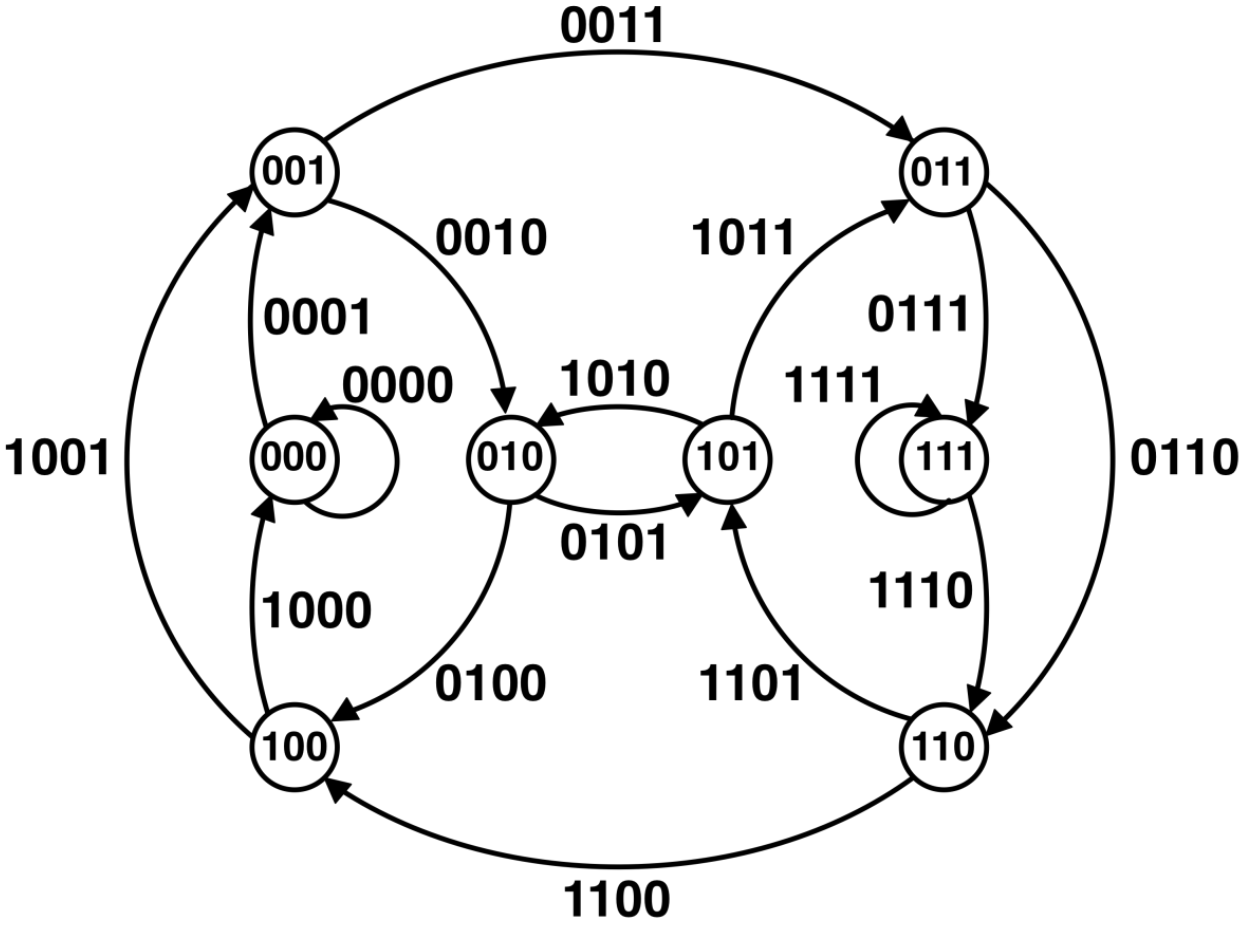


From Hamilton  to Euler  to de Bruijn 





De Bruijn Graph for 4-Universal String

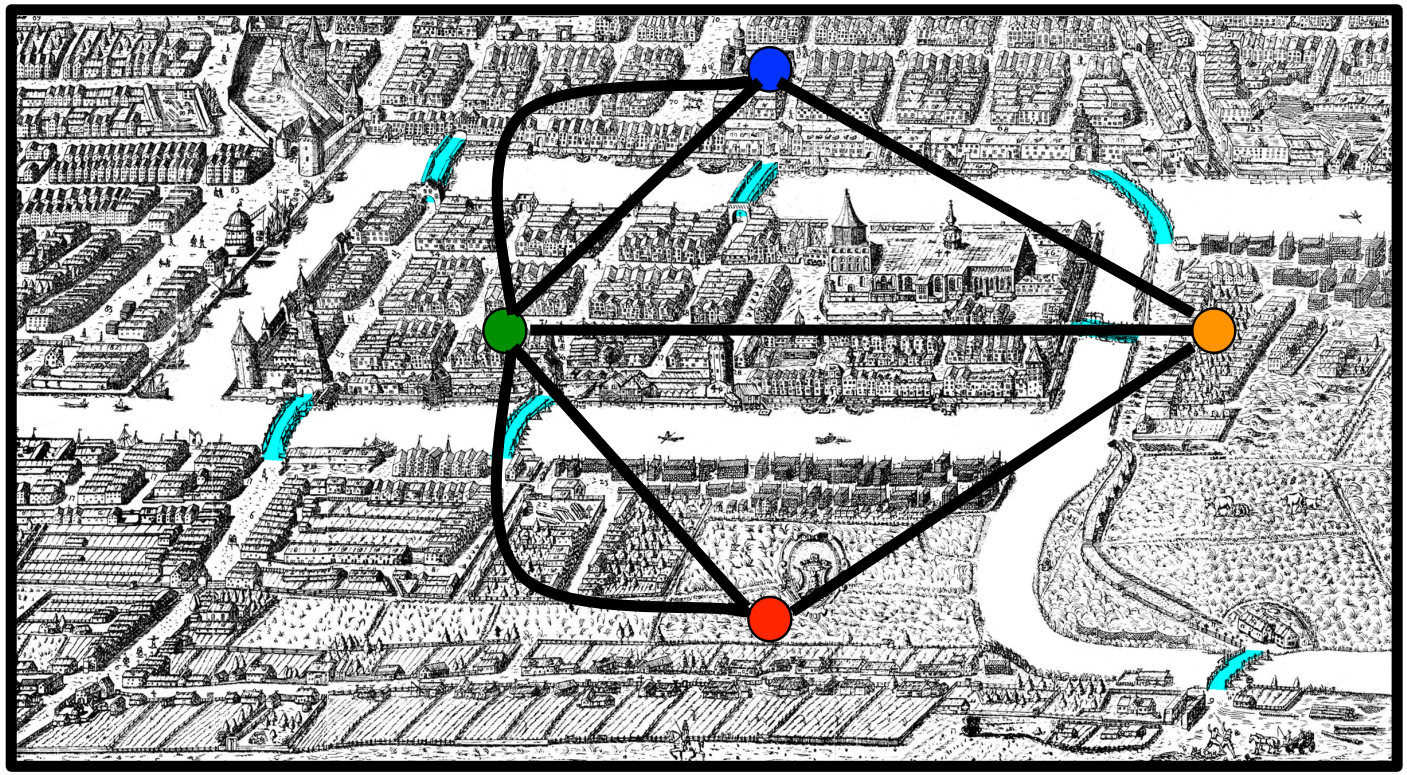


Does it have an Eulerian cycle? If yes, how can we find it?

Eulerian CYCLE Problem

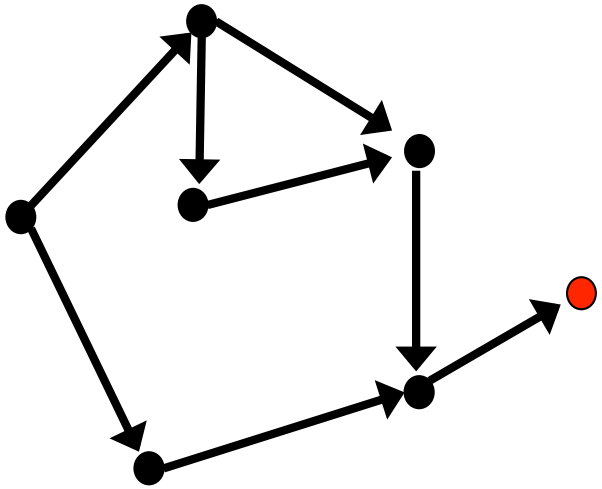
Eulerian CYCLE Problem. Find an Eulerian cycle in a graph.

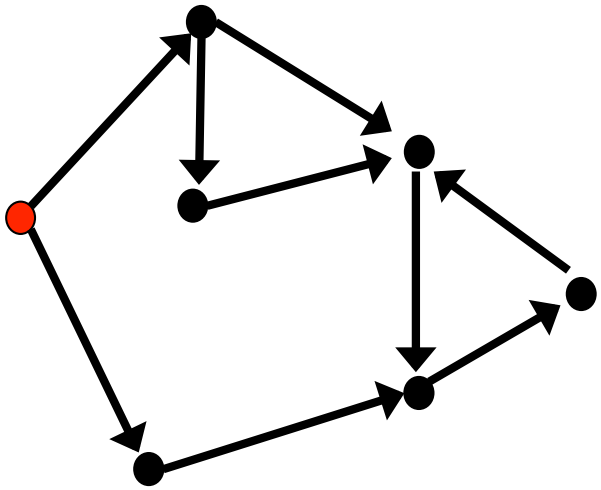
- Input. A graph.
- Output. A cycle visiting every edge in the graph exactly once.

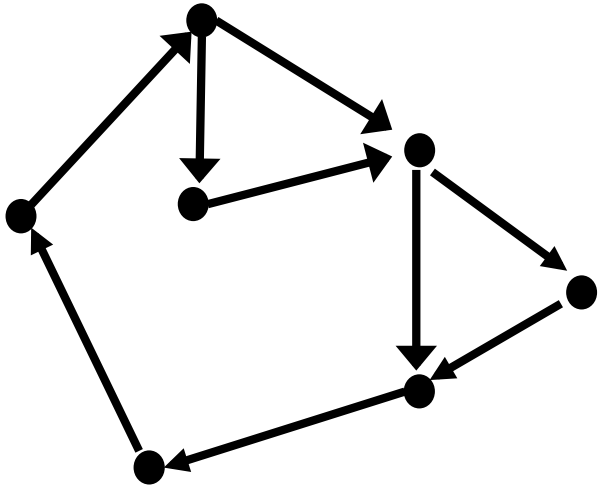


A Graph is **Eulerian** if It Contains an Eulerian Cycle.

Is this graph Eulerian?

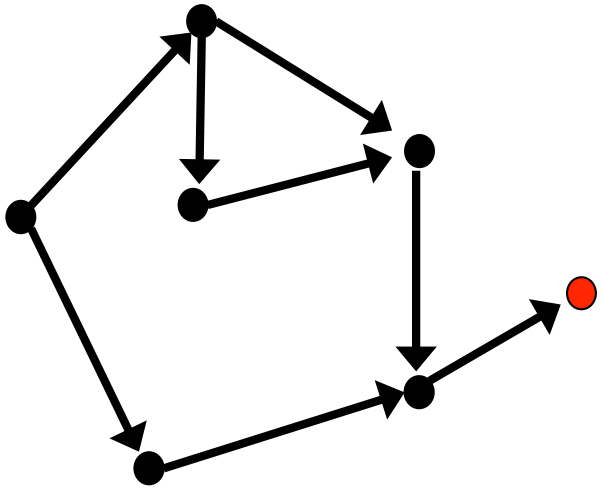


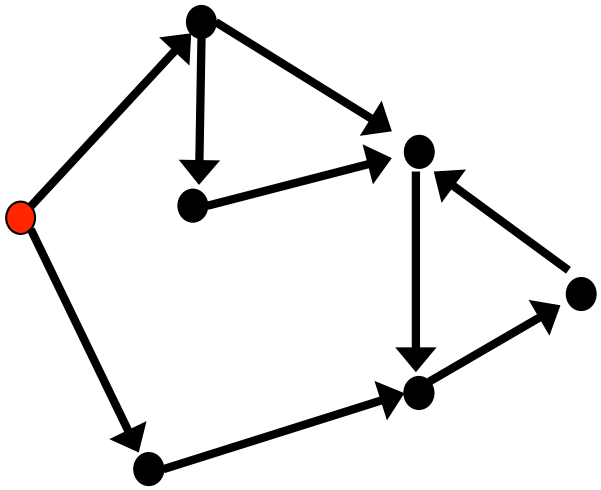


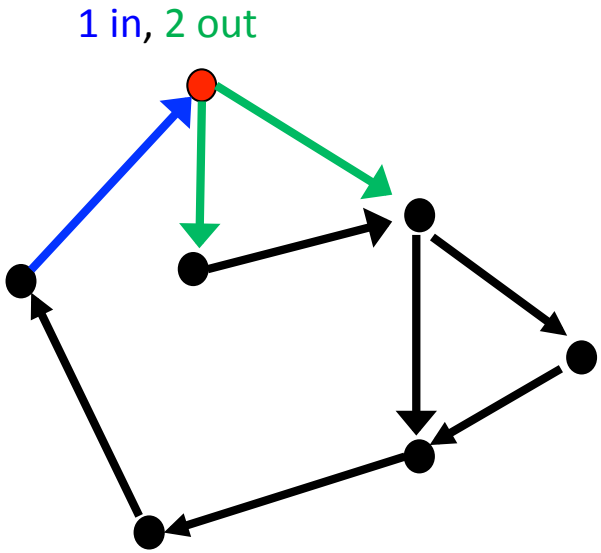


A Graph is **Eulerian** if It Contains an Eulerian Cycle.

Is this graph Eulerian?



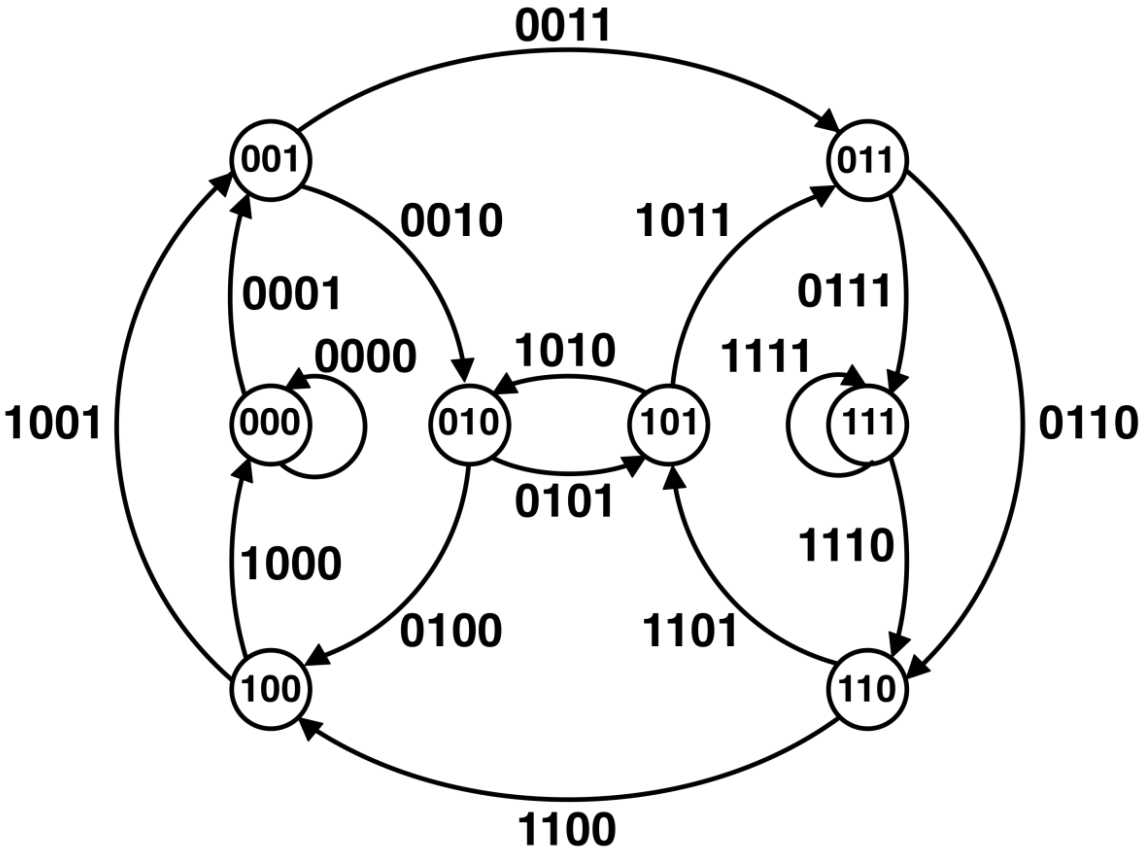




A graph is balanced if indegree = outdegree for each node

Euler's Theorem

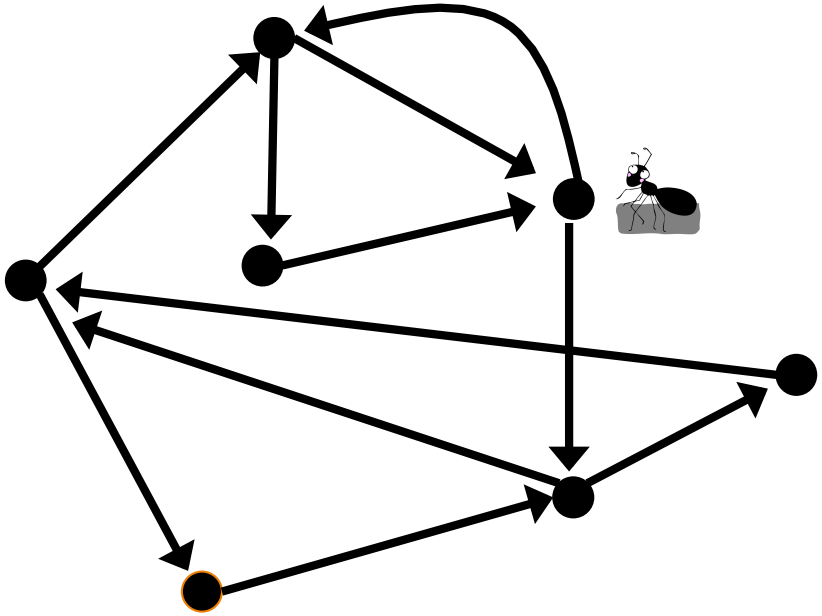
- Every Eulerian graph is balanced
- **Every balanced* graph is Eulerian**



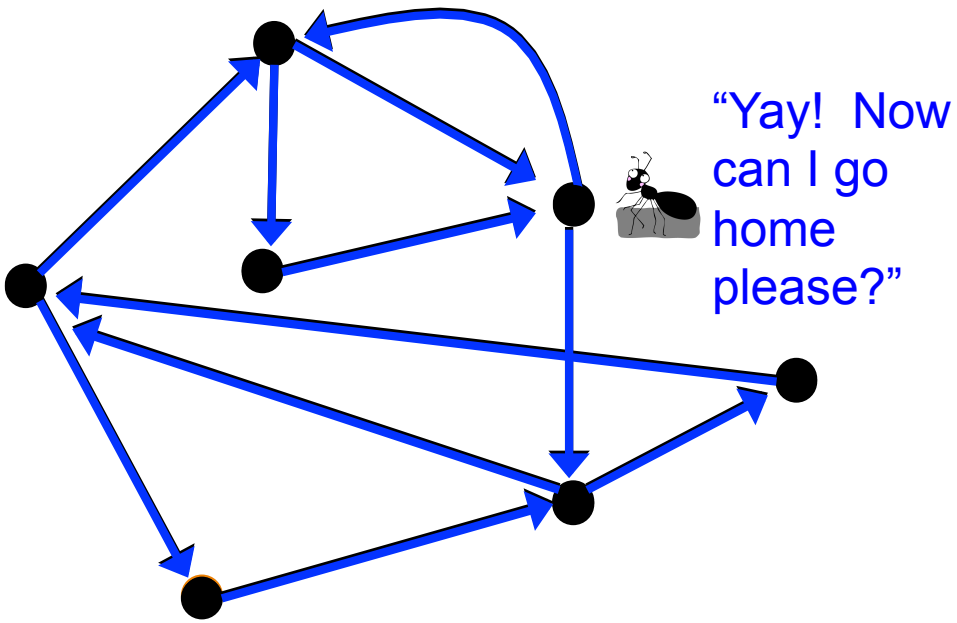
(*) and strongly connected, of course!

Recruiting an Ant to Prove Euler's Theorem

Let an ant randomly walk through the graph.
The ant cannot use the same edge twice!

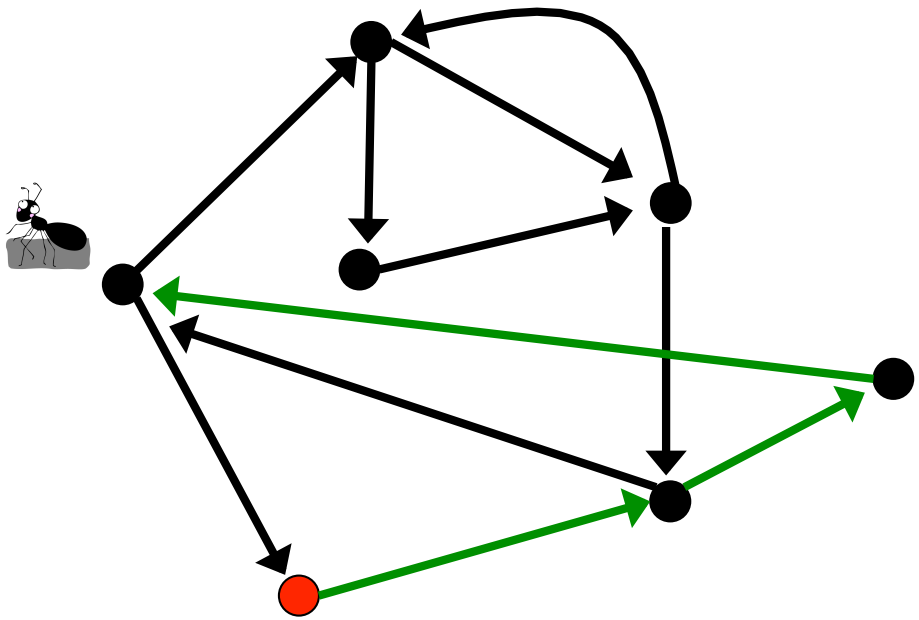


If Ant Was a Genius…



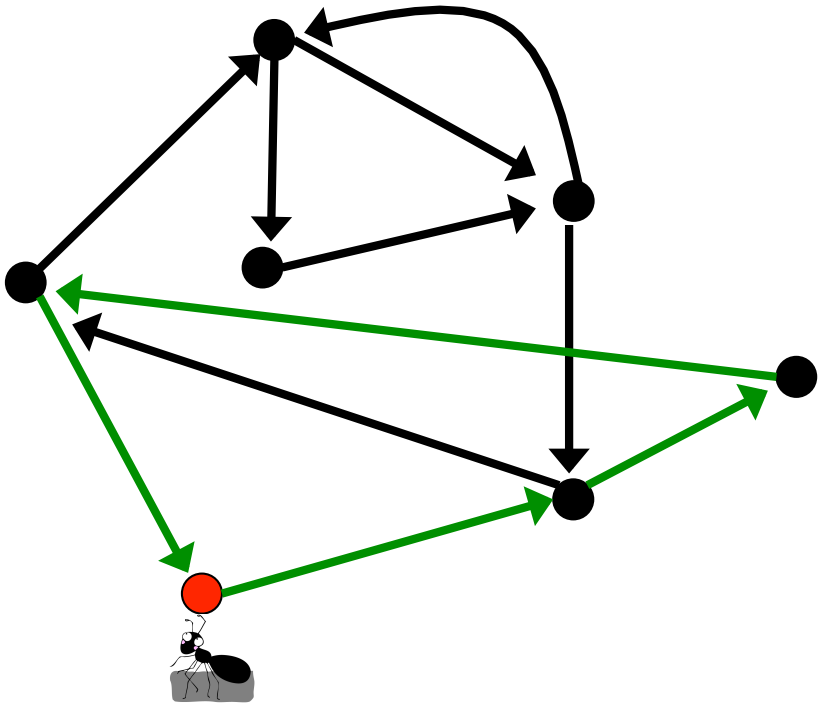
A Less Intelligent Ant Would Randomly Choose a Node and Start Walking…

Can it get stuck? **In what node?**



The Ant Has Completed a Cycle BUT has not Proven Euler's theorem yet…

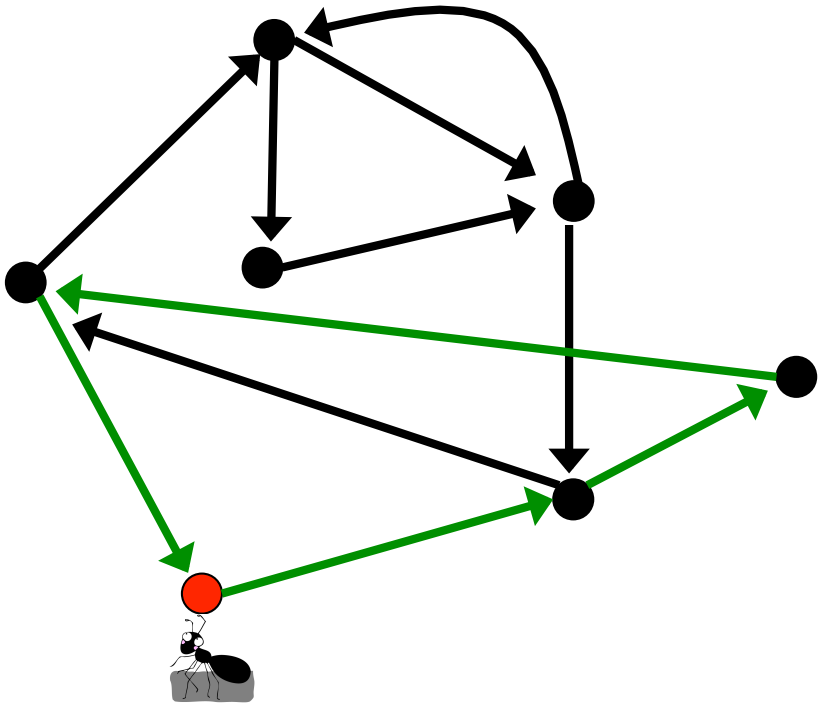
The constructed cycle is not Eulerian. **Can we enlarge it?**



Let's Start at a Different Node in the Green Cycle

Let's start at a node with still unexplored edges.

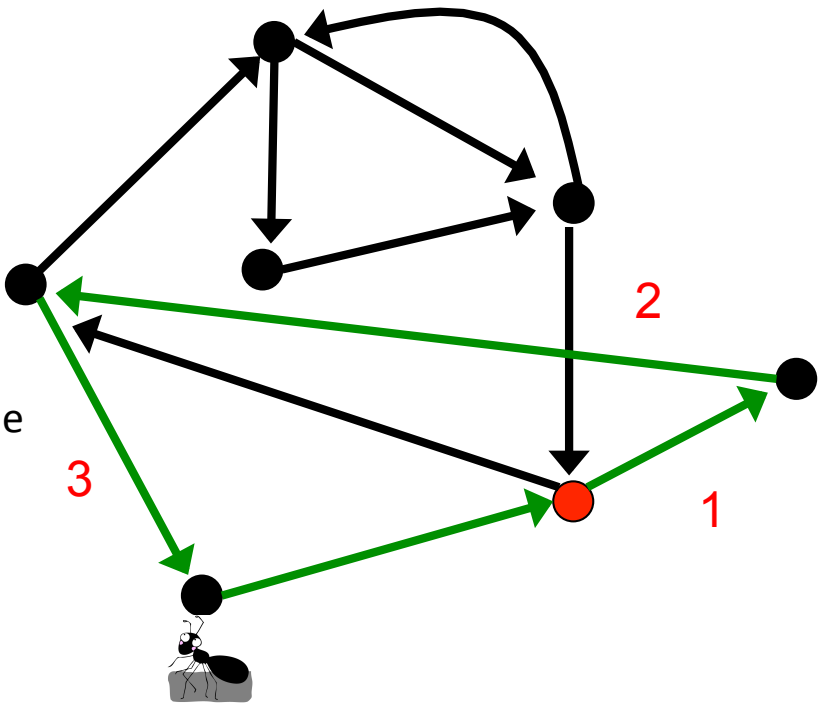
"Why should I start at a different node? Backtracking? I'm not evolved to walk backwards! And what difference does it make???"



An Ant Traversing Previously Constructed Cycle

Starting at a node that has an unused edge, traverse the already constructed (green cycle) and return back to the starting node.

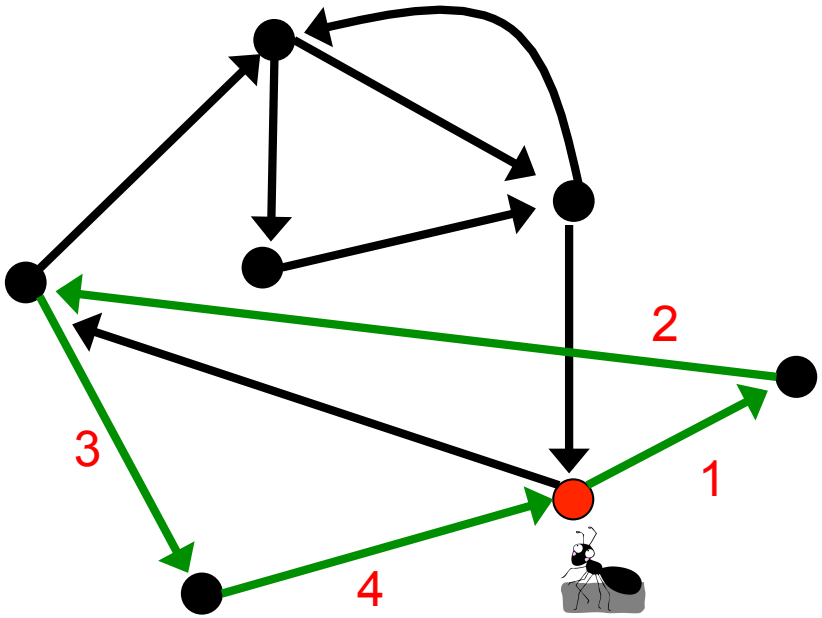
"Why do I have to walk along the same cycle again??? Can I see something new?"



I Returned Back BUT… I Can Continue Walking!

Starting at a node that has an unused edge, traverse the already constructed (green cycle) and return back to the starting node.

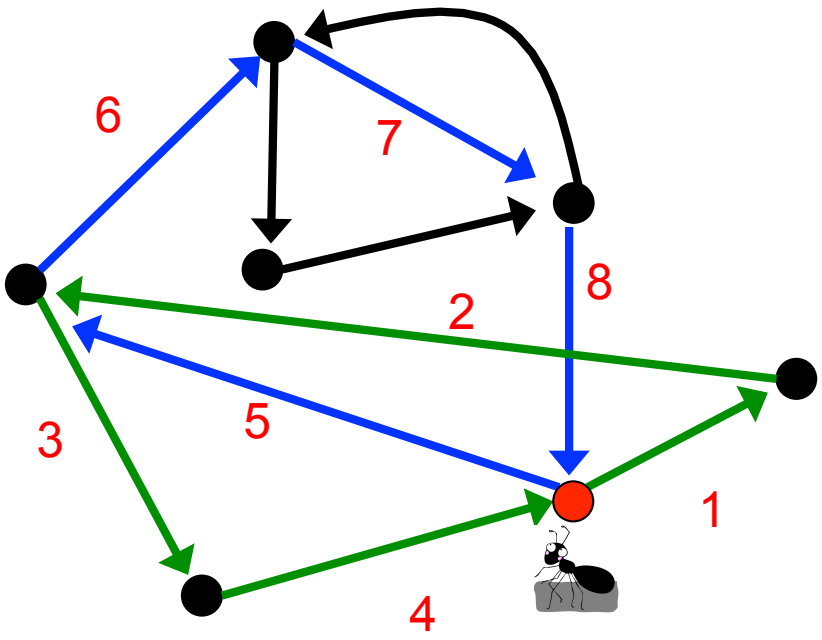
After completing the cycle, start random exploration of still untraversed edges in the graph.



Stuck Again!

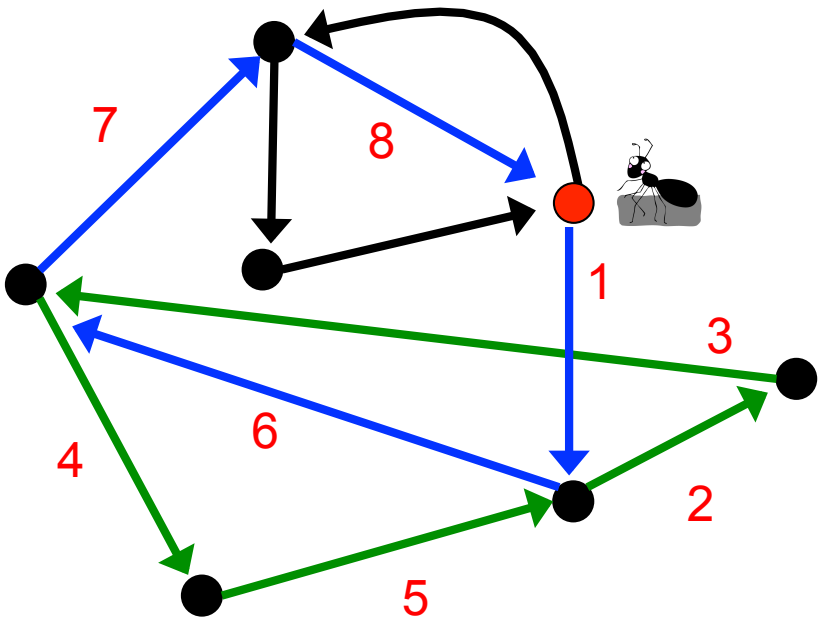
No Eulerian cycle yet… can we enlarge the green-blue cycle?

The ant should walk along the constructed cycle starting at yet another node. Which one?



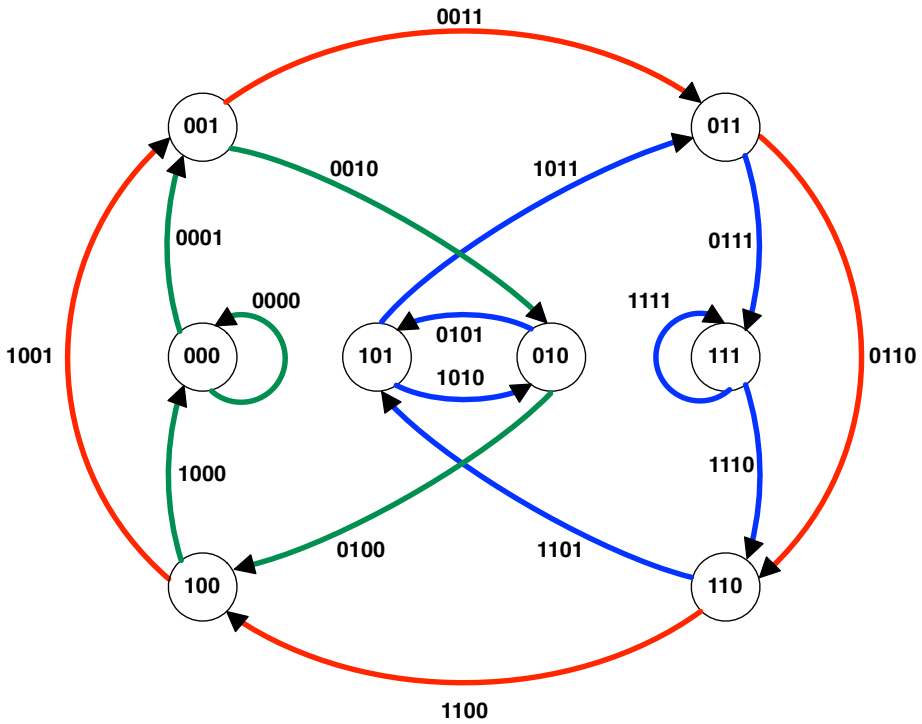
I Returned Back BUT… I Can Continue Walking!

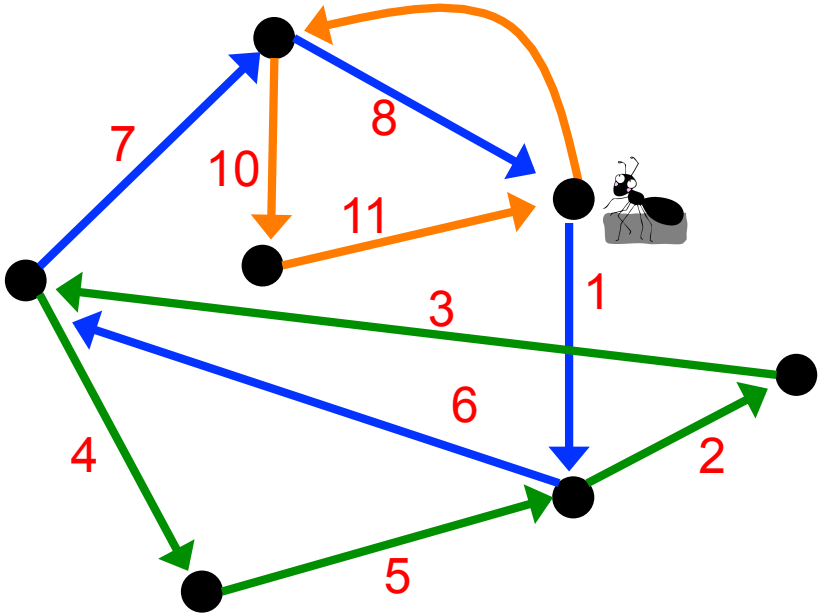
"Hmm, maybe these instructions were not that stupid…"



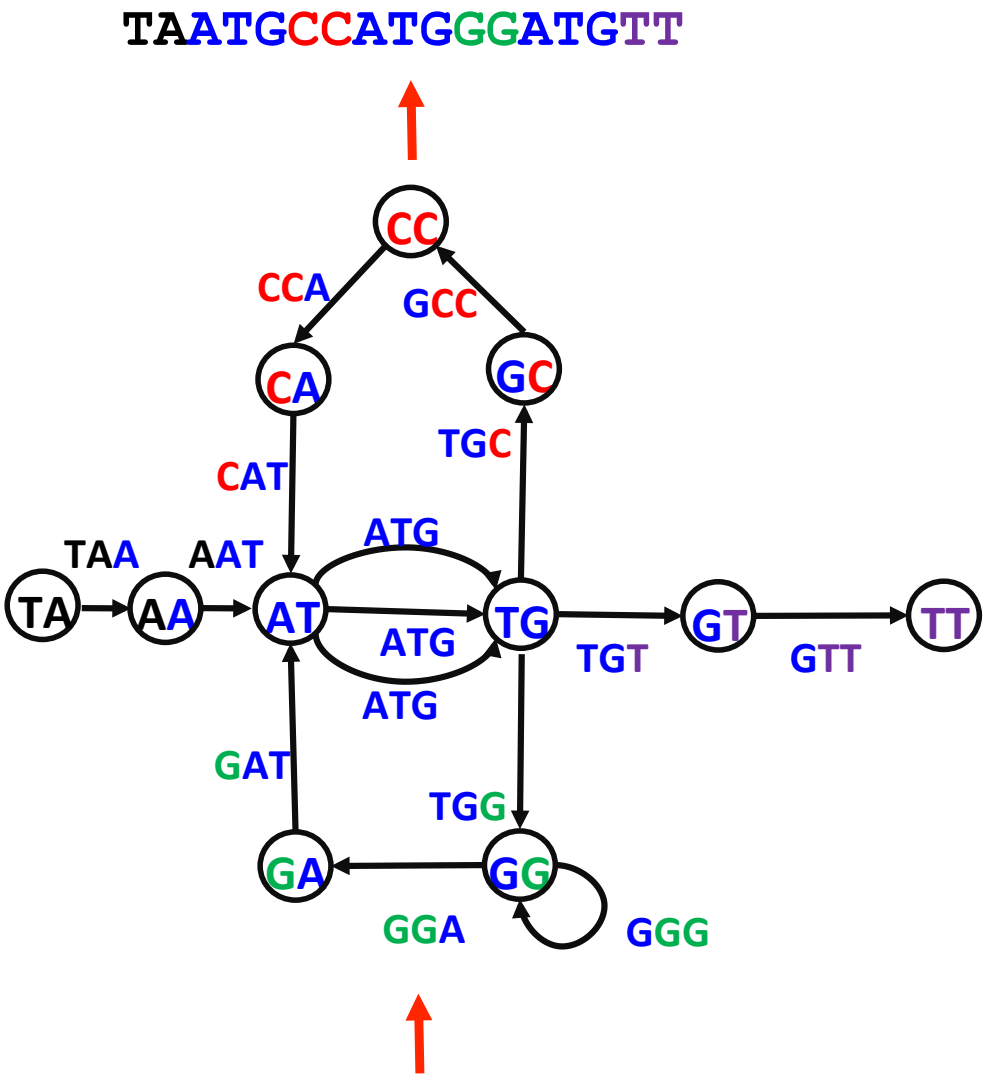
I Proved Euler's Theorem!

EulerianCycle(Balanced*Graph*)
  form a *Cycle* by randomly walking in *BalancedGraph* (avoiding already visited edges)
    while *Cycle* is not Eulerian
      select a node newStart in Cycle with still unexplored outgoing edges
      form a *Cycle'* by traversing *Cycle* from newStart and randomly walking afterwards
      Cycle ← *Cycle'*
  return *Cycle*



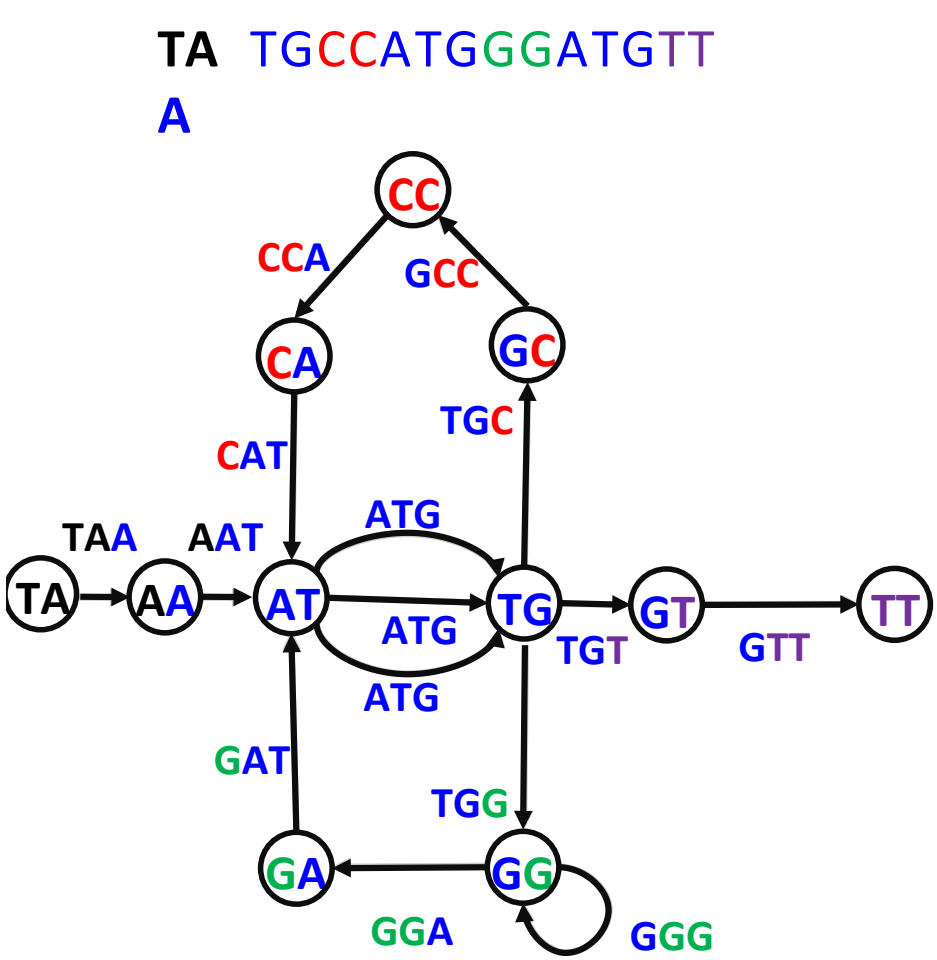


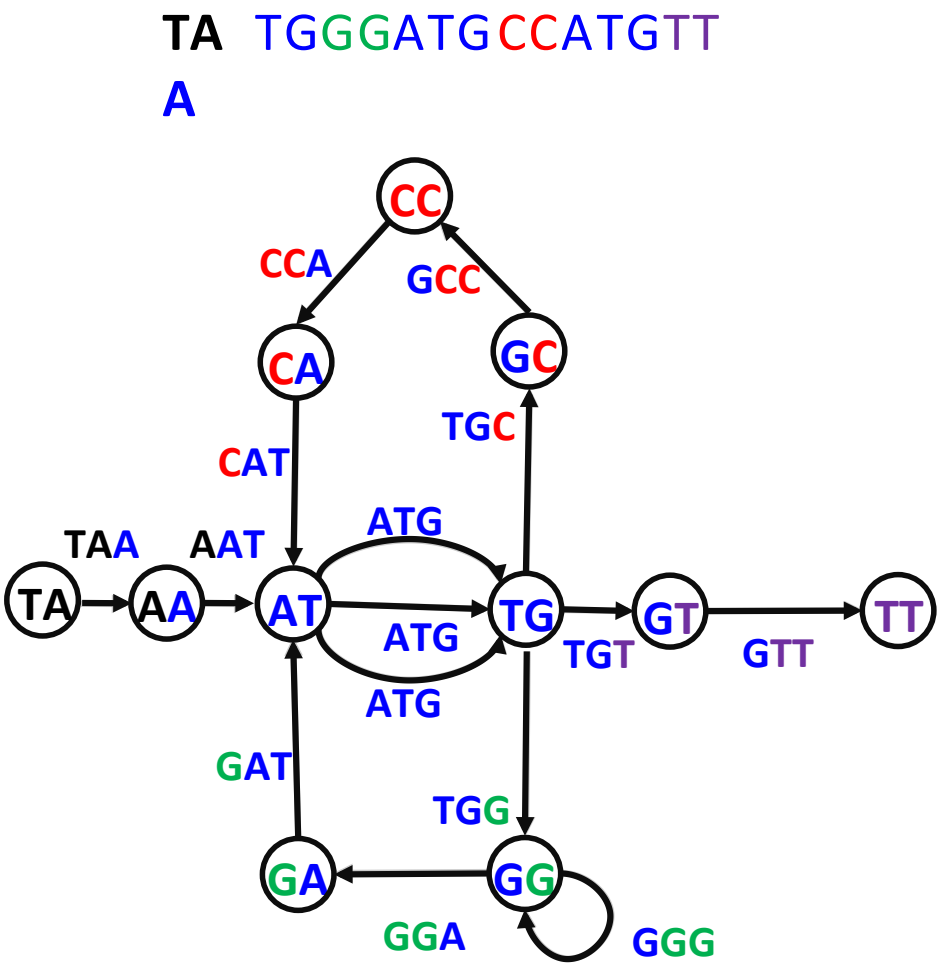
From Reads to de Bruijn Graph to Genome



AAT ATG ATG ATG CAT CCA GAT GCC GGA GGG GTT TAA TGC TGG TGT

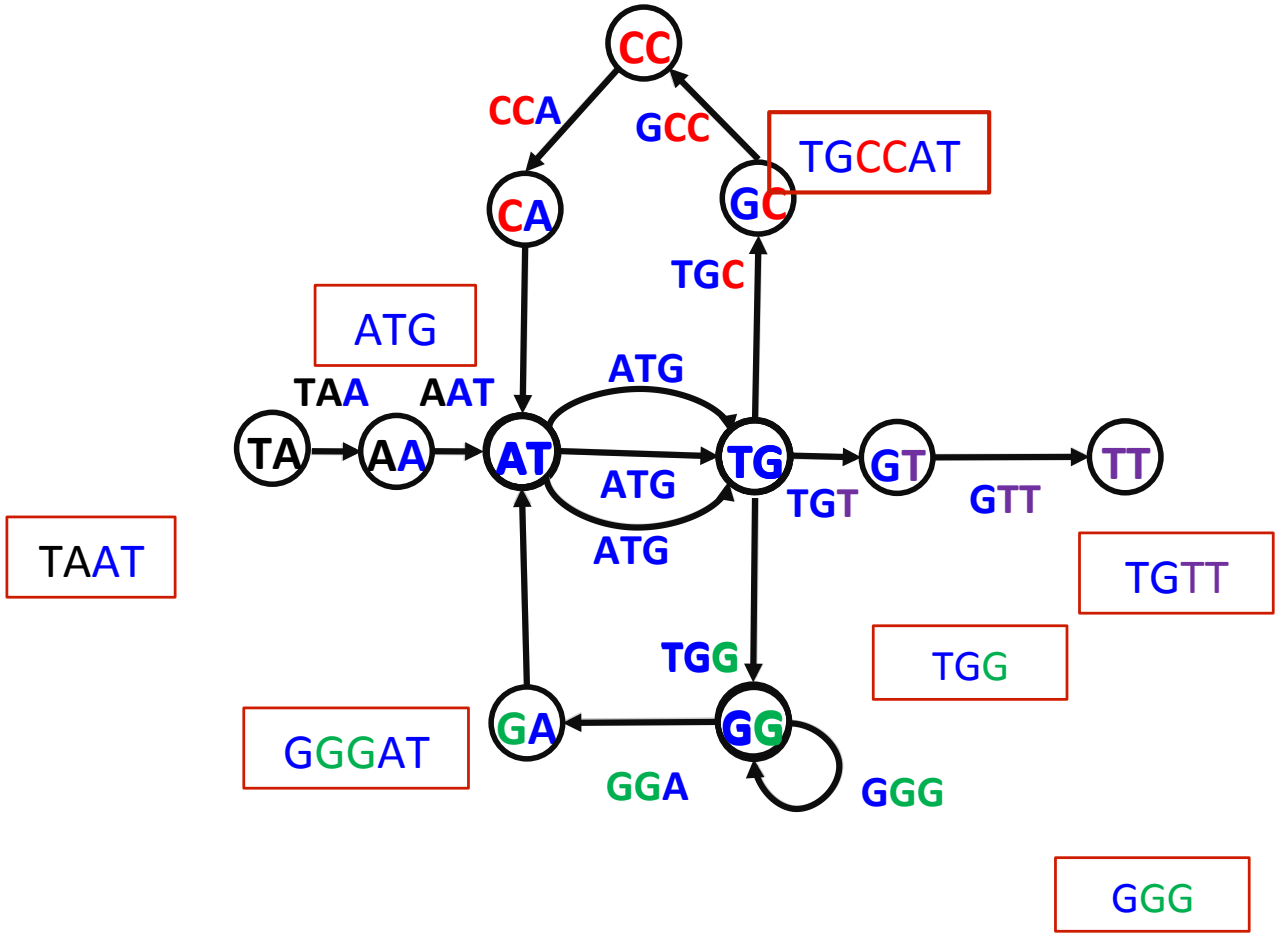
Multiple Eulerian Paths



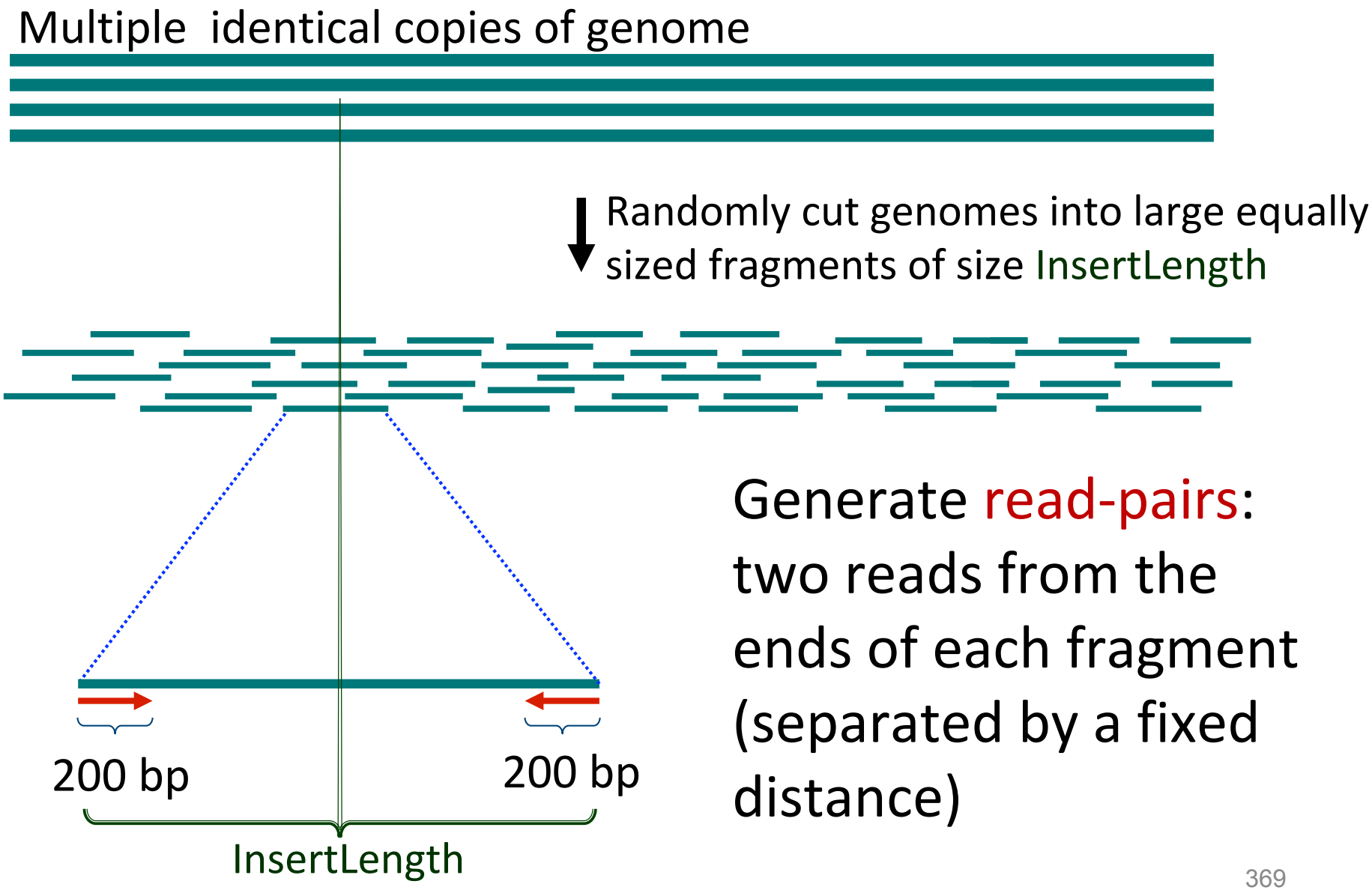


Breaking Genome into Contigs

TA TGCCATGGGATGTT
A



DNA Sequencing with Read-pairs



Generate read-pairs: two reads from the ends of each fragment (separated by a fixed distance)

From *k*-mers to Paired *k*-mers

A paired k-mer is a pair of k-mers at a fixed distance d apart in Genome.
E.g. TCA and TCC are at distance d=11 apart.

Disclaimers:
1. In reality, Read1 and Read2 are typically sampled from different strands:
(→ ..... ← rather than → ..... →)
2. In reality, the distance d between reads is measured with errors.

What is PairedComposition(**TAATGCCATGGGATGTT**)?

```
TAA GCC
 AAT CCA
  ATG CAT
   TGC ATG
    GCC TGG
     CCA GGG
      CAT GGA
       ATG GAT
        TGG ATG
         GGG TGT
          GGA GTT
```

Representing a paired 3-mer **TAA** **GCC** as a 2-line expression:

```
TAA
GCC
```

```
TAA  AAT  ATG  TGC  GCC  CCA  CAT  ATG  TGG  GGG  GGA
GCC  CCA  CAT  ATG  TGG  GGG  GGA  GAT  ATG  TGT  GTT
```

PairedComposition(**TAATGCCATGGGATGTT**)

```
TAA GCC
 AAT CCA
  ATG CAT
   TGC ATG
    GCC TGG
     CCA GGG
      CAT GGA
       ATG GAT
        TGG ATG
         GGG TGT
          GGA GTT
```

```
TAA  AAT  ATG  TGC  GCC  CCA  CAT  ATG  TGG  GGG  GGA
GCC  CCA  CAT  ATG  TGG  GGG  GGA  GAT  ATG  TGT  GTT

AAT  ATG  ATG  CAT  CCA  GCC  GGA  GGG  TAA  TGC  TGG
CCA  CAT  GAT  GGA  GGG  TGG  GTT  TGT  GCC  ATG  ATG
```

Representing PairedComposition in lexicographic order

String Reconstruction from Read-Pairs Problem

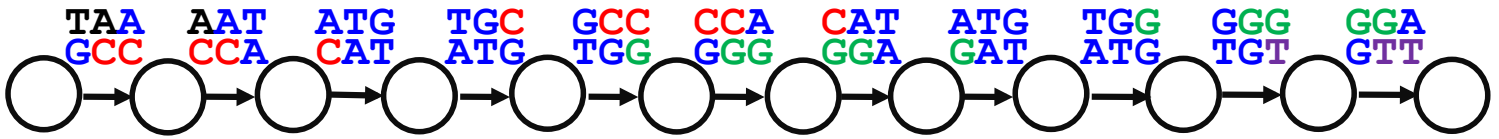
String Reconstruction from Read-Pairs Problem. Reconstruct a string from its paired *k*-mers.

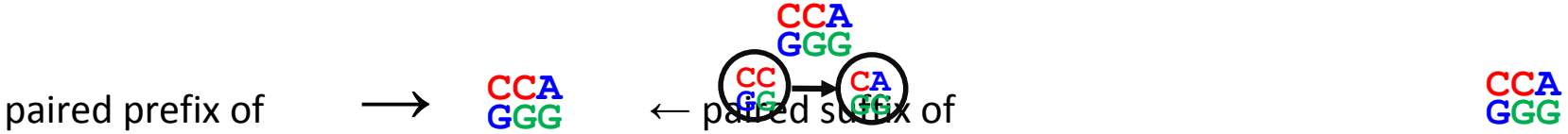
- **Input.** A collection of paired *k*-mers.
- **Output.** A string *Text* such that *PairedComposition*(*Text*) is equal to the collection of paired k-mers.

How Would de Bruijn Assemble Paired *k*-mers?

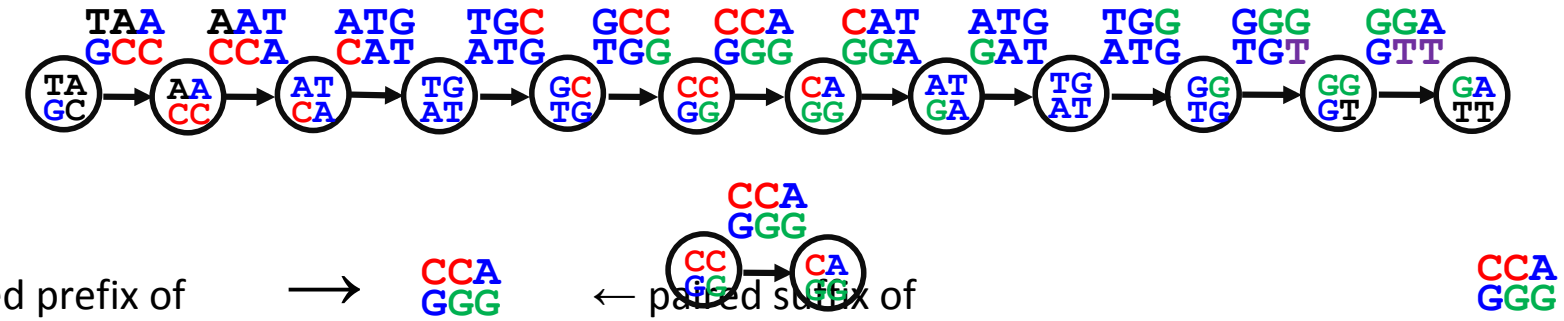
Representing Genome **TAATGCCATGGGATGTT** as a Path

```
TAA GCC
 AAT CCA
  ATG CAT
   TGC ATG
    GCC TGG
     CCA GGG
      CAT GGA
       ATG GAT
        TGG ATG
         GGG TGT
          GGA GTT
```

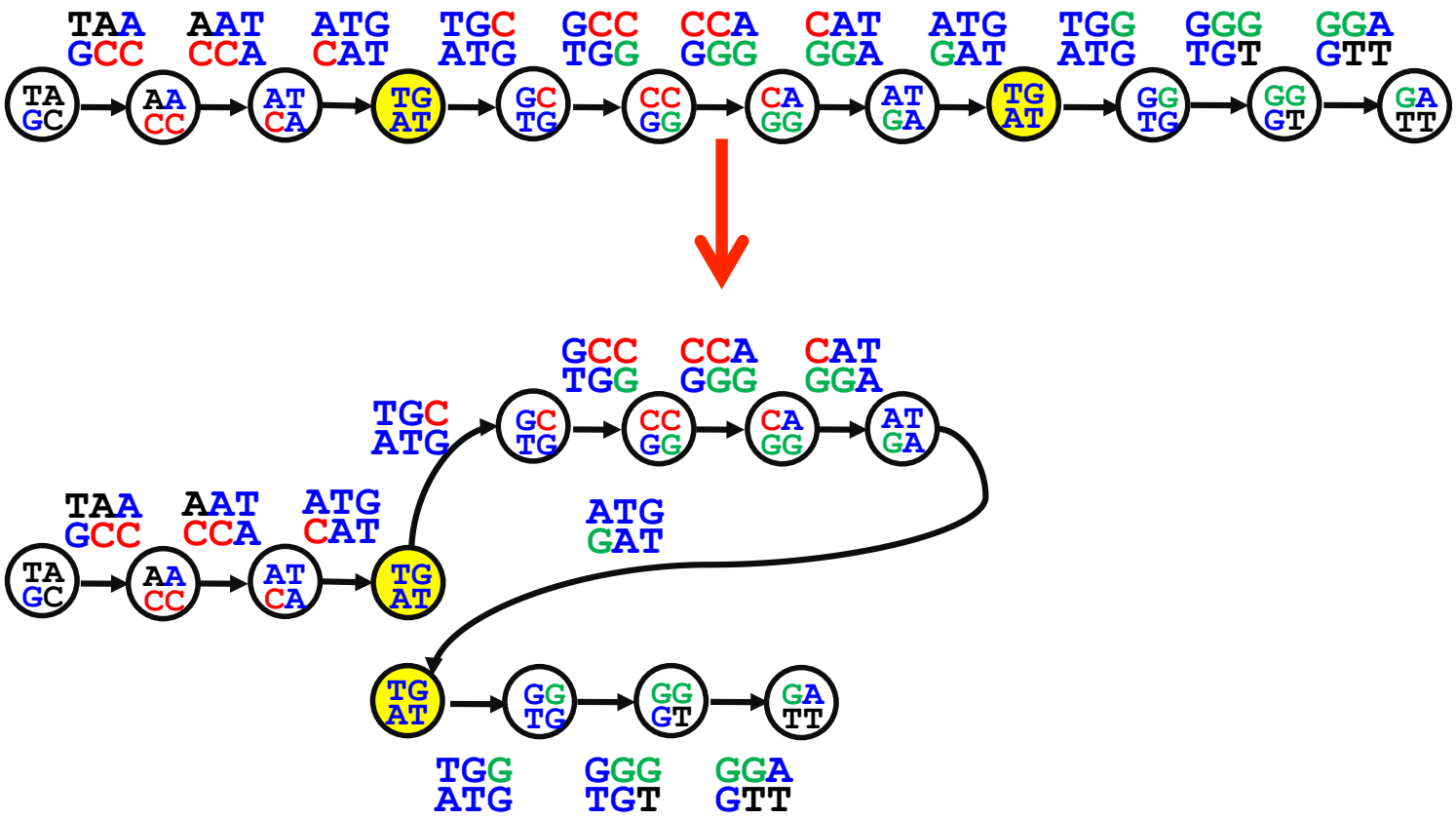




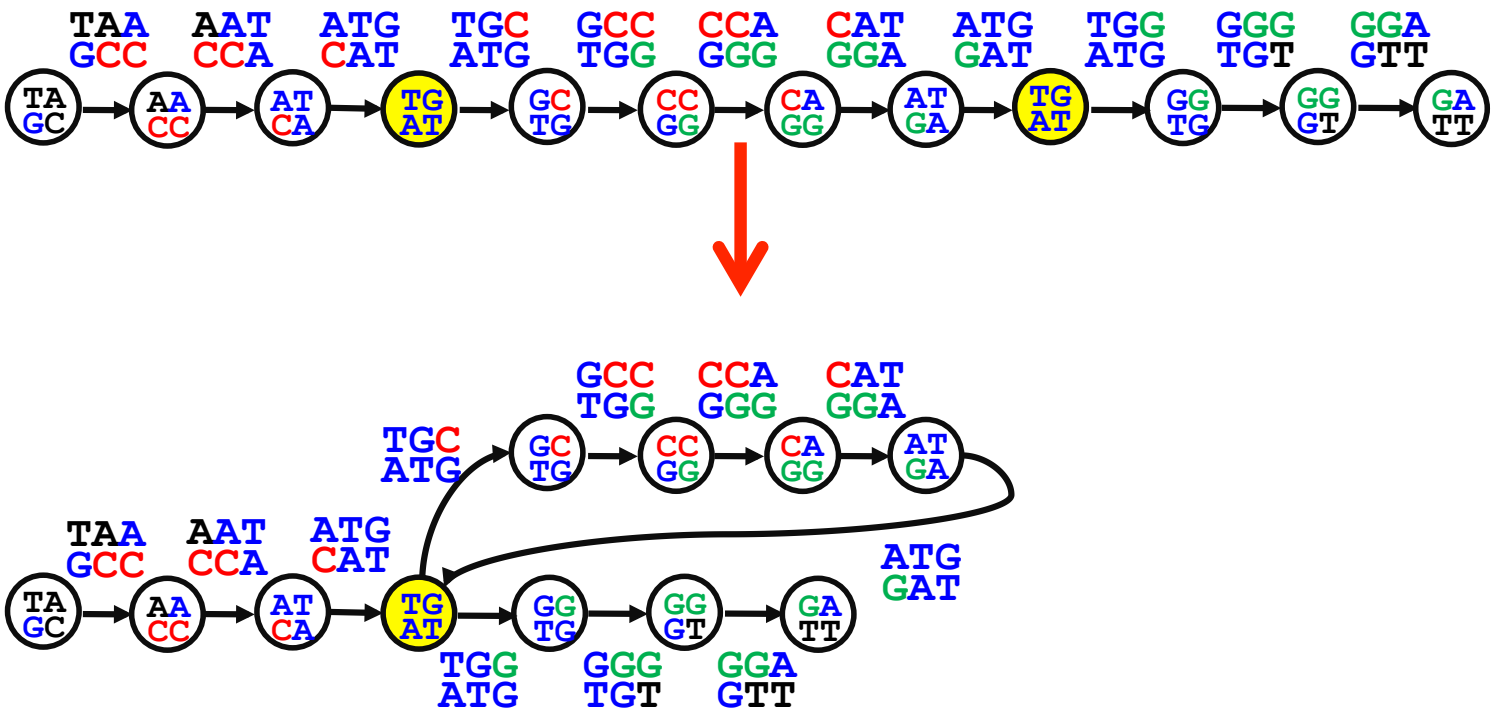
Labeling Nodes by Paired Prefixes and Suffixes



Glue nodes with identical labels

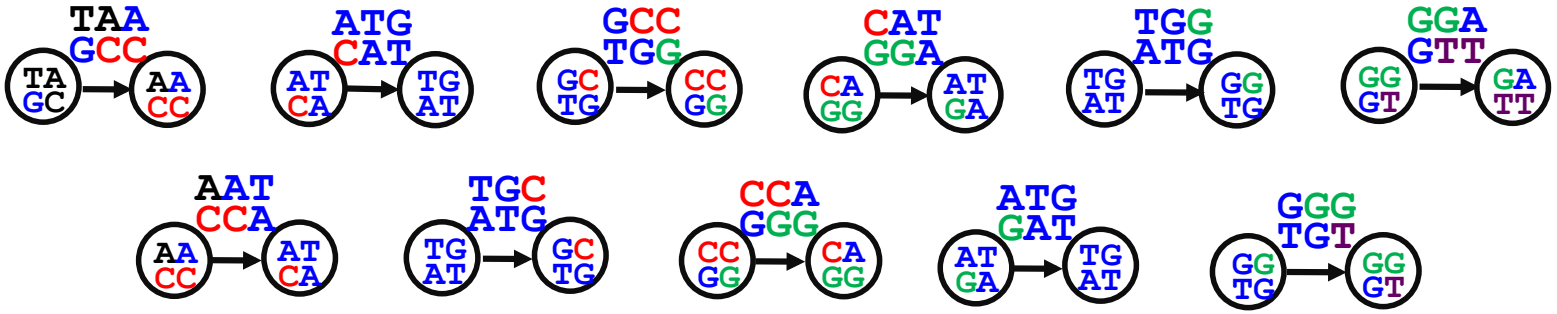


Glue nodes with identical labels



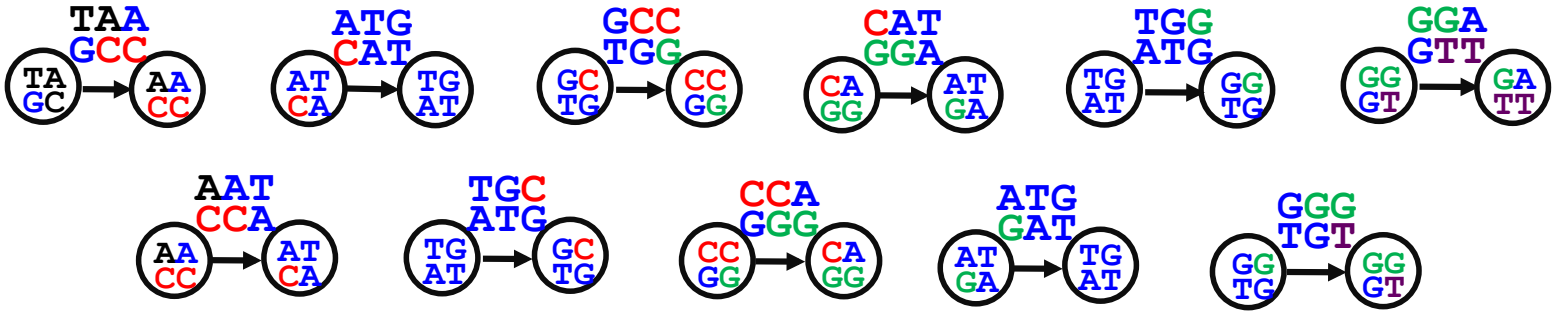
Paired de Bruijn Graph from the Genome

Constructing Paired de Bruijn Graph



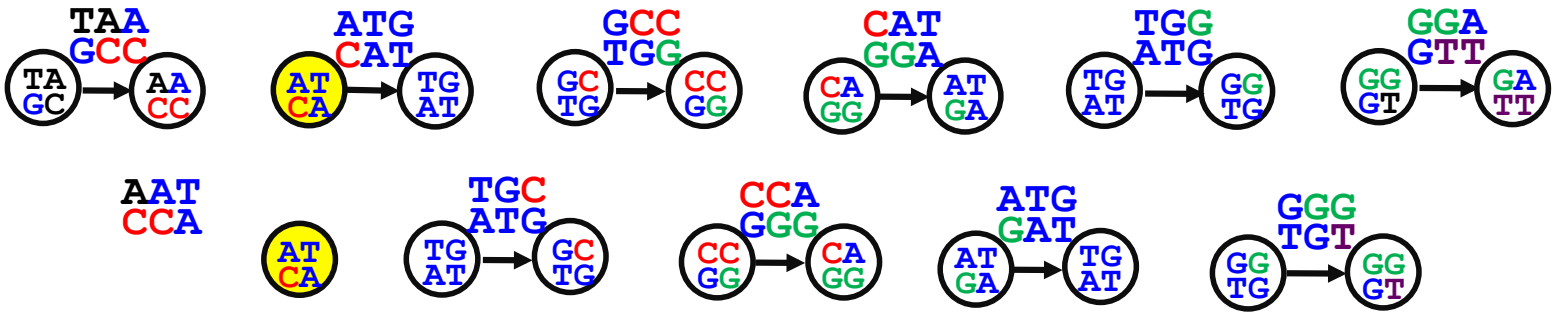
paired prefix of → CCA / GGG ← paired suffix of CCA / GGG

Constructing Paired de Bruijn Graph

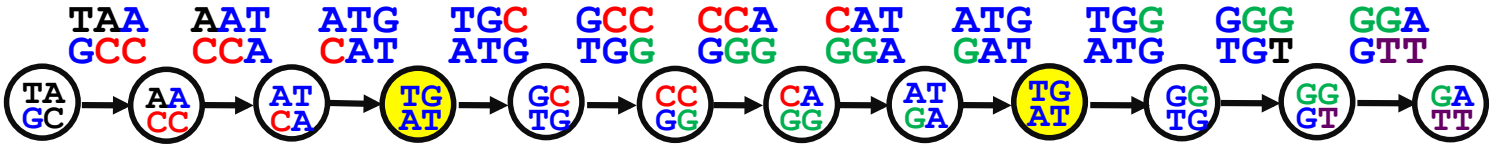


- **Paired de Bruijn graph for a collection of paired *k*-mers:**
  - Represent every paired *k*-mer as an edge between its paired prefix and paired suffix.
  - Glue **ALL** nodes with identical labels.

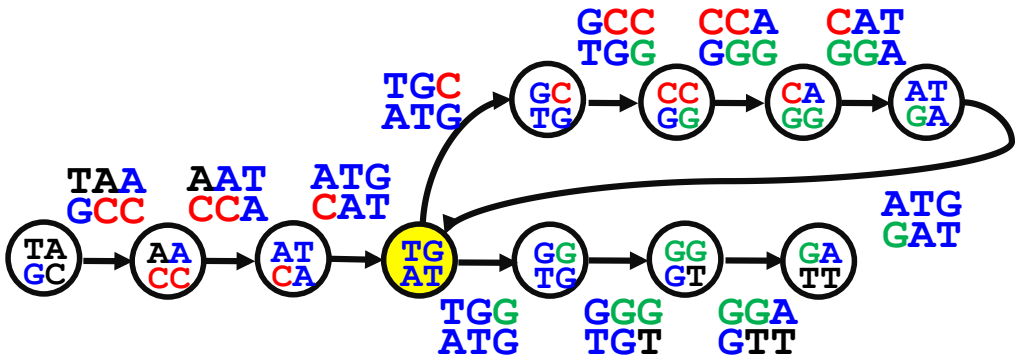
Constructing Paired de Bruijn Graph



We Are Not Done with Gluing Yet



Constructing Paired de Bruijn Graph



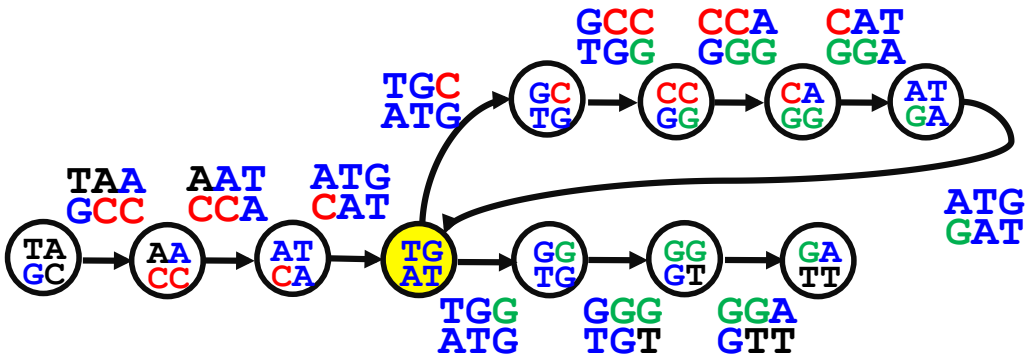
Paired de Bruijn Graph from read-pairs

- **Paired de Bruijn graph for a collection of paired *k*-mers:**
  - Represent every paired *k*-mer as an edge between its paired prefix and paired suffix.
  - Glue **ALL** nodes with identical labels.

Which Graph Represents a Better Assembly?

Unique genome reconstruction

TAATGCCATGGGATGTT

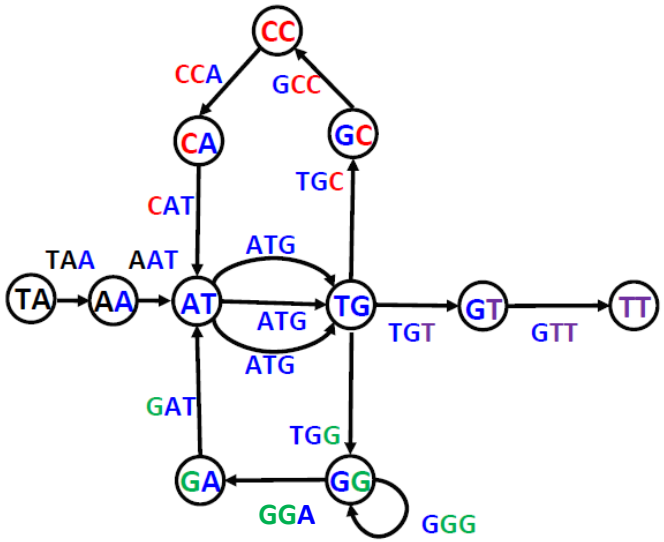


Paired de Bruijn Graph

Multiple genome reconstructions

TAATGCCATGGGATGTT

TAATGGGATGCCATGTT



De Bruijn Graph

Some Ridiculously Unrealistic Assumptions

- Perfect coverage of genome by reads (every *k*-mer from the genome is represented by a read)
- Reads are error-free.
- Multiplicities of *k*-mers are known
- Distances between reads within read-pairs are exact.

Some Ridiculously Unrealistic Assumptions

- **Imperfect** coverage of genome by reads (every *k*-mer from the genome is represented by a read)
- Reads are **error-prone**.
- Multiplicities of *k*-mers are **unknown.**
- Distances between reads within read-pairs are **inexact.**
- **Etc., etc., etc.**

1st Unrealistic Assumption: Perfect Coverage

```
atgccgtatggacaacgact
atgccgtatg
  gccgtatgga
     gtatggacaa
          gacaacgact
```

250-nucleotide reads generated by Illumina technology capture only a small fraction of 250-mers from the genome, thus violating the key assumption of the de Bruijn graphs.

Breaking Reads into Shorter *k*-mers

```
atgccgtatggacaacgact
atgccgtatg
  gccgtatgga
     gtatggacaa
          gacaacgact
```

```
atgccgtatggacaacgact
atgcc
 tgccg
  gccgt
   ccgta
    cgtat
     gtatg
      tatgg
       atgga
        tggac
         ggaca
          gacaa
           acaac
            caacg
             aacga
              acgac
               cgact
```

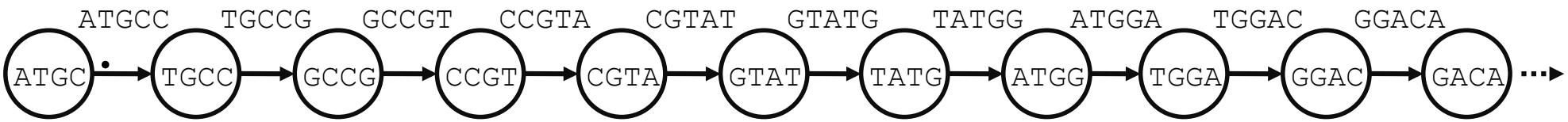
2nd Unrealistic Assumption: Error-free Reads

```
atgccgtatggacaacgact
atgccgtatg
  gccgtatgga
     gtatggacaa
          gacaacgact
    cgtaCggaca
```

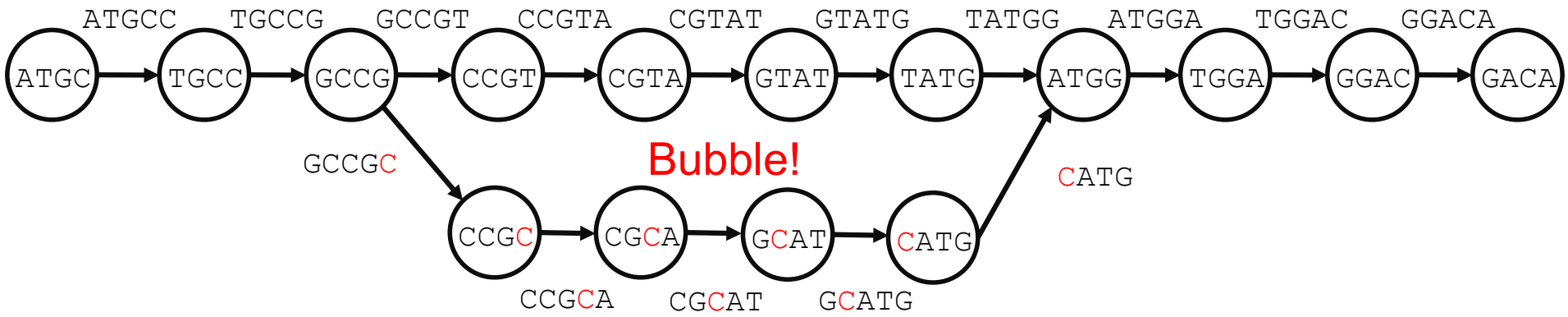
Erroneous read
(change of t into C)

```
atgccgtatggacaacgact
atgcc
 tgccg
  gccgt
   ccgta
    cgtat
     gtatg
      tatgg
       atgga
        tggac
         ggaca
          gacaa
           acaac
            caacg
             aacga
              acgac
               cgact
cgtaC
 gtaCg
  taCgg
   aCgga
    Cggac
```

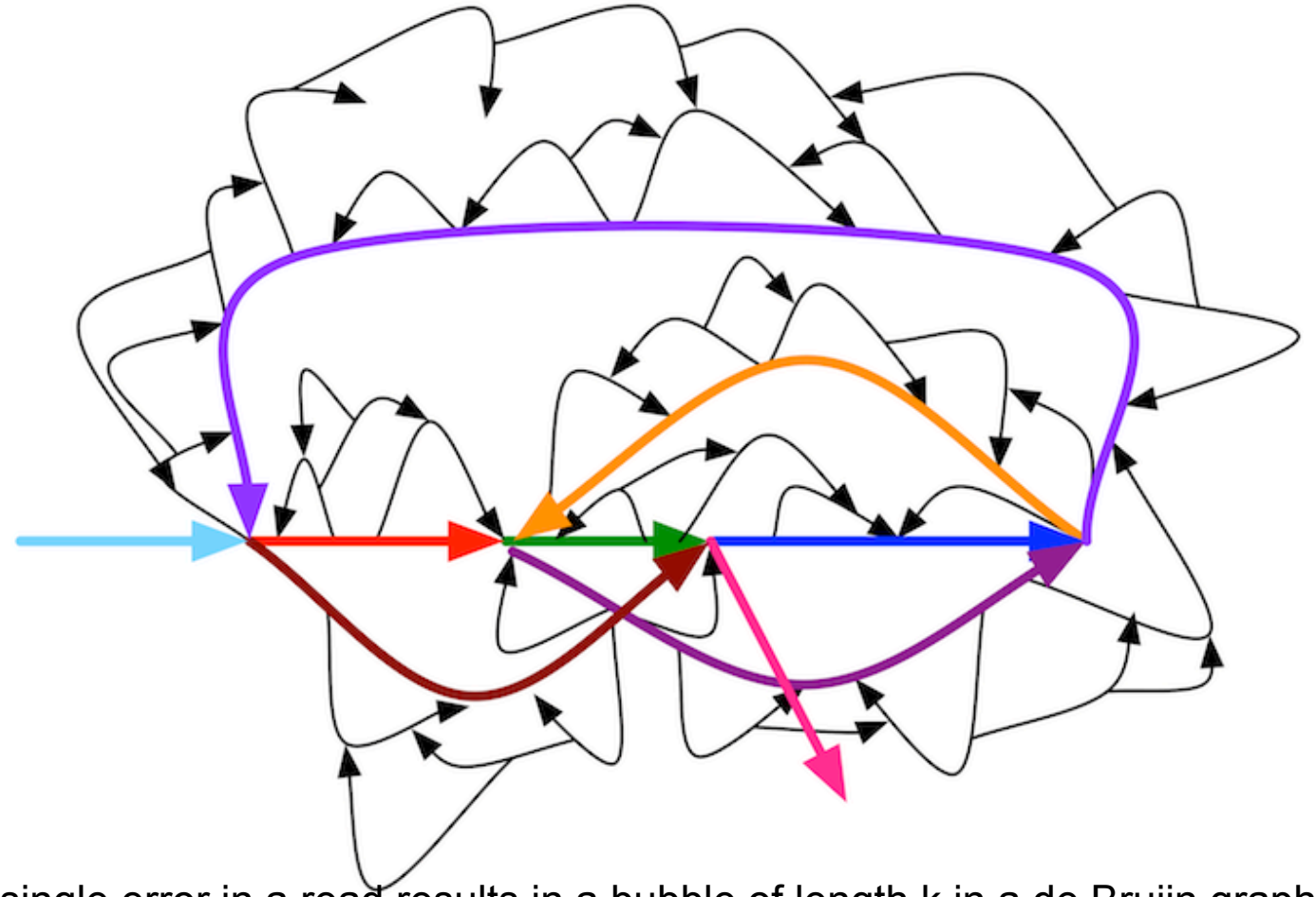
De Bruijn Graph of ATGGCGTGCAATG… Constructed from Error-Free Reads



Errors in Reads Lead to **Bubbles** in the De Bruijn Graph

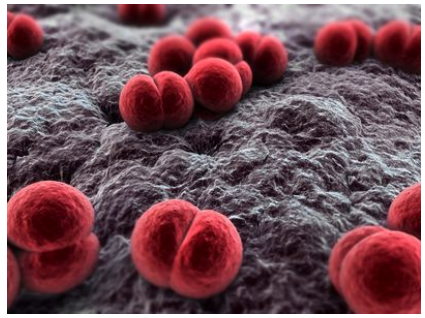


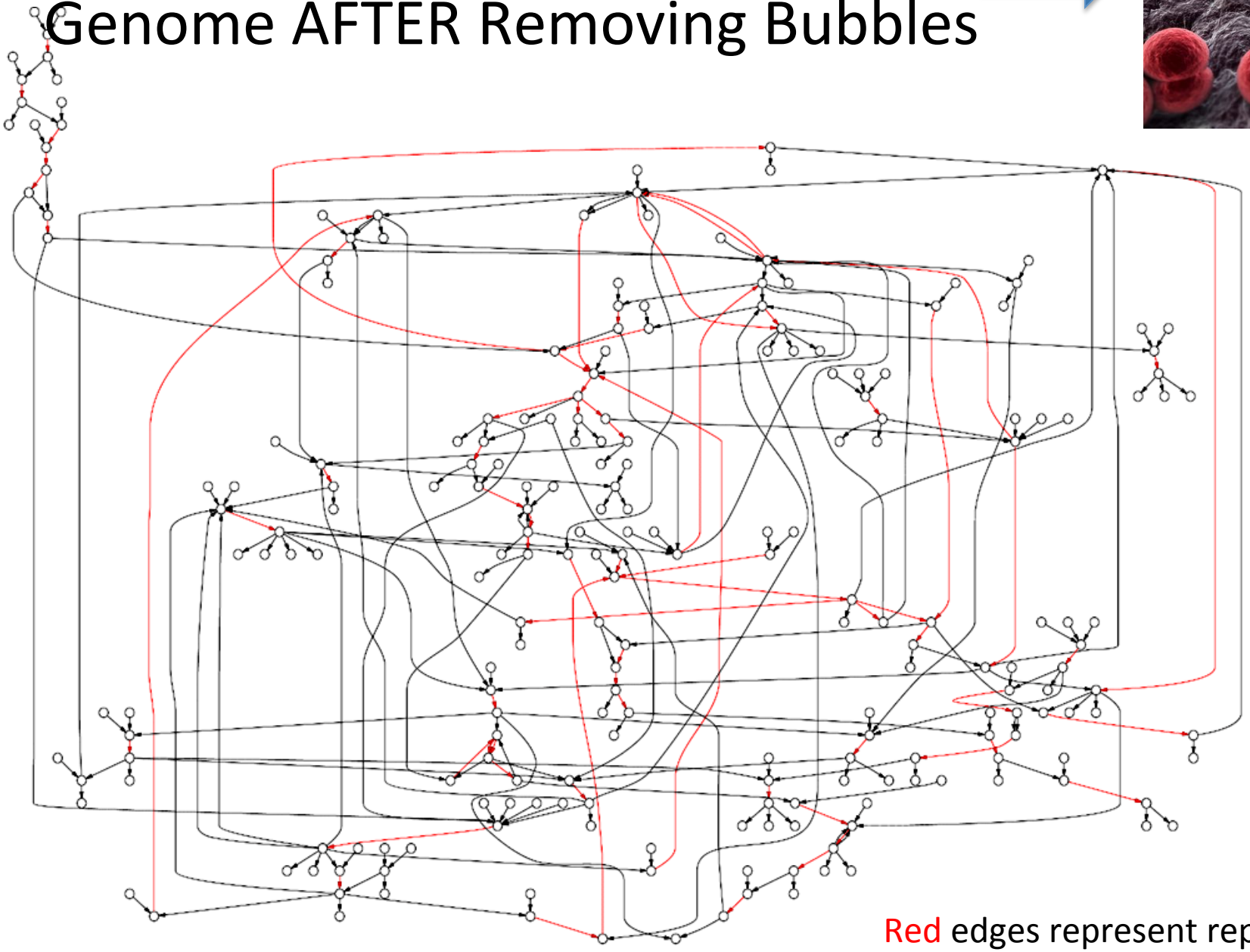
Bubble Explosion



A single error in a read results in a bubble of length k in a de Bruijn graph constructed from k-mers. Multiple errors in various reads may form longer bubbles, but since the error rate in reads is rather small (less than 1% per nucleotide in Illumina reads), most bubbles are small.

De Bruin Graph of *N. meningitidis* Genome AFTER Removing Bubbles





Red edges represent repeats

Example and RECAP

Input: GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTTT

Copy: GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTTT
GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTTT
GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTTT
GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTTT

Fragment: GGCGTCTA TATCTCGG CTCTAGGCCCTC ATTTTTT
GGC GTCTATAT CTCGGCTCTAGGCCCTCA TTTTTT
GGCGTC TATATCT CGGCTCTAGGCCCT CATTTTTT
GGCGTCTAT ATCTCGGCTCTAG GCCCTCA TTTTTT

CTAGGCCCTCAATTTTT
CTCTAGGCCCTCAATTTTT
GGCTCTAGGCCCTCATTTTTT
CTCGGCTCTAGCCCCTCATTTT
TATCTCGACTCTAGGCCCTCA
TATCTCGACTCTAGGCC
TCTATATCTCGGCTCTAGG
GGCGTCTATATCTCG
GGCGTCGATATCT
GGCGTCTATATCT
GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTTT

177 nucleotides

35 nucleotides

Average coverage = 177 / 35 ≈ 7x

Example and RECAP

"*k*-mer" is a substring of length *k*

S: GGCGATTCATCG

A 4-mer of *S*: ATTC

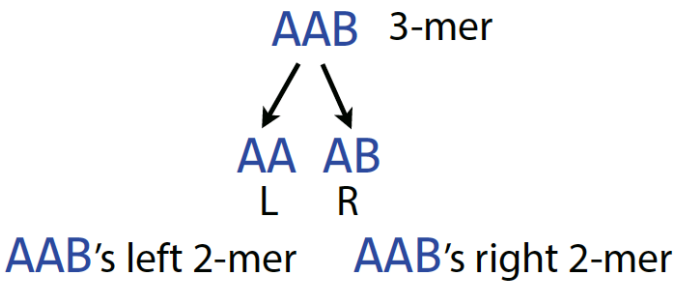
All 3-mers of *S*:
GGC
GCG
CGA
GAT
ATT
TTC
TCA
CAT
ATC
TCG

mer: from Greek meaning "part"

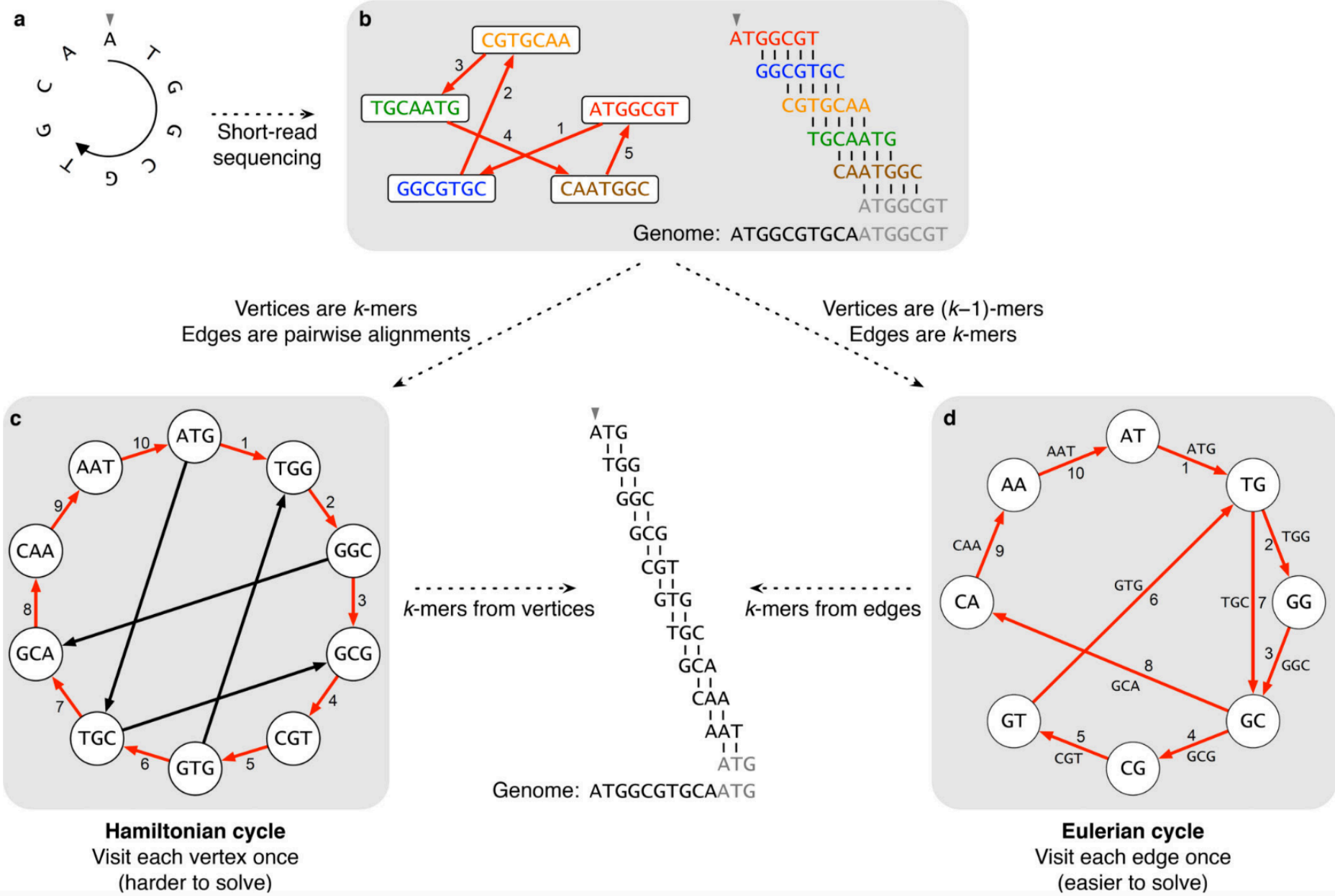
I'll use "*k*-1-mer" to refer to a substring of length *k* - 1

AAA, AAB, ABB, BBB, BBA

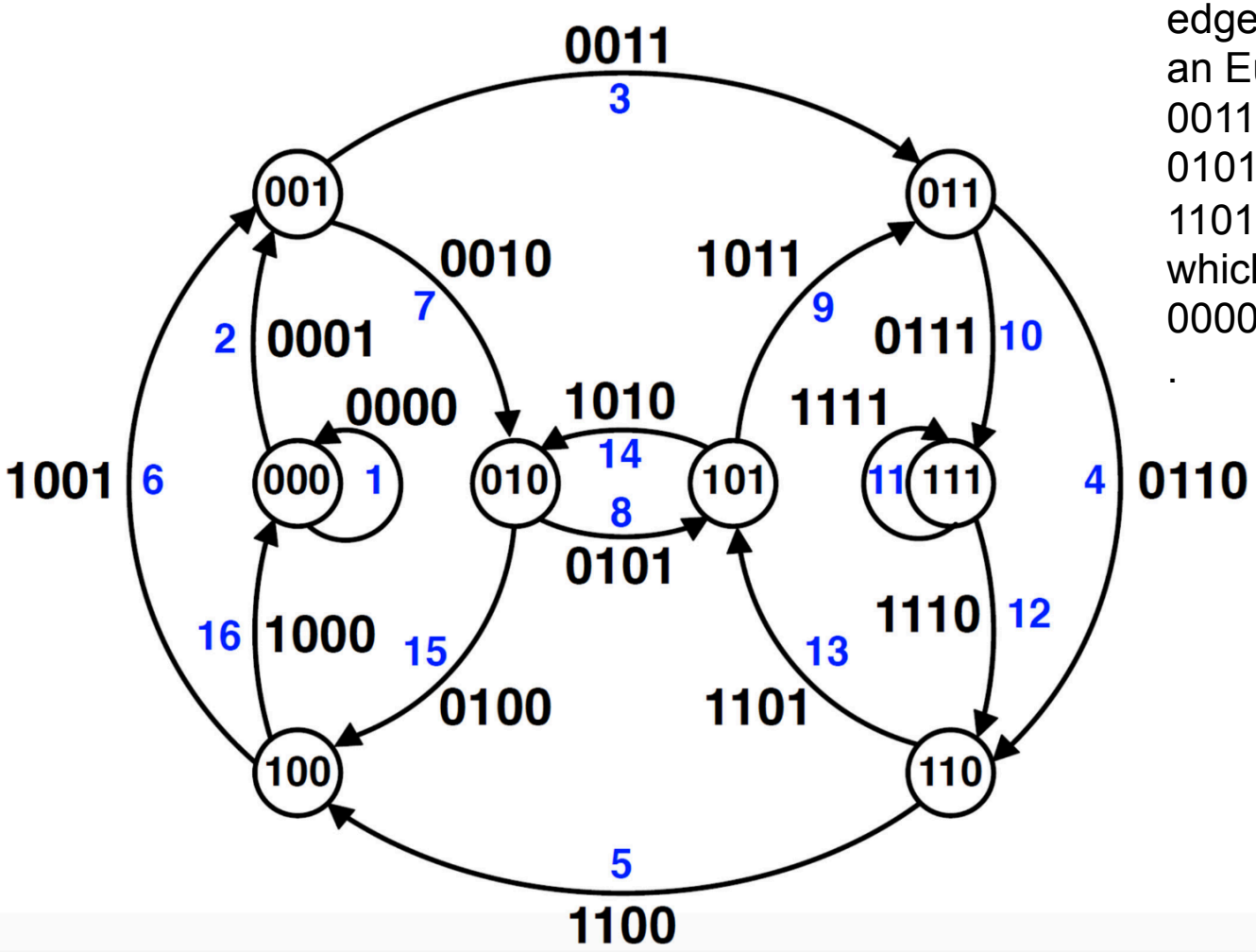
AAB is a *k*-mer (*k* = 3). AA is its *left* *k*-1-mer, and AB is its right *k*-1-mer.



Example and RECAP



Example and RECAP



The de Bruijn graph for k = 4 and a 2-character alphabet composed of the digits 0 and 1.
This graph has an Eulerian cycle since each node has indegree and outdegree equal to 2.
Following the blue numbered edges in order 1, 2, ..., 16 gives an Eulerian cycle 0000, 0001, 0011, 0110, 1100, 1001, 0010, 0101, 1011, 0111, 1111, 1110, 1101, 1010, 0100, 1000,
which spells the cyclic superstring 0000110010111101
.

Example and RECAP

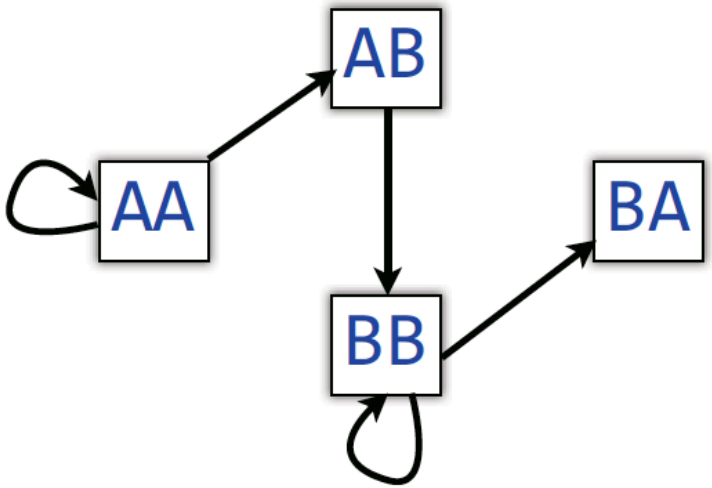
AAABBBA

take all 3-mers: AAA, AAB, ABB, BBB, BBA

form L/R 2-mers: AA, AA, AA, AB, AB, BB, BB, BB, BB, BA

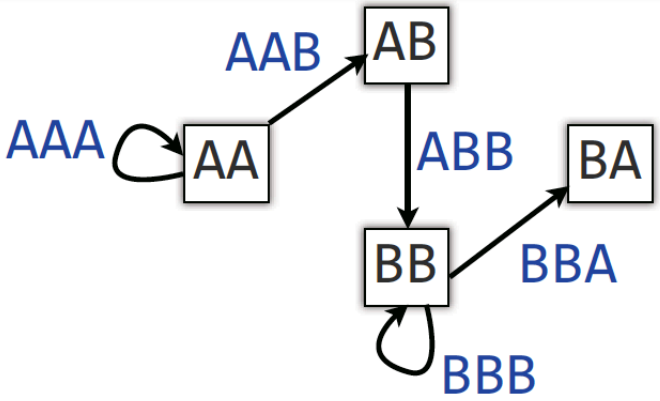
L R L R L R L R L R

Let 2-mers be nodes in a new graph. Draw a directed edge from each left 2-mer to corresponding right 2-mer:

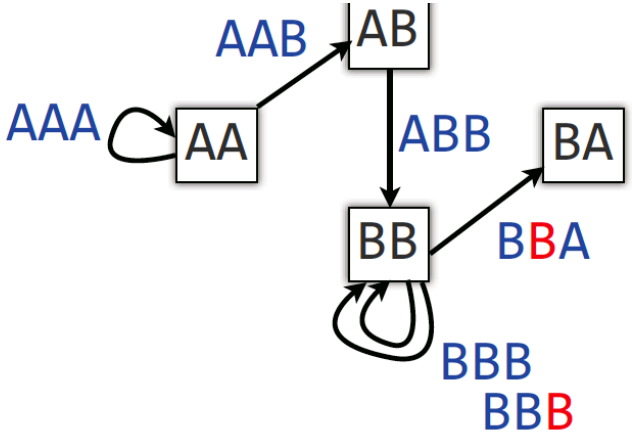


Each *edge* in this graph corresponds to a length-3 input string

Example and RECAP



An edge corresponds to an overlap (of length *k*-2) between two *k*-1 mers.
More precisely, it corresponds to a *k*-mer from the input.



If we add one more B to our input string: AAABBBBA, and rebuild the
De Bruijn graph accordingly, we get a *multiedge*.

Example and RECAP

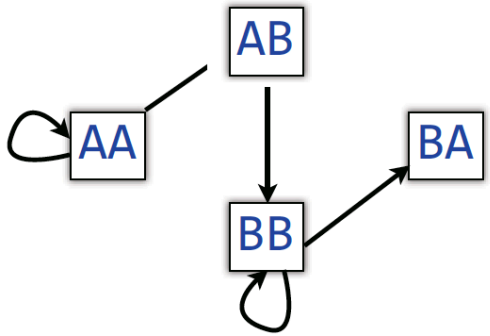
Node is *balanced* if indegree equals outdegree

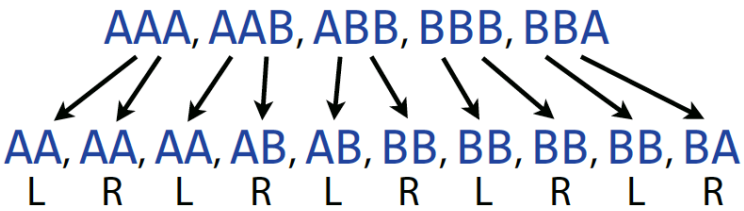
Node is *semi-balanced* if indegree differs from outdegree by 1

Graph is *connected* if each node can be reached by some other node

Eulerian walk visits each edge exactly once

Not all graphs have Eulerian walks. Graphs that do are *Eulerian*.
(For simplicity, we won't distinguish Eulerian from semi-Eulerian.)





Is it Eulerian? Yes

Argument 1: AA → AA → AB → BB → BB → BA

Argument 2: AA and BA are semi-balanced, AB and BB are balanced

De Bruijn graph

Example and RECAP

A procedure for making a De Bruijn graph for a genome

Assume *perfect sequencing* where each length-*k* substring is sequenced exactly once with no errors

Pick a substring length *k*: 5

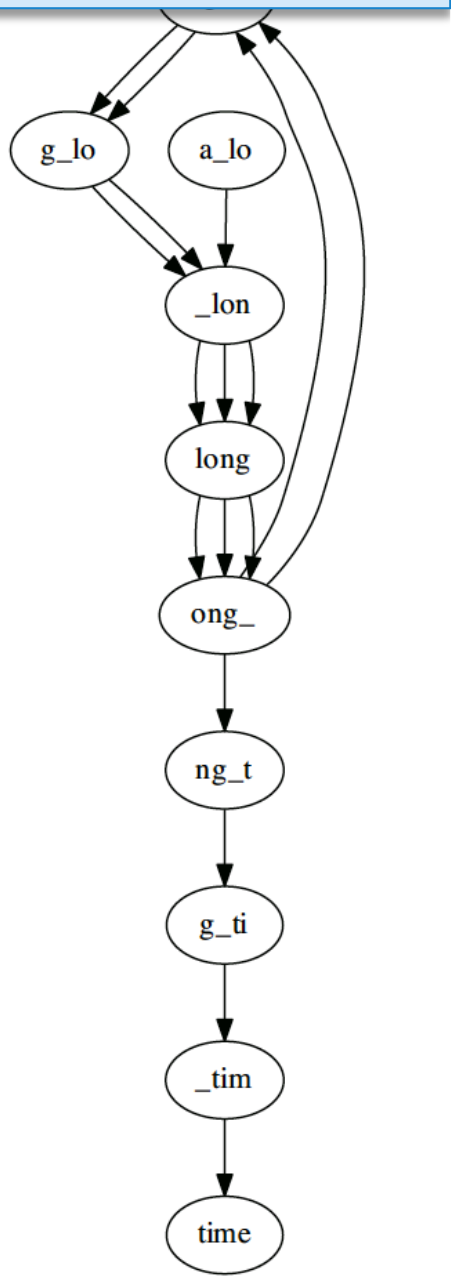
Start with each read: `a_long_long_long_time`

Take each *k* mer and split into left and right *k*-1 mers

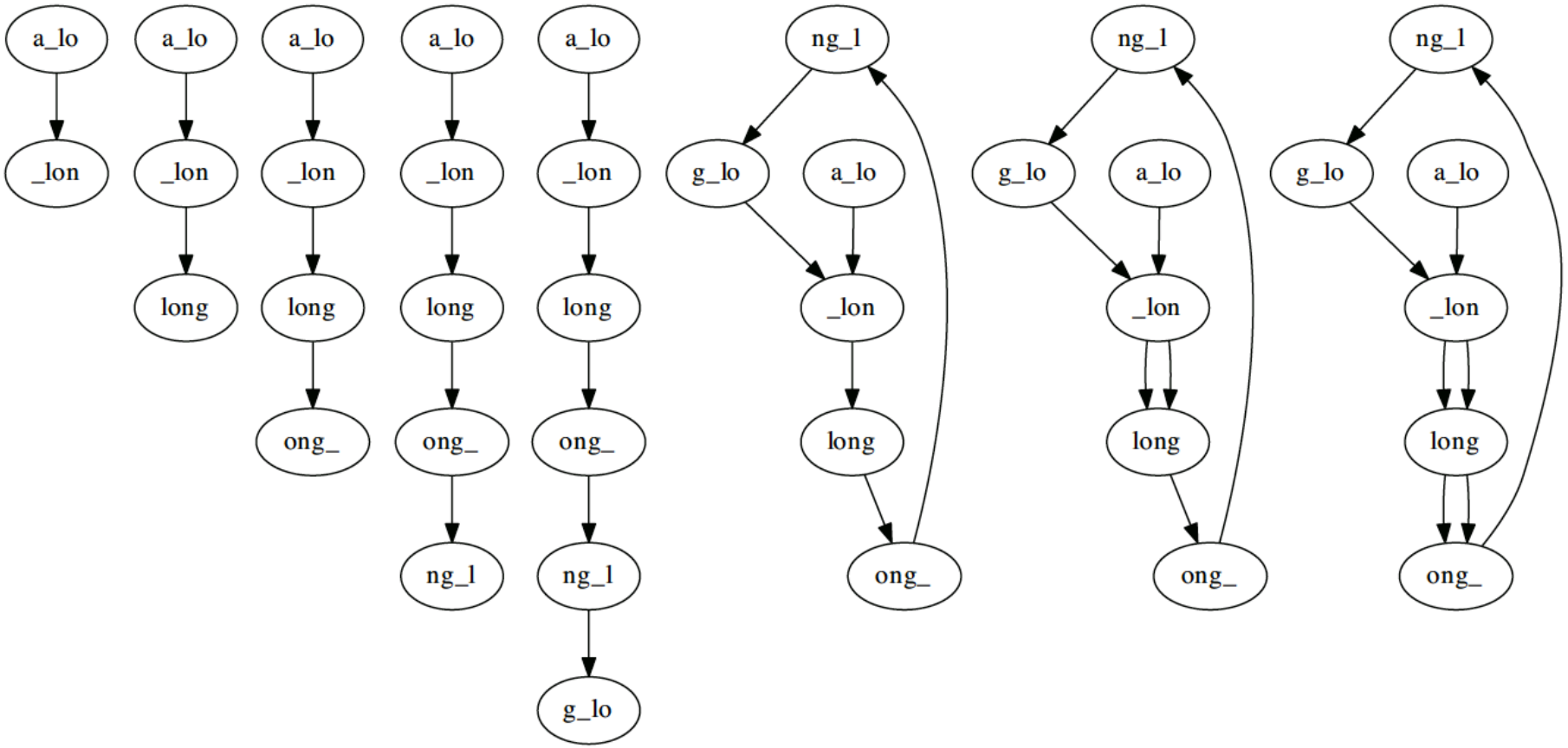
`long_`

`long ong_`

Add k-1 mers as nodes to De Bruijn graph (if not already there), add edge from left *k*-1 mer to right *k*-1 mer



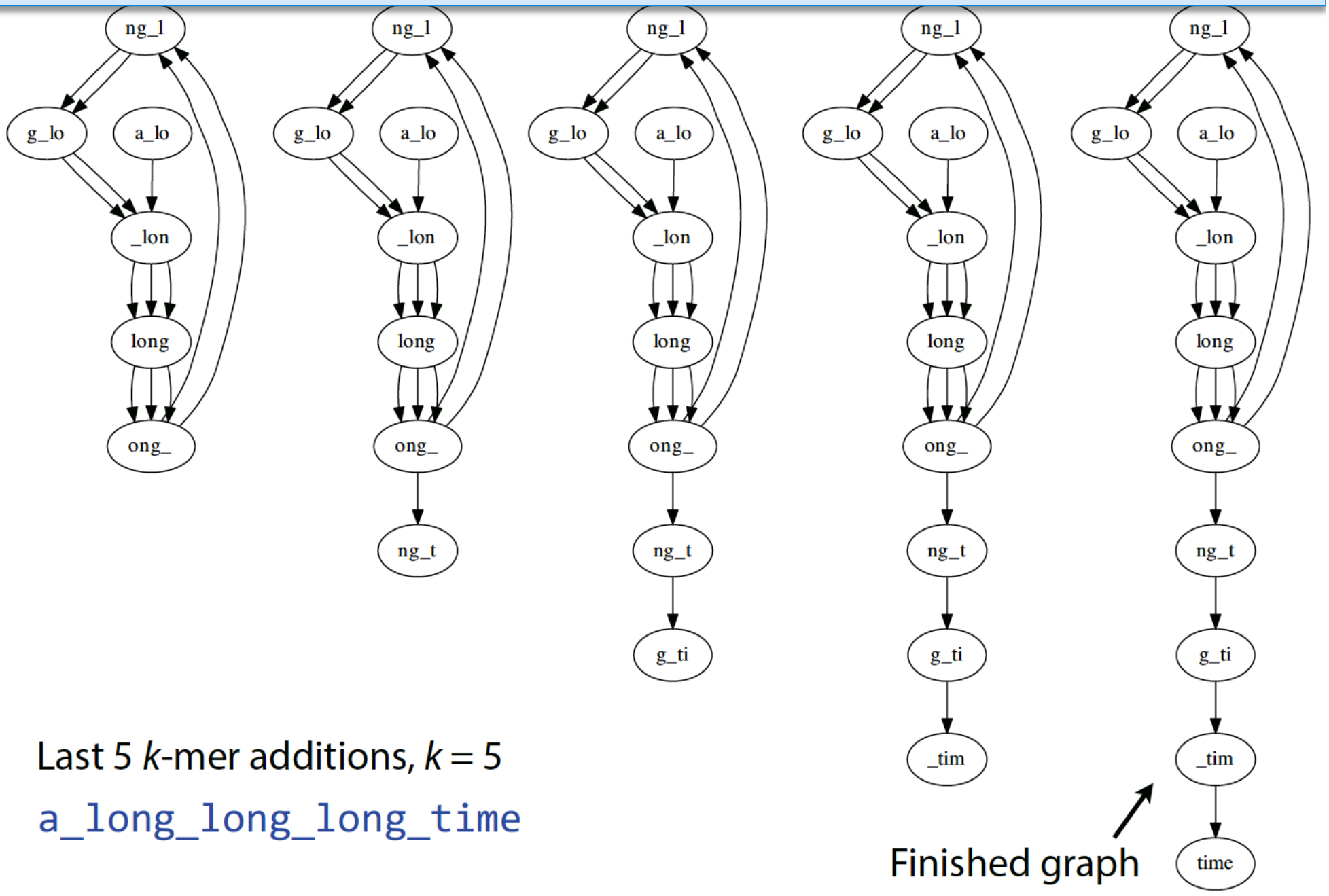
Example and RECAP



First 8 *k*-mer additions, $k = 5$

`a_long_long_long_time`

Example and RECAP



Last 5 *k*-mer additions, $k = 5$

`a_long_long_long_time`

Finished graph

De Bruijn graph

Example and RECAP

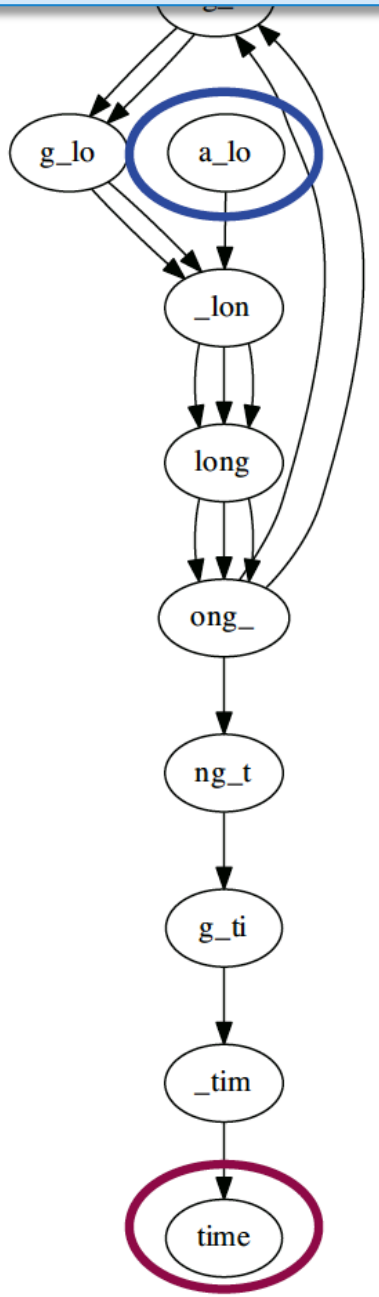
With perfect sequencing, this procedure always yields an Eulerian graph. Why?

Node for *k*-1-mer from left end is semi-balanced with one more outgoing edge than incoming *

Node for *k*-1-mer at right end is semi-balanced with one more incoming than outgoing *

Other nodes are balanced since # times *k*-1-mer occurs as a left *k*-1-mer = # times it occurs as a right *k*-1-mer

* Unless genome is circular

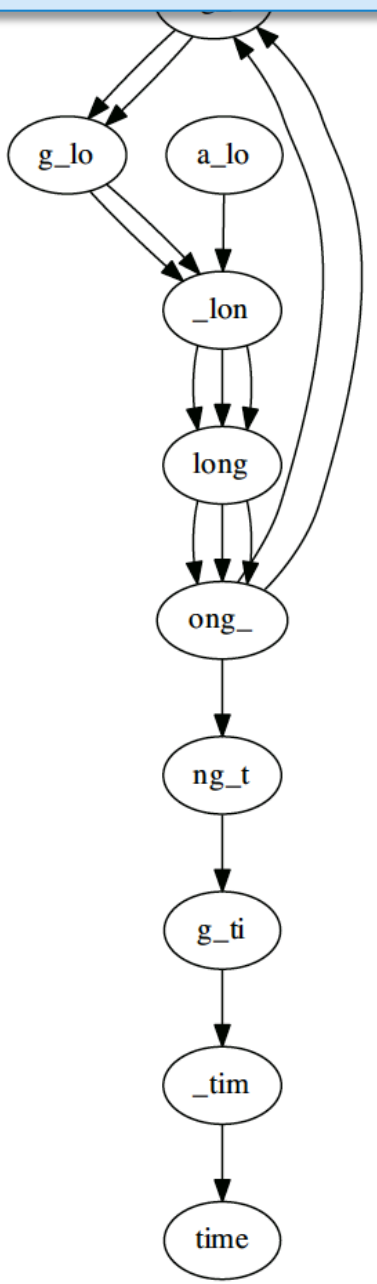


De Bruijn graph

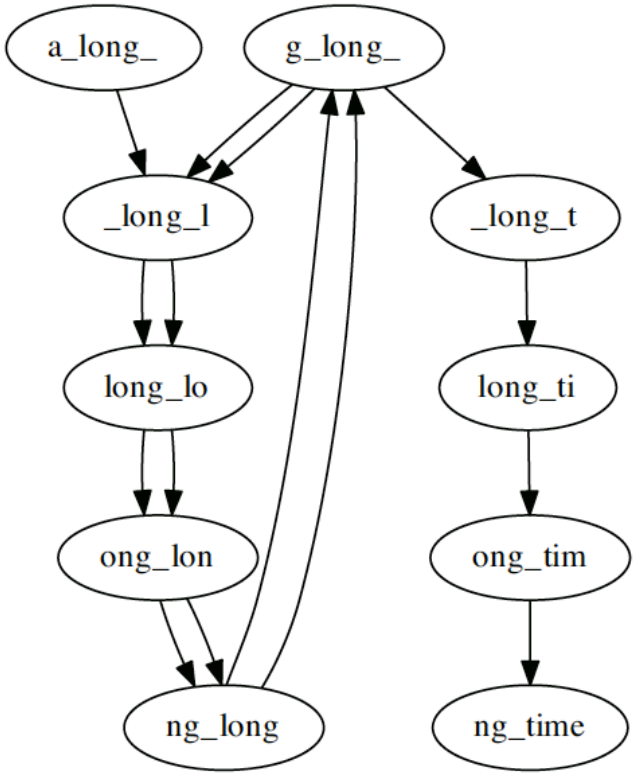
Example and RECAP

Assuming perfect sequencing, procedure yields graph with Eulerian walk that can be found efficiently.

We saw cases where Eulerian walk corresponds to the original superstring. Is this always the case?



Example and RECAP



How much work to build graph?

For each *k*-mer, add 1 edge and up to 2 nodes

Reasonable to say this is O(1) expected work

Assume hash map encodes nodes & edges

Assume *k*-1-mers fit in O(1) machine words, and hashing O(1) machine words is O(1) work

Querying / adding a key is O(1) expected work

O(1) expected work for 1 *k*-mer, O(*N*) overall

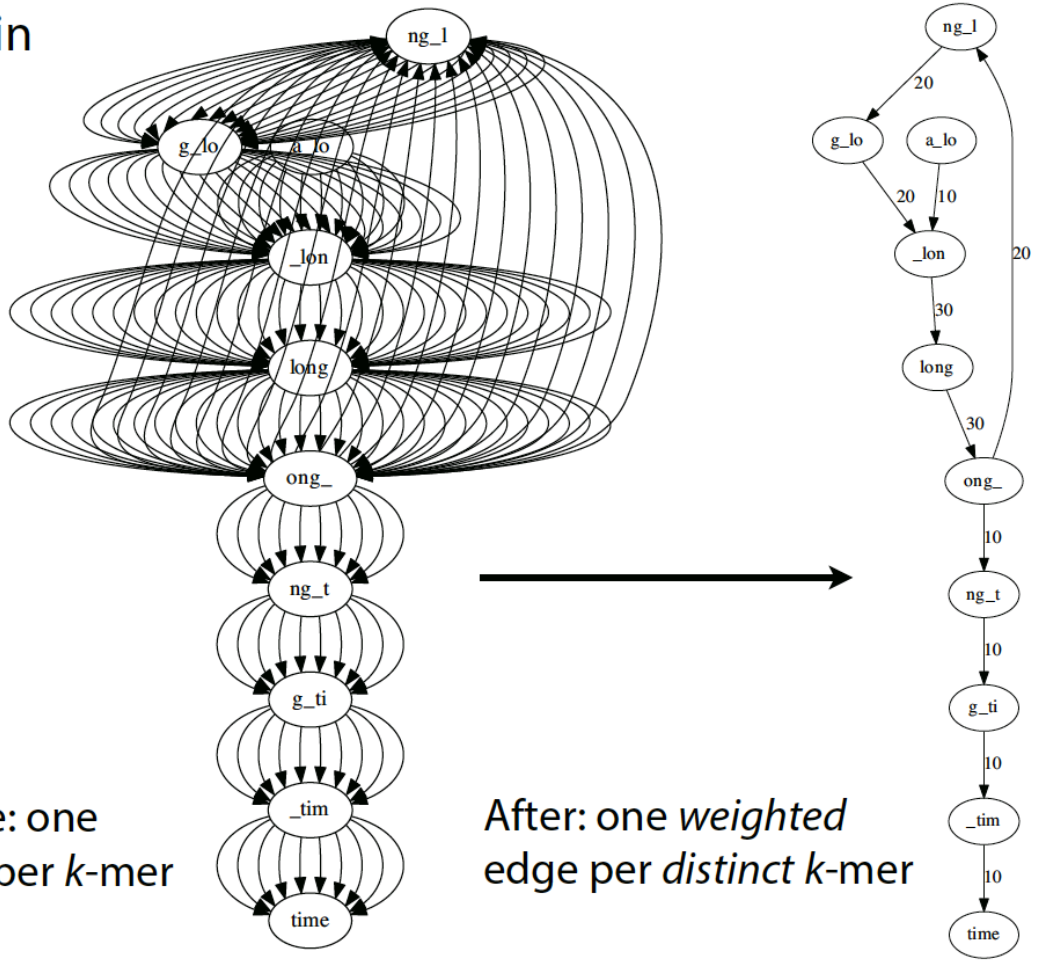
Example and RECAP

In typical assembly projects, average coverage is ~ 30 - 50

Same edge might appear in dozens of copies; let's use edge *weights* instead

Weight = # times *k*-mer occurs

Using weights, there's one *weighted* edge for each *distinct k*-mer



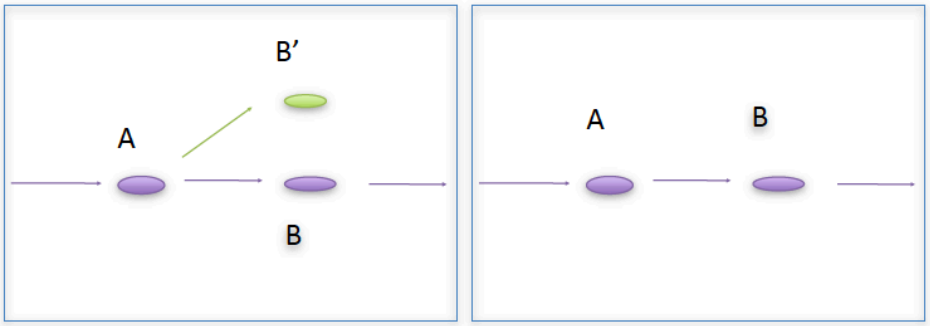
Before: one edge per *k*-mer

After: one *weighted* edge per *distinct k*-mer

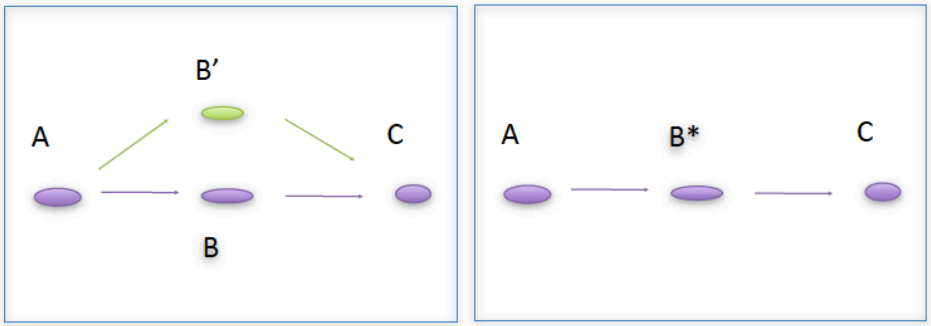
References: https://ocw.mit.edu/courses/biology/7-91j-foundations-of-computational-and-systems-biology-spring-2014/lecture-slides/MIT7_91JS14_Lecture6.pdf
http://nbviewer.jupyter.org/github/BenLangmead/comp-genomics-class/blob/master/notebooks/CG_deBruijn.ipynb

Example and RECAP

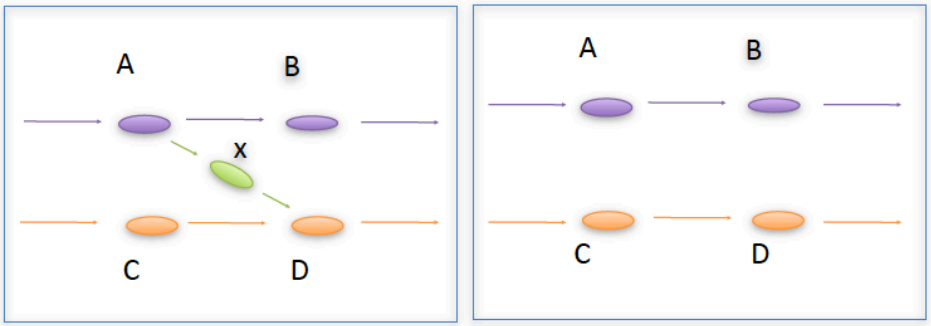
– Errors at end of read
  - Trim off ‘dead-end’ tips



– Errors in middle of read
  - Pop Bubbles



– Chimeric Edges
  - Clip short, low coverage nodes



Example and RECAP

"It was the best of times, it was the worst of times, it was the age of wisdom, it was the age of foolishness, it was the epoch of belief, it was the epoch of incredulity,.... "

Dickens, Charles. A Tale of Two Cities. 1859. London: Chapman Hall

itwasthebestoftimesitwastheworstoftimesitwastheageofwisdomitwastheageoffoolishness...

How do we assemble?

fincreduli geoffoolis ltwasthebe ltwasthebe geofwisdom itwastheep epochofinc timesitwas stheepocho nessitwast wastheageo theepochof stheepocho hofincredu estoftimes eoffoolish lishnessit hofbeliefi pochofincr itwasthewo twastheage toftimesit domitwasth ochofbelie eepochofbe eepochofbe astheworst chofincred theageofwi iefitwasth ssitwasthe astheepoch efitwasthe wisdomitwa ageoffooli twasthewor ochofbelie sdomitwast sitwasthea eepochofbe ffoolishne eofwisdomi hebestofti stheageoff twastheepo eworstofti stoftimesi theepochof esitwasthe heepochofi theepochof sdomitwast astheworst rstoftimes worstoftim stheepocho geoffoolis ffoolishne timesitwas lishnessit stheageoff eworstofti orstoftime fwisdomitw wastheageo heageofwis increduli ishnessitw twastheepo wasthewors astheepoch heworstoft ofbeliefit wastheageo heepochofi pochofincr heageofwis stheageofw fincreduli astheageof wisdomitwa wastheageo astheepoch olishnessi astheepoch itwastheep twastheage wisdomitwa fbeliefitw bestoftime epochofbel theepochof sthebestof lishnessit hofbeliefi ltwasthebe ishnessitw sitwasthew ageofwisdo twastheage esitwasthe twastheage shnessitwa fincreduli fbeliefitw theepochof mesitwasth domitwasth ochofbelie heageofwis oftimesitw stheepocho bestoftime twastheage foolishnes ftimesitwa thebestoft itwastheag theepochof itwasthewo ofbeliefit bestoftime mitwasthea imesitwast timesitwas orstoftime estoftimes twasthebes stoftimesi sdomitwast wisdomitwa theworstof astheworst sitwasthew theageoffo eepochofbe

...etc. to 10's of millions of reads

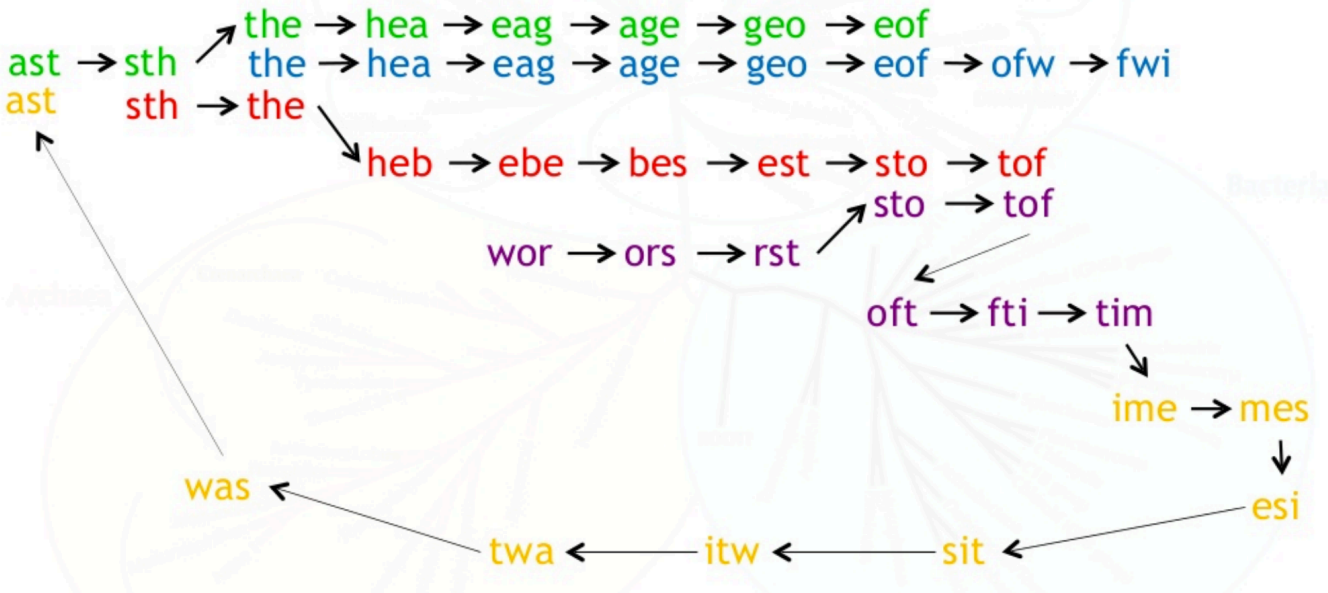
Example and RECAP

Step 1:
Convert reads into "Kmers"

Kmer: a substring of defined length

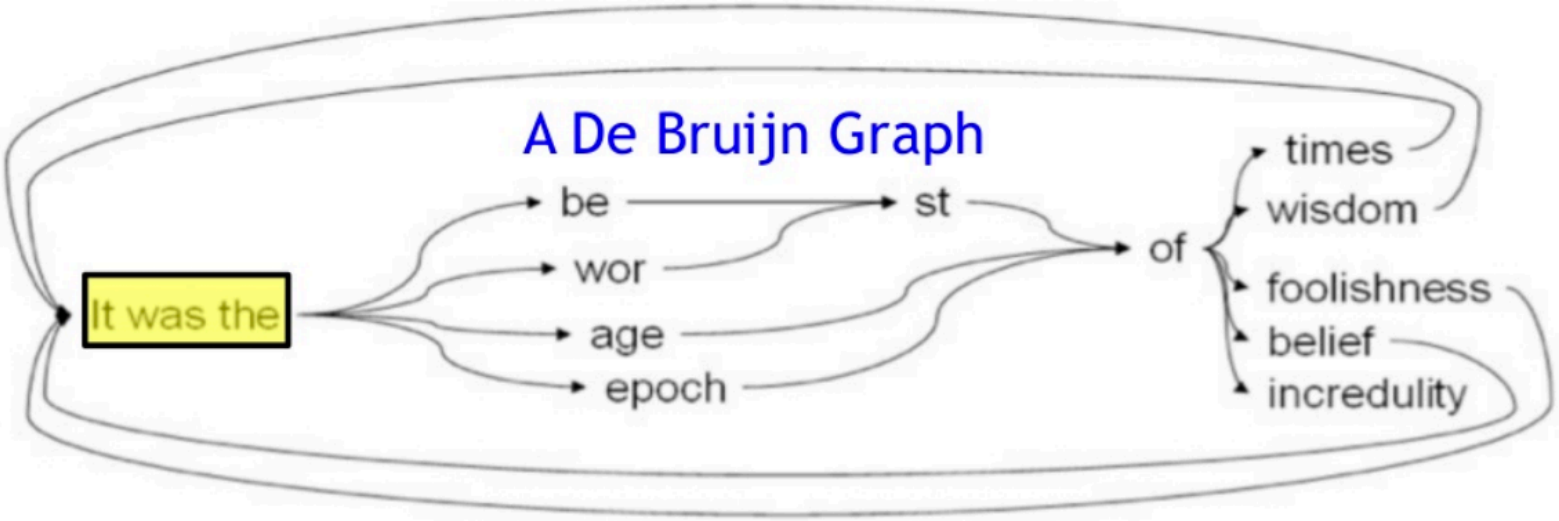
Reads:	theageofwi	sthebestof	astheageof	worstoftim	imesitwast
Kmers : (k=3)	the	sth	ast	wor	ime
	hea	the	sth	ors	mes
	eag	heb	the	rst	esi
	age	ebe	hea	sto	sit
	geo	bes	eag	tof	itw
	eof	est	age	oft	twa
	ofw	sto	geo	fti	was

Step 2:
Build a De-Bruijn graph from the kmers



Example and RECAP

Step 3:
Simplify the graph as much as possible:



De Bruijn assemblies 'broken' by repeats longer than kmer

"It was the best of times, it was the worst of times, it was the age of wisdom, it was the age of foolishness, it was the epoch of belief, it was the epoch of incredulity,.... "

Example and RECAP

The final assembly (k=3)

wor times itwasthe foolishness st wisdom

incredulity age epoch be of belief

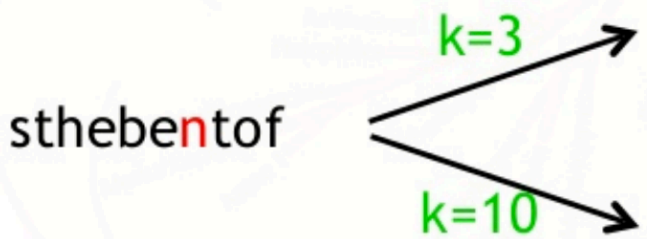
Repeat with a longer "kmer" length

A better assembly (k=20)

itwasthebestoftimesitwastheworstoftimesitwastheageofwisdomitwastheageoffoolis...

Why not always use longest 'k' possible?

Sequencing errors:



sth the ebe ent tof
heb ben nto

Mostly unaffected kmers

sthebentof

100% wrong kmer

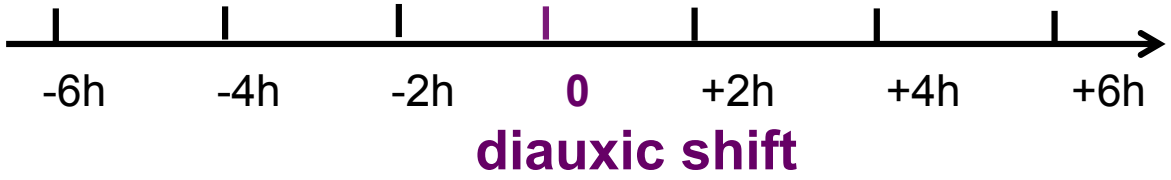
Slides from Presentation by Alicia Clum genomebiology.jgi-psf.org/Content/MGM-13.Sep2012/.../3.clum.ppt

Clustering Algorithms

- Clustering as an optimization problem
- The Lloyd algorithm for *k*-means clustering
- From Hard to Soft Clustering
- From Coin Flipping to *k*-means Clustering
- Expectation Maximization
- Soft *k*-means Clustering
- Hierarchical Clustering
- Markov Clustering Algorithm
- Stochastic Neighbor Embedding

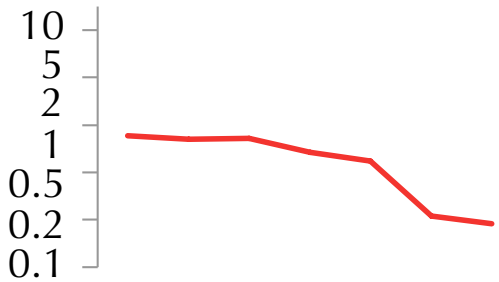
Measuring 3 Genes at 7 Checkpoints

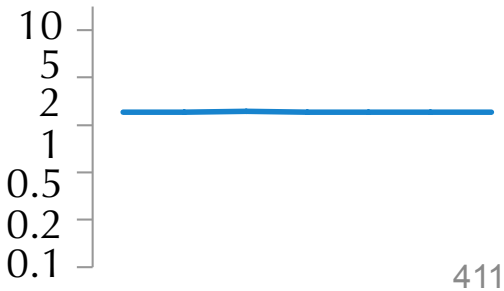
Measure expression of various yeast genes at 7 checkpoints:



YLR258W	1.1	1.4	1.4	3.7	4.0	10.0	5.9
YPL012W	1.1	0.8	0.9	0.4	0.3	0.1	0.1
YPR055W	1.1	1.1	1.1	1.1	1.1	1.1	1.1

e_{ij} = **expression level** of gene *i* at checkpoint *j*





Google

YLR258W yeast

Tutti | Immagini | Shopping | Video | Maps | Altro | Impostazioni | Strumenti

Circa 4.730 risultati (0,40 secondi)

GSY2 | SGD
https://www.yeastgenome.org/locus/S000004248 ▾ Traduci questa pagina
30 ago 2005 - Standard Name: GSY2; Systematic Name: **YLR258W**; SGD ID: SGD: .... of **yeast** glycogen synthase-2 by COOH-terminal phosphorylation.

YLR258W - SGD-Wiki
https://wiki.yeastgenome.org/index.php/YLR258W ▾ Traduci questa pagina
23 gen 2012 - Description of **YLR258W**: Glycogen synthase, similar to Gsy1p; expression ... of **yeast** glycogen synthase-2 by COOH-terminal phosphorylation.

GSY2 Protein | SGD
https://www.yeastgenome.org/locus/S000004248/protein ▾ Traduci questa pagina
... Database (SGD) provides comprehensive integrated biological information for the budding **yeast** Saccharomyces cerevisiae. ... GSY2 / **YLR258W** Protein.

GSY2 - Glycogen [starch] synthase isoform 2 - Saccharomyces ...
https://www.uniprot.org/uniprot/P27472 ▾ Traduci questa pagina
Saccharomyces cerevisiae (strain ATCC 204508 / S288c) (Baker's **yeast**). Status .... BioCyc[], **YEAST**:**YLR258W**-MONOMER ... Ordered Locus Names:**YLR258W**.

```
   1 ATGTCCCGTG ACCTACAAAA CCATTTGTTA TTCGAGACTG CGACTGAGGT TGCTAATAGG
  61 GTTGGTGGTA TTTACTCCGT GCTAAAATCG AAGGCTCCCA TTACGGTTGC CCAGTATAAA
 121 GACCATTACC ACTTGATAGG GCCCTTAAAT AAAGCCACTT ATCAAAATGA AGTTGATATA
 181 CTAGATTGGA AGAAGCCTGA AGCCTTTTCC GATGAAATGA GGCCAGTGCA GCATGCCCTG
 241 CAAACAATGG AATCTAGAGG AGTTCATTTT GTTTATGGGA GGTGGCTGAT TGAAGGTGCT
 301 CCAAAAGTAA TACTTTTTGA CTTGGATTCT GTGAGAGGTT ATTCGAATGA ATGGAAGGGT
 361 GATTTATGGT CATTAGTAGG AATTCCCTCT CCTGAGAATG ATTTCGAGAC GAATGATGCT
 421 ATCCTATTGG GGTATACAGT CGCTTGGTTT CTAGGCGAAG TGGCTCATCT CGATTCACAA
 481 CACGCAATTG TTGCGCACTT TCACGAATGG TTGGCCGGTG TTGCGTTACC ATTATGCCGT
 541 AAAAGGCGTA TCGATGTAGT TACCATTTTC ACCACTCATG CTACTTTATT GGGACGGTAT
 601 TTATGCGCCT CCGGCAGTTT CGATTTTTAC AATTGTTTAG AATCTGTTGA TGTTGATCAC
 661 GAAGCTGGCA GATTTGGCAT ATACCATCGT TATTGTATAG AGAGAGCGGC GGCTCATTCT
 721 GCAGACGTGT TCACTACGGT GTCACAAATA ACTGCTTTTG AAGCGGAACA TCTTTTGAAA
 781 AGAAAACCAG ATGGGATTTT GCCTAATGGA CTGAACGTCA TCAAATTTCA AGCATTTCAT
 841 GAGTTCCAAA ATTTGCATGC TTTGAAAAAA GAAAAAATCA ATGACTTTGT AAGAGGCCAT
 901 TTTCATGGTT GCTTCGATTT CGATCTAGAC AACACCTTGT ACTTTTTTAT TGCTGGTAGA
 961 TATGAGTATA AAAATAAGGG TGCTGACATG TTTATTGAGG CTCTAGCGCG TTTGAACTAC
1021 AGATTAAAAG TATCCGGATC CAAAAAAACT GTGGTAGCGT TTATTGTCAT GCCCGCCAAA
1081 AATAATTCCT TCACTGTTGA AGCATTGAAG GGCCAGGCTG AGGTGAGGGC GTTAGAAAAT
1141 ACTGTACATG AAGTGACTAC TTCAATTGGT AAAAGAATAT TCGATCATGC TATCAGGTAC
1201 CCCCACAATG GACTGACGAC GGAATTACCA ACCGATTTGG GTGAATTACT AAAGAGTTCG
1261 GATAAAGTTA TGTTAAAGAG ACGTATTTTG GCTTTGAGAA GGCCGGAGGG ACAGTTACCC
1321 CCAATAGTTA CACACAATAT GGTCGATGAC GCTAATGACC TGATTTTAAA TAAAATCAGA
1381 CAAGTTCAAT TGTTCAATAG CCCAAGTGAT CGTGTTAAAA TGATCTTCCA TCCAGAATTT
1441 TTGAACGCTA ATAATCCGAT CCTTGGTTTA GATTATGATG AGTTCGTTCG TGGTTGCCAT
1501 TTGGGTGTTT TCCCTTCATA CTACGAGCCT TGGGGTTACA CACCTGCAGA ATGTACAGTA
1561 ATGGGTGTTC CCTCCATCAC GACAAATGTC TCTGGTTTCG GTGCCTATAT GGAAGACTTG
1621 ATCGAAACCA ACCAAGCGAA AGATTACGGT ATTTATATTG TGGATCGTCG TTTCAAGGCA
1681 CCTGATGAAT CTGTGGAACA ATTAGTTGAC TACATGGAAG AATTTGTAAA AAAGACAAGA
1741 AGGCAAAGAA TTAATCAAAG AAATAGAACT GAAAGACTCT CCGACTTACT GGACTGGAAG
1801 AGAATGGGTC TCGAATACGT CAAGGCAAGG CAGTTAGCAT TAAGAAGAGG CTATCCTGAT
1861 CAGTTCAGAG AGCTCGTTGG TGAAGAACTA AATGATTCCA ACATGGATGC TTTAGCAGGC
1921 GGAAAGAAAT TGAAAGTTGC AAGACCGCTT AGTGTACCTG GCTCACCAAG AGATTTGAGA
1981 TCAAACAGCA CAGTCTACAT GACCCCTGGT GATTTGGGTA CTCTGCAGGA GGTTAATAAC
2041 GCGGACGATT ATTTTTCATT GGGAGTGAAT CCTGCAGCTG ACGATGACGA CGATGGCCCA
2101 TATGCTGATG ACAGTTAA
```

Download Sequence (.fsa) | Custom Sequence Retrieval

Analyze | Sequence | Function | Literature | Community

Protein | Gene Ontology | Phenotype | Interactions | Regulation | Expression | Literature

GSY2 / YLR258W Sequence

Sequence Help

Protein Product: glycogen (starch) synthase GSY2
Feature Type: ORF , Verified
Description: Glycogen synthase; expression induced by glucose limitation, nitrogen starvation, heat shock, and stationary phase; activity regulated by cAMP-dependent, Snf1p and Pho85p kinases as well as by the Gac1p-Glc7p phosphatase; GSY2 has a paralog, GSY1, that arose from the whole genome duplication; relocalizes from cytoplasm to plasma membrane upon DNA replication stress [1 2 3 4 5 6 7 8 9 10]
Paralog: GSY1 [10]
EC Number: 2.4.1.11

Reference Strain: S288C

View in: JBrowse

GSY2 Location: Chromosome XII 660716..662833

https://www.ncbi.nlm.nih.gov/geo/

NCBI Resources ▾ How To ▾

GEO Home | Documentation ▾ | Query & Browse ▾ | Email GEO

Gene Expression Omnibus

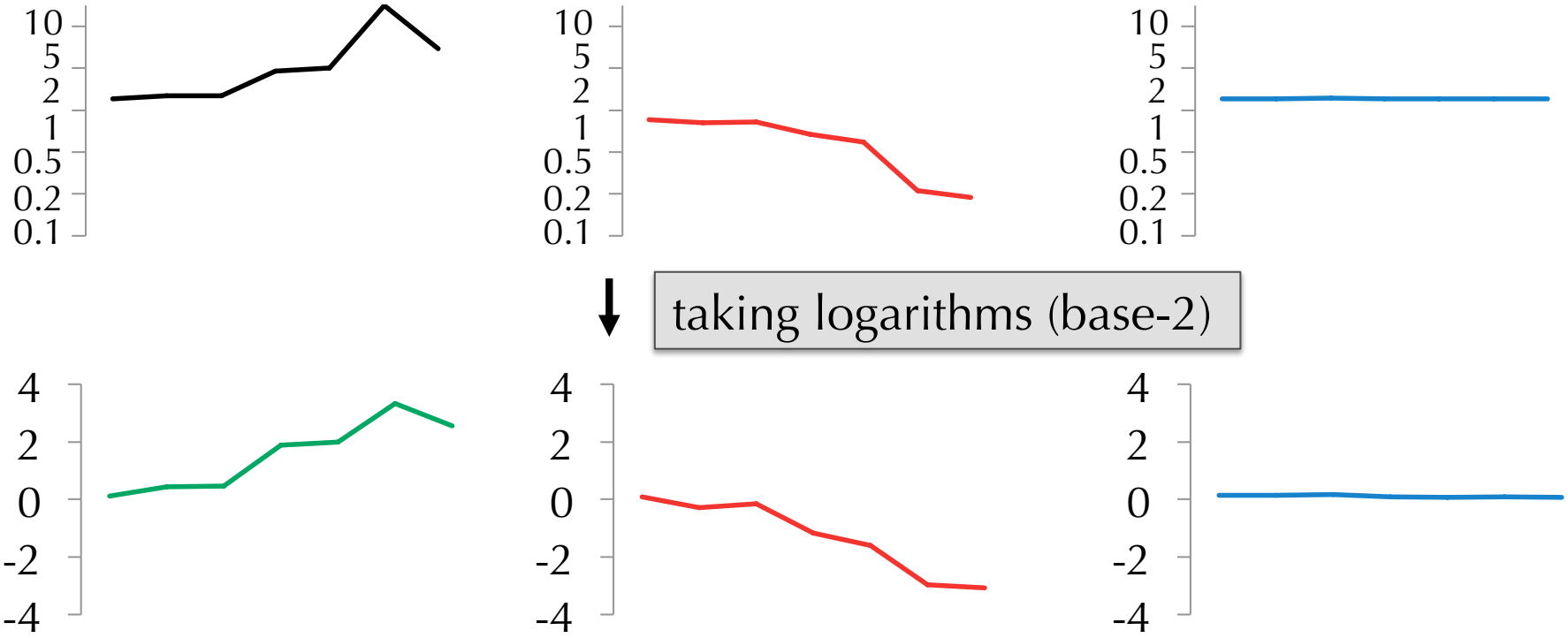
GEO is a public functional genomics data repository supporting MIAME-compliant data submissions. Array- and sequence-based data are accepted. Tools are provided to help users query and download experiments and curated gene expression profiles.

Getting Started
- Overview
- FAQ
- About GEO DataSets
- About GEO Profiles
- About GEO2R Analysis
- How to Construct a Query
- How to Download Data

Tools
- Search for Studies at GEO DataSets
- Search for Gene Expression at GEO Profiles
- Search GEO Documentation
- Analyze a Study with GEO2R
- Studies with Genome Data Viewer Tracks
- Programmatic Access
- FTP Site

Switching to Logarithms of Expression Levels

YLR258W	1.1	1.4	1.4	3.7	4.0	10.0	5.9
YPL012W	1.1	0.8	0.9	0.4	0.3	0.1	0.1
YPR055W	1.1	1.1	1.1	1.1	1.1	1.1	1.1

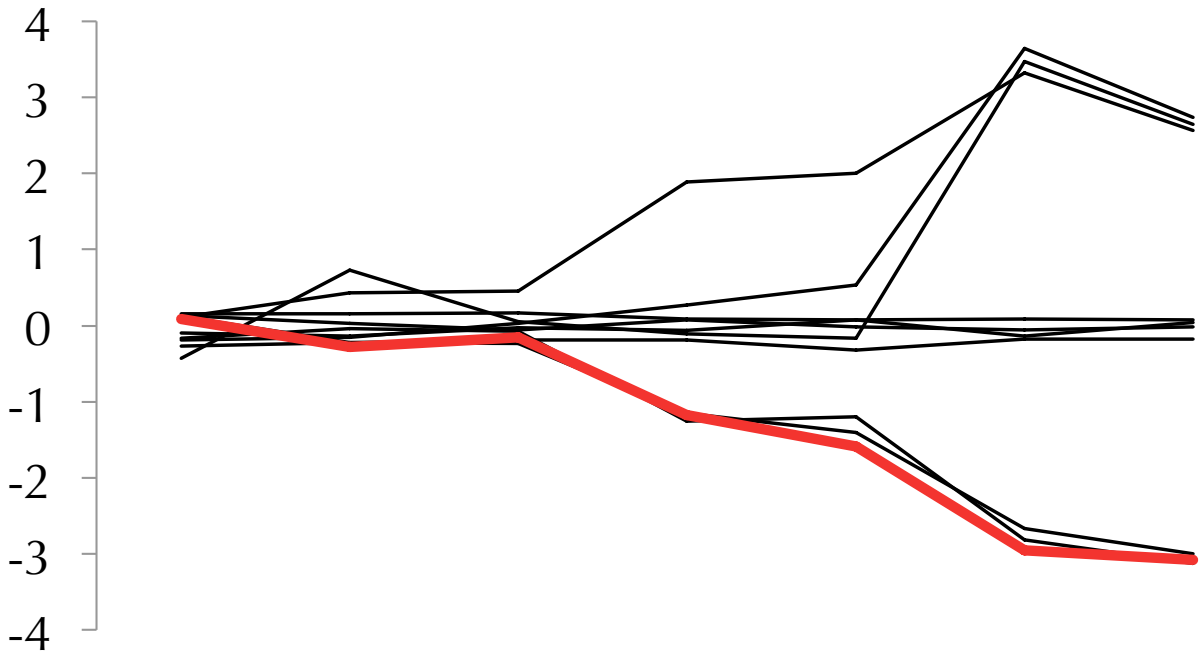


YLR258W	0.1	0.4	0.5	1.9	2.0	3.3	2.6
YPL012W	0.1	-0.3	-0.2	-1.2	-1.6	-3.0	-3.1
YPR055W	0.2	0.2	0.2	0.1	0.1	0.1	0.1

Gene Expression Matrix

YLR361C	0.14	0.03	-0.06	0.07	-0.01	-0.06	-0.01
YMR290C	0.12	-0.23	-0.24	-1.16	-1.40	-2.67	-3.00
YNR065C	-0.10	-0.14	-0.03	-0.06	-0.07	-0.14	-0.04
YGR043C	-0.43	-0.73	-0.06	-0.11	-0.16	3.47	2.64
YLR258W	0.11	0.43	0.45	1.89	2.00	3.32	2.56
YPL012W	0.09	-0.28	-0.15	-1.18	-1.59	-2.96	-3.08
YNL141W	-0.16	-0.04	-0.07	-1.26	-1.20	-2.82	-3.13
YJL028W	-0.28	-0.23	-0.19	-0.19	-0.32	-0.18	-0.18
YKL026C	-0.19	-0.15	0.03	0.27	0.54	3.64	2.74
YPR055W	0.15	0.15	0.17	0.09	0.07	0.09	0.07

gene expression vector



Gene Expression Matrix

YLR361C	0.14	0.03	-0.06	0.07	-0.01	-0.06	-0.01
YMR290C	0.12	-0.23	-0.24	-1.16	-1.40	-2.67	-3.00
YNR065C	-0.10	-0.14	-0.03	-0.06	-0.07	-0.14	-0.04
YGR043C	-0.43	-0.73	-0.06	-0.11	-0.16	3.47	2.64
YLR258W	0.11	0.43	0.45	1.89	2.00	3.32	2.56
YPL012W	0.09	-0.28	-0.15	-1.18	-1.59	-2.96	-3.08
YNL141W	-0.16	-0.04	-0.07	-1.26	-1.20	-2.82	-3.13
YJL028W	-0.28	-0.23	-0.19	-0.19	-0.32	-0.18	-0.18
YKL026C	-0.19	-0.15	0.03	0.27	0.54	3.64	2.74
YPR055W	0.15	0.15	0.17	0.09	0.07	0.09	0.07

1997: Joseph deRisi measured expression of 6,400 yeast genes at 7 checkpoints before and after the diauxic shift.

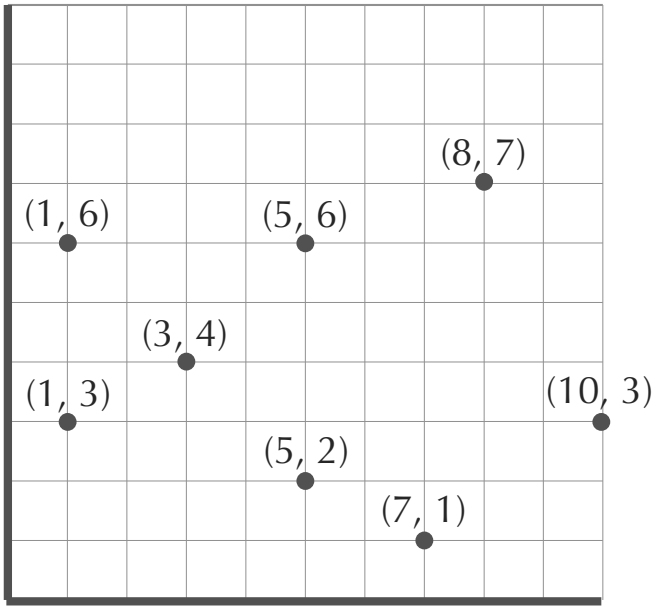
6,400 x 7 gene expression matrix

Goal: partition all yeast genes into clusters so that:

- genes in the *same* cluster have similar behavior
- genes in *different* clusters have different behavior

Genes as Points in Multidimensional Space

YLR361C	0.14	0.03	-0.06	0.07	-0.01	-0.06	-0.01
YMR290C	0.12	-0.23	-0.24	-1.16	-1.40	-2.67	-3.00
YNR065C	-0.10	-0.14	-0.03	-0.06	-0.07	-0.14	-0.04
YGR043C	-0.43	-0.73	-0.06	-0.11	-0.16	3.47	2.64
YLR258W	0.11	0.43	0.45	1.89	2.00	3.32	2.56
YPL012W	0.09	-0.28	-0.15	-1.18	-1.59	-2.96	-3.08
YNL141W	-0.16	-0.04	-0.07	-1.26	-1.20	-2.82	-3.13
YJL028W	-0.28	-0.23	-0.19	-0.19	-0.32	-0.18	-0.18
YKL026C	-0.19	-0.15	0.03	0.27	0.54	3.64	2.74
YPR055W	0.15	0.15	0.17	0.09	0.07	0.09	0.07



n x *m*
gene expression
matrix

n points in
m-dimensional
space

Gene Expression and Cancer Diagnostics

MammaPrint: a test that evaluates the likelihood of breast cancer recurrence based on the expression of just 70 genes.

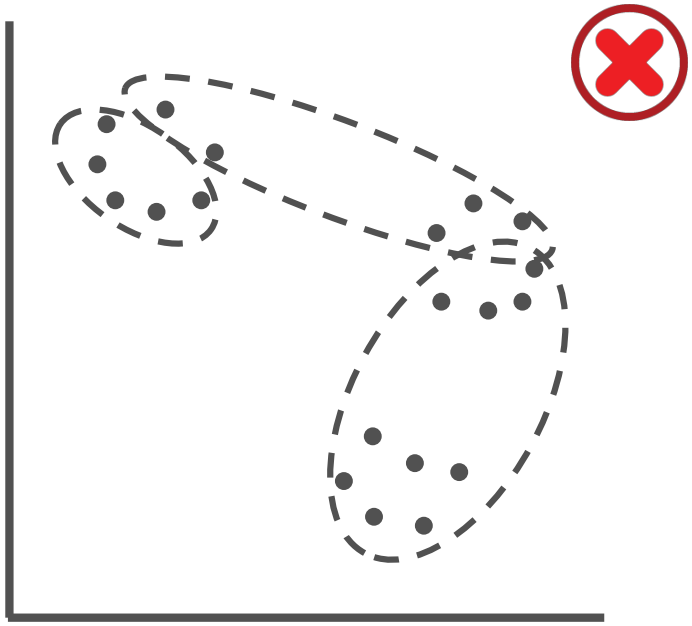
But how did scientists discover these 70 human genes?

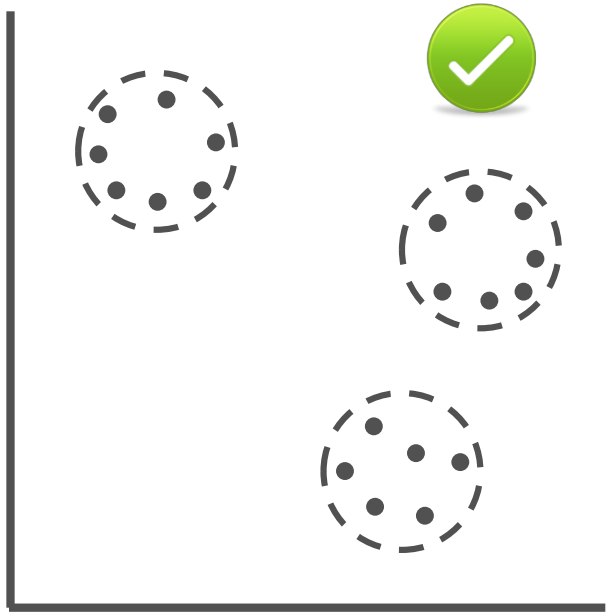
Toward a Computational Problem

Good Clustering Principle: Elements within the same cluster are closer to each other than elements in different clusters.

Toward a Computational Problem

- distance between elements in the same cluster $< \Delta$
- distance between elements in different clusters $> \Delta$

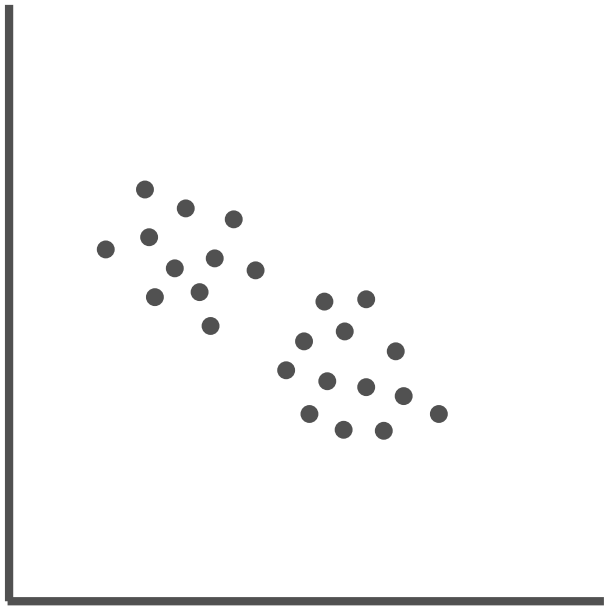




Clustering Problem

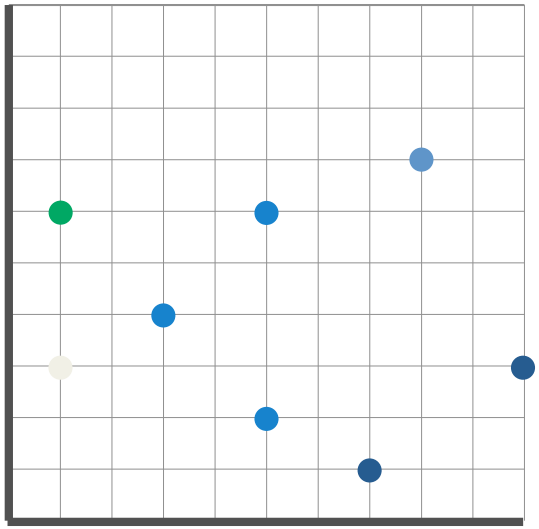
Clustering Problem: *Partition a set of expression vectors into clusters*.

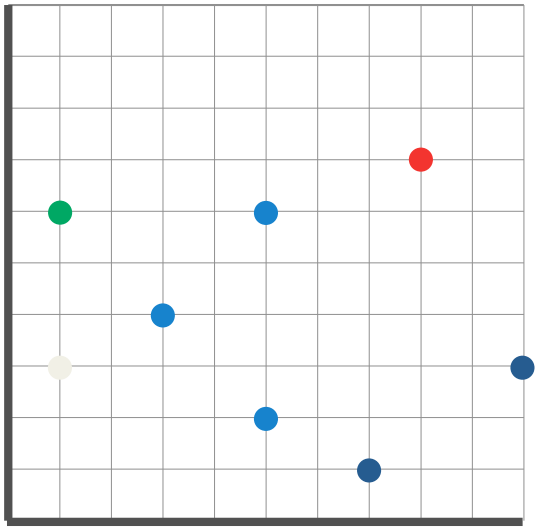
- **Input**: A collection of *n* vectors and an integer *k*.
- **Output**: Partition of *n* vectors into *k* disjoint clusters satisfying the Good Clustering Principle.

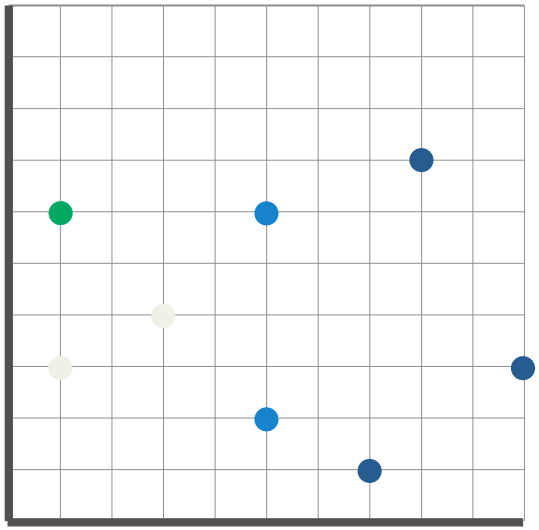


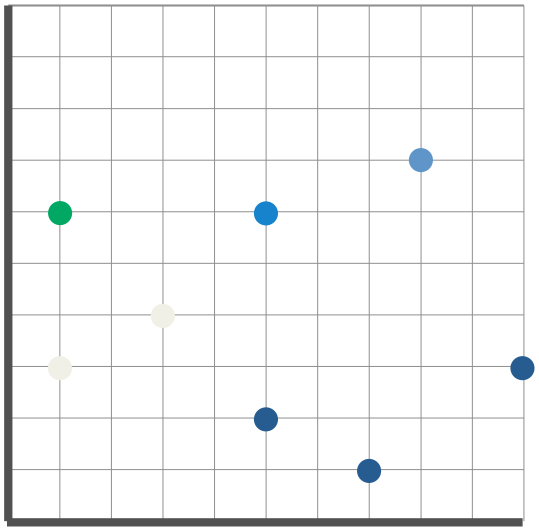
Any partition into two clusters **does not** satisfy the Good Clustering Principle!

What is the “best” partition into three clusters?





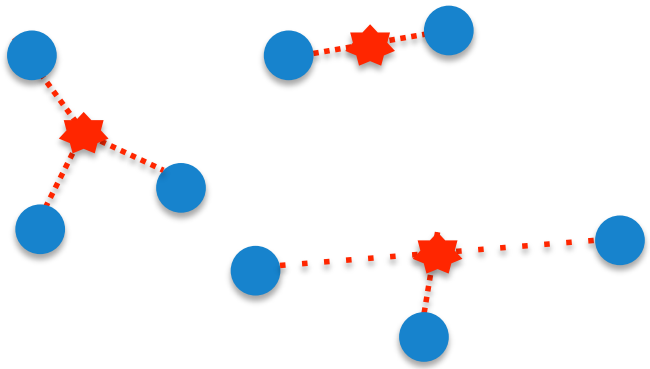




Clustering as Finding Centers

Goal: partition a set *Data* into *k* clusters.

Equivalent goal: find a set of *k* points *Centers* that will serve as the "centers" of the *k* clusters in *Data*.

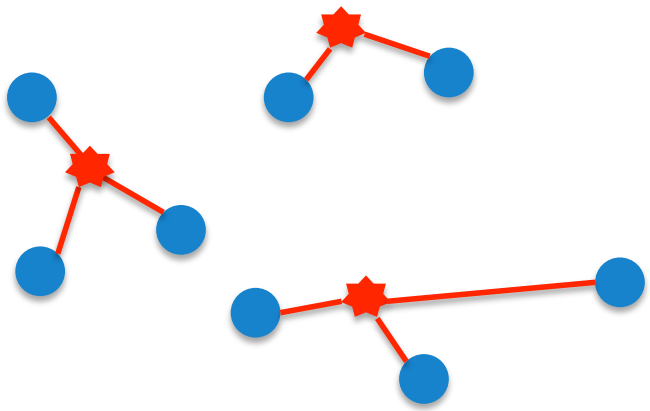


Clustering as Finding Centers

Goal: partition a set *Data* into *k* clusters.

Equivalent goal: find a set of *k* points *Centers* that will serve as the "centers" of the *k* clusters in *Data* and will minimize some notion of distance from *Centers* to *Data* .

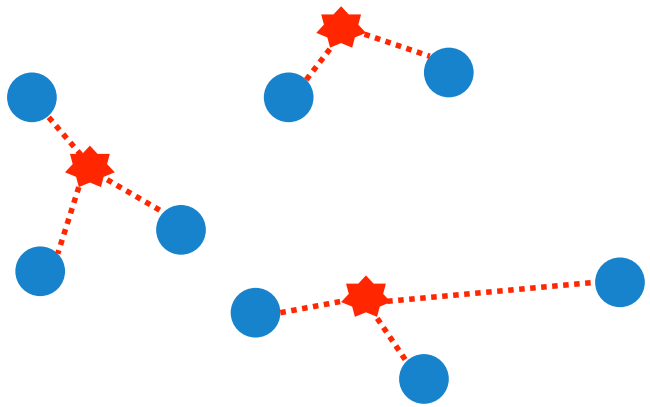
What is the "distance" from *Centers* to *Data*?



Distance from a *Single* *DataPoint* to *Centers*

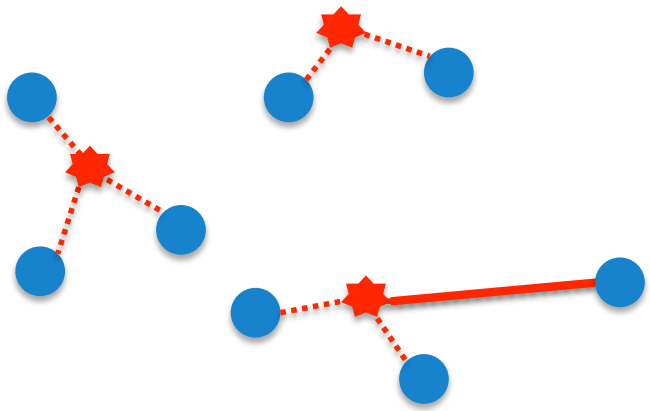
The distance from *DataPoint* in *Data* to *Centers* is the distance from *DataPoint* to the closest center:

$$d(\textit{DataPoint}, \textit{Centers}) = \min_{\text{all points } x \text{ from } \textit{Centers}} d(\textit{DataPoint}, x)$$



Distance from *Data* to *Centers*

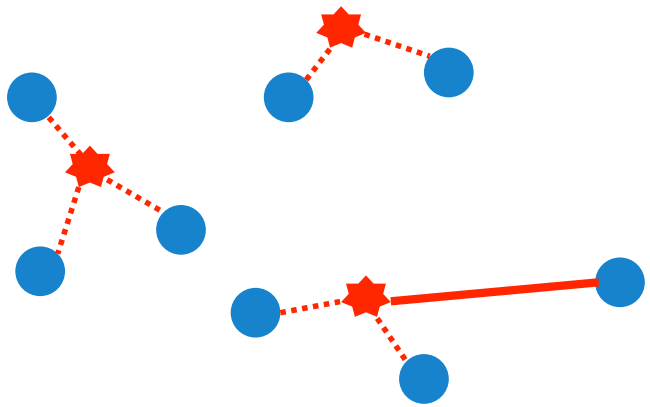
$$MaxDistance(Data, Centers) = \max_{\text{all points } DataPoint \text{ from } Data} d(DataPoint, Centers)$$



k-Center Clustering Problem

***k*-Center Clustering Problem.** Given a set of points *Data*, find *k* centers minimizing *MaxDistance*(*Data*, *Centers*).

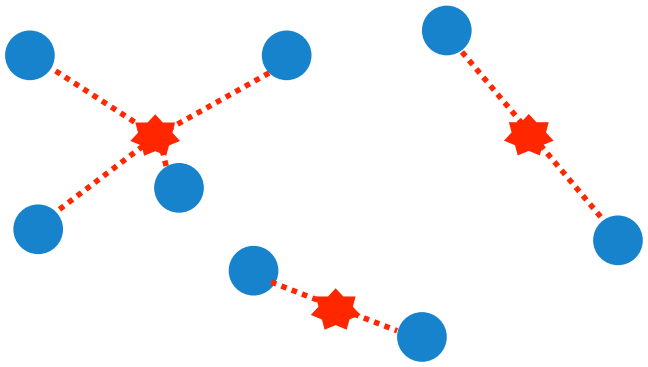
- **Input:** A set of points *Data* and an integer *k*.
- **Output:** A set of *k* points *Centers* that minimizes *MaxDistance*(*DataPoints*, *Centers*) over all possible choices of *Centers*.



k-Center Clustering Problem

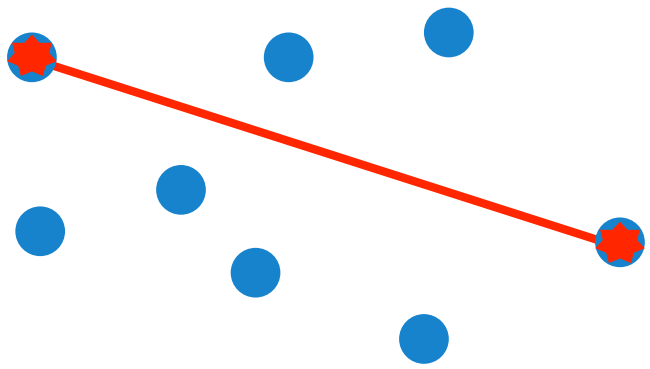
***k*-Center Clustering Problem.** Given a set of points *Data*, find *k* centers minimizing *MaxDistance*(*Data*, *Centers*).

- **Input:** A set of points *Data* and an integer *k*.
- **Output:** A set of *k* points *Centers* that minimizes *MaxDistance*(*DataPoints*, *Centers*) over all possible choices of *Centers*.



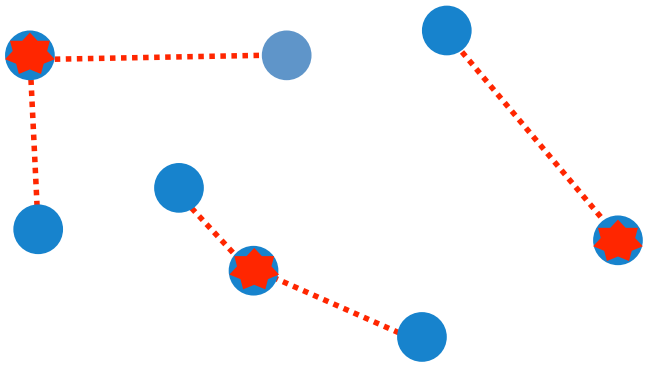
k-Center Clustering Heuristic

FarthestFirstTraversal(*Data*, *k*)
  Centers ← the set consisting of a single *DataPoint* from *Data*
  while *Centers* have fewer than *k* points
    DataPoint ← a point in *Data* maximizing *d*(*DataPoint*, *Centers*)
                  among all data points
    add *DataPoint* to *Centers*



k-Center Clustering Heuristic

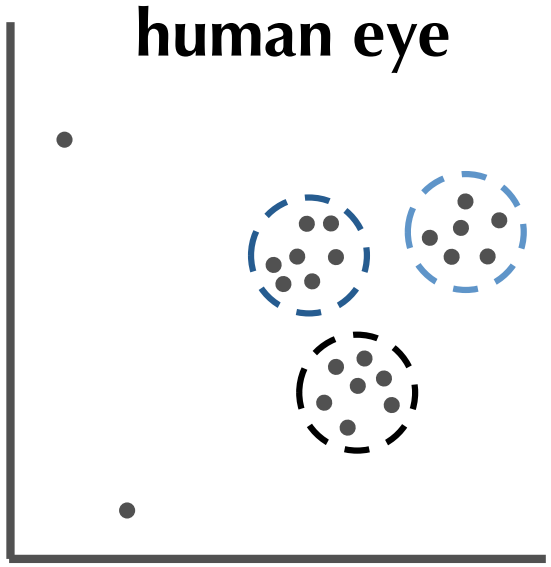
FarthestFirstTraversal(*Data*, *k*)
  Centers ← the set consisting of a single *DataPoint* from *Data*
  while *Centers* have fewer than *k* points
    DataPoint ← a point in *Data* maximizing *d*(*DataPoint*, *Centers*)
                  among all data points
    add *DataPoint* to *Centers*

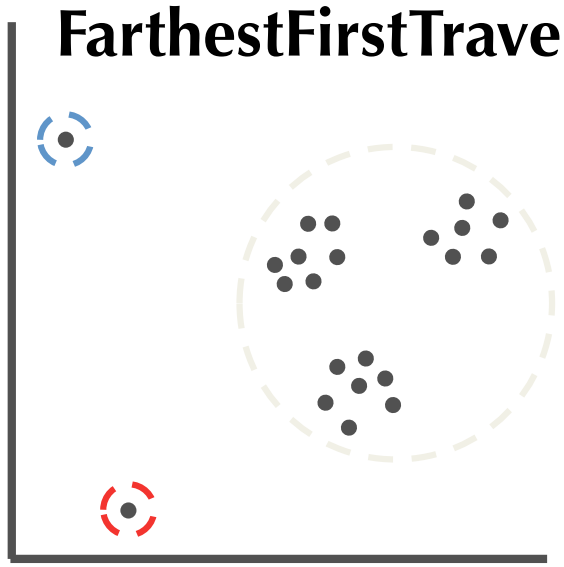


What Is Wrong with **FarthestFirstTraversal**?

FarthestFirstTraversal selects *Centers* that minimize *MaxDistance*(*Data*, *Centers*).

But biologists are interested in **typical** rather than **maximum** deviations, since maximum deviations may represent **outliers** (experimental errors).





Modifying the Objective Function

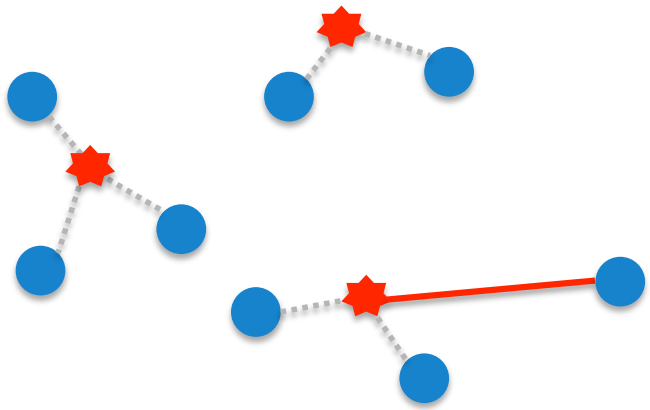
The **maximal distance** between *Data* and *Centers*:

$$MaxDistance(Data, Centers) = \max_{DataPoint \text{ from } Data} d(DataPoint, Centers)$$

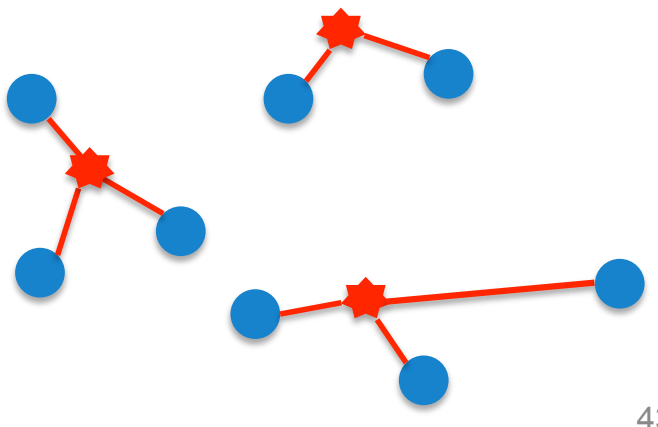
The **squared error distortion** between *Data* and *Centers*:

$$Distortion(Data, Centers) = \sum_{DataPoint \text{ from } Data} d(DataPoint, Centers)^2/n$$

A single data point contributes to *MaxDistance*



All data points contribute to *Distortion*



k-Means Clustering Problem

***k*-Center Clustering Problem:**

Input: A set of points *Data* and an integer *k*.

Output: A set of *k* points *Centers* that minimizes

MaxDistance(*DataPoints*,*Centers*)

over all choices of *Centers*.

***k*-Means Clustering Problem:**

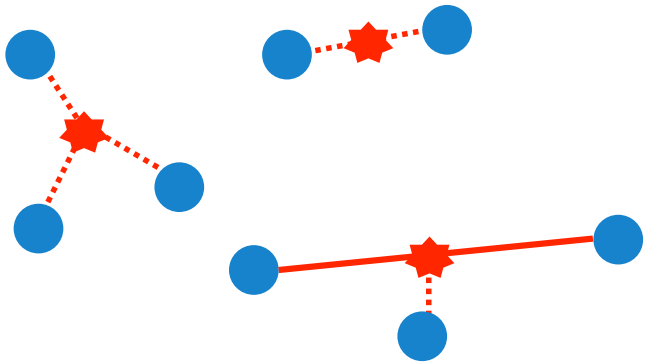
Input: A set of points *Data* and an integer *k*.

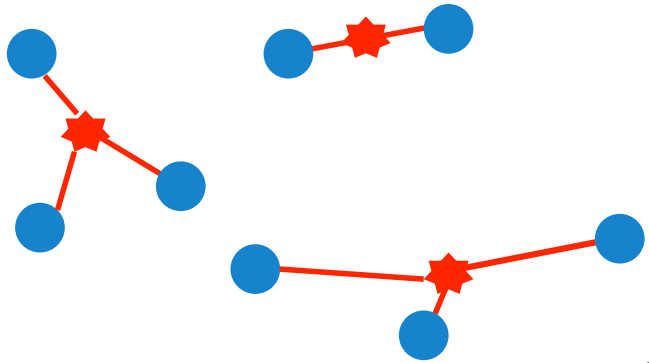
Output: A set of *k* points *Centers* that minimizes

Distortion(*Data*,*Centers*)

over all choices of *Centers*.

NP-Hard for $k > 1$



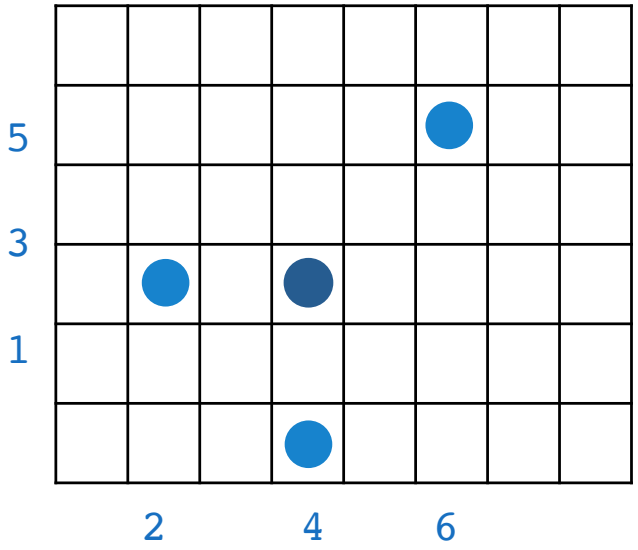


k-Means Clustering for *k* = 1

Center of Gravity Theorem: The center of gravity of points *Data* is the only point solving the 1-Means Clustering Problem.

The **center of gravity** of points *Data* is

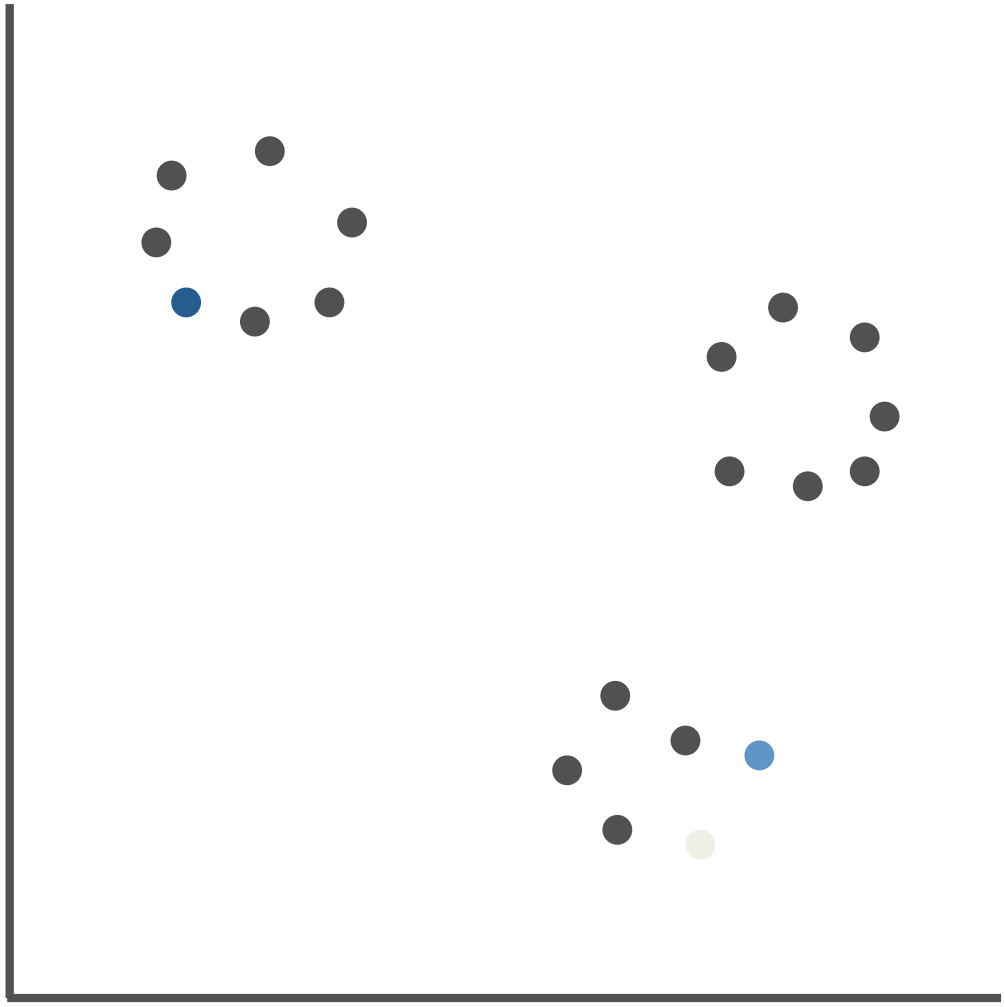
$\sum_{\text{all points } DataPoint \text{ in } Data} DataPoint$ / #points in *Data*



i-th coordinate of the center of gravity = the average of the *i*-th coordinates of datapoints:

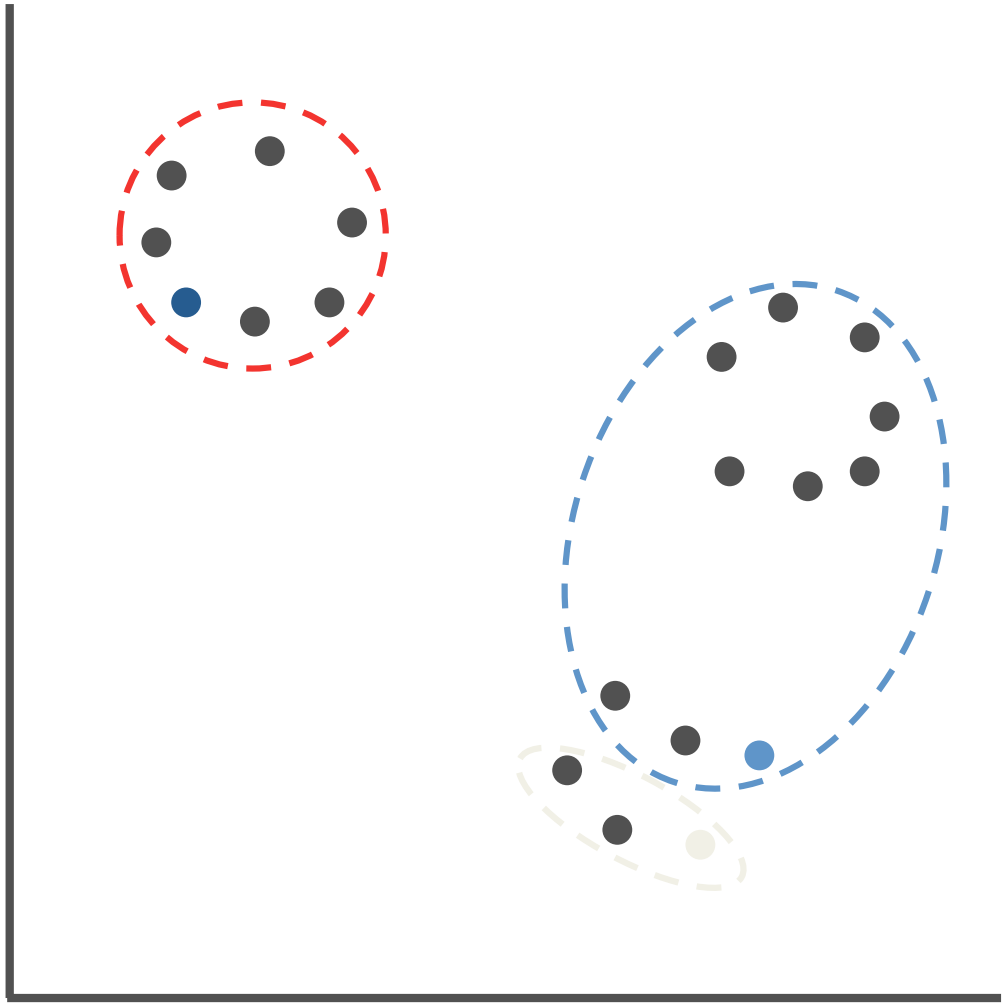
((2+4+6)/3, (3+1+5)/3) = (4, 3)

The Lloyd Algorithm in Action



Select *k* arbitrary data points as *Centers*

The Lloyd Algorithm in Action

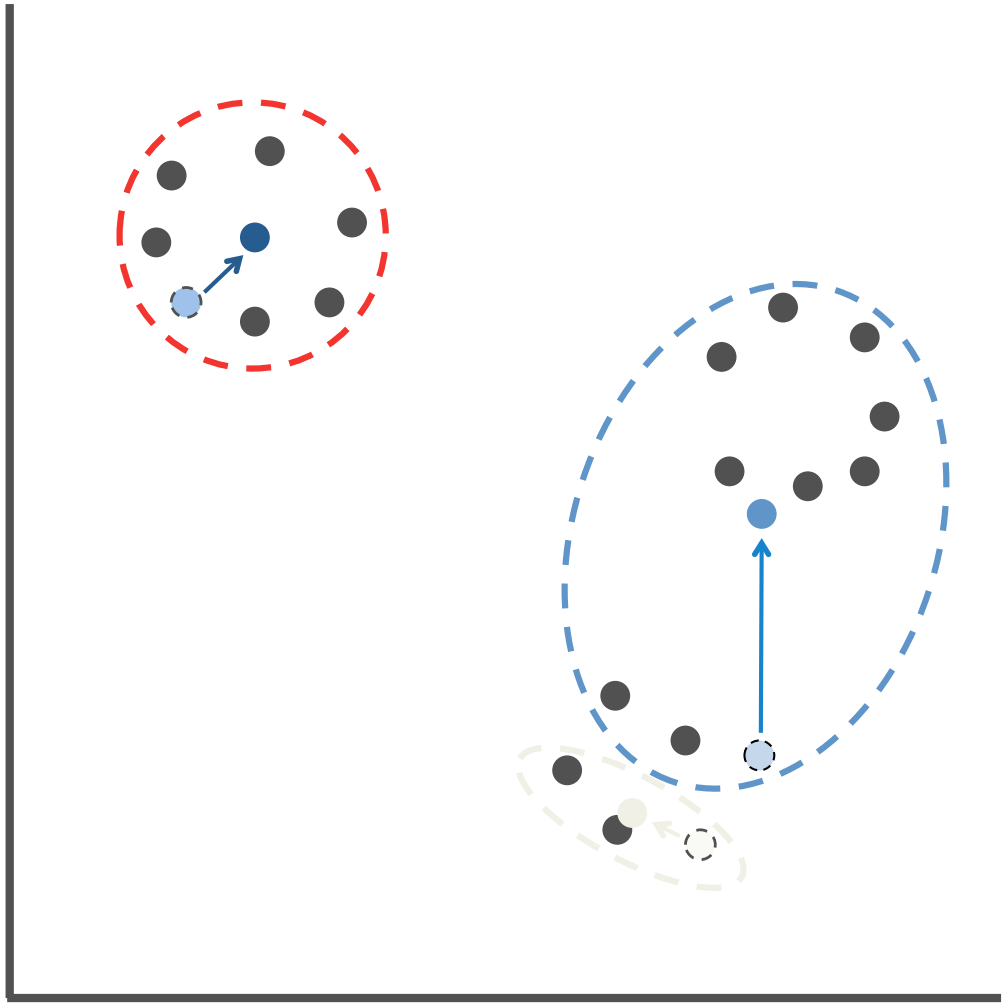


Clusters

Centers

assign each data point to its nearest center

The Lloyd Algorithm in Action

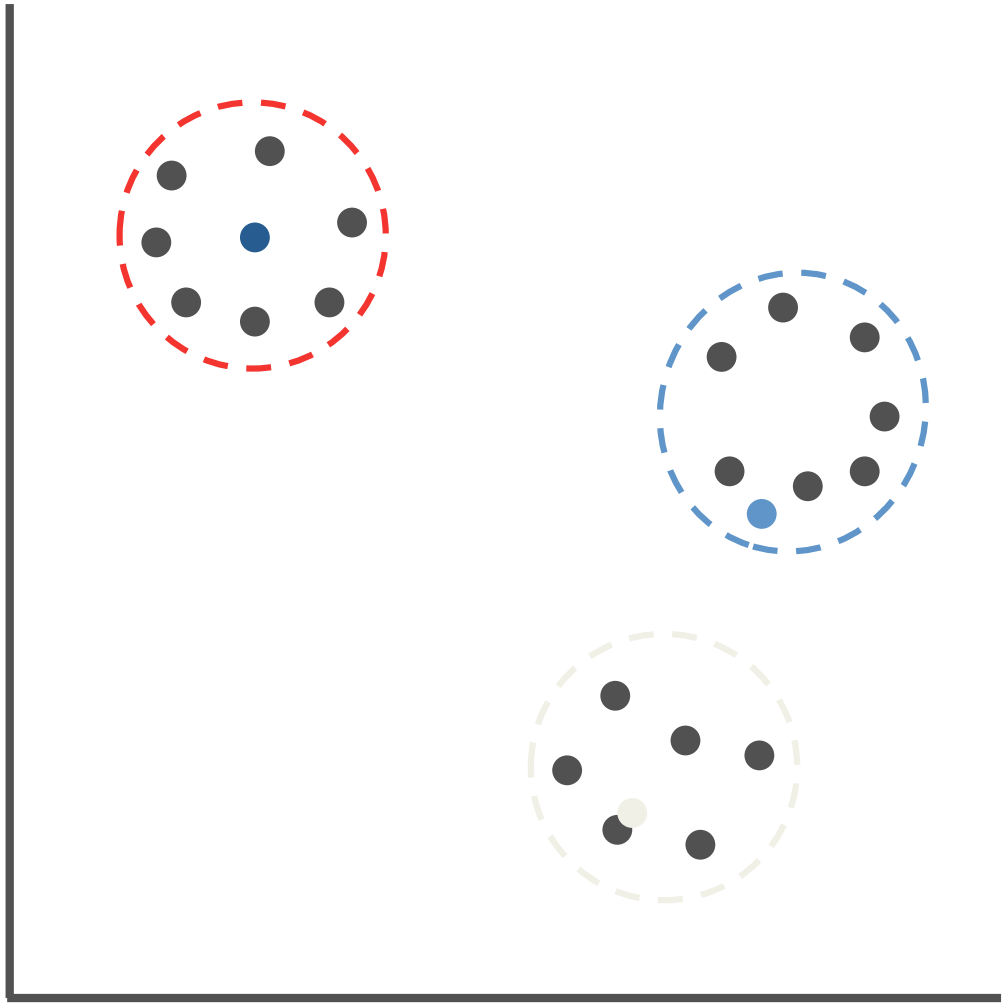


Clusters

Centers

new centers ← clusters' centers of gravity

The Lloyd Algorithm in Action



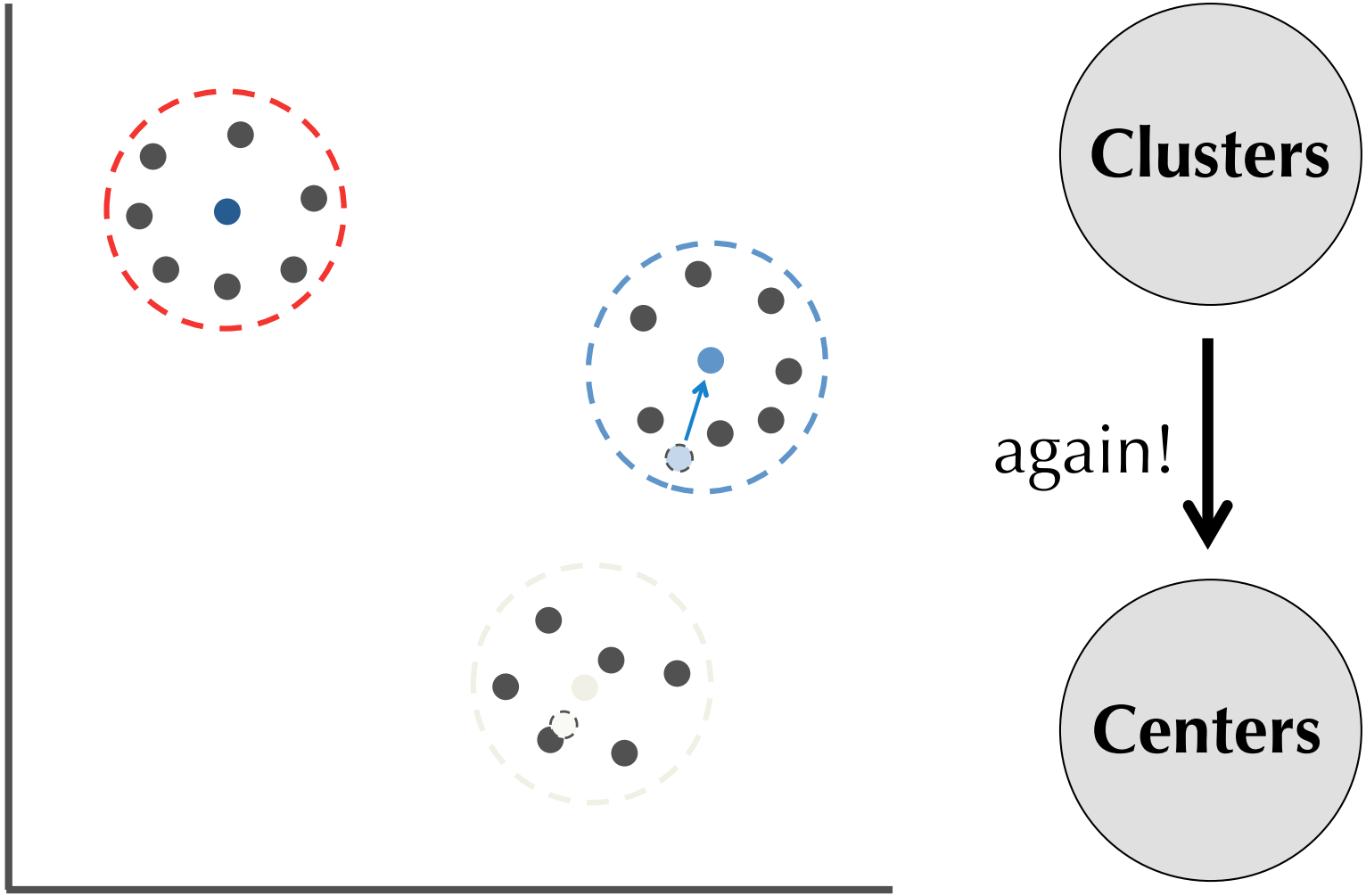
Clusters

again!

Centers

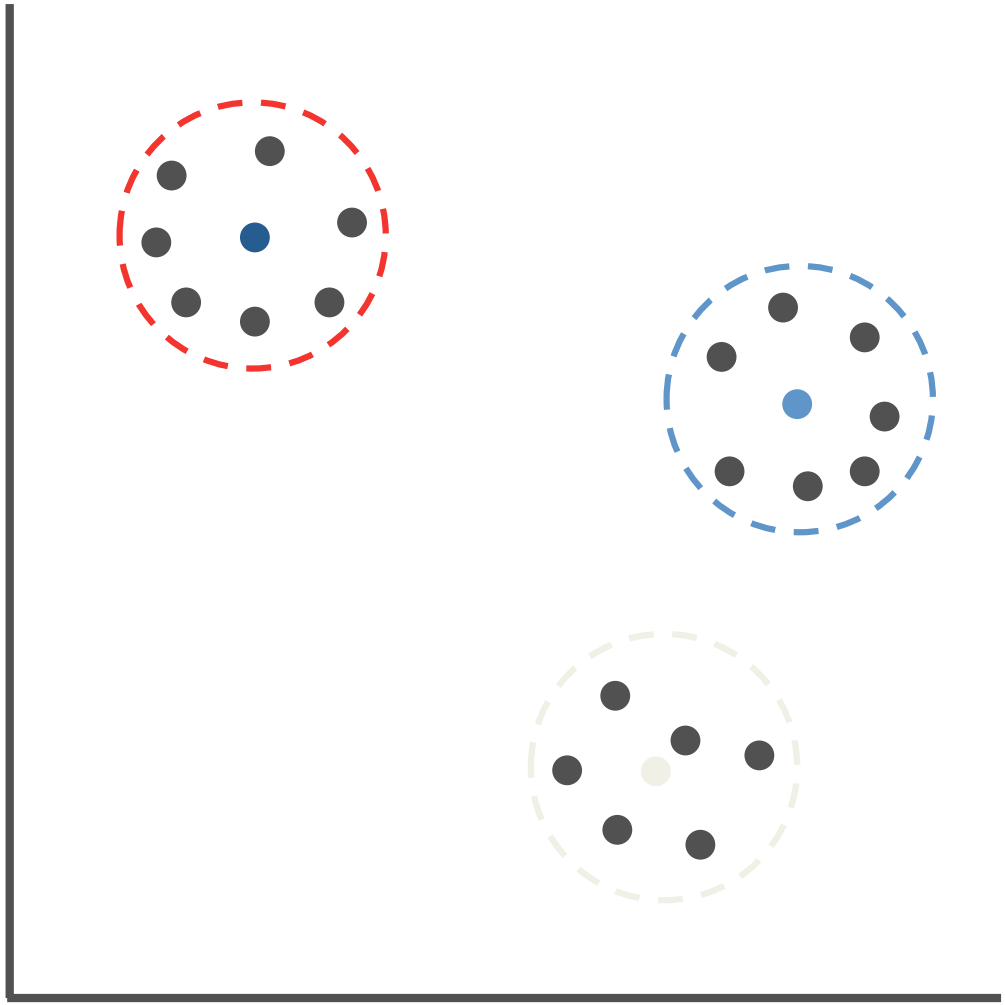
assign each data point to its nearest center

The Lloyd Algorithm in Action



new centers ← clusters' centers of gravity

The Lloyd Algorithm in Action



Clusters

again!

Centers

assign each data point to its nearest center

The Lloyd Algorithm

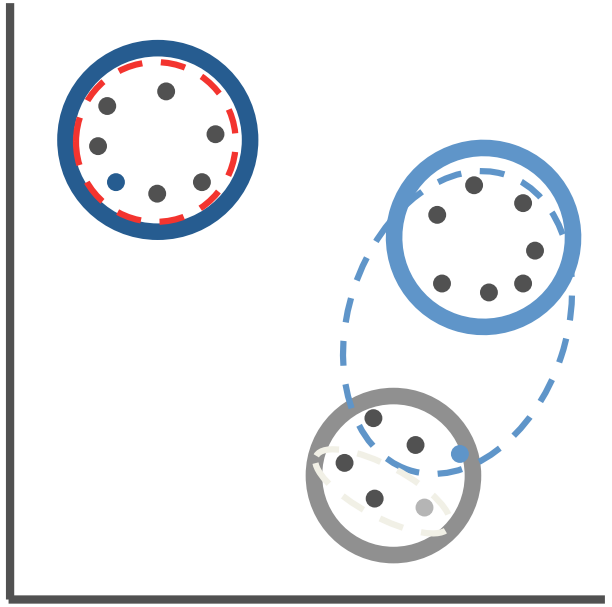
Select *k* arbitrary data points as *Centers* and then iteratively performs the following two steps:

- **Centers to Clusters**: Assign each data point to the cluster corresponding to its nearest center (ties are broken arbitrarily).
- **Clusters to Centers**: After the assignment of data points to *k* clusters, compute new centers as clusters' center of gravity.

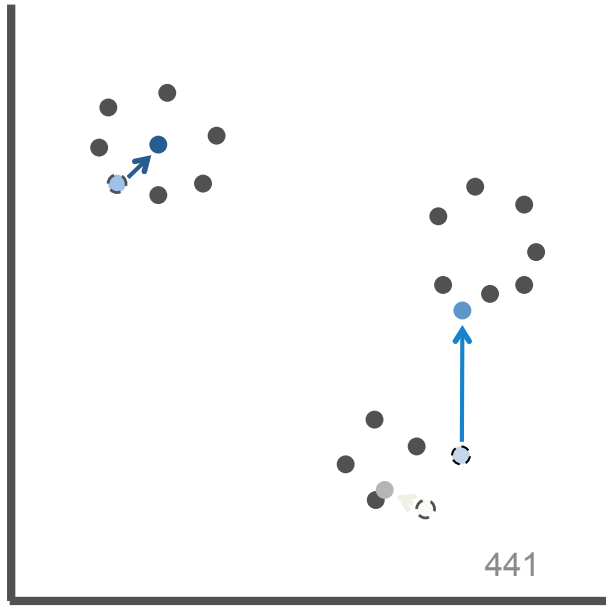
The Lloyd algorithm terminates when the centers stop moving (**convergence**).

Must the Lloyd Algorithm Converge?

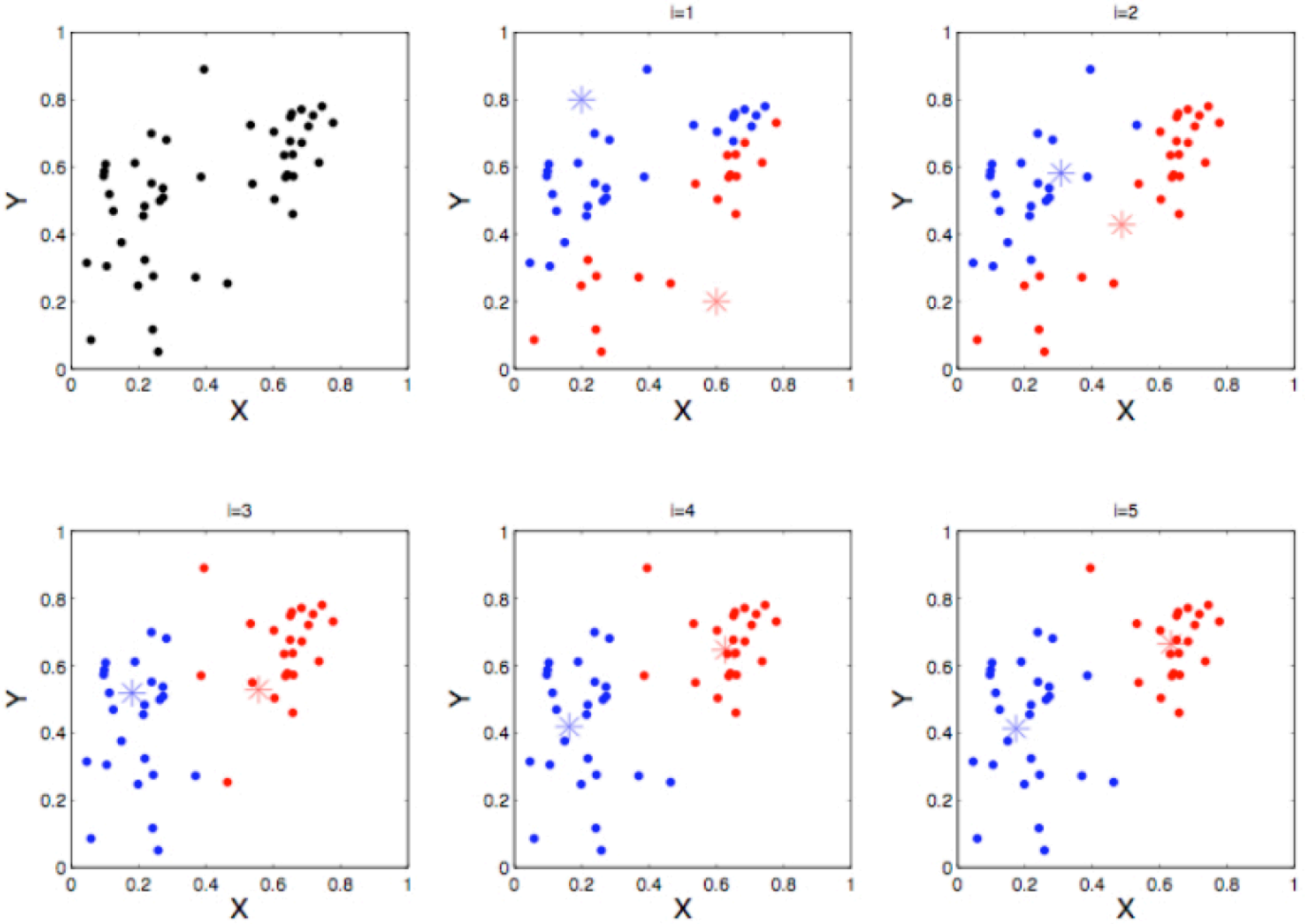
- If a data point is assigned to a new center during the **Centers to Clusters** step:
  - the squared error distortion is reduced because this center must be closer to the point than the previous center was.



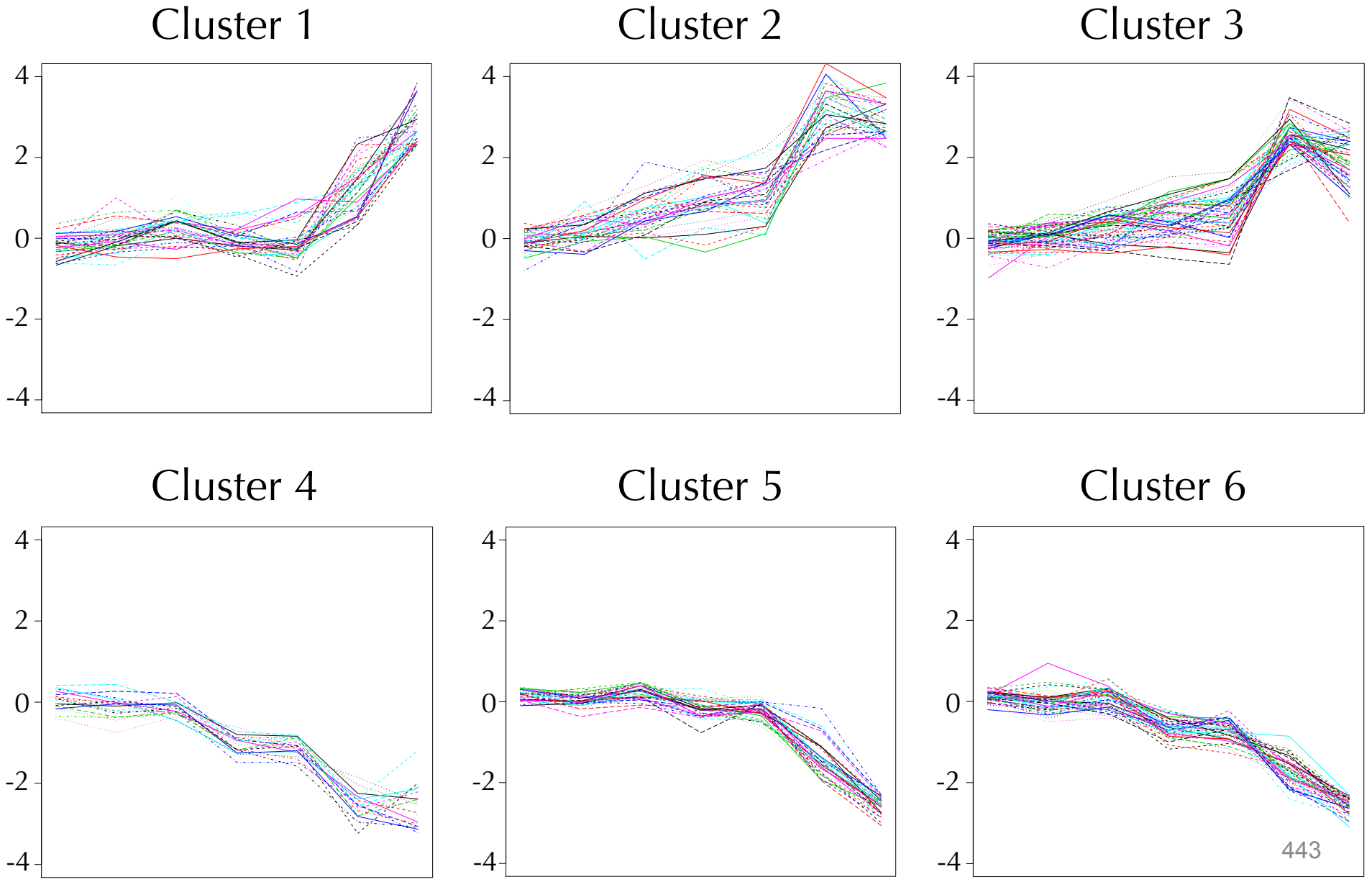
- If a center is moved during the **Clusters to Centers** step:
  - the squared error distortion is reduced since the center of gravity is the *only point* minimizing the distortion (the Center of Gravity Theorem).



RECAP

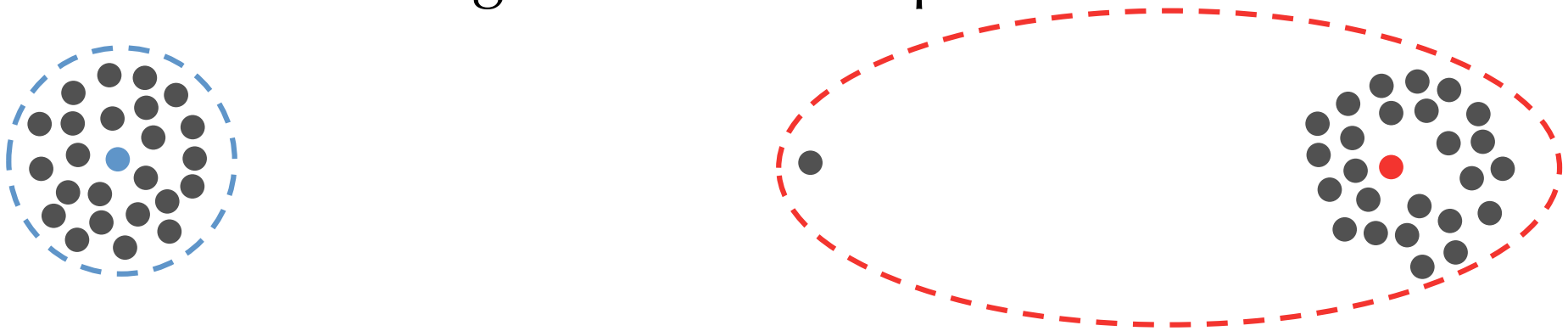


Clustering Yeast Genes



Soft vs. Hard Clustering

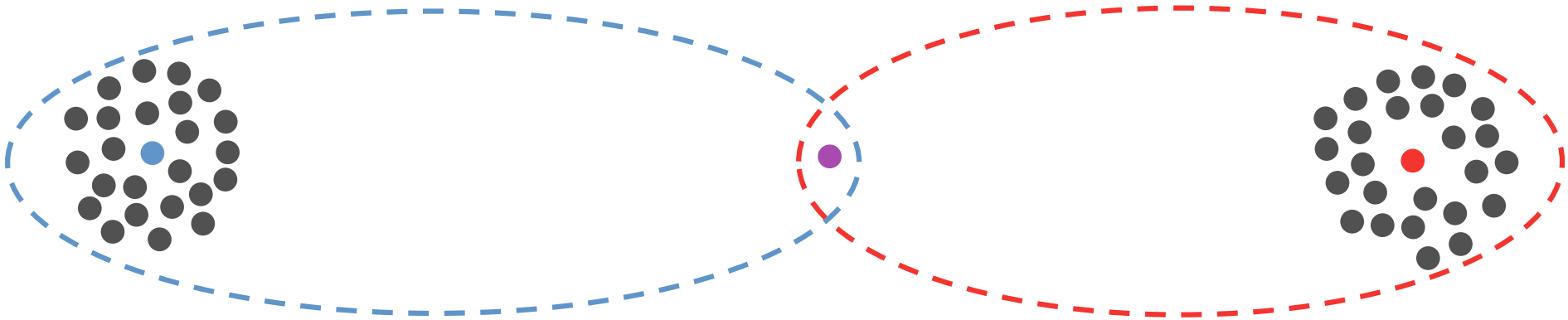
- The Lloyd algorithm assigns the midpoint either to the red or to the blue cluster.
  - "**hard**" assignment of data points to clusters.



Midpoint: A point approximately halfway between two clusters.

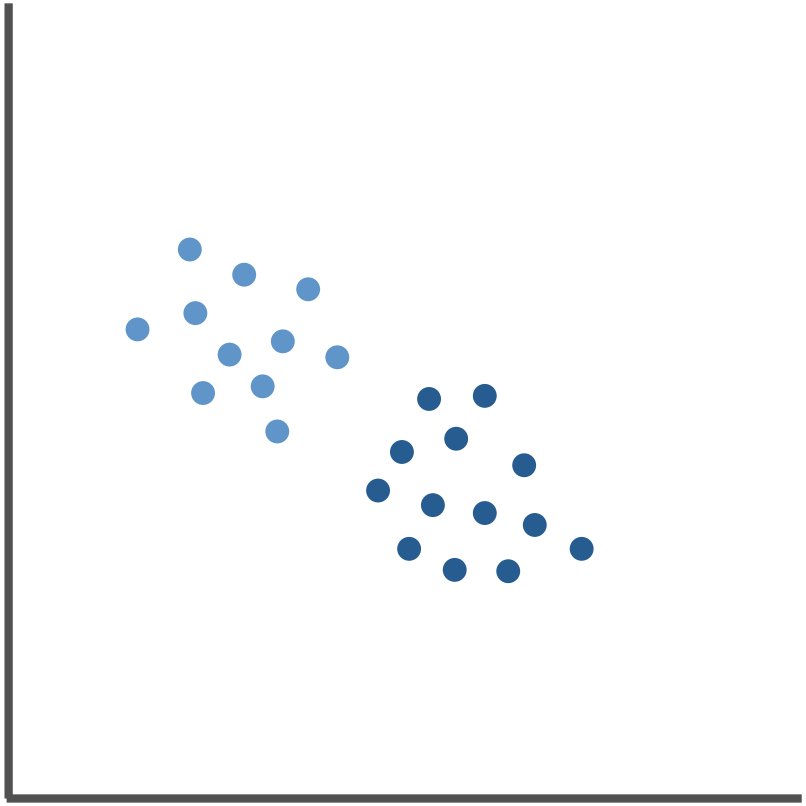
Soft vs. Hard Clustering

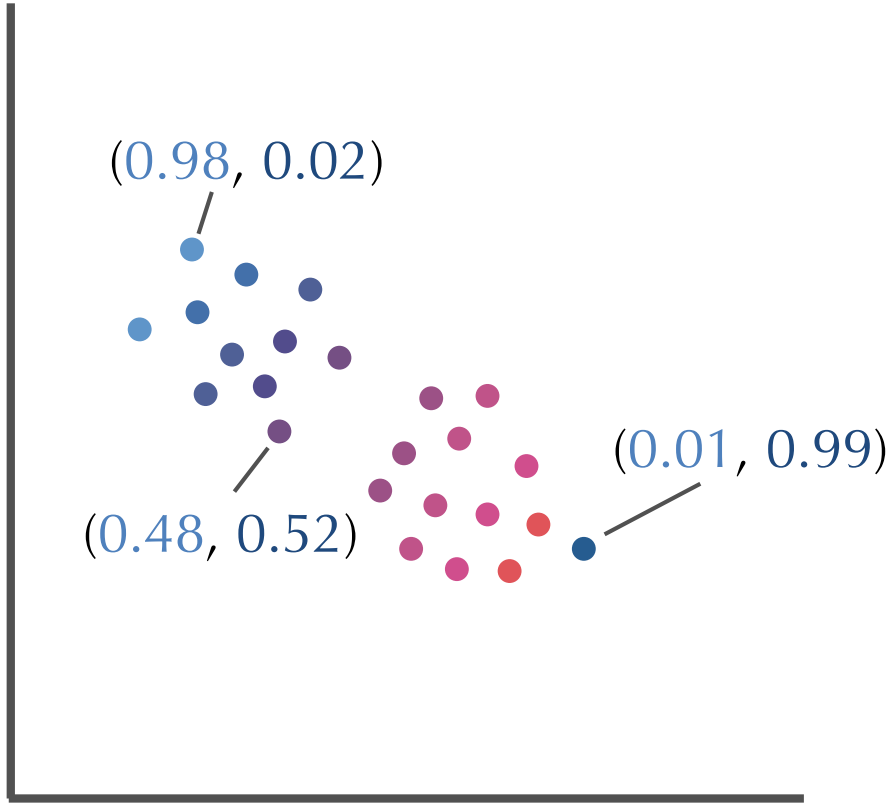
- The Lloyd algorithm assigns the midpoint either to the red or to the blue cluster.
  - "**hard**" assignment of data points to clusters.



- Can we color the midpoint half-red and half-blue?
  - "**soft**" assignment of data points to clusters.

Soft vs. Hard Clustering





Hard choices: points are colored red or blue depending on their cluster membership.

Soft choices: points are assigned "red" and "blue" *responsibilities* r_{blue} and r_{red} ($r_{\text{blue}} + r_{\text{red}} = 1$)

Flipping One Biased Coin

- We flip a loaded coin with an **unknown bias θ** (probability that the coin lands on heads).
- The coin lands on heads ***i* out of *n*** times.
- For each bias, we can compute the probability of the resulting sequence of flips.

Probability of generating the given sequence of flips is

$$\Pr(\text{sequence}|\theta) = \theta^{i} * (1-\theta)^{n-i}$$

This expression is maximized at **θ= *i*/*n*** *(*most likely bias)

Flipping Two Biased Coins

A *B*

Data	
HTTTHTTHTH	**0.4**
HHHHTHHHHH	**0.9**
HTHHHHHTHH	**0.8**
HTTTTTHHTT	**0.3**
THHHTHHHTH	**0.7**

Goal: estimate the probabilities θ_A and θ_B

If We Knew Which Coin Was Used in Each Sequence…

	Data	*HiddenVector*
HTTTHTTHTH	0.4	1
HHHHTHHHHH	0.9	0
HTHHHHHTHH	0.8	0
HTTTTTHHTT	0.3	1
THHHTHHHTH	0.7	0

Goal: estimate *Parameters* = (θ_A, θ_B) when *HiddenVector* is given

If We Knew Which Coin Was Used in Each Sequence…

	Data	*HiddenVector*
HTTTHTTHTH	0.4	1
HHHHTHHHHH	0.9	0
HTHHHHHTHH	0.8	0
HTTTTTHHTT	0.3	1
THHHTHHHTH	0.7	0

θ_A = fraction of heads generated in all flips with coin A =
(4+3) / (10+10) = (0.4+0.3) / 2 = 0.35

θ_B = fraction of heads generated in all flips with coin B =
(9+8+7) / (10+10+10) = (0.9+0.8+0.7) / (1+1+1) = 0.80

Parameters as a Dot-Product

	Data		***HiddenVector***	***Parameters***=(θ_A, θ_B)
HTTTHTTHTH	**0.4**	*	**1**	
HHHHTHHHHH	0.9	*	0	
HTHHHHHTHH	0.8	*	0	(0.35, 0.80)
HTTTTTHHTT	**0.3**	*	**1**	
THHHTHHHTH	0.7	*	0	

θ_A = fraction of heads generated in all flips with coin A =

= (4+3) / (10+10) = (0.4+0.3) / 2 = 0.35

(0.4*1+0.9*0+0.8*0+0.3*1+0.7*0)/ (1+0+0+1+0) = 0.35

$\sum_{\text{all data points } i} Data_i * HiddenVector_i \;/\; \sum_{\text{all data points } i} HiddenVector_i = 0.35$

$Data * HiddenVector \;/\; (1,\mathbf{1},\ldots,*1) * HiddenVector = 0.35$

1 refers to a vector (1,1, … ,1) consisting of all 1s

Parameters as a Dot-Product

	Data		*HiddenVector*	***Parameters***=(θ_A, θ_B)
HTTTHTTHTH	0.4	*	1	
HHHHTHHHHH	0.9	*	0	
HTHHHHHTHH	0.8	*	0	(0.35, 0.80)
HTTTTTHHTT	0.3	*	1	
THHHTHHHTH	0.7	*	0	

θ_B = fraction of heads generated in all flips with coin *B*

= (9+8+7) / (10+10+10) = (0.9+0.8+0.7) /(1+1+1) = 0.80

(0.5*0+0.9*1+0.8*1+0.4*0+0.7*1) / (0+1+1+0+1) = 0.80

$\sum_{\text{all points } i}$ *Data*$_i$ * (1- *HiddenVector*$_i$) / $\sum_{\text{all points } i}$ (1- *HiddenVector*$_i$)=

Data * (1-*HiddenVector*) / **1** * (1 - *HiddenVector*)

Parameters as a Dot-Product

	Data		*HiddenVector*	***Parameters***=(θ_A, θ_B)
HTTTHTTHTH	**0.4**	*	**1**	
HHHHTHHHHH	**0.9**	*	**0**	
HTHHHHHTHH	**0.8**	*	**0**	(0.35, 0.80)
HTTTTTHHTT	**0.3**	*	**1**	
THHHTHHHTH	**0.7**	*	**0**	

θ_A = fraction of heads generated in all flips with coin *A*

= (0.4+0.3)/2=0.35

= *Data * HiddenVector / **1** * HiddenVector*

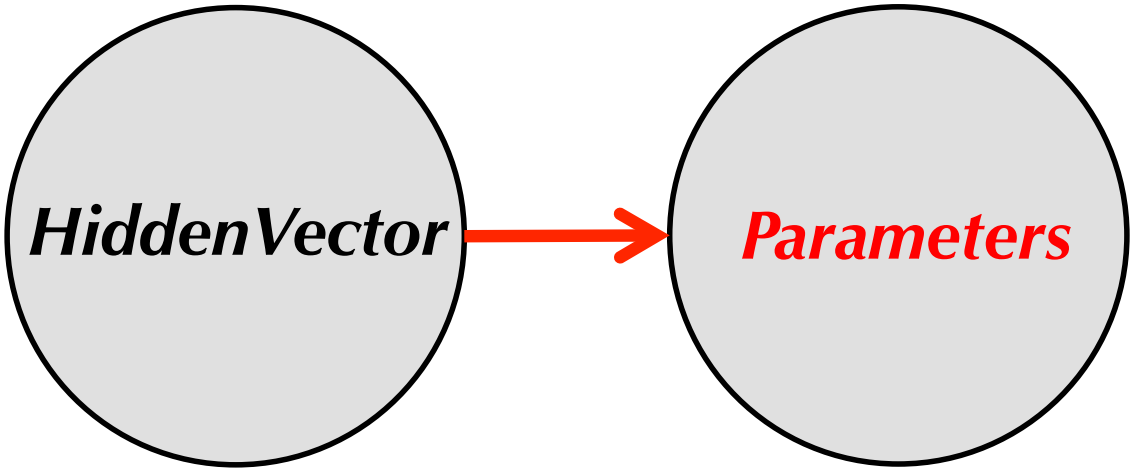
θ_B = fraction of heads generated in all flips with coin *B*

= (0.9+0.8+0.7)/3=0.80

= *Data * (**1**-HiddenVector) / **1** * (**1** - HiddenVector)*

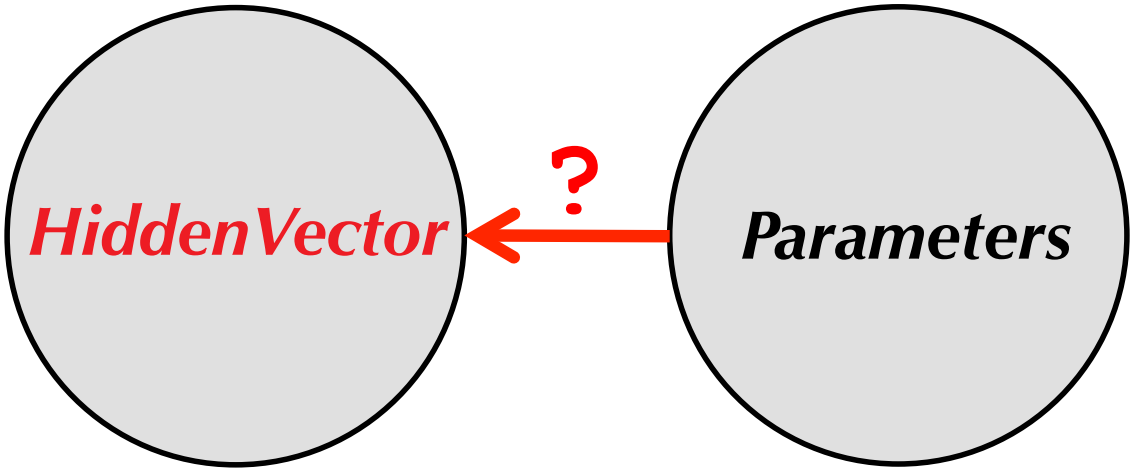
Data, HiddenVector, Parameters

Data	*HiddenVector*	*Parameters*=(θ_A, θ_B)
0.4	1	
0.9	0	
0.8	0	→ (0.35, 0.80)
0.3	1	
0.7	0	



Data, HiddenVector, Parameters

Data	*HiddenVector*	*Parameters*=(θ_A, θ_B)
0.4	?	
0.9	?	
0.8	?	← (0.35, 0.80)
0.3	?	
0.7	?	



From *Data & Parameters to HiddenVector*

Data	***HiddenVector***	***Parameters***=(θ_A, θ_B)
0.4	?	
0.9	?	
0.8	?	← (0.35, 0.80)
0.3	?	
0.7	?	

Which coin is more likely to generate the 1st sequence (with 4 H)?

$\Pr(1^{st}\text{ sequence}|\theta_A)=\theta_A^4(1-\theta_A)^6 = 0.35^4 \bullet 0.65^6 \approx 0.00113$ >
$\Pr(1^{st}\text{ sequence}|\theta_B)= \theta_B^4(1-\theta_B)^6 = 0.80^4 \bullet 0.20^6 \approx 0.00003$

From *Data & Parameters to HiddenVector*

Data	*HiddenVector*	*Parameters*=(θ_A, θ_B)
0.4	1	
0.9	?	
0.8	?	← (0.35, 0.80)
0.3	?	
0.7	?	

Which coin is more likely to generate the 1st sequence (with 4 H)?

$\Pr(1^{st}\text{ sequence}|\theta_A)=\theta_A^4(1-\theta_A)^6 = 0.35^4 \bullet 0.65^6 \approx 0.00113$ >
$\Pr(1^{st}\text{ sequence}|\theta_B)= \theta_B^4(1-\theta_B)^6 = 0.80^4 \bullet 0.20^6 \approx 0.00003$

From *Data & Parameters* to *HiddenVector*

Data	*HiddenVector*	*Parameters*=(θ_A, θ_B)
0.4	1	
0.9	?	
0.8	?	← (0.35, 0.80)
0.3	?	
0.7	?	

Which coin is more likely to generate the 2nd sequence (with 9 H)?

$\Pr(2^{nd}\text{ sequence}|\theta_A) = \theta_A^9 (1-\theta_A)^1 = 0.35^9 \bullet 0.65^1 \approx 0.00005$ <
$\Pr(2^{nd}\text{ sequence}|\theta_B) = \theta_B^9 (1-\theta_B)^1 = 0.80^9 \bullet 0.20^1 \approx 0.02684$

From *Data & Parameters* to *HiddenVector*

Data	*HiddenVector*	*Parameters*=(θ_A, θ_B)
0.4	1	
0.9	0	
0.8	?	← (0.35, 0.80)
0.3	?	
0.7	?	

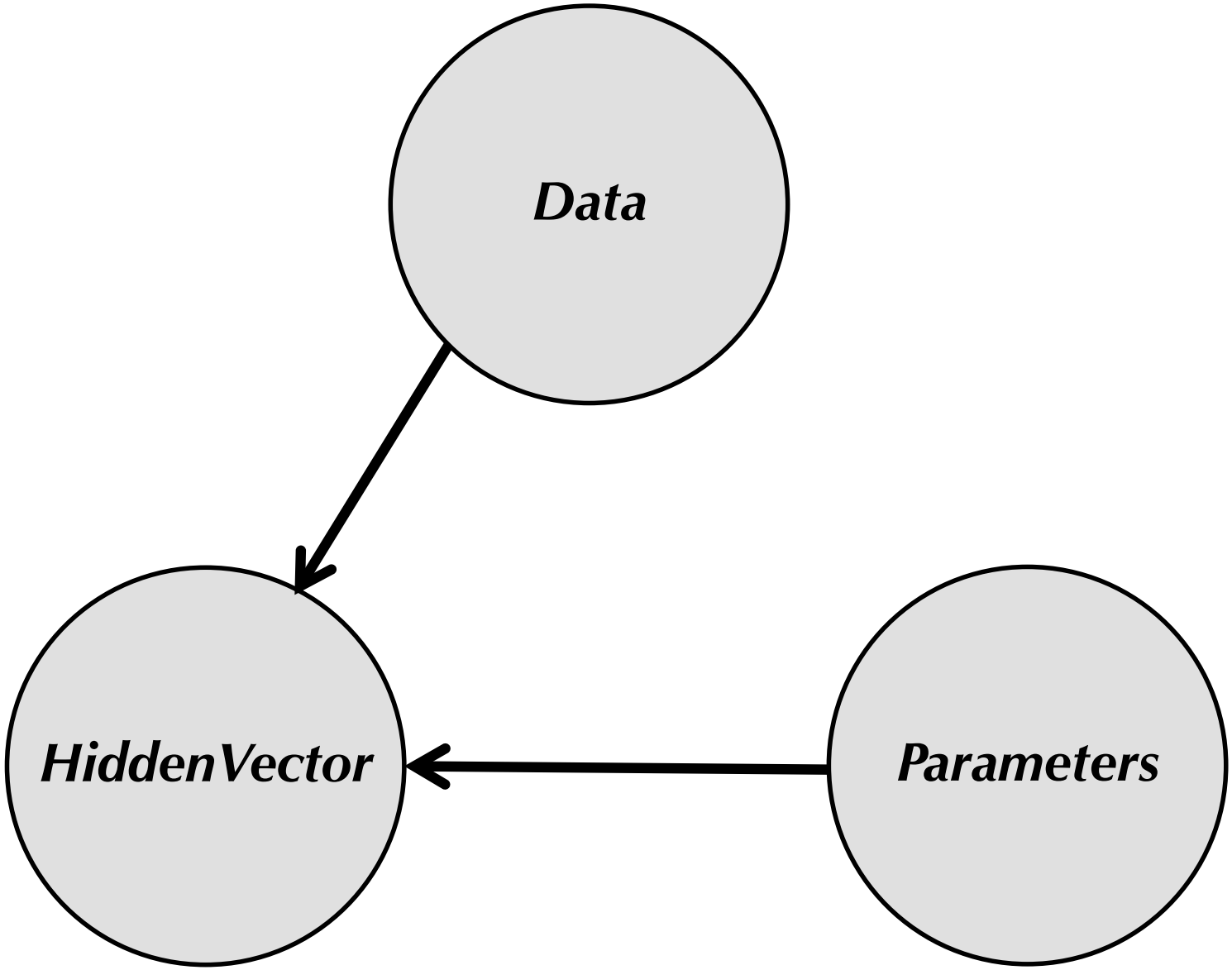
Which coin is more likely to generate the 2nd sequence (with 9 H)?

$\Pr(2^{nd}\text{ sequence}|\theta_A) = \theta_A^9 (1-\theta_A)^1 = 0.35^9 \bullet 0.65^1 \approx 0.00005$ <
$\Pr(2^{nd}\text{ sequence}|\theta_B) = \theta_B^9 (1-\theta_B)^1 = 0.80^9 \bullet 0.20^1 \approx 0.02684$

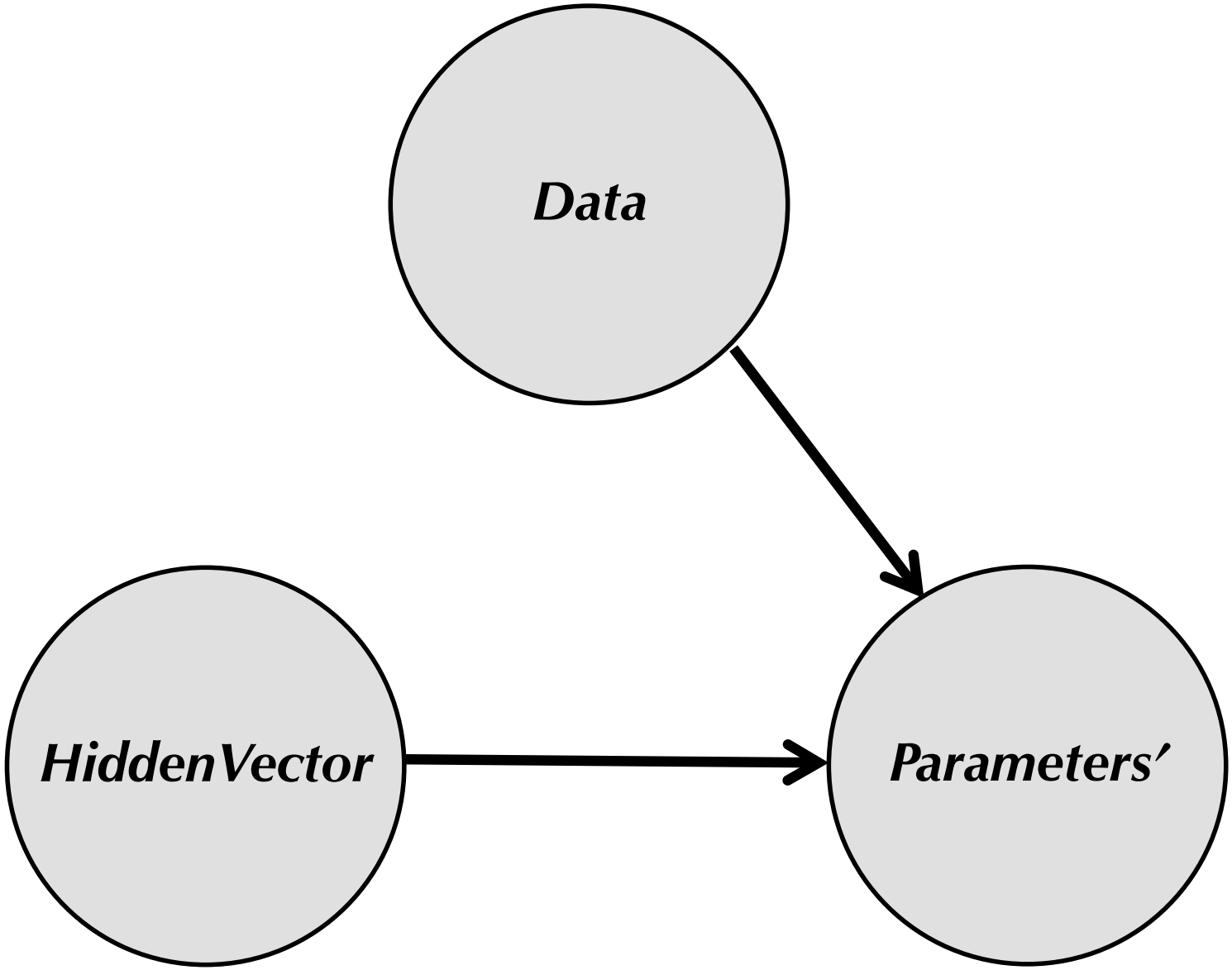
HiddenVector Reconstructed!

Data	*HiddenVector*	*Parameters*=(θ_A, θ_B)
0.4	1	
0.9	0	
0.8	0	← (0.35, 0.80)
0.3	1	
0.7	0	

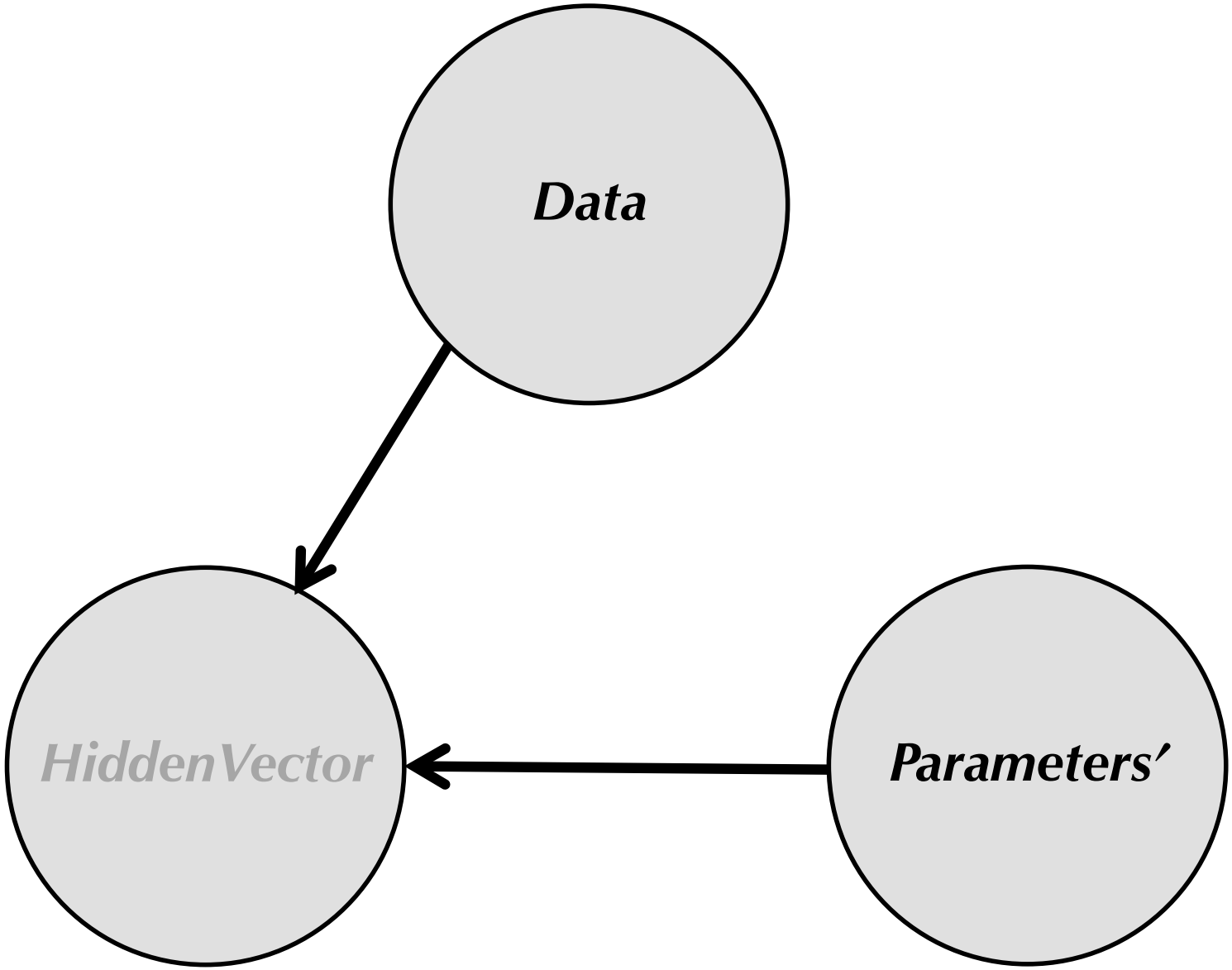
Reconstructing *HiddenVector* and *Parameters*



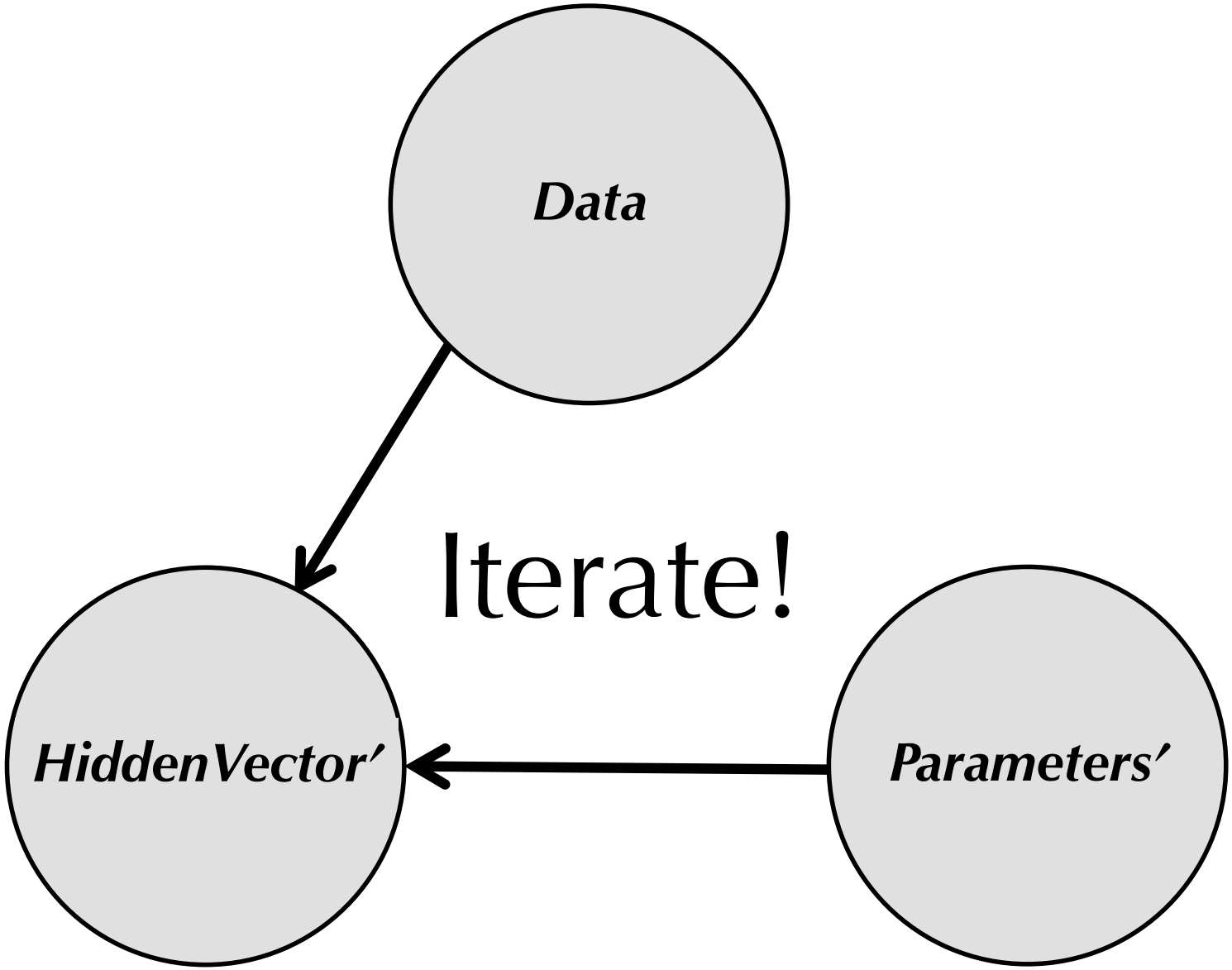
Reconstructing *HiddenVector* and *Parameters*



Reconstructing *HiddenVector* and *Parameters*



Reconstructing *HiddenVector* and *Parameters*

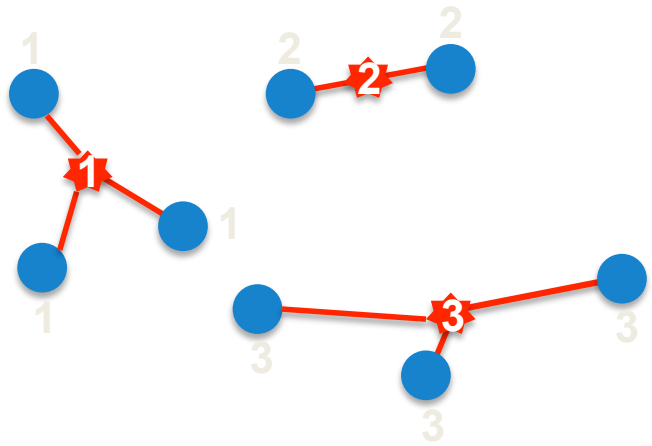


From Coin Flipping to k-means Clustering: Where Are *Data, HiddenVector,* and *Parameters*?

Data: data points $Data = (Data_1,\ldots,Data_n)$

Parameters: $Centers = (Center_1,\ldots,Center_k)$

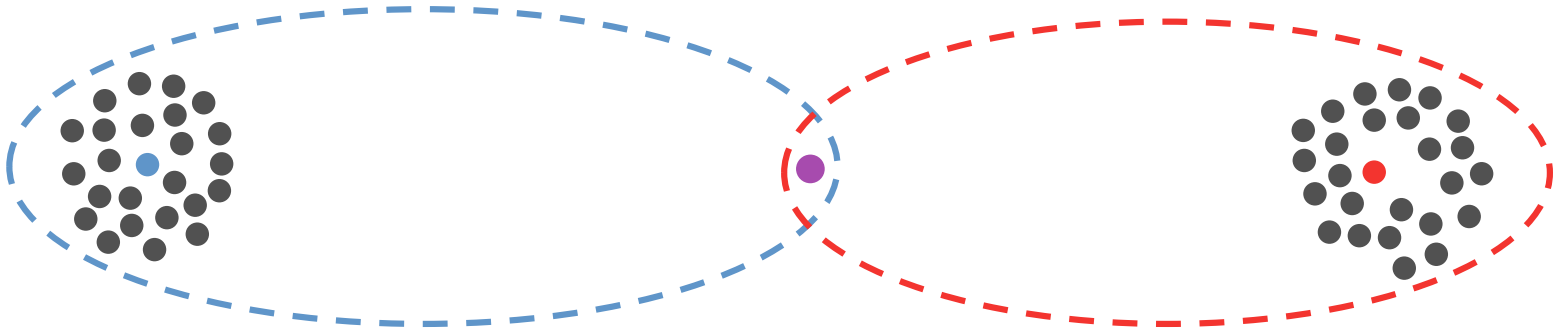
HiddenVector: assignments of data points to k centers (n-dimensional vector with coordinates varying from 1 to k).



Coin Flipping and Soft Clustering

- **Coin flipping**: how would you select between coins A and B if $\Pr(\text{sequence}|\theta_A) = \Pr(\text{sequence}|\theta_B)$?

- ***k*-means clustering:** what cluster would you assign a data point it to if it is a midpoint of centers C_1 and C_2?



Soft assignments: assigning C_1 and C_2 "responsibility" ≈0.5 for a midpoint.

From *Data & Parameters to HiddenVector*

Data	***HiddenVector***	***Parameters*** = (θ_A, θ_B)
0.4	?	
0.9	?	
0.8	?	← (0.60, 0.82)
0.3	?	
0.7	?	

Which coin is more likely to have generated the first sequence (with 4 H)?

$\Pr(1^{st}\text{ sequence}|\theta_A) = \theta_A^5 (1-\theta_A)^5 = 0.60^4 \bullet 0.40^6 \approx 0.000531 >$
$\Pr(1^{st}\text{ sequence}|\theta_B) = \theta_B^5 (1-\theta_B)^5 = 0.82^4 \bullet 0.18^6 \approx 0.000015$

Memory Flash:
From *Data & Parameters to HiddenVector*

Data	***HiddenVector***	***Parameters*** = (θ_A, θ_B)
0.4	1	
0.9	?	
0.8	?	← (0.60, 0.82)
0.3	?	
0.7	?	

Which coin is more likely to have generated the first sequence (with 4 H)?

$\Pr(1^{st}\text{ sequence}|\theta_A) = \theta_A^5 (1-\theta_A)^5 = 0.60^4 \bullet 0.40^6 \approx 0.000531 >$
$\Pr(1^{st}\text{ sequence}|\theta_B) = \theta_B^5 (1-\theta_B)^5 = 0.82^4 \bullet 0.18^6 \approx 0.000015$

From *Data & Parameters* to ***HiddenMatrix***

Data	***HiddenMatrix***		***Parameters*** = (θ_A, θ_B)
0.4	0.97	0.03	
0.9	?		
0.8	?	←	(0.60, 0.82)
0.3	?		
0.7	?		

What are the **responsibilities** of coins for this sequence?

$$\Pr(1^{st}\ \text{sequence}|\theta_A) \approx 0.000531 >$$
$$\Pr(1^{st}\ \text{sequence}|\theta_B) \approx 0.000015$$

$$0.000531 / (0.000531 + 0.000015) \approx 0.97$$
$$0.000015 / (0.000531 + 0.000015) \approx 0.03$$

From *Data & Parameters* to ***HiddenMatrix***

Data	*HiddenMatrix*		*Parameters* = (θ_A, θ_B)
0.4	0.97	0.03	
0.9	0.12	0.88	
0.8	?		← (0.60, 0.82)
0.3	?		
0.7	?		

What are the responsibilities of coins for the 2nd sequence?

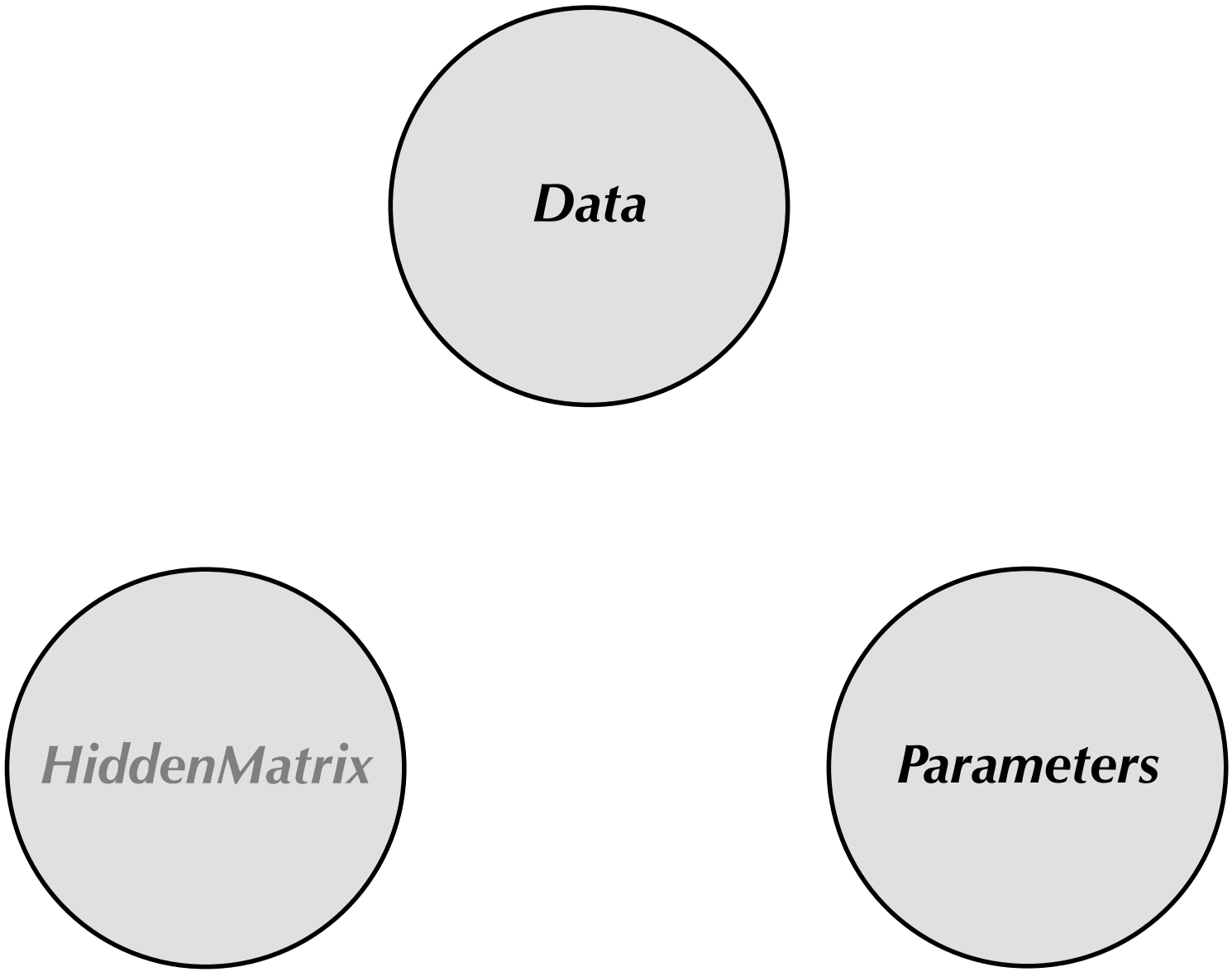
$\Pr(2^{nd}\text{ sequence}|\theta_A) \approx 0.0040 <$
$\Pr(2^{nd}\text{ sequence}|\theta_B) \approx 0.0302$

$0.0040 / (0.0040 + 0.0302) = 0.12$
$0.0342 / (0.0040 + 0.0342) = 0.88$

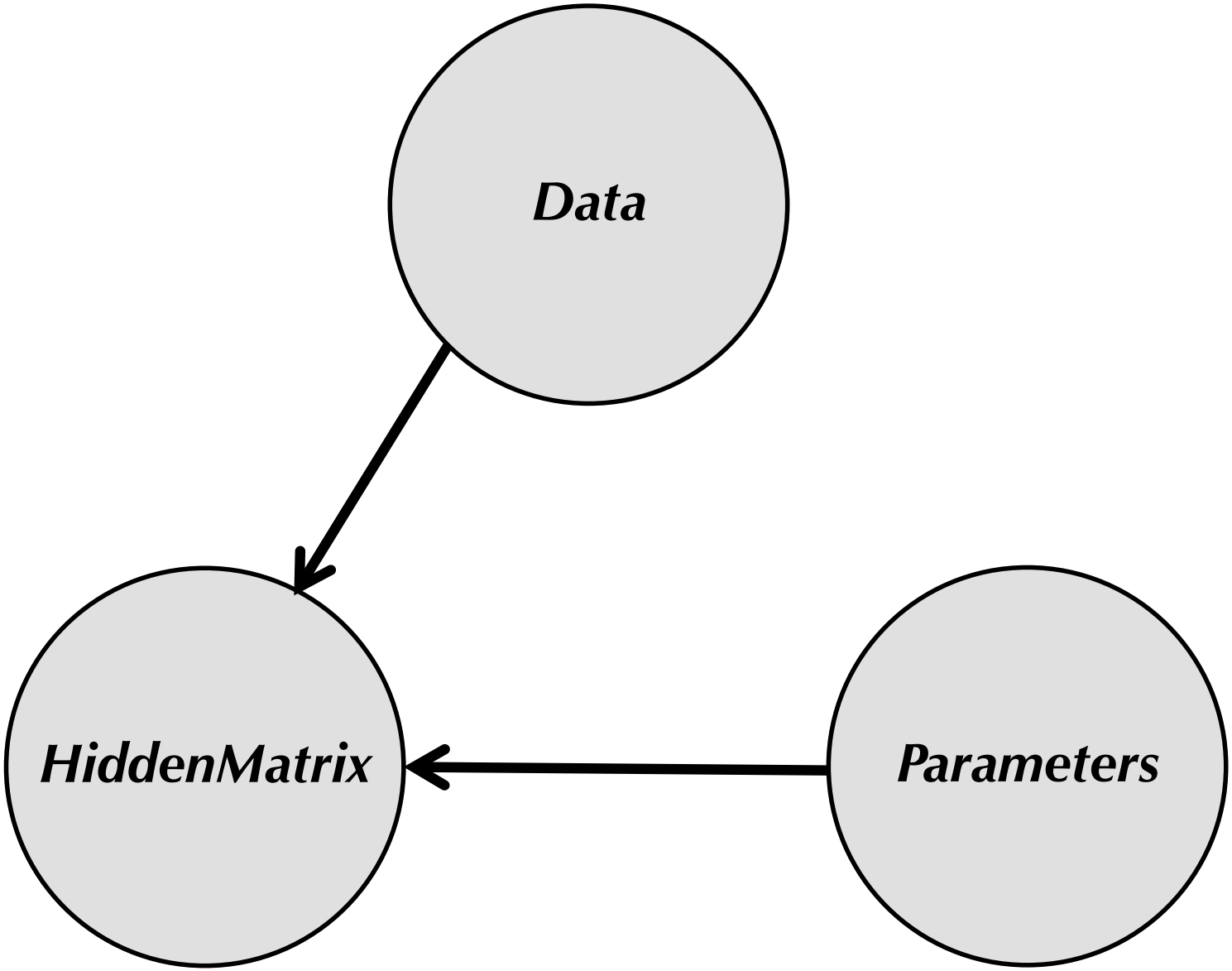
HiddenMatrix Reconstructed!

Data	*HiddenMatrix*		*Parameters* = (θ_A, θ_B)
0.4	0.97	0.03	
0.9	0.12	0.88	
0.8	0.29	0.71	← (0.60, 0.82)
0.3	0.99	0.01	
0.7	0.55	0.45	

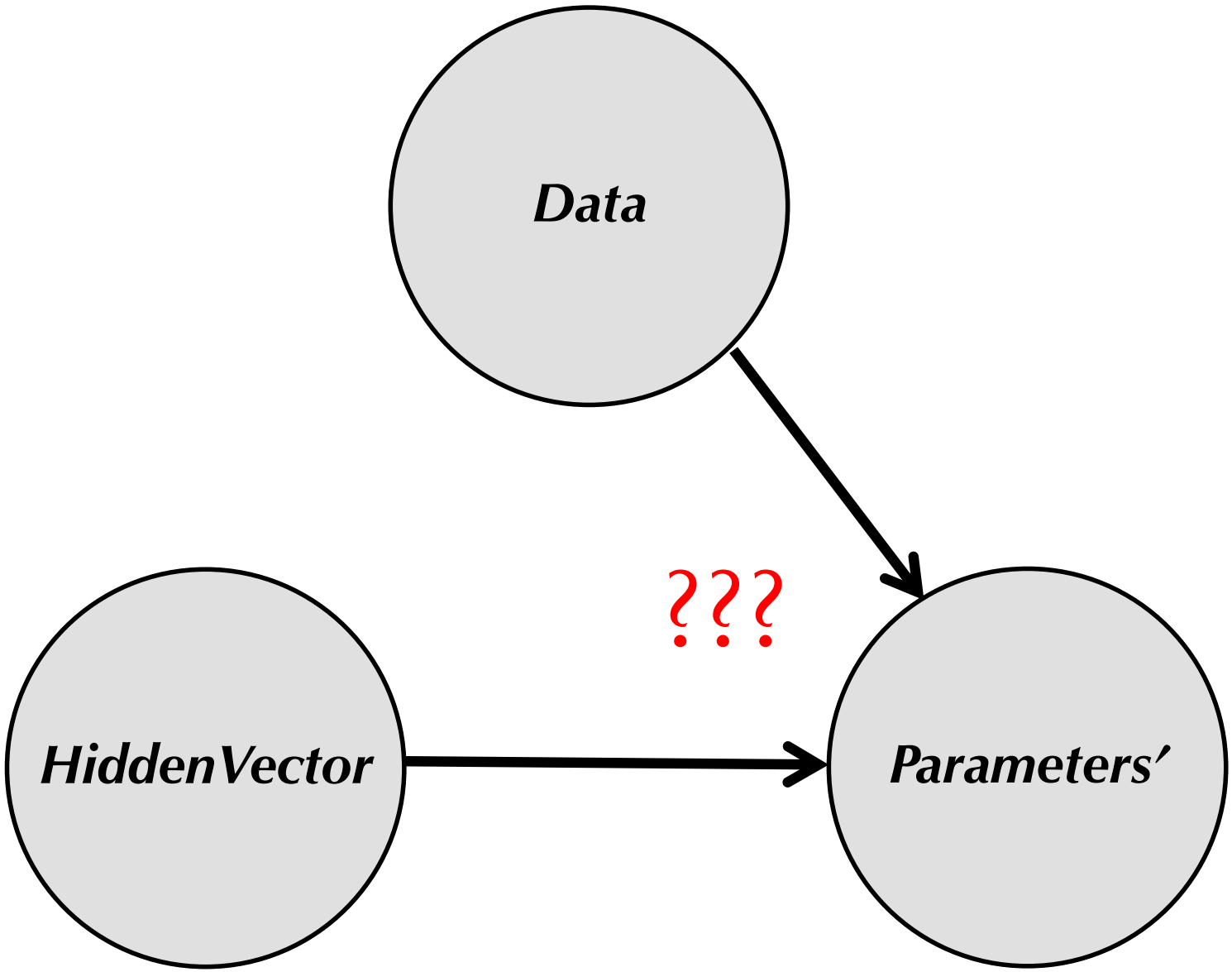
Expectation Maximization Algorithm



E-step



M-step



Memory Flash: Dot Product

	Data		***HiddenVector***	***Parameters***=(θ_A, θ_B)
HTTTHTTHTH	0.4	*	1	
HHHHTHHHHH	0.9	*	0	
HTHHHHHTHH	0.8	*	0	???
HTTTTTHHTT	0.3	*	1	
THHHTHHHTH	0.7	*	0	

θ_A = *Data* * *HiddenVector* / **1** * *HiddenVector*

θ_B = *Data* * (**1**-*HiddenVector*) / **1** * (**1**-*HiddenVector*)

From *Data* & *HiddenMatrix* to ***Parameters***

	Data	***HiddenVector***	***Parameters***=(θ_A,θ_B)
HTTTHTTHTH	**0.4**	**1**	
HHHHTHHHHH	**0.9**	**0**	
HTHHHHHTHH	**0.8**	**0**	
HTTTTTHHTT	**0.3**	**1**	
THHHTHHHTH	**0.7**	**0**	

θ_A = *Data* * *HiddenVector* / **1** * *HiddenVector*

θ_B = *Data* * (**1**-*HiddenVector*) / **1** * (**1**-*HiddenVector*)

HiddenVector = (1 0 0 1 0)

What is *HiddenMatrix* corresponding to this *HiddenVector*?

From *Data* & *HiddenMatrix* to ***Parameters***

	Data	***HiddenVector***	***Parameters***=(θ_A,θ_B)
HTTTHTTHTH	0.4	1	
HHHHTHHHHH	0.9	0	
HTHHHHHTHH	0.8	0	
HTTTTTHHTT	0.3	1	
THHHTHHHTH	0.7	0	

θ_A = *Data* * *HiddenVector* / **1** * *HiddenVector*

θ_A = *Data* * 1st row of *HiddenMatrix* / **1***1st row of *HiddenMatrix*

θ_B = *Data* * (**1**-*HiddenVector*) / **1** * (**1**-*HiddenVector*)

θ_B = *Data* * 2nd row of *HiddenMatrix* / **1***2nd row of *HiddenMatrix*

HiddenVector = (1 0 0 1 0)

Hidden Matrix = 1 0 0 1 0 = *HiddenVector*
0 1 1 0 1 = **1** - *HiddenVector*

From *Data* & *HiddenMatrix* to ***Parameters***

	Data	***HiddenMatrix***	***Parameters***=(θ_A,θ_B)
HTTTHTTHTH	**0.4**	**0.97**	**0.03**
HHHHTHHHHH	**0.9**	**0.12**	**0.88**
HTHHHHHTHH	**0.8**	**0.29**	**0.71**
HTTTTTHHTT	**0.3**	**0.99**	**0.01**
THHHTHHHTH	**0.7**	**0.55**	**0.45**

θ_A = *Data* * *HiddenVector* / **1** * *HiddenVector*

θ_A = *Data* * 1st row of *HiddenMatrix* / **1***1st row of *HiddenMatrix*

θ_B = *Data* * (**1**-*HiddenVector*) / **1** * (**1**-*HiddenVector*)

θ_B = *Data* * 2nd row of *HiddenMatrix* / **1***2nd row of *HiddenMatrix*

HiddenVector = (1 0 0 1 0)

Hidden Matrix =

.97	.03	.29	.99	.55
.03	.97	.71	.01	.45

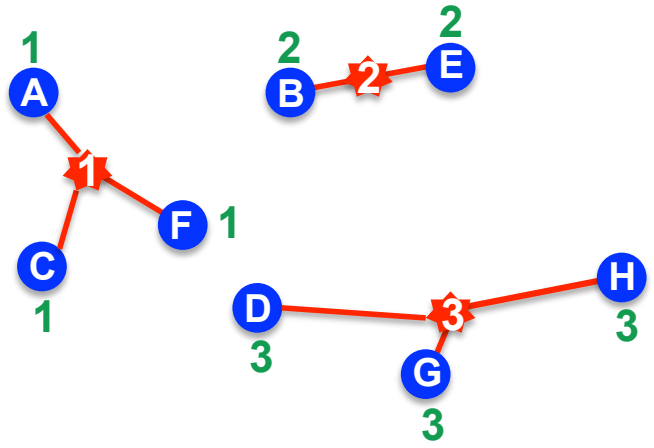
From *HiddenVector* to *HiddenMatrix*

Data: data points $Data = \{Data_1, \ldots, Data_n\}$
Parameters: $Centers = \{Center_1, \ldots, Center_k\}$
HiddenVector: assignments of data points to centers

H	A	B	C	D	E	F	G
HiddenVector	1	2	1	3	2	1	3	3

1	1	0	1	0	0	1	0	0
2	0	1	0	0	1	0	0	0
3	0	0	0	1	0	0	1	1

HiddenMatrix

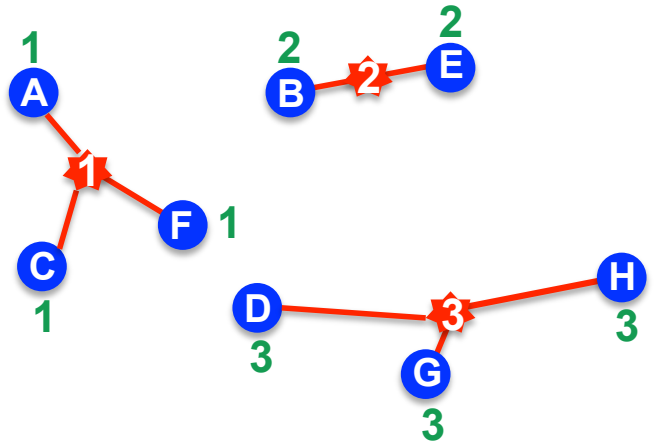


From *HiddenVector* to *HiddenMatrix*

Data: data points $Data = \{Data_1, \ldots, Data_n\}$
Parameters: $Centers = \{Center_1, \ldots, Center_k\}$
$HiddenMatrix_{i,j}$: responsibility of center *i* for data point *j*

HiddenMatrix

	A	B	C	D	E	F	G
1	H 0.7	0	1	0	0	1	0	0
2	0.2	1	0	0	1	0	0	0
3	0.1	0	0	1	0	0	1	1

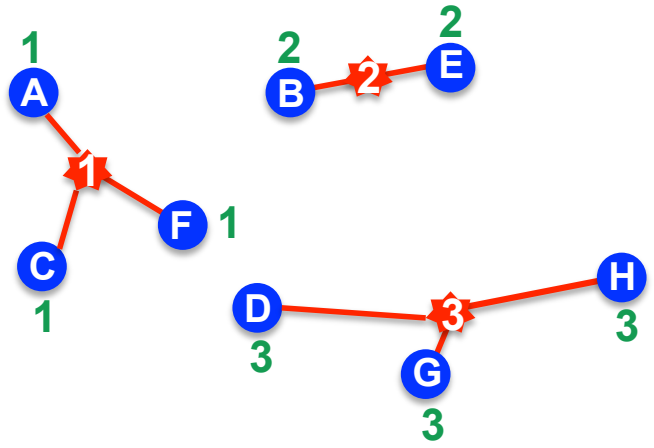


From *HiddenVector* to *HiddenMatrix*

Data: data points $Data = \{Data_1, \ldots, Data_n\}$
Parameters: $Centers = \{Center_1, \ldots, Center_k\}$
$HiddenMatrix_{i,j}$: responsibility of center *i* for data point *j*

HiddenMatrix

	A	B	C	D	E	F	G	H
1	0.70	0.15	0.73	0.40	0.15	0.80	0.05	0.05
2	0.20	0.80	0.17	0.20	0.80	0.10	0.05	0.20
3	0.10	0.05	0.10	0.40	0.05	0.10	0.90	0.75



Responsibilities and the Law of Gravitation

planets

stars

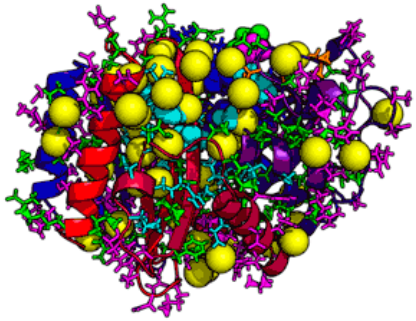
0.70	0.15	0.73	0.40	0.15	0.80	0.05	0.05
0.20	0.80	0.17	0.20	0.80	0.10	0.05	0.20
0.10	0.05	0.10	0.40	0.05	0.10	0.90	0.75

responsibility of star *i* for a planet *j* is proportional to the pull (Newtonian law of gravitation):

$$Force_{i,j} = 1/distance(Data_j, Center_i)^2$$

$$HiddenMatrix_{ij} := Force_{i,j} \,/\, \sum_{\text{all centers } j} Force_{i,j}$$

Responsibilities and Statistical Mechanics



data points

centers

0.70	0.15	0.73	0.40	0.15	0.80	0.05	0.05
0.20	0.80	0.17	0.20	0.80	0.10	0.05	0.20
0.10	0.05	0.10	0.40	0.05	0.10	0.90	0.75

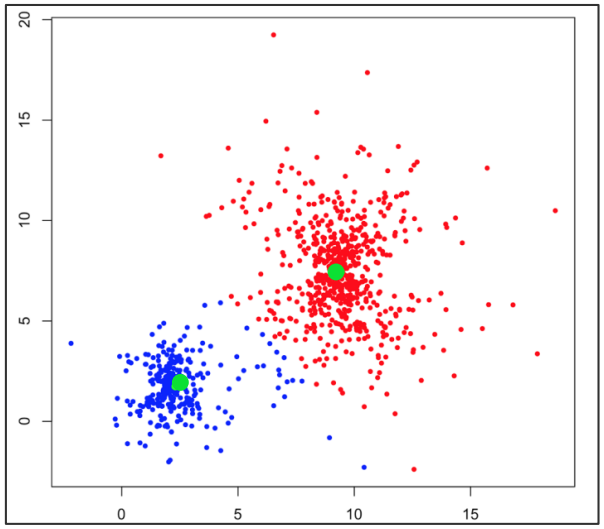
responsibility of center *i* for a data point *j* is proportional to

$$Force_{i,j} = e^{-\beta \cdot \text{distance}(Data j,\ Center i)}$$

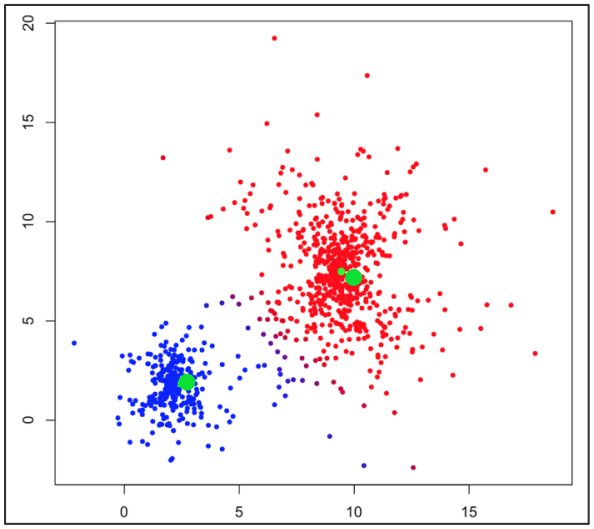
where β is a **stiffness parameter**.

$$HiddenMatrix_{ij} := Force_{i,j} \,/\, \sum_{\text{all centers } j} Force_{i,j}$$

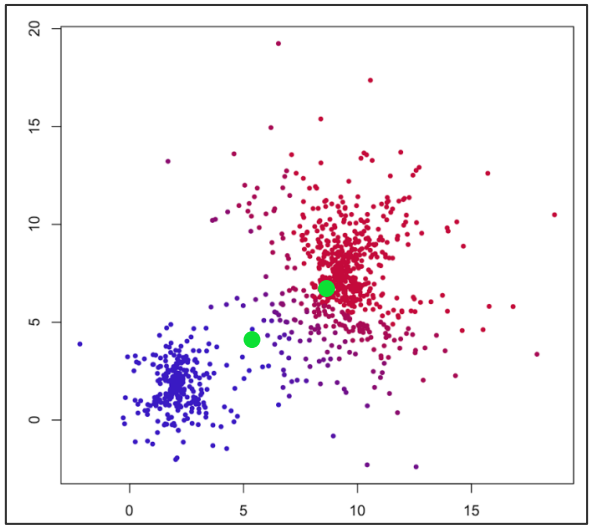
How Does Stiffness Affect Clustering?



Hard *k*-means clustering



Soft *k*-means clustering (stiffness β=1)



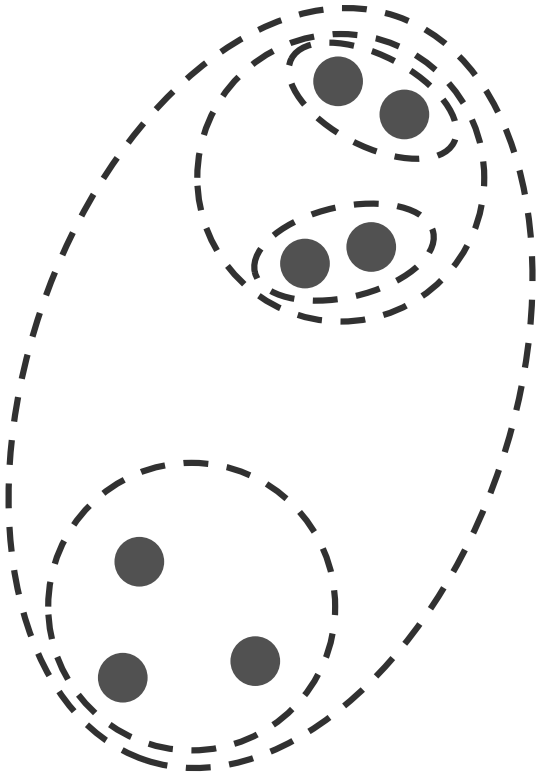
Soft *k*-means clustering (stiffness β= 0.3)

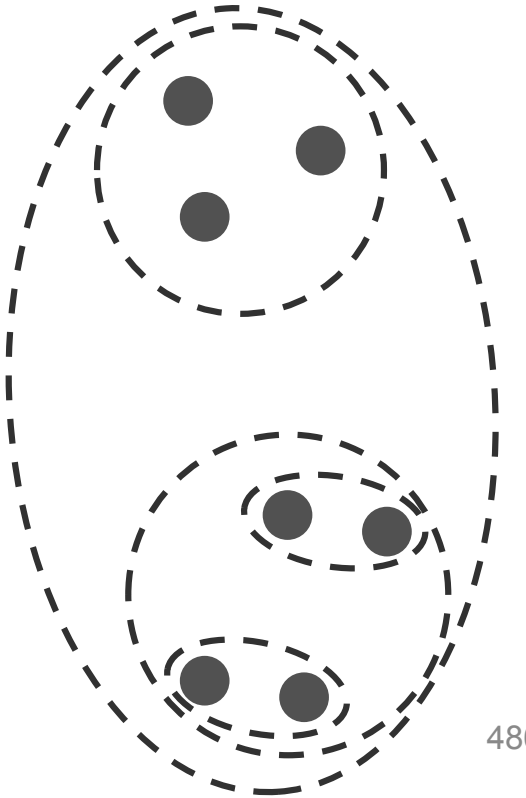
Stratification of Clusters

Clusters often have **subclusters**, which have subsubclusters, and so on.

Stratification of Clusters

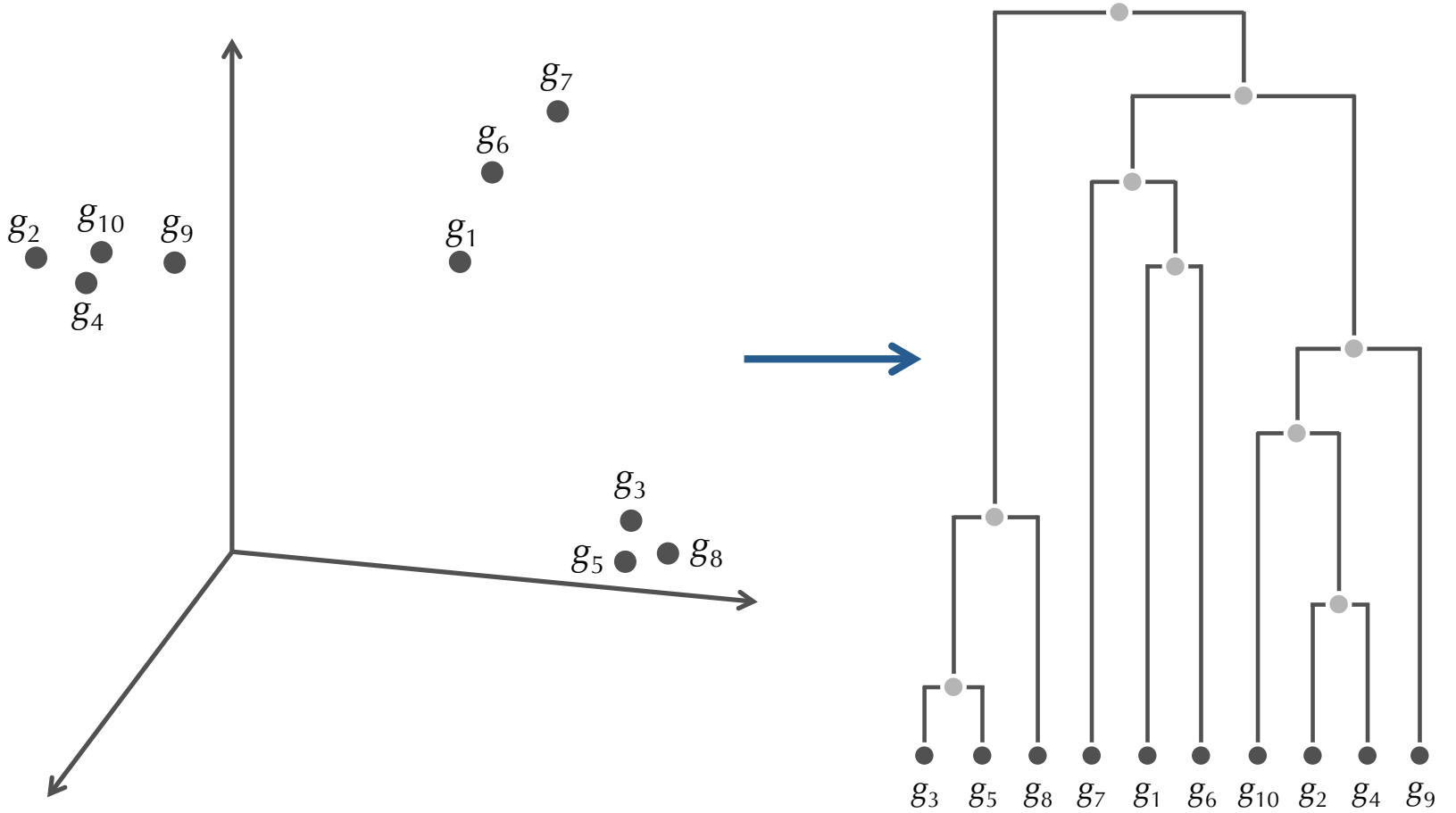
Clusters often have **subclusters**, which have sub-subclusters, and so on.





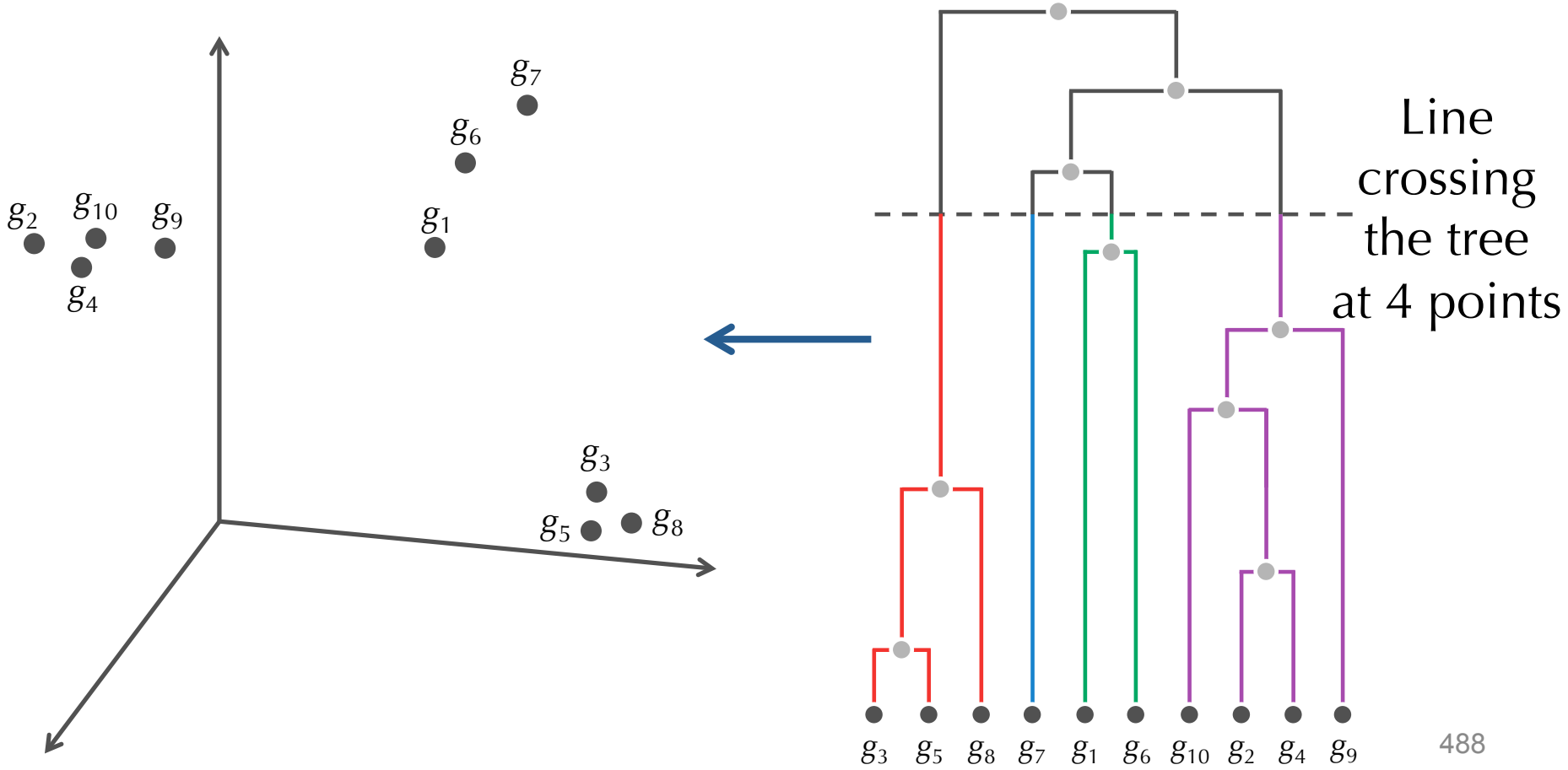
From Data to a Tree

To capture stratification, the **hierarchical clustering** algorithm organizes *n* data points into a tree.



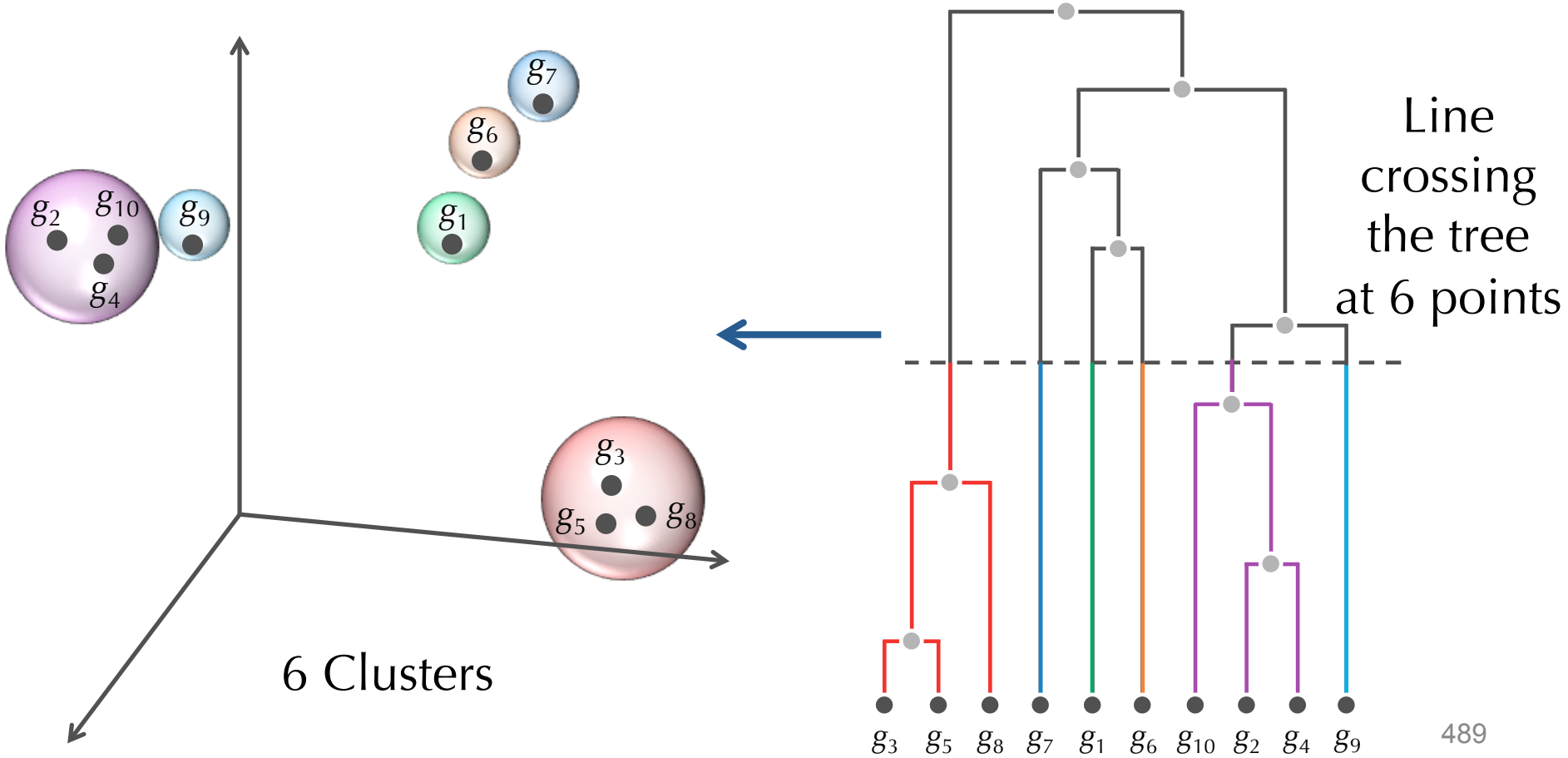
From a Tree to a Partition into 4 Clusters

To capture stratification, the **hierarchical clustering** algorithm organizes *n* data points into a tree.



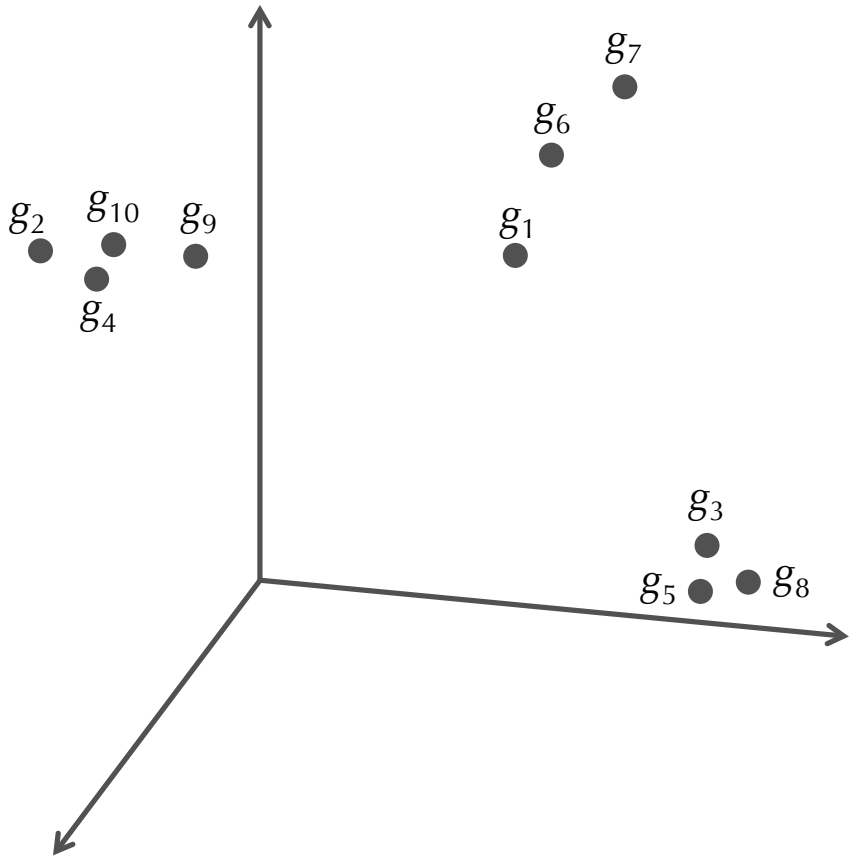
From a Tree to a Partition into 6 Clusters

To capture stratification, the **hierarchical clustering** algorithm first organizes *n* data points into a tree.



Constructing the Tree

Hierarchical clustering starts from a transformation of *n* x *m* expression matrix into *n* x *n* **similarity matrix** or **distance matrix.**

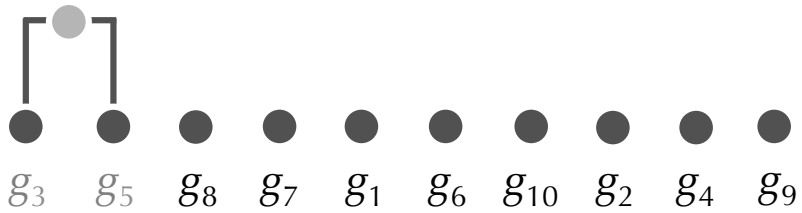


Distance Matrix

	g_1	g_2	g_3	g_4	g_5	g_6	g_7	g_8	g_9	g_{10}
g_1	0.0	8.1	9.2	7.7	9.3	2.3	5.1	10.2	6.1	7.0
g_2	8.1	0.0	12.0	0.9	12.0	9.5	10.1	12.8	2.0	1.0
g_3	9.2	12.0	0.0	11.2	0.7	11.1	8.1	1.1	10.5	11.5
g_4	7.7	0.9	11.2	0.0	11.2	9.2	9.5	12.0	1.6	1.1
g_5	9.3	12.0	0.7	11.2	0.0	11.2	8.5	1.0	10.6	11.6
g_6	2.3	9.5	11.1	9.2	11.2	0.0	5.6	12.1	7.7	8.5
g_7	5.1	10.1	8.1	9.5	8.5	5.6	0.0	9.1	8.3	9.3
g_8	10.2	12.8	1.1	12.0	1.0	12.1	9.1	0.0	11.4	12.4
g_9	6.1	2.0	10.5	1.6	10.6	7.7	8.3	11.4	0.0	1.1
g_{10}	7.0	1.0	11.5	1.1	11.6	8.5	9.3	12.4	1.1	0.0

Constructing the Tree

Identify the two closest clusters and merge them.

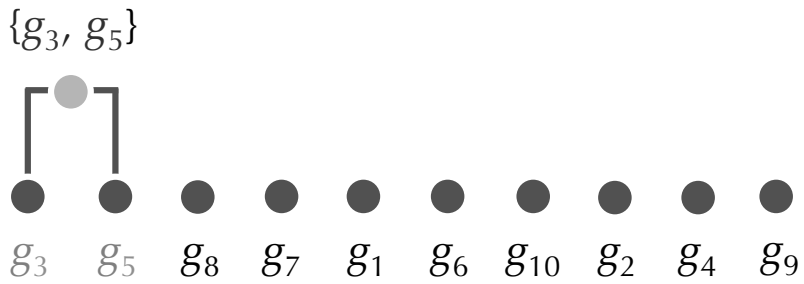


	g_1	g_2	g_3	g_4	g_5	g_6	g_7	g_8	g_9	g_{10}
g_1	0.0	8.1	9.2	7.7	9.3	2.3	5.1	10.2	6.1	7.0
g_2	8.1	0.0	12.0	0.9	12.0	9.5	10.1	12.8	2.0	1.0
g_3	9.2	12.0	0.0	11.2	**0.7**	11.1	8.1	1.1	10.5	11.5
g_4	7.7	0.9	11.2	0.0	11.2	9.2	9.5	12.0	1.6	1.1
g_5	9.3	12.0	0.7	11.2	0.0	11.2	8.5	1.0	10.6	11.6
g_6	2.3	9.5	11.1	9.2	11.2	0.0	5.6	12.1	7.7	8.5
g_7	5.1	10.1	8.1	9.5	8.5	5.6	0.0	9.1	8.3	9.3
g_8	10.2	12.8	1.1	12.0	1.0	12.1	9.1	0.0	11.4	12.4
g_9	6.1	2.0	10.5	1.6	10.6	7.7	8.3	11.4	0.0	1.1
g_{10}	7.0	1.0	11.5	1.1	11.6	8.5	9.3	12.4	1.1	0.0

Constructing the Tree

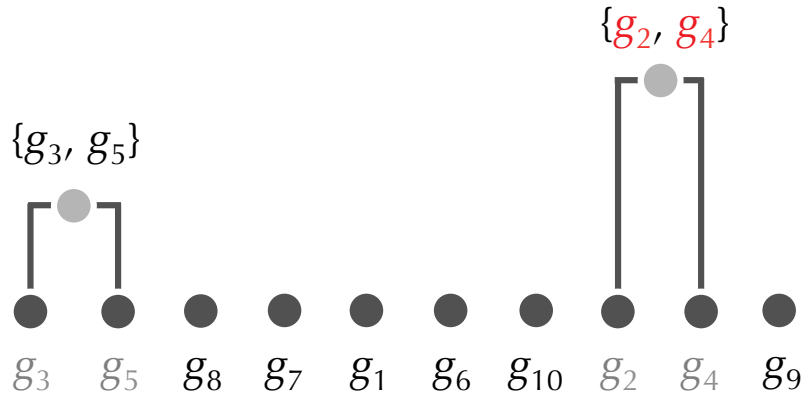
Recompute the distance between two clusters as average distance between elements in the cluster.

	g_1	g_2	g_3, g_5	g_4	g_6	g_7	g_8	g_9	g_{10}
g_1	0.0	8.1	9.2	7.7	2.3	5.1	10.2	6.1	7.0
g_2	8.1	0.0	12.0	0.9	9.5	10.1	12.8	2.0	1.0
g_3, g_5	9.2	12.0	0.0	11.2	11.1	8.1	1.0	10.5	11.5
g_4	7.7	0.9	11.2	0.0	9.2	9.5	12.0	1.6	1.1
g_6	2.3	9.5	11.1	9.2	0.0	5.6	12.1	7.7	8.5
g_7	5.1	10.1	8.1	9.5	5.6	0.0	9.1	8.3	9.3
g_8	10.2	12.8	1.0	12.0	12.1	9.1	0.0	11.4	12.4
g_9	6.1	2.0	10.5	1.6	7.7	8.3	11.4	0.0	1.1
g_{10}	7.0	1.0	11.5	1.1	8.5	9.3	12.4	1.1	0.0



Constructing the Tree

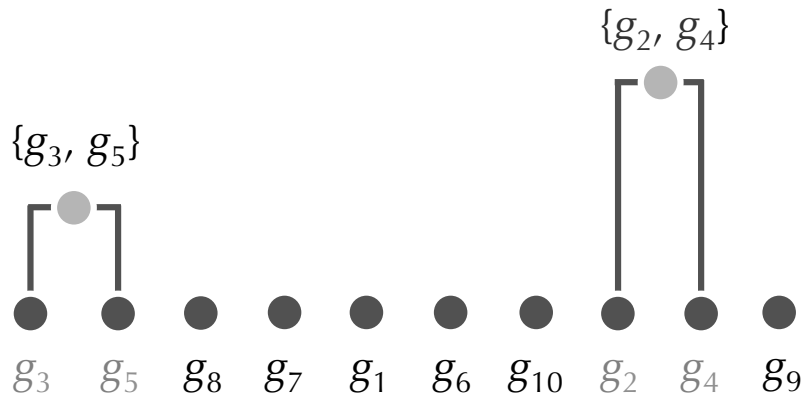
Identify the two closest clusters and merge them.



	g_1	g_2	g_3, g_5	g_4	g_6	g_7	g_8	g_9	g_{10}
g_1	0.0	8.1	9.2	7.7	2.3	5.1	10.2	6.1	7.0
g_2	8.1	0.0	12.0	**0.9**	9.5	10.1	12.8	2.0	1.0
g_3, g_5	9.2	12.0	0.0	11.2	11.1	8.1	1.0	10.5	11.5
g_4	7.7	0.9	11.2	0.0	9.2	9.5	12.0	1.6	1.1
g_6	2.3	9.5	11.1	9.2	0.0	5.6	12.1	7.7	8.5
g_7	5.1	10.1	8.1	9.5	5.6	0.0	9.1	8.3	9.3
g_8	10.2	12.8	1.0	12.0	12.1	9.1	0.0	11.4	12.4
g_9	6.1	2.0	10.5	1.6	7.7	8.3	11.4	0.0	1.1
g_{10}	7.0	1.0	11.5	1.1	8.5	9.3	12.4	1.1	0.0

Constructing the Tree

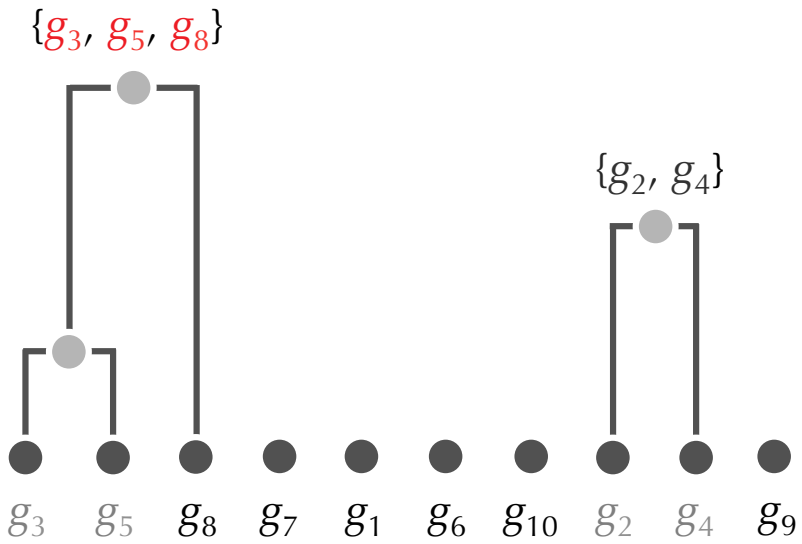
Recompute the distance between two clusters (as average distance between elements in the cluster).



	g_1	g_2, g_4	g_3, g_5	g_6	g_7	g_8	g_9	g_{10}
g_1	0.0	7.7	9.2	2.3	5.1	10.2	6.1	7.0
g_2, g_4	7.7	0.0	11.2	9.2	9.5	12.0	1.6	1.0
g_3, g_5	9.2	11.2	0.0	11.1	8.1	1.0	10.5	11.5
g_6	2.3	9.2	11.1	0.0	5.6	12.1	7.7	8.5
g_7	5.1	9.5	8.1	5.6	0.0	9.1	8.3	9.3
g_8	10.2	12.0	1.0	12.1	9.1	0.0	11.4	12.4
g_9	6.1	1.6	10.5	7.7	8.3	11.4	0.0	1.1
g_{10}	7.0	1.0	11.5	8.5	9.3	12.4	1.1	0.0

Constructing the Tree

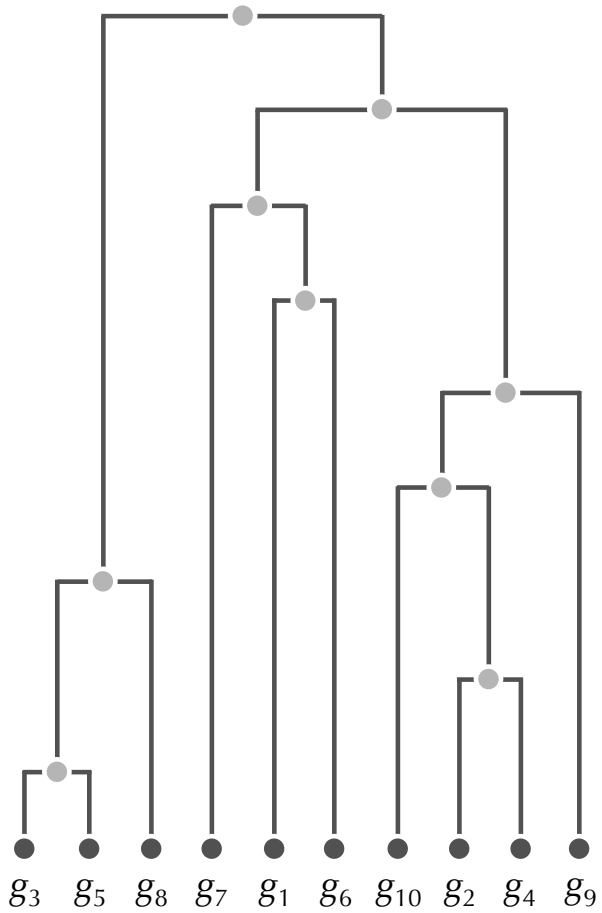
Identify the two closest clusters and merge them.



	g_1	g_2, g_4	g_3, g_5	g_6	g_7	g_8	g_9	g_{10}
g_1	0.0	7.7	9.2	2.3	5.1	10.2	6.1	7.0
g_2, g_4	7.7	0.0	11.2	9.2	9.5	12.0	1.6	1.0
g_3, g_5	9.2	11.2	0.0	11.1	8.1	**1.0**	10.5	11.5
g_6	2.3	9.2	11.1	0.0	5.6	12.1	7.7	8.5
g_7	5.1	9.5	8.1	5.6	0.0	9.1	8.3	9.3
g_8	10.2	12.0	1.0	12.1	9.1	0.0	11.4	12.4
g_9	6.1	1.6	10.5	7.7	8.3	11.4	0.0	1.1
g_{10}	7.0	1.0	11.5	8.5	9.3	12.4	1.1	0.0

Constructing the Tree

Iterate until all elements form a single cluster (root).



Constructing a Tree from a Distance Matrix *D*

HierarchicalClustering (*D, n*)
  Clusters ← *n* single-element clusters labeled 1 to *n*
  T ← a graph with the *n* isolated nodes labeled 1 to *n*
  while there is more than one cluster
    find the two closest clusters C_i and C_j
    merge C_i and C_j into a new cluster C_{new} with $|C_i| + |C_j|$ elements
    add a new node labeled by cluster C_{new} to *T*
    connect node C_{new} to C_i and C_j by directed edges
    remove the rows and columns of *D* corresponding to C_i and C_j
    remove C_i and C_j from *Clusters*
    add a row and column to *D* for the cluster C_{new} by computing
      $D(C_{new}, C)$ for each cluster *C* in *Clusters*
    add C_{new} to *Clusters*
  assign root in *T* as a node with no incoming edges
  return *T*

Different Distance Functions Result in Different Trees

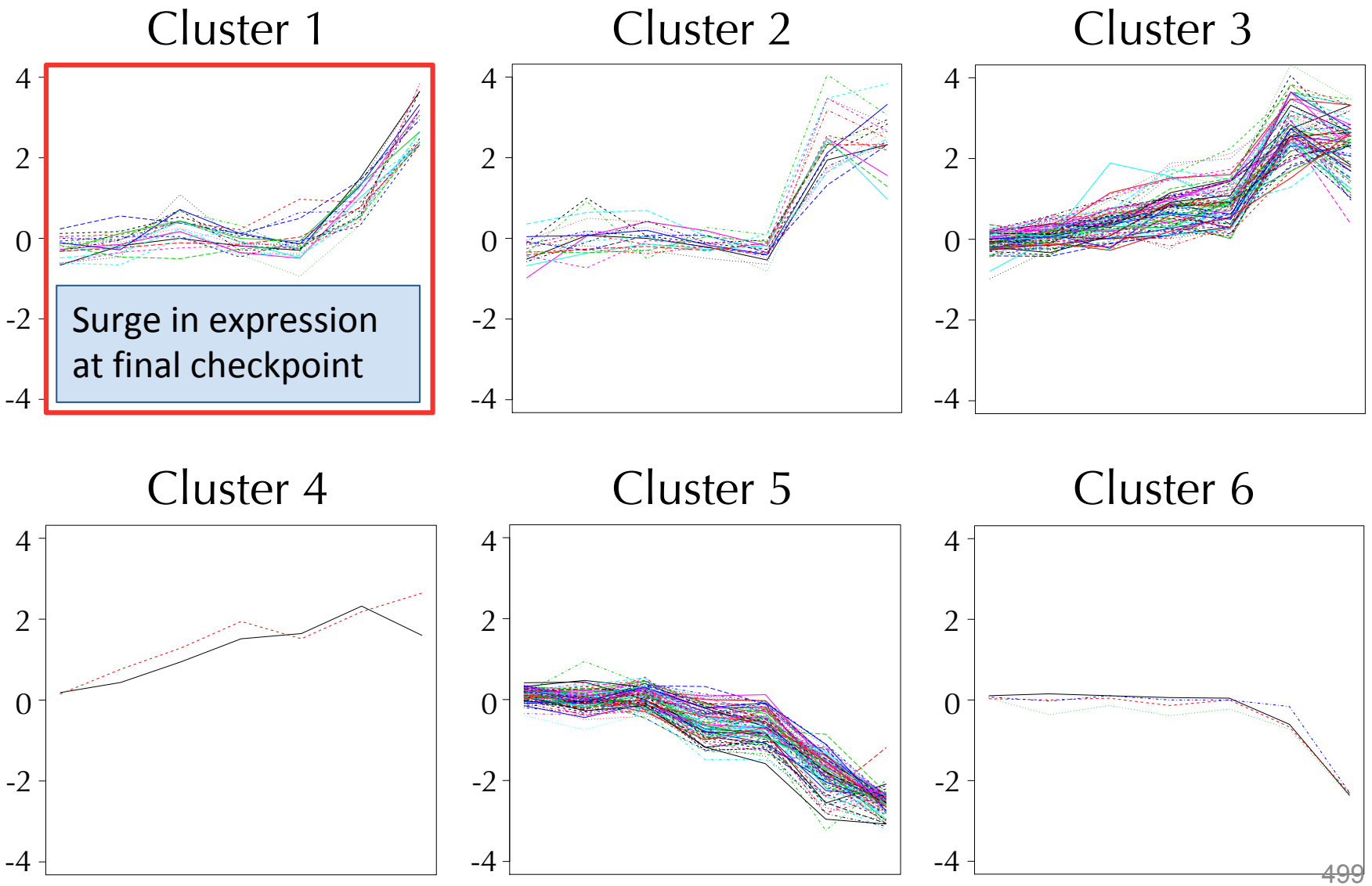
Average distance between elements of two clusters:

$$D_{avg}(C_1, C_2) = (\sum_{\text{all points } i \text{ and } j \text{ in clusters } C1 \text{ and } C2, \text{ respectively}} D_{i,j}) / (|C_1| * |C_2|)$$

Minimum distance between elements of two clusters:

$$D_{min}(C_1, C_2) = \min_{\text{all points } i \text{ and } j \text{ in clusters } C1 \text{ and } C2, \text{ respectively}} D_{i,j}$$

Clusters Constructed by **HierarchicalClustering**



Markov Clustering Algorithm

Unlike most clustering algorithms, the MCL (micans.org/mcl) does not require the number of expected clusters to be specified beforehand. The basic idea underlying the algorithm is that dense clusters correspond to regions with a larger number of paths.

You can find the code at micans.org/mcl

Enright AJ, Van Dongen S, Ouzounis CA. An efficient algorithm for large-scale detection of protein families. Nucleic Acids Res. 2002 30:1575-84.

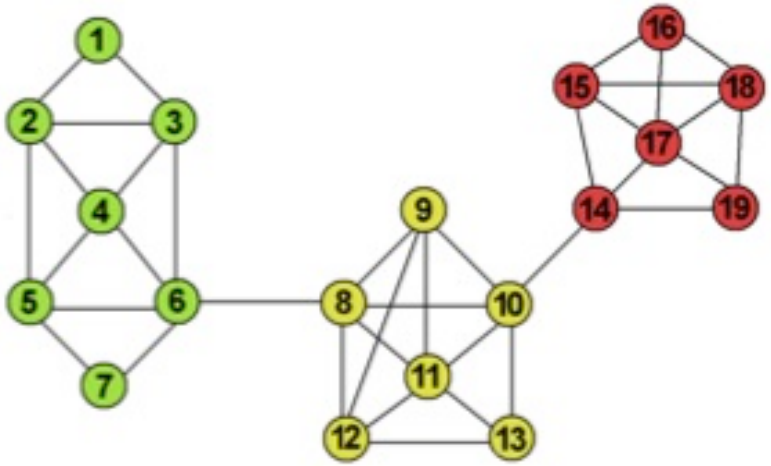
Markov Clustering Algorithm

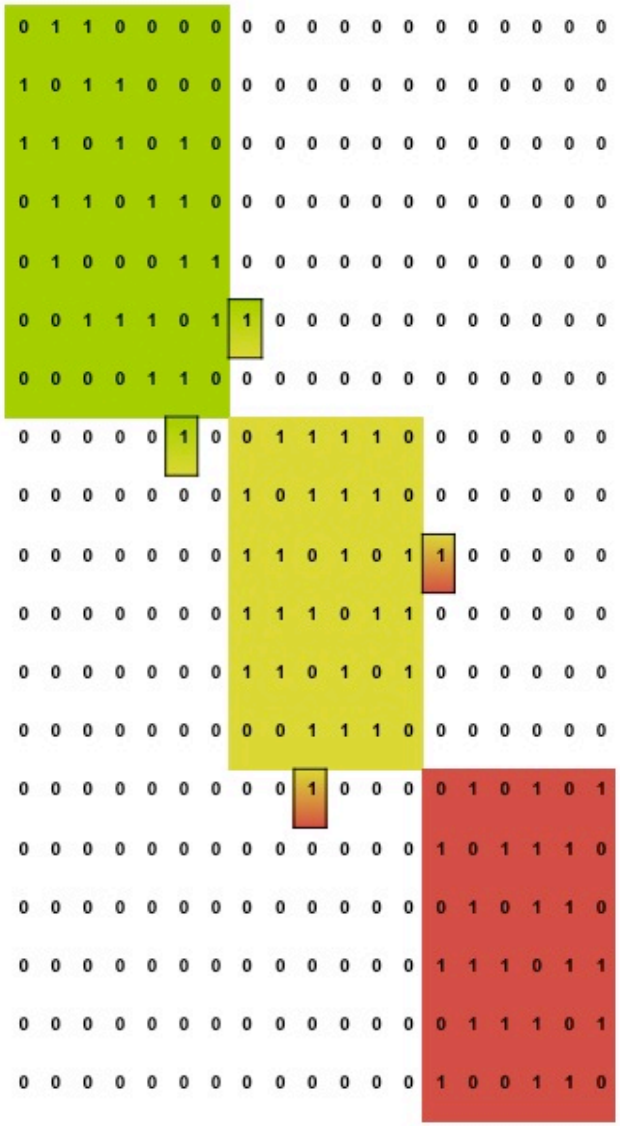
We take a random walk on the graph described by the similarity matrix, but after each step we weaken the links between distant nodes and strengthen the links between nearby nodes.

A random walk has a higher probability to stay inside the cluster than to leave it soon. **The crucial point lies in boosting this effect by an iterative alternation of expansion and inflation steps. An inflation parameter is responsible for both strengthening and weakening of current, i.e. Strengthens strong currents, and weakens already weak currents. An expansion parameter, r, controls the extent of this strengthening / weakening. In the end, this influences the granularity of clusters.**

Markov Clustering Algorithm

Matrix representation

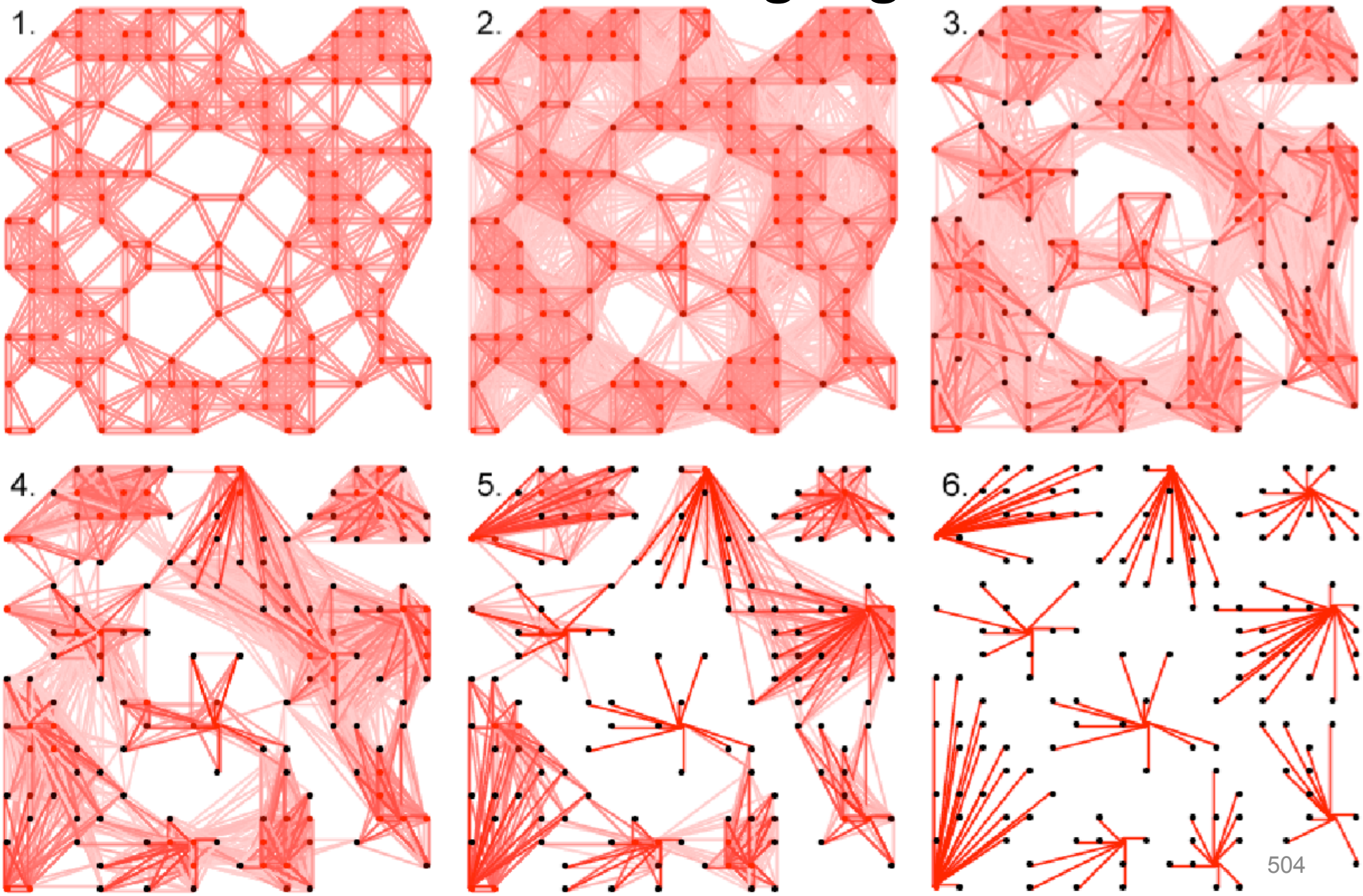




Markov Clustering Algorithm

1. Input is an un-directed graph, with power parameter e (usually =2), and inflation parameter r (usually =2).
2. Create the associated adjacency matrix
3. Normalize the matrix; $M'_{pq} = \frac{M_{pq}}{\sum_i M_{iq}}$
4. Expand by taking the e-th power of the matrix; for example, if $e = 2$ just multiply the matrix by itself.
5. Inflate by taking inflation of the resulting matrix with parameter r : $M_{pq} = \frac{(M_{pq})^r}{\sum_i (M_{iq})^r}$
6. Repeat steps 4 and 5 until a steady state is reached (convergence).

Markov Clustering Algorithm



Markov Clustering Algorithm

The number of steps to converge is not proven, but experimentally shown to be 10 to 100 steps, and mostly consist of sparse matrices after the first few steps.

The expansion step of MCL has time complexity $O(n^3)$. The inflation has complexity $O(n^2)$. However, the matrices are generally very sparse, or at least the vast majority of the entries are near zero. Pruning in MCL involves setting near-zero matrix entries to zero, and can allow sparse matrix operations to improve the speed of the algorithm vastly.

Markov Clustering Algorithm

Input : A weighted undirected graph $G = (V, E)$, expansion parameter e, inflation parameter r

Output : A partitioning of V into disjoint components

$M \leftarrow M(G)$

while M *is not fixpoint* **do**

  $M \leftarrow M^e$

  forall $i \in V$ **do**

    forall $j \in V$ **do**

      $M[i][j] \leftarrow M[i][j]^r$

    forall $j \in V$ **do**

      $M[i][j] \leftarrow \frac{M[i][j]}{\sum_{k \in V} M[i][k]}$

$H \leftarrow$ graph induced by non-zero entries of M

$C \leftarrow$ clustering induced by connected components of H

Stochastic Neighbor Embedding : key points

A popular method for exploring high-dimensional data is something called t-SNE, introduced by van der Maaten and Hinton in 2008. The technique has become widespread in the field of machine learning, since it has an almost magical ability to create compelling two-dimensonal “maps” from data with hundreds or even thousands of dimensions.

The goal is to take a set of points in a high-dimensional space and find a faithful representation of those points in a lower-dimensional space, typically the 2D plane. The algorithm is non-linear and adapts to the underlying data, performing different transformations on different regions. Those differences can be a major source of confusion.

Stochastic Neighbor Embedding : key points

A second feature of t-SNE is a tuneable parameter, “perplexity,” which says (loosely) how to balance attention between local and global aspects of your data. The parameter is, in a sense, a guess about the number of close neighbors each point has. The original paper says, “The performance of SNE is fairly robust to changes in the perplexity, and typical values are between 5 and 50.” But the story is more nuanced than that. Getting the most from t-SNE may mean analyzing multiple plots with different perplexities.

Stochastic Neighbor Embedding : key points

scikit learn

Home | Installation | Documentation | Examples

Google Custom Search

Previous (Multi-dimensi...) | Next (Comparison of...) | Up (Examples)

scikit-learn v0.20.0
Other versions

Please **cite us** if you use the software.

t-SNE: The effect of various perplexity values on the shape

Note: Click here to download the full example code

t-SNE: The effect of various perplexity values on the shape

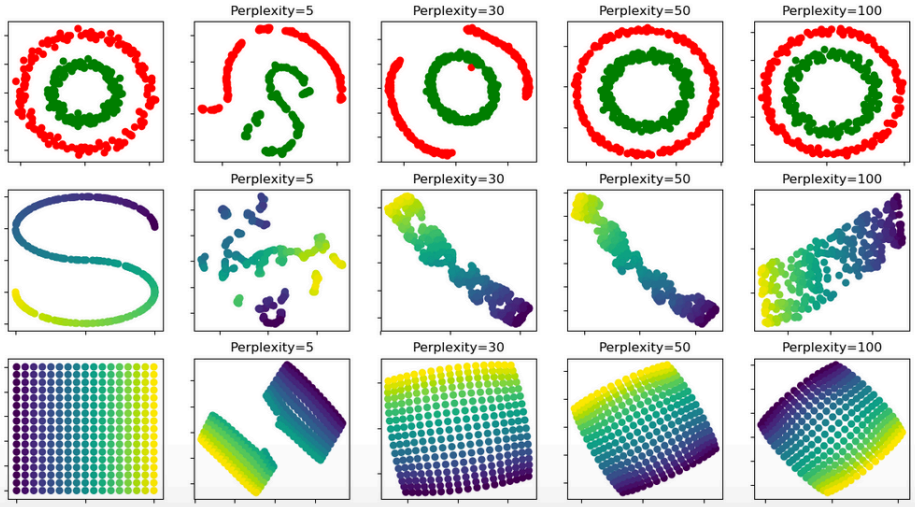
An illustration of t-SNE on the two concentric circles and the S-curve datasets for different perplexity values.

We observe a tendency towards clearer shapes as the preplexity value increases.

The size, the distance and the shape of clusters may vary upon initialization, perplexity values and does not always convey a meaning.

As shown below, t-SNE for higher perplexities finds meaningful topology of two concentric circles, however the size and the distance of the circles varies slightly from the original. Contrary to the two circles dataset, the shapes visually diverge from S-curve topology on the S-curve dataset even for larger perplexity values.

For further details, “How to Use t-SNE Effectively” http://distill.pub/2016/misread-tsne/ provides a good discussion of the effects of various parameters, as well as interactive plots to explore those effects.

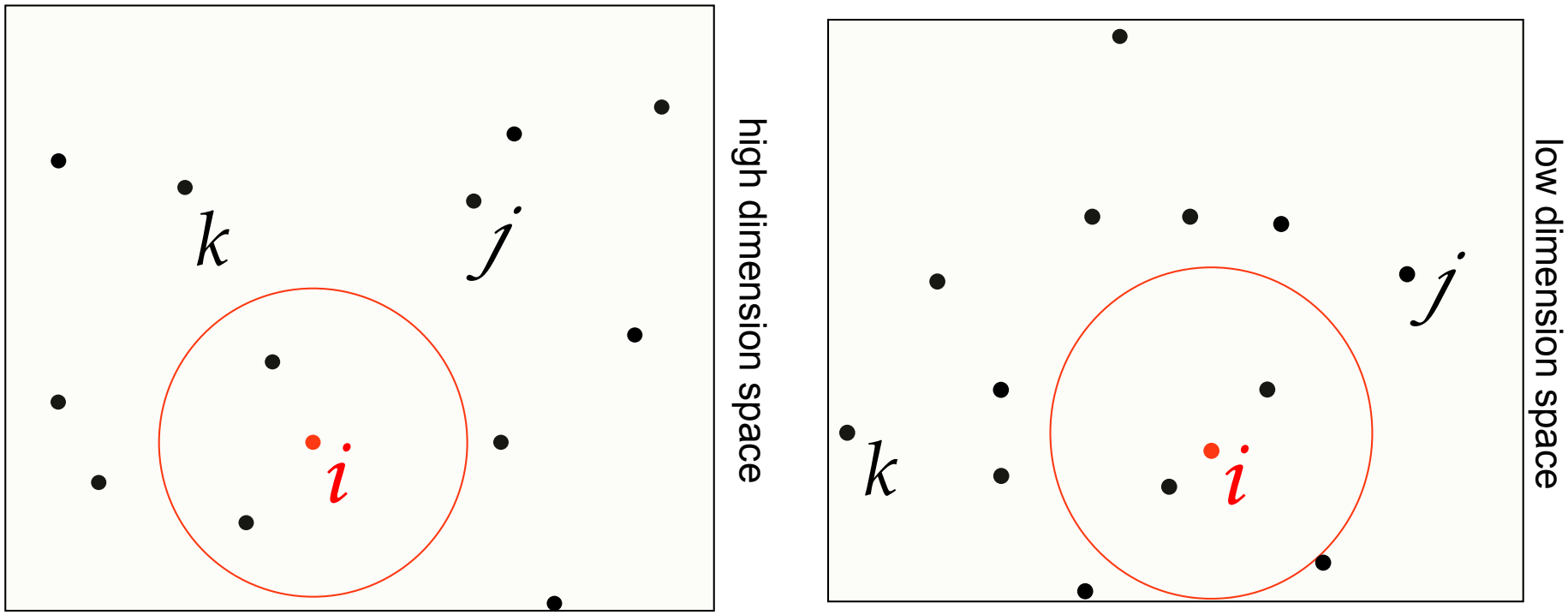


Stochastic Neighbor Embedding : key points

First convert each high-dimensional similarity into the probability that one data point will pick the other data point as its neighbor. To evaluate a map:

- Use the pairwise distances in the low-dimensional map to define the probability that a map point will pick another map point as its neighbor.
- Compute the Kullback-Leibler divergence between the probabilities in the high-dimensional and low-dimensional spaces.
- Each point in high-Dimension has a conditional probability of picking each other point as its neighbor.
- The distribution over neighbors is based on the high-Dimension pairwise distances.

Stochastic Neighbor Embedding



Evaluate this representation by seeing how well the low-Dimension probabilities model the high-Dimension ones.

Stochastic Neighbor Embedding

Stochastic Neighbor Embedding (SNE) is the process of constructing conditional probabilities representing the similarity between high dimensional data points using their Euclidean distances. The conditional probability $p_{j|i}$ for points x_j and x_i is defined by the equation

$$p_{j|i} = \frac{exp(\frac{-||x_i - x_j||^2}{2\sigma_i^2})}{\sum_{k \neq i} exp(\frac{-||x_i - x_j||^2}{2\sigma_i^2})}$$

Stochastic Neighbor Embedding

Similarity is ultimately the probability that x_i would define x_j as a neighbor, in which a neighborhood is defined by a Gaussian probability density centered at x_i. where σ_i is the variance of the x_i-centered distribution.

A large $p_{j|i}$ is indicative of close, or similar, data points, and a very small $p_{j|i}$ means that x_j is not likely a neighbor of x_i.

Instead of using a Gaussian distribution, t-SNE assumes the closely-related Student-t distribution to compute the pairwise conditional probabilities in a low-dimensional space more efficiently.

Stochastic Neighbor Embedding

The t-SNE algorithm improves upon the original SNE algorithm by implementing a cost function with a simpler gradient that uses the Kullback-Leibler divergence (DKL) between the high-dimensional joint probability distribution P and a low-dimensional Student-t based joint probability distribution Q (Equation 2) . The gradient is explicitly defined in Equation 3.

equation 2

$$q_{ij} = \frac{(1 + ||x_i - x_j||^2)^{-1}}{\sum_{k \neq l} (1 + ||y_k - y_l||^2)^{-1}}$$

equation 3

$$\frac{\delta C}{\delta \mathscr{Y}} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)(1 + ||y_i - y_j||^2)^{-1}$$

Stochastic Neighbor Embedding

With higher-dimensional data, one runs the risk of overcrowding the projection such that dissimilarities between points cannot be faithfully plotted due to a lack of space in the two-dimensional map to reduce the high-dimensional data.

The use of the heavy-tailed Student-t distribution mitigates this issue because it converts the moderate distances that, when mapped to a two-dimensional plane tend to be too close to x_i, to probabilities that map the points an appropriately greater distance away.

Stochastic Neighbor Embedding

Algorithm 1: Standard t-distributed Stochastic Neighbor Embedding Algorithm.

Data: : data set $X = x_1, x_2, ..., x_n$,

cost function parameters: perplexity *Perp*;

optimization parameters: number of iterations T, learning rate η, momentum $\alpha(t)$;

Result: low-dimensional data representation $\mathcal{Y}^{(T)} = y_1, y_2, y_n$.

begin

compute pairwise affinities $p_{j|i}$ with perplexity *Perp* (Equation 1)

set $p_i j = \frac{p_{j|i} + p_{i|j}}{2n}$;

sample initial solution $\mathcal{Y}^{(0)} = y_1, y_2, y_n$ from $\mathcal{N}(0, 10\_4I)$;

for $t = 1$ **to** T **do**

compute low-dimensional affinities q_{ij} (Equation 2)

compute gradient $\frac{\delta C}{\delta \mathcal{Y}}$ (Equation 3)

set $\mathcal{Y}^{(t)} = \mathcal{Y}^{(t-1)} + \eta \frac{\delta C}{\delta \mathcal{Y}} + \alpha(t)(\mathcal{Y}^{t-1)} - \mathcal{Y}^{t-2)})$;

end

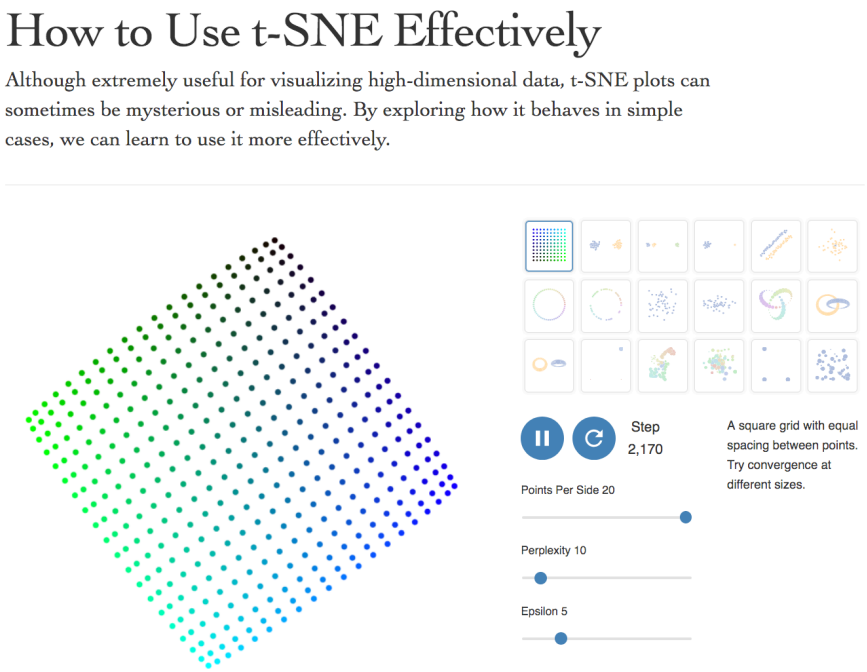
end

References on t-SNE

- t-SNE main paper: , L.J.P. van der Maaten and G.E. Hinton. Visualizing High-Dimensional Data Using t-SNE. Journal of Machine Learning Research 9(Nov):2579-2605, 2008
- [useful video: https://lvdmaaten.github.io/tsne/)https://youtu.be/RJVL80Gg3lA?list=UUtXKDgv1AVoG88PLl8nGXmw](https://youtu.be/RJVL80Gg3lA?list=UUtXKDgv1AVoG88PLl8nGXmw))

.

- how to use: https://distill.pub/2016/misread-tsne/



Burrows – Wheeler Transform

Burrows (left), Wheeler (right)
both at the Computer Laboratory

Bowtie
An ultrafast memory-efficient short read aligner

Burrows-Wheeler Aligner

Home

Introduction

BWA is a software package for mapping low-divergent sequences against a large reference genome, such as the human genome. It consists of three algorithms: BWA-backtrack, BWA-SW and BWA-MEM. The first algorithm is designed for Illumina sequence reads up to 100bp, while the rest two for longer sequences ranged from 70bp to 1Mbp. BWA-MEM and BWA-SW share similar features such as long-read support and split alignment, but BWA-MEM, which is the latest, is generally recommended for high-quality queries as it is faster and more accurate. BWA-MEM also has better performance than BWA-backtrack for 70-100bp Illumina reads.

BWA:

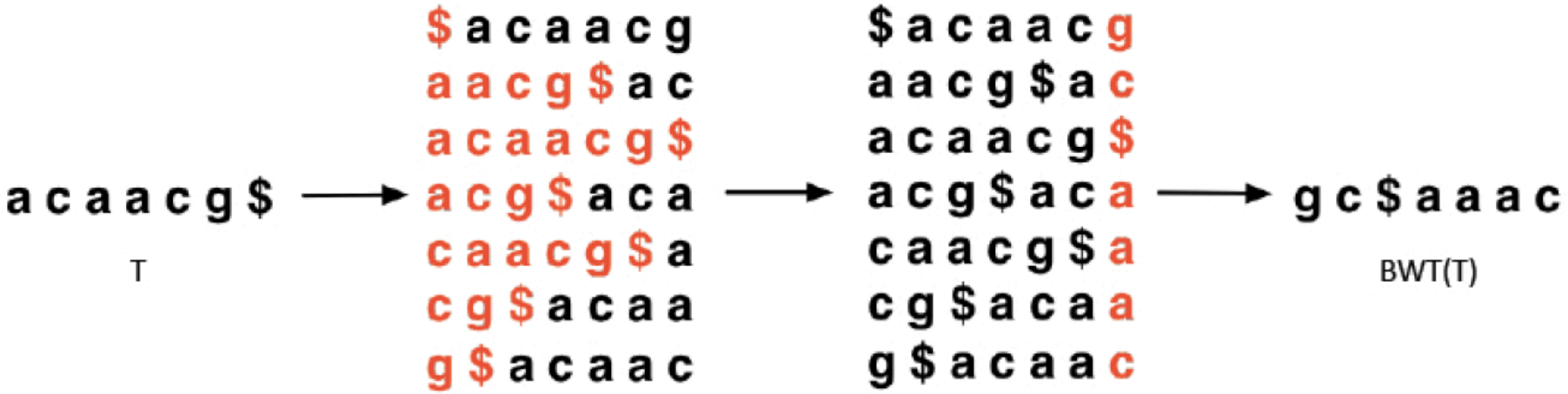
SF project page
SF download page
Mailing list
BWA maual page
Repository

Links:

SAMtools

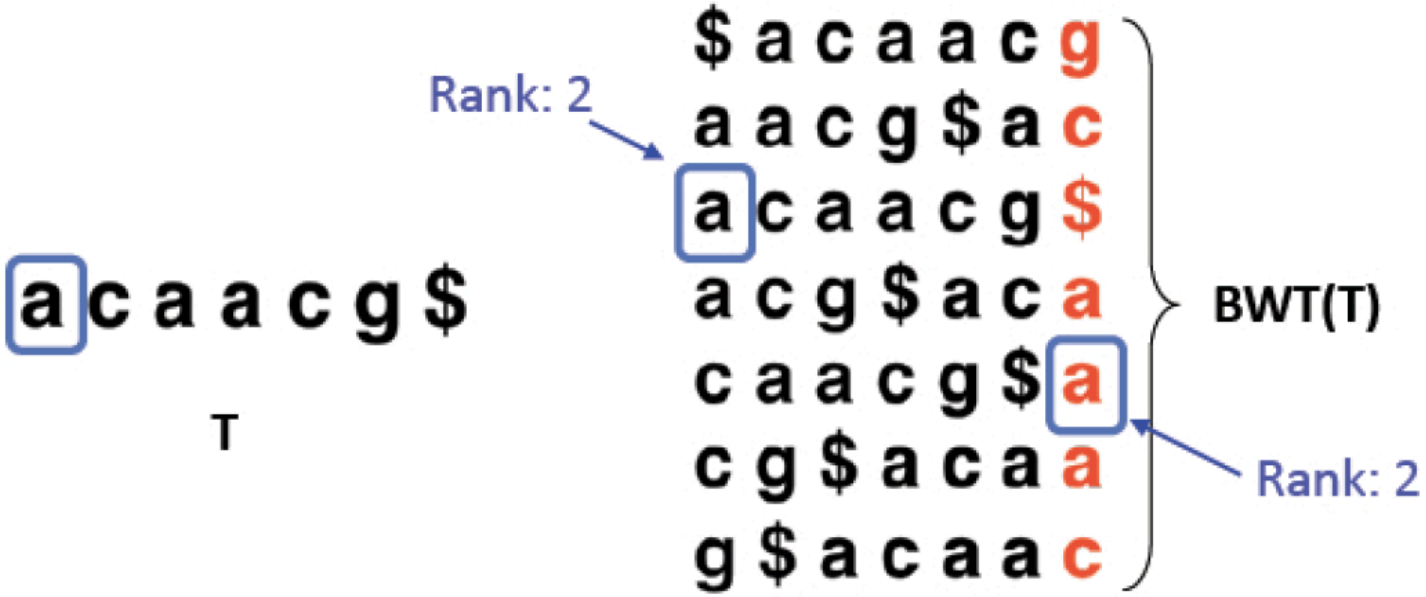
Burrows Wheeler Transform

Three steps: 1) Form a N*N matrix by cyclically rotating (left) the given text to form the rows of the matrix. Here we use '$' as a sentinel (lexicographically greatest character in the alphabet and occurs exactly once in the text but it is not a must). 2) Sort the matrix according to the alphabetic order. Note that the cycle and the sort procedures of the Burrows-Wheeler induces a partial clustering of similar characters providing the means for compression. 3) The last column of the matrix is BWT(T) (we need also the row number where the original string ends up).



BWT

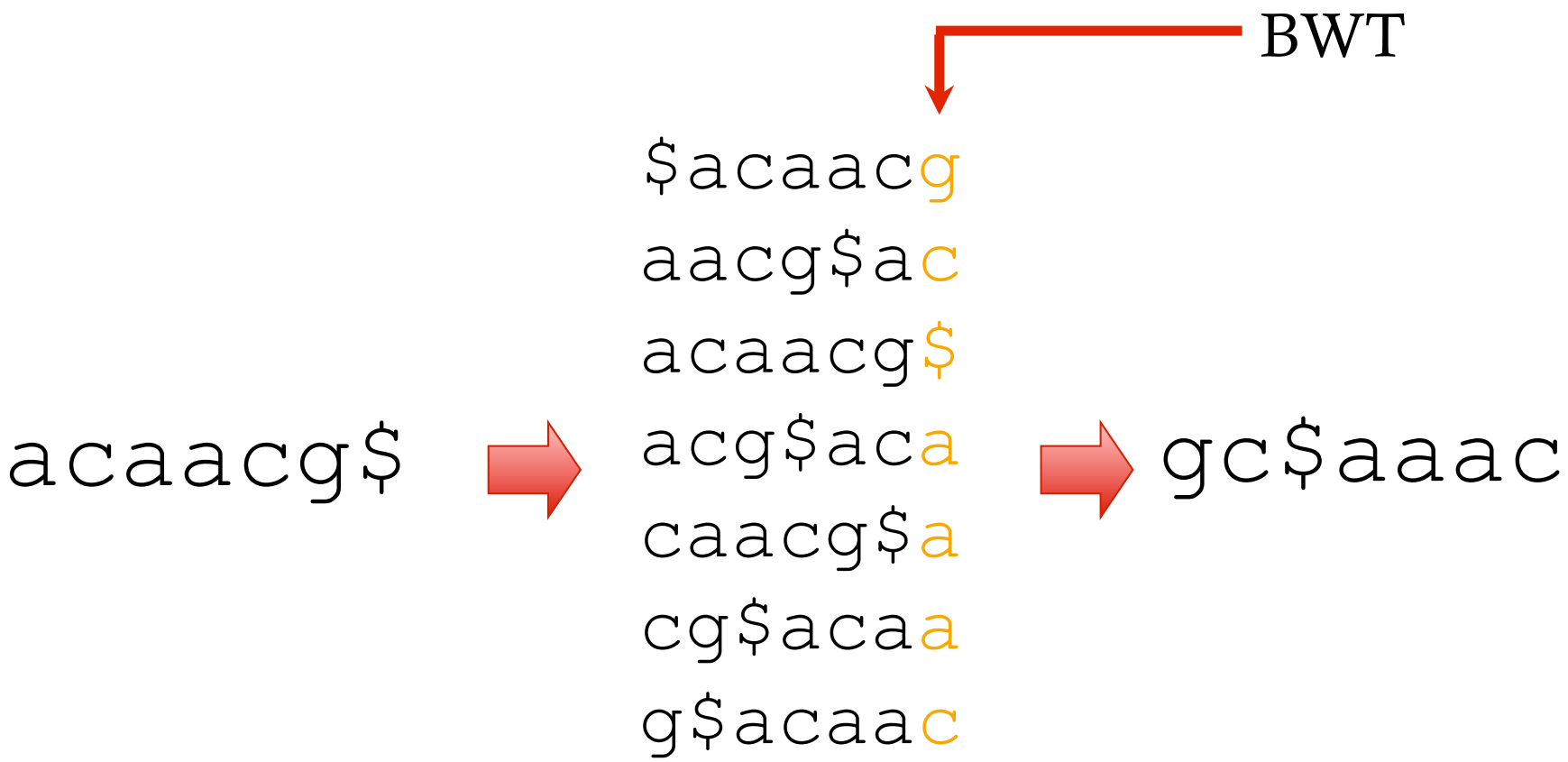
Property that makes BWT(T) reversible is LF Mapping: the ith occurrence of a character in Last column is same text occurrence as the ith occurrence in the First column (i.e. the sorting strategy preserves the relative order in both last column and first column).



BWT

To recreate T from BWT(T), repeatedly apply the rule: T = BWT[LF(i)] + T; i = LF(i) where LF(i) maps row i to row whose first character corresponds to i"s last per LF Mapping. First step: $S = 2$; $T = \$$. Second step: s = LF[2] =6; $T = g\$$. Third step: s = LF[6] = 5; $T = cg\$$.

Burrows-Wheeler Transform (BWT)



Burrows-Wheeler Matrix (BWM)

Burrows-Wheeler Matrix

$acaacg
aacg$ac
acaacg$
acg$aca
caacg$a
cg$acaa
g$acaac

Burrows-Wheeler Matrix

	$acaacg
3	aacg$ac
1	acaacg$
4	acg$aca
2	caacg$a
5	cg$acaa
6	g$acaac

See the suffix array?

Key observation

$a^1c^1a^2a^3c^2g^1\$^1$

“last first (LF) mapping”

The *i*-th occurrence of character X in the last column corresponds to the same text character as the *i*-th occurrence of X in the first column.

$^1\$acaacg^1$

$^2aacg\$ac^1$

$^1acaacg\$^1$

$^3acg\$aca^2$

$^1caacg\$a^1$

$^2cg\$acaa^3$

$^1g\$acaac^2$

Burrow Wheeler Transform

Genome Assembly

- Why do we map reads?
- Using the Trie
- From a Trie to a Suffix Tree
- String Compression and the Burrows-Wheeler Transform
- Inverting Burrows-Wheeler
- Using Burrows-Wheeler for Pattern Matching
- Finding the Matched Patterns
- Setting Up Checkpoints
- Inexact Matching

Toward a Computational Problem

- **Reference genome**: database genome used for comparison.

- **Question:** How can we assemble individual genomes efficiently using the reference?

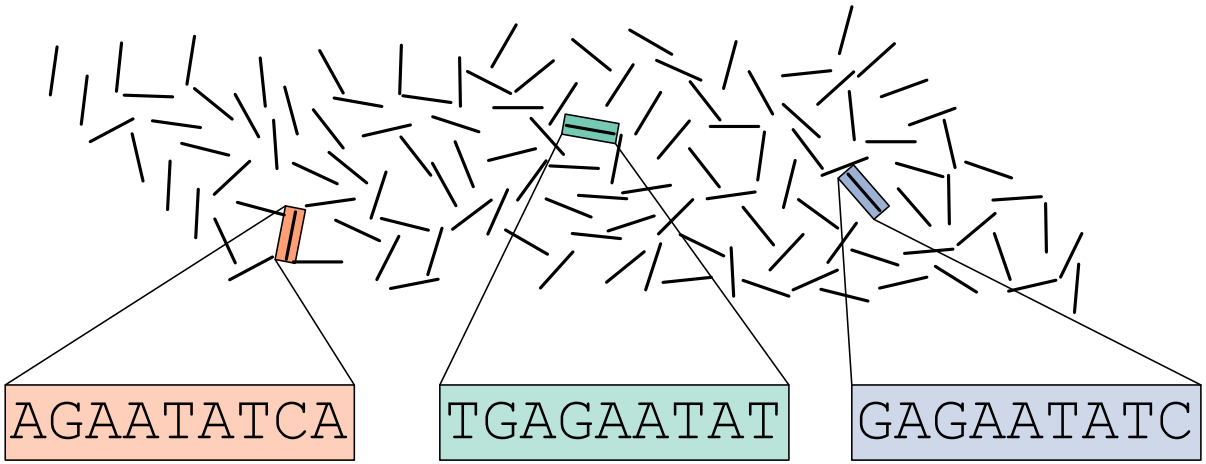
CTGATGATGGACTACGCTACTACTGCTAGCTGTAT Individual

CTGAGGATGGACTACGCTACTACTGATAGCTGTTT Reference

Why Not Use Assembly?

Multiple copies of a genome

Shatter the genome into reads



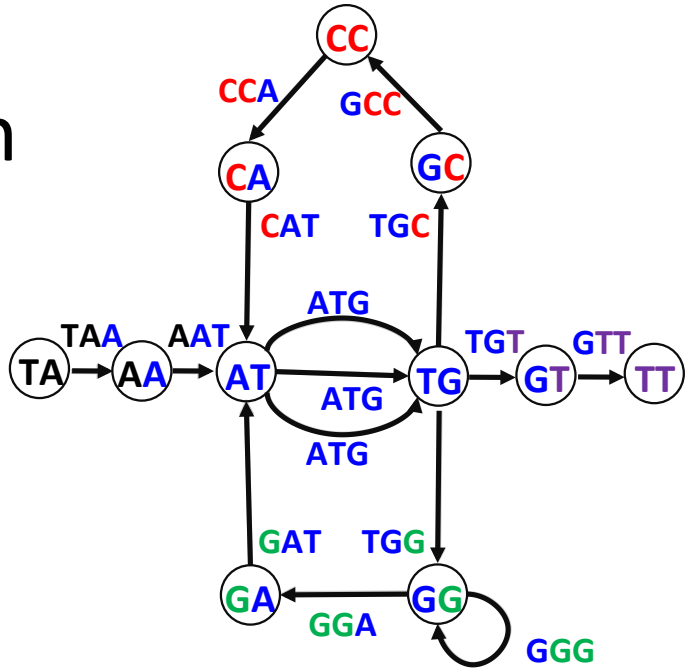
Sequence the reads

Assemble the genome with overlapping reads

AGAATATCA
GAGAATATC
TGAGAATAT
...TGAGAATATCA...

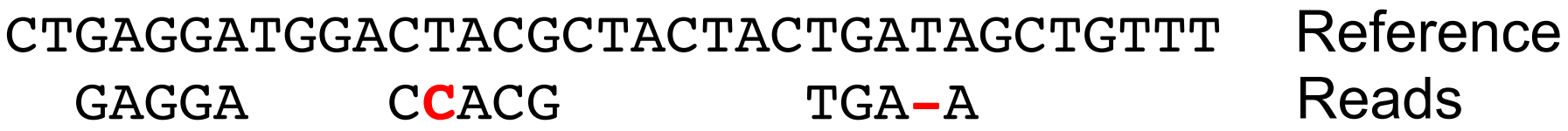
Why Not Use Assembly?

- Constructing a de Bruijn graph takes a lot of memory.
- Hope: a machine in a clinic that would collect and map reads in 10 minutes.
- Idea: use existing structure of reference genome to help us sequence a patient's genome.



Read Mapping

- **Read mapping**: determine where each read has high similarity to the reference genome.



Why Not Use Alignment?

- **Fitting alignment:** align each read *Pattern* to the best substring of *Genome*.

- Has runtime O(|*Pattern*| * |*Genome*|) for each *Pattern*.

- Has runtime O(|*Patterns*| * |*Genome*|) for a collection of *Patterns*.

Exact Pattern Matching

- Focus on a simple question: where do the reads match the reference genome *exactly*?

- **Single Pattern Matching Problem:**
  - **Input:** A string *Pattern* and a string *Genome.*
  - **Output:** All positions in *Genome* where *Pattern* appears as a substring.

Exact Pattern Matching

- Focus on a simple question: where do the reads match the reference genome *exactly*?

- **Multiple Pattern Matching Problem:**
  - **Input:** A collection of strings *Patterns* and a string *Genome.*
  - **Output:** All positions in *Genome* where a string from *Patterns* appears as a substring.

A Brute Force Approach

- We can simply iterate a brute force approach method, sliding each *Pattern* down *Genome.*

```
panamabananas    Genome
   nana          Pattern
```

- **Note**: we use words instead of DNA strings for convenience.

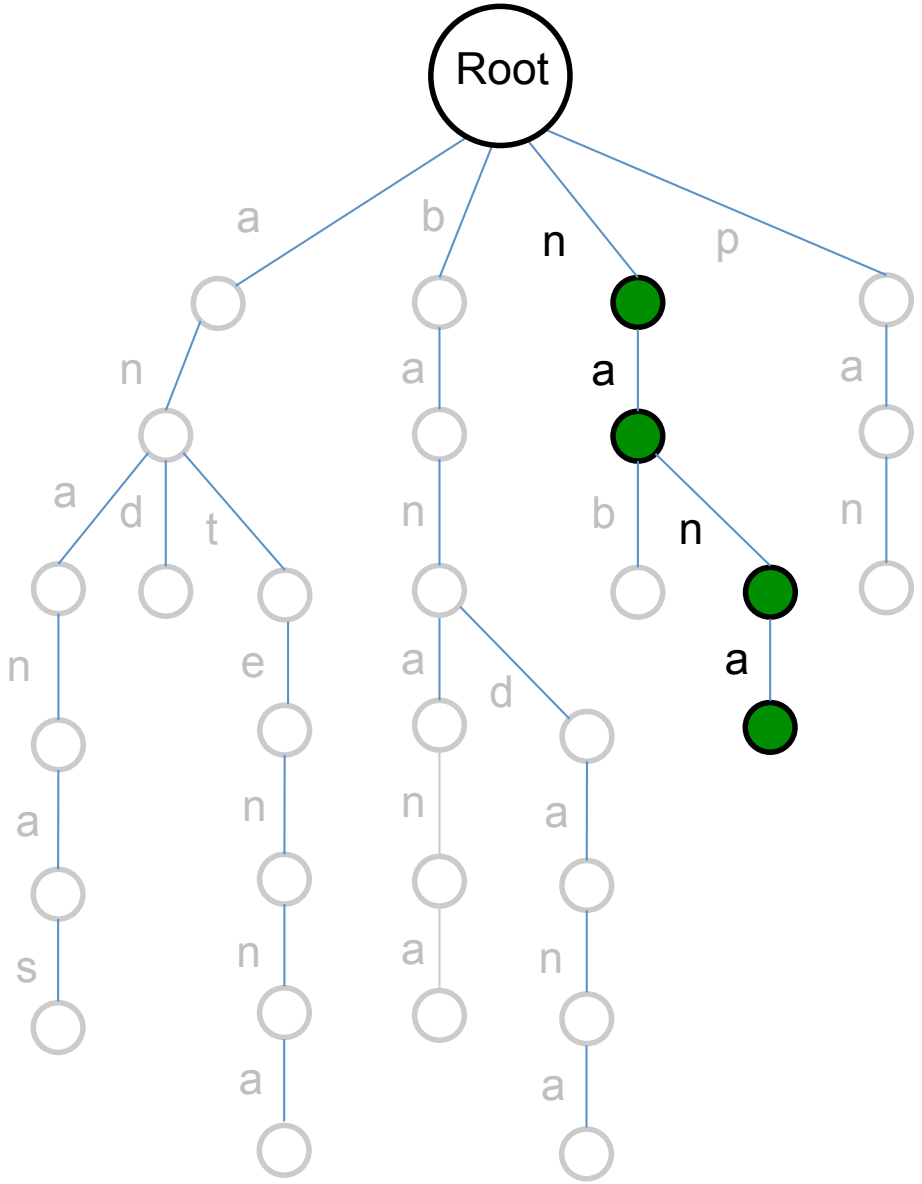
Brute Force Is Too Slow

- The runtime of the brute force approach is too high!
  - Single *Pattern*: O(|*Genome*| * |*Pattern*|)
  - Multiple *Patterns*: O(|*Genome*| * |*Patterns*|)
  - |*Patterns*| = combined length of *Patterns*

Processing Patterns into a Trie

- Idea: combine reads into a graph. Each substring of the genome can match at most one read. So each read will correspond to a unique path through this graph.

- The resulting graph is called a **trie**.



Patterns

banana
pan
and
nab
antenna
bandana
ananas
nana

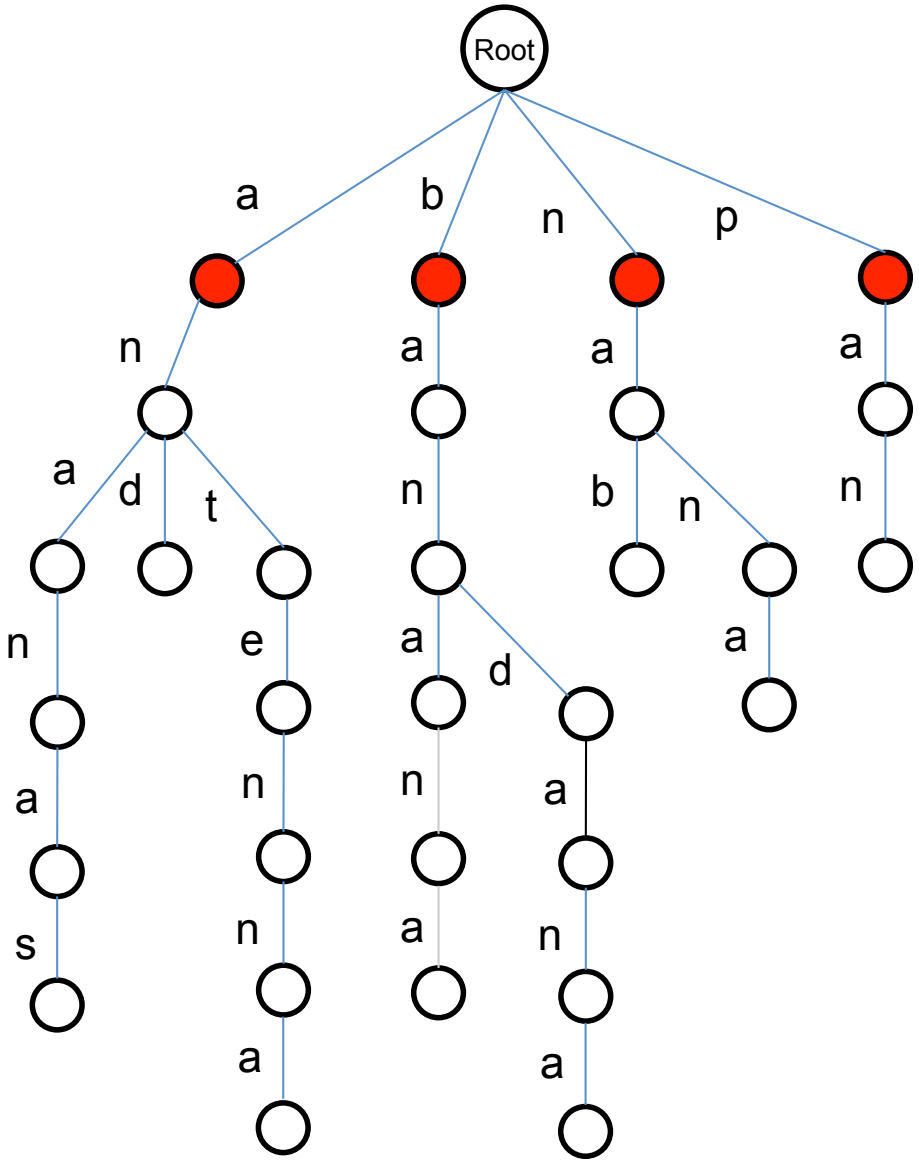
Using the Trie for Pattern Matching

- **TrieMatching**: Slide the trie down the genome.

- At each position, walk down the trie and see if we can reach a leaf by matching symbols.

- Analogy: bus stops

p a n a m a b a n a n a **s**

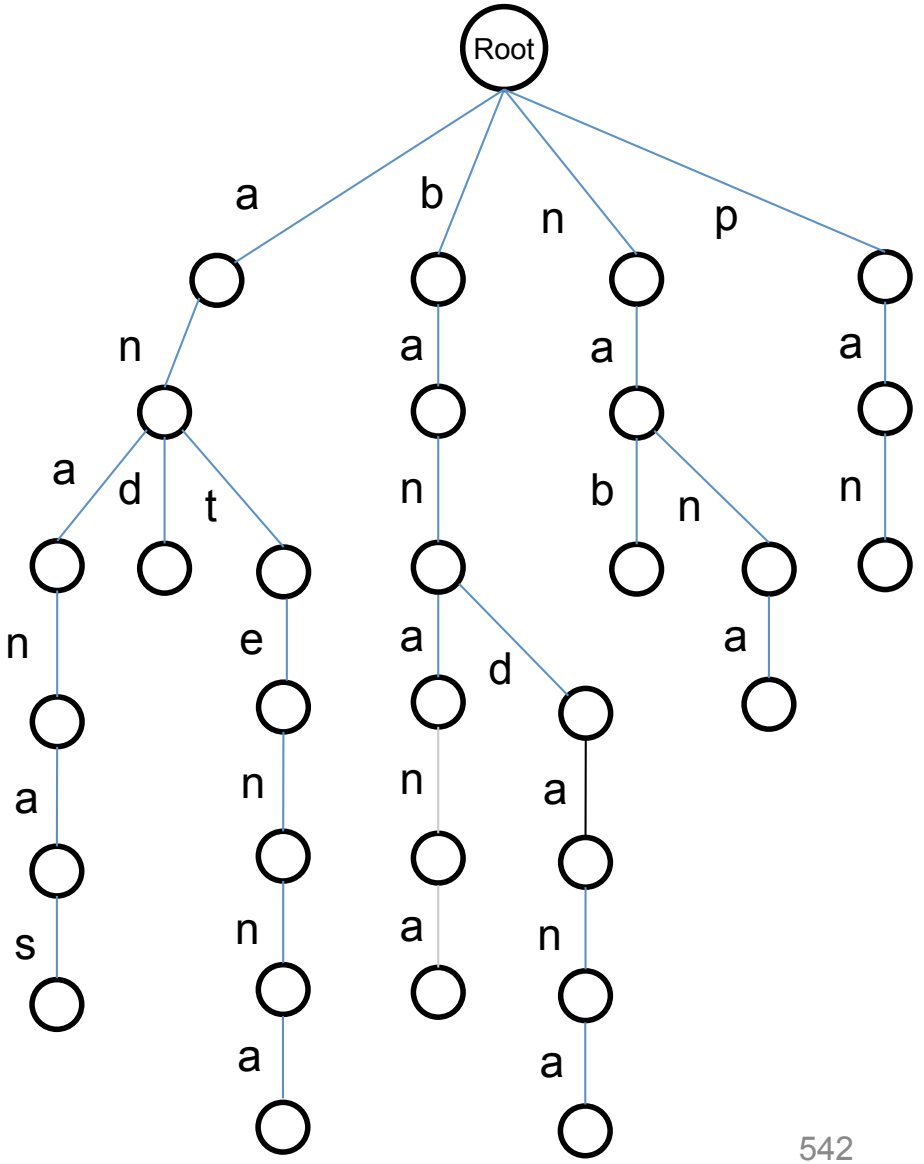


Success!

- Runtime of Brute Force:
  - Total: O(|*Genome*|*|*Patterns*|)
- Runtime of Trie Matching:
  - Trie Construction: O(|*Patterns*|)
  - Pattern Matching: O(|*Genome*| * |*LongestPattern*|)

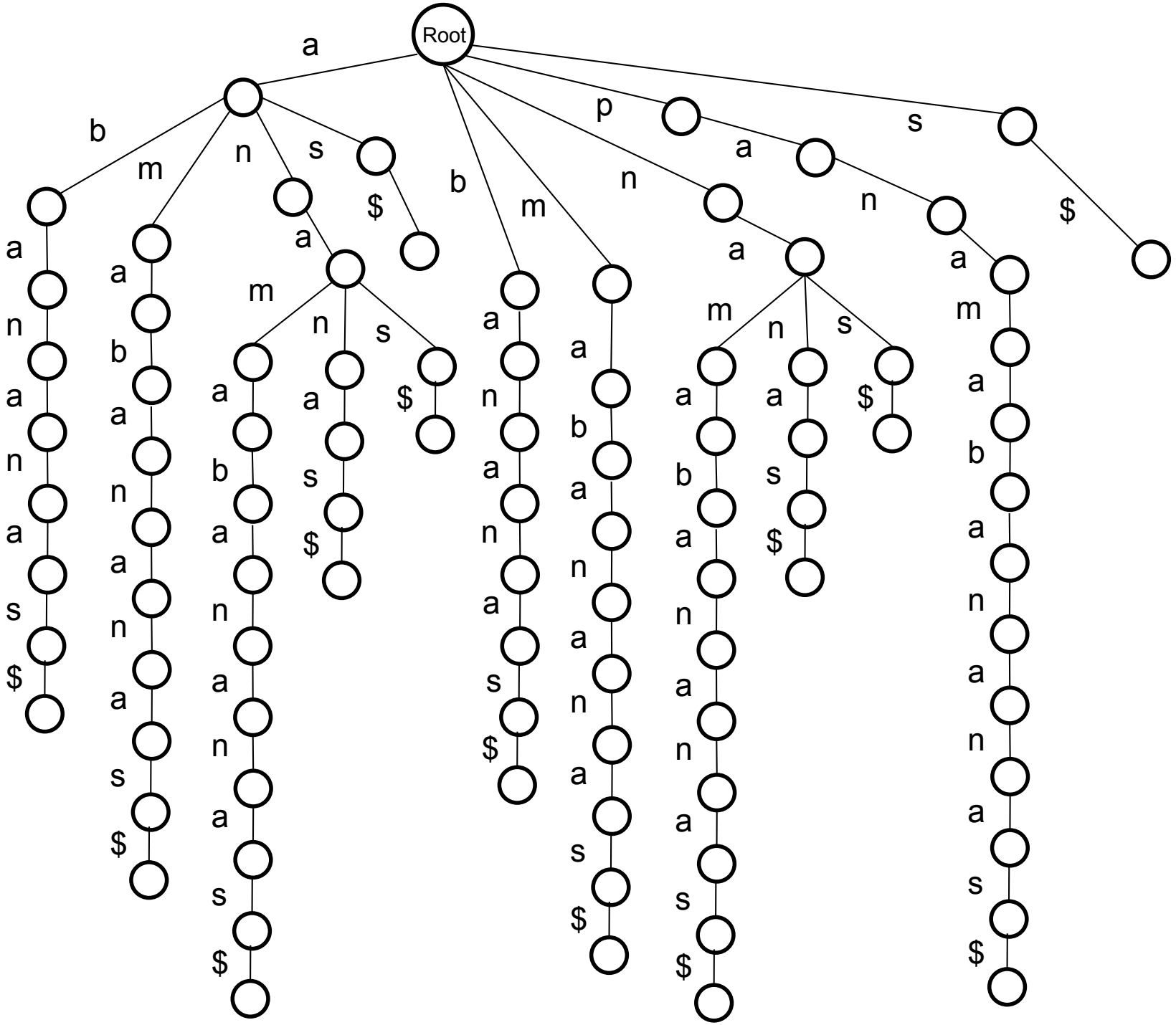
Memory Analysis of TrieMatching

- Son completely forgot about memory!
- Our trie: 30 edges, $|Patterns| = 39$
- Worst case: # edges = **$O(|Patterns|)$**



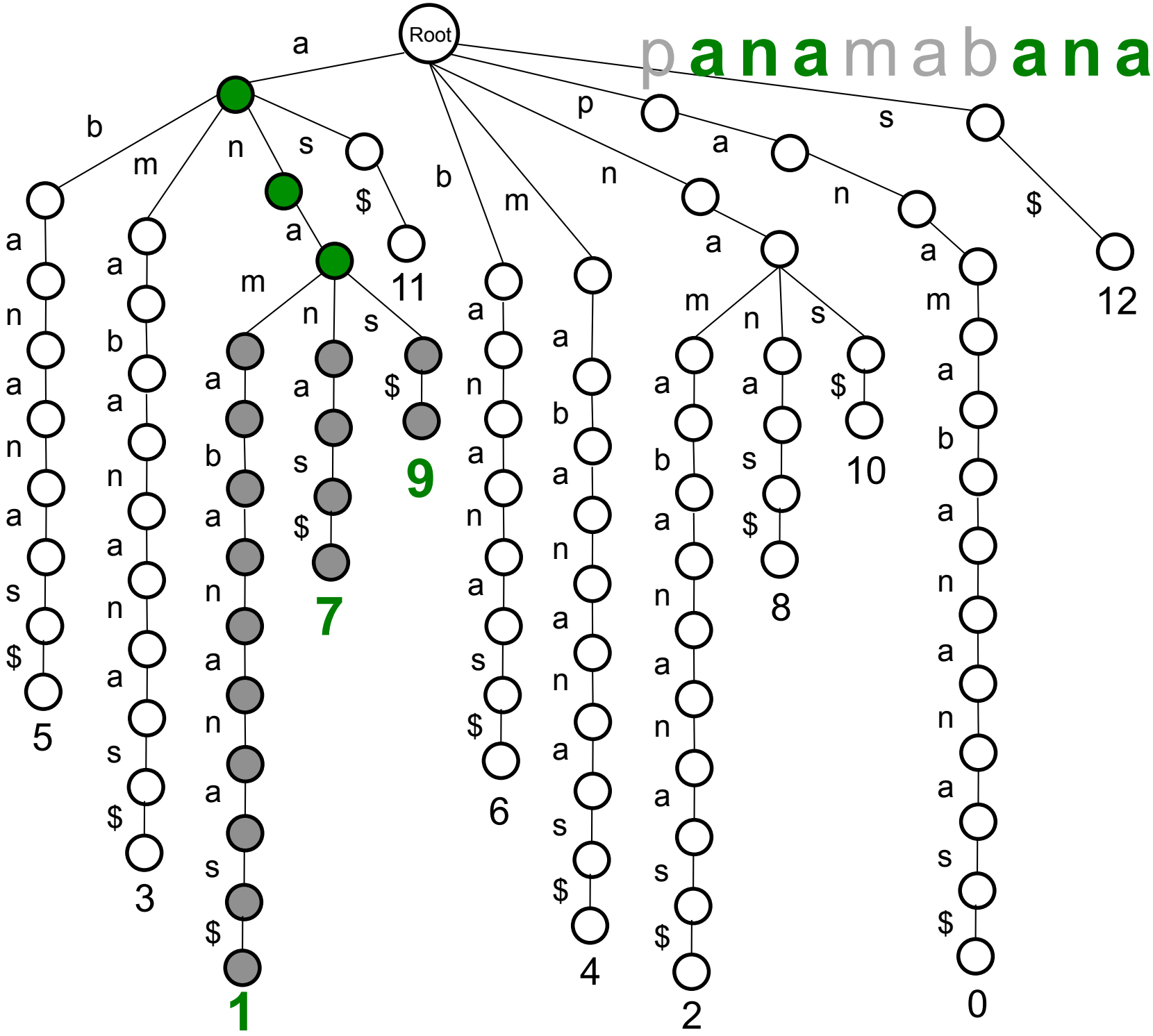
Preprocessing the Genome

- What if instead we create a data structure from the genome itself?
- Split *Genome* into all its suffixes. (Show matching “banana” by finding the suffix “bananas”.)
- How can we combine these suffixes into a data structure?
- Let’s use a trie!


Root
a
b
m
n
s
$
p
a
n
a
$
s
b
m
a
n
a
b
a
n
a
s
$
a
b
a
n
a
n
a
s
$
a
m
n
s
a
b
a
n
a
n
a
s
$
a
s
$
$
a
n
a
n
a
s
$
a
b
a
n
a
n
a
s
$
a
m
n
s
a
b
a
n
a
n
a
s
$
$
a
s
$
a
m
a
b
a
n
a
n
a
s
$

The Suffix Trie and Pattern Matching

- For each *Pattern*, see if *Pattern* can be spelled out from the root downward in the suffix trie.


Root
panamabanana$
a
b
m
n
s
$
p
a
n
a
m
a
b
a
n
a
n
a
s
$
5
3
1
7
9
11
6
4
2
8
10
0
12

Memory Trouble Once Again

- Worst case: the suffix trie holds O(|*Suffixes*|) nodes.

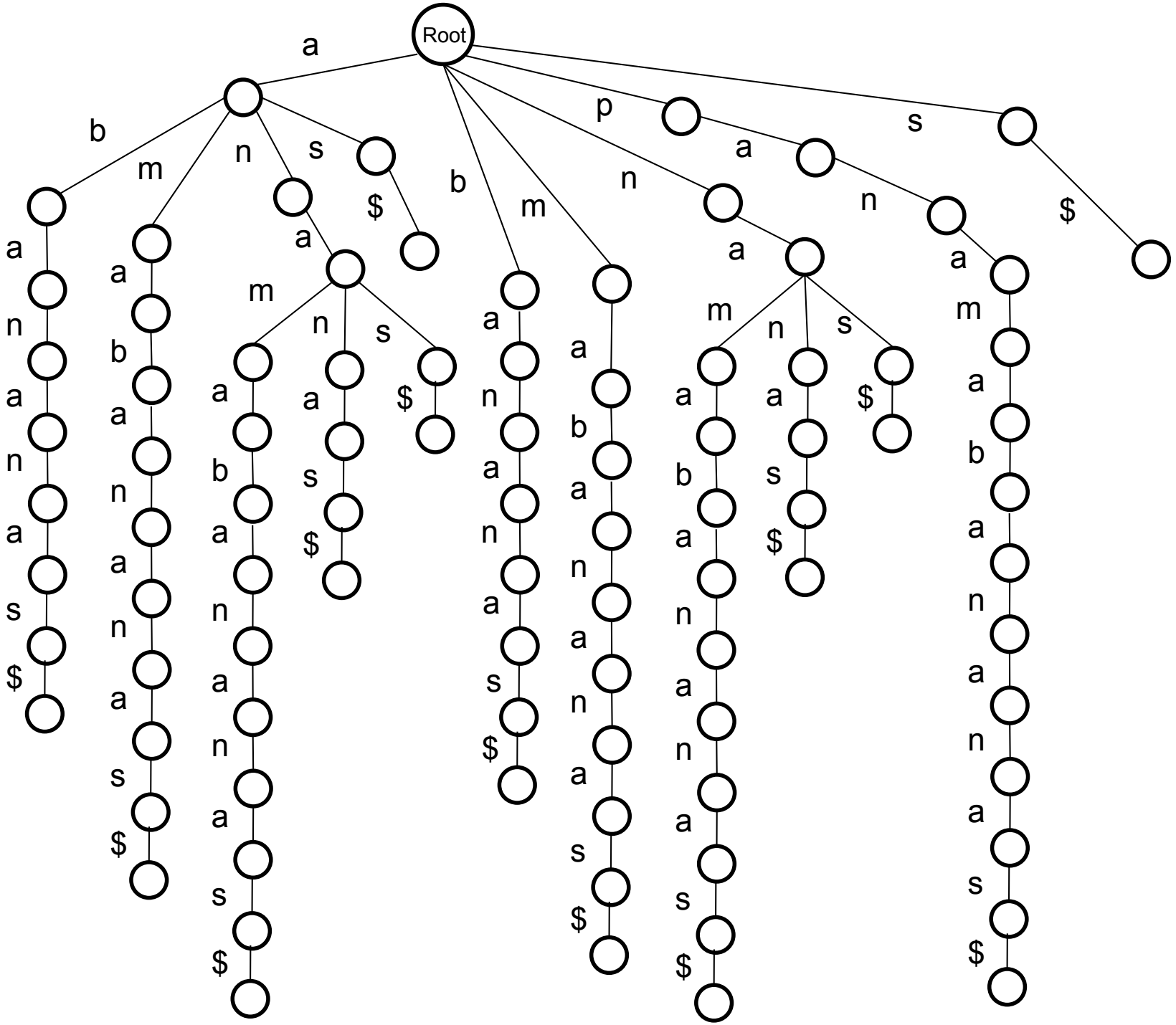
- For a *Genome* of length *n*, |*Suffixes*| = $n(n-1)/2 = O(n^2)$

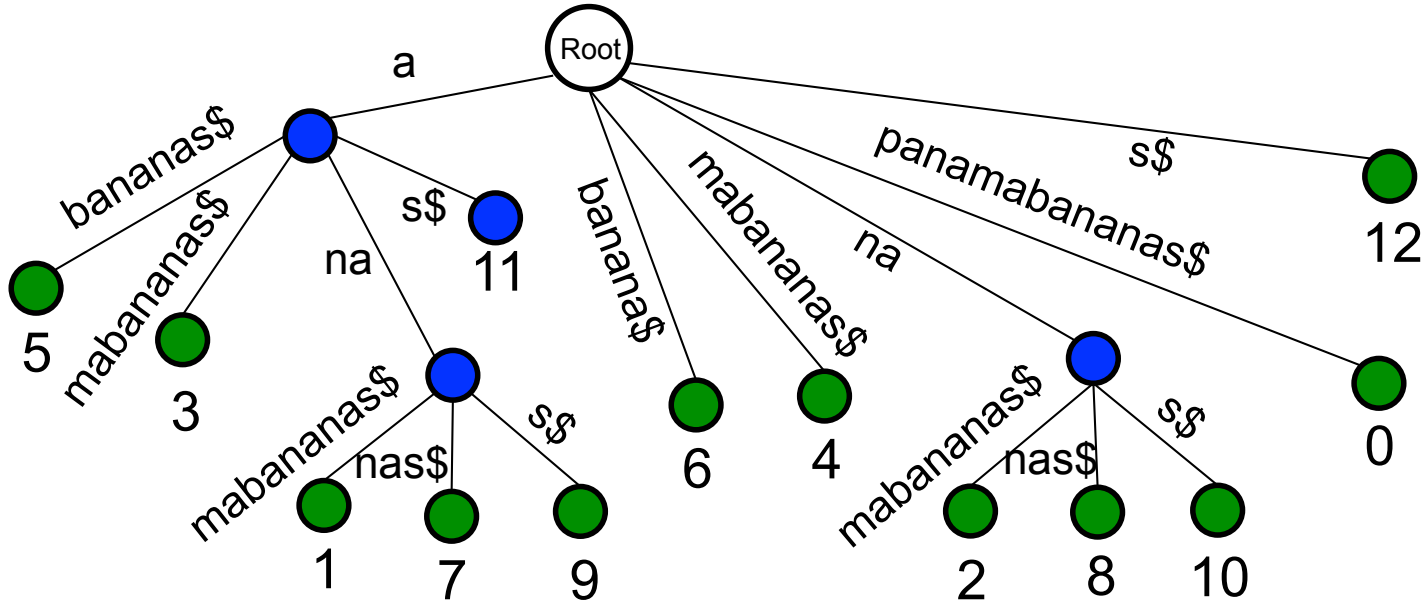
Suffixes

```
panamabananas$
anamabananas$
namabananas$
amabananas$
mabananas$
abananas$
bananas$
ananas$
nanas$
anas$
nas$
as$
s$
$
```

Compressing the Trie

- This doesn’t mean that our idea was bad!
- To reduce memory, we can compress each “nonbranching path” of the tree into an edge.


Root
a
b
m
n
s
$
p
a
n
a
$
s
b
m
a
n
a
n
a
s
$
a
b
a
n
a
n
a
s
$
a
m
n
s
a
b
a
n
a
n
a
s
$
$
a
n
a
n
a
s
$
a
b
a
n
a
n
a
s
$
a
m
n
s
a
b
a
n
a
n
a
s
$
$
a
m
a
b
a
n
a
n
a
s
$



- This data structure is called a **suffix tree**.

- For any *Genome*, # nodes < 2|*Genome*|.
  - **# leaves** = |*Genome*|;
  - **# internal nodes** < |*Genome*| – 1

Runtime and Memory Analysis

- Runtime:
  - $O(|Genome|^2)$ to construct the suffix tree.
  - $O(|Genome| + |Patterns|)$ to find pattern matches.

- Memory:
  - $O(|Genome|^2)$ to construct the suffix tree.
  - $O(|Genome|)$ to store the suffix tree.

Runtime and Memory Analysis

- Runtime:
  - O(|*Genome*|) to construct the suffix tree ***directly.***
  - O(|*Genome*| + |*Patterns*|) to find pattern matches.
  - **Total: O(|*Genome*| + |*Patterns*|)**

- Memory:
  - O(|*Genome*|) to construct the suffix tree ***directly.***
  - O(|*Genome*|) to store the suffix tree.
  - **Total: O(|*Genome*| + |*Patterns*|)**

We are Not Finished Yet

- I am happy with the suffix tree, but I am not completely satisfied.
  - Runtime: O(|*Genome*| + |*Patterns*|)
  - Memory: O(|*Genome*|)

- However, big-O notation ignores constants!
  - The best known suffix tree implementations require ~ 20 times the length of |*Genome*|.
  - Can we reduce this constant factor?

Genome Compression

- Idea: decrease the amount of memory required to hold *Genome*.

- This indicates that we need methods of **compressing** a large genome, which is seemingly a separate problem.

Idea #1: Run-Length Encoding

- **Run-length encoding:** compresses a run of *n* identical symbols.

Genome

GGGGGGGGGGCCCCCCCCCCCAAAAAAATTTTTTTTTTTTTTTTCCCCCG

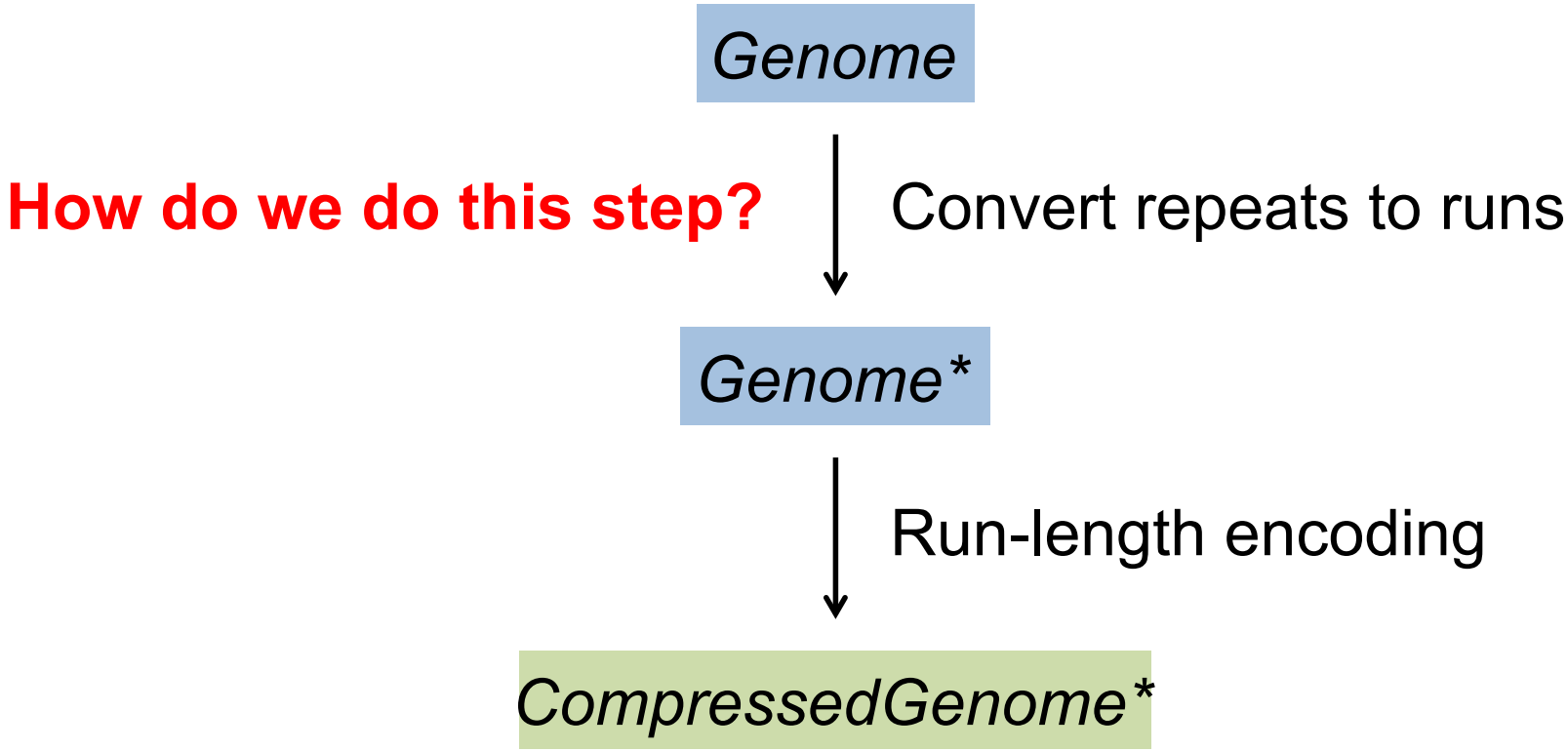
10G11C7A15T5C1G

Run-length encoding

- Problem: Genomes don't have lots of runs…

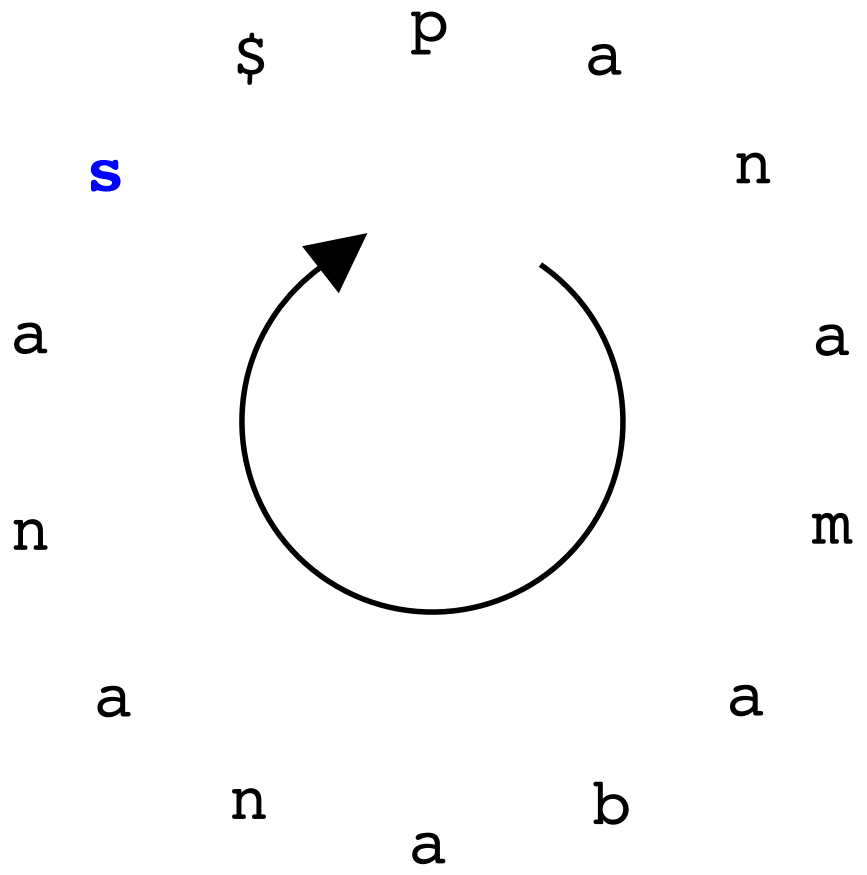
Converting Repeats to Runs

- …but they do have lots of repeats!



The Burrows-Wheeler Transform

```
panamabananas$
$panamabananas
s$panamabanana
```



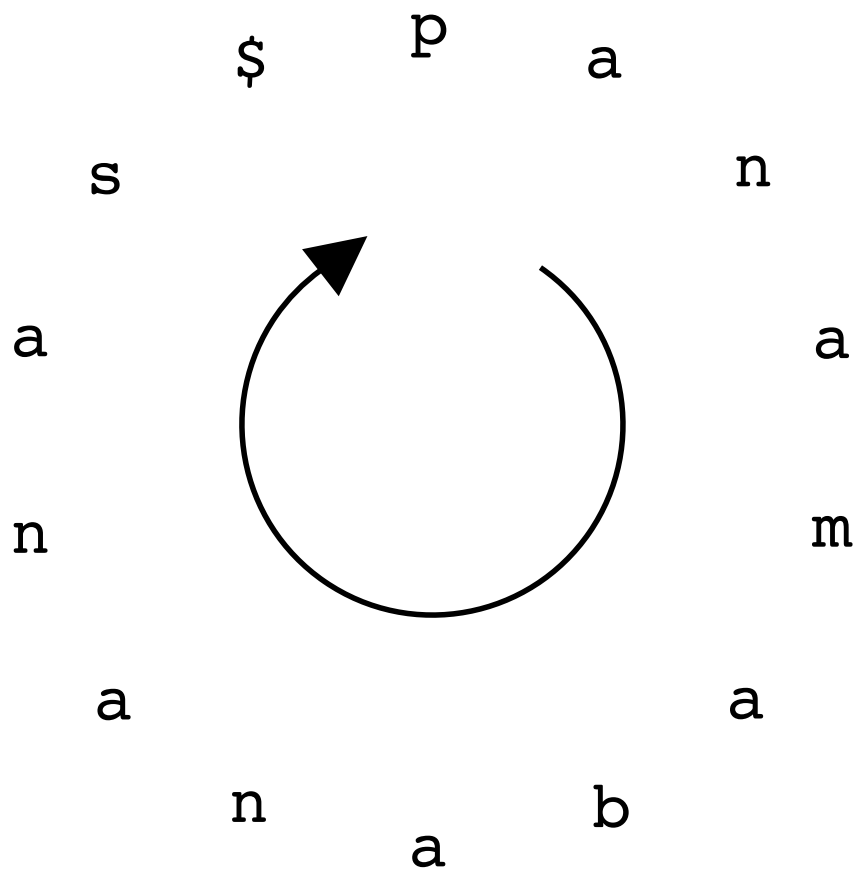
Form all cyclic rotations of "panamabananas$"

Burrows, Michael and Wheeler, David J. (1994), A block sorting lossless data compression algorithm, Technical Report 124, Digital Equipment Corporation

Li, H and Durbin, R (2009) Fast and accurate short read alignment with Burrows-Wheeler transform. Bioinformatics 25:1754-60.

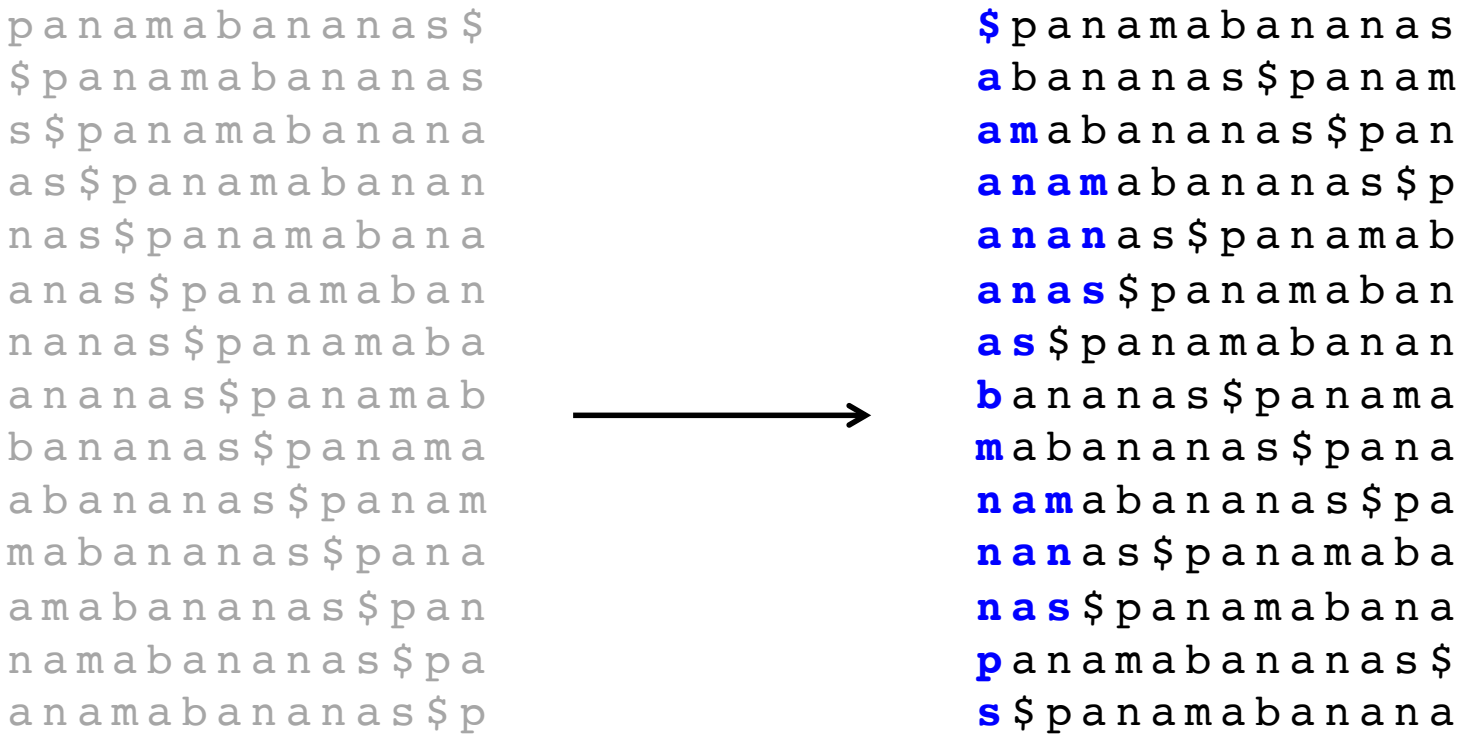
The Burrows-Wheeler Transform

```
panamabananas$
$panamabananas
s$panamabanana
as$panamabanan
nas$panamabana
anas$panamaban
nanas$panamaba
ananas$panamab
bananas$panama
abananas$panam
mabananas$pana
amabananas$pan
namabananas$pa
anamabananas$p
```



Form all cyclic rotations of
"panamabananas$"

The Burrows-Wheeler Transform



Form all cyclic rotations of "panamabananas$"

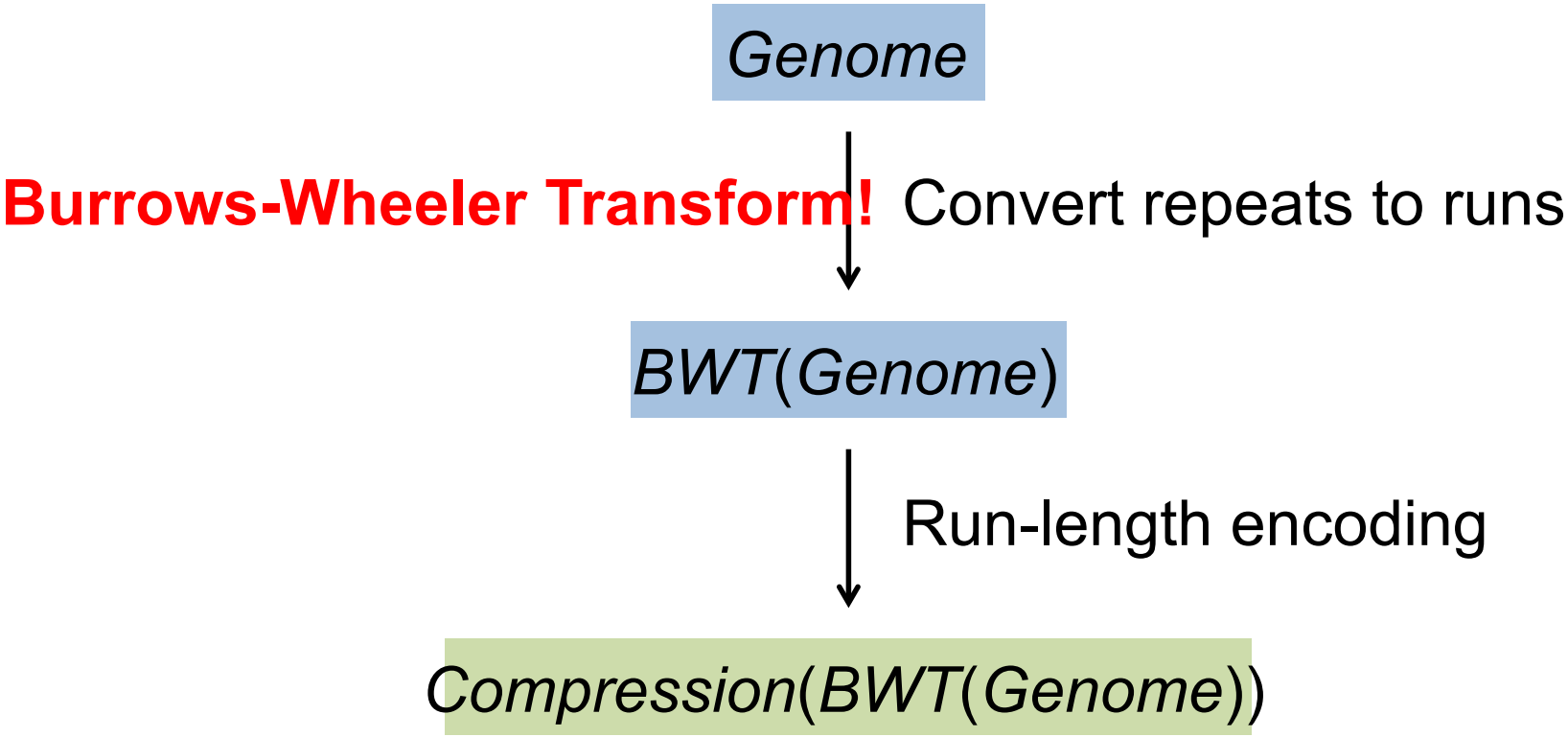
Sort the strings lexicographically ($ comes first)

The Burrows-Wheeler Transform

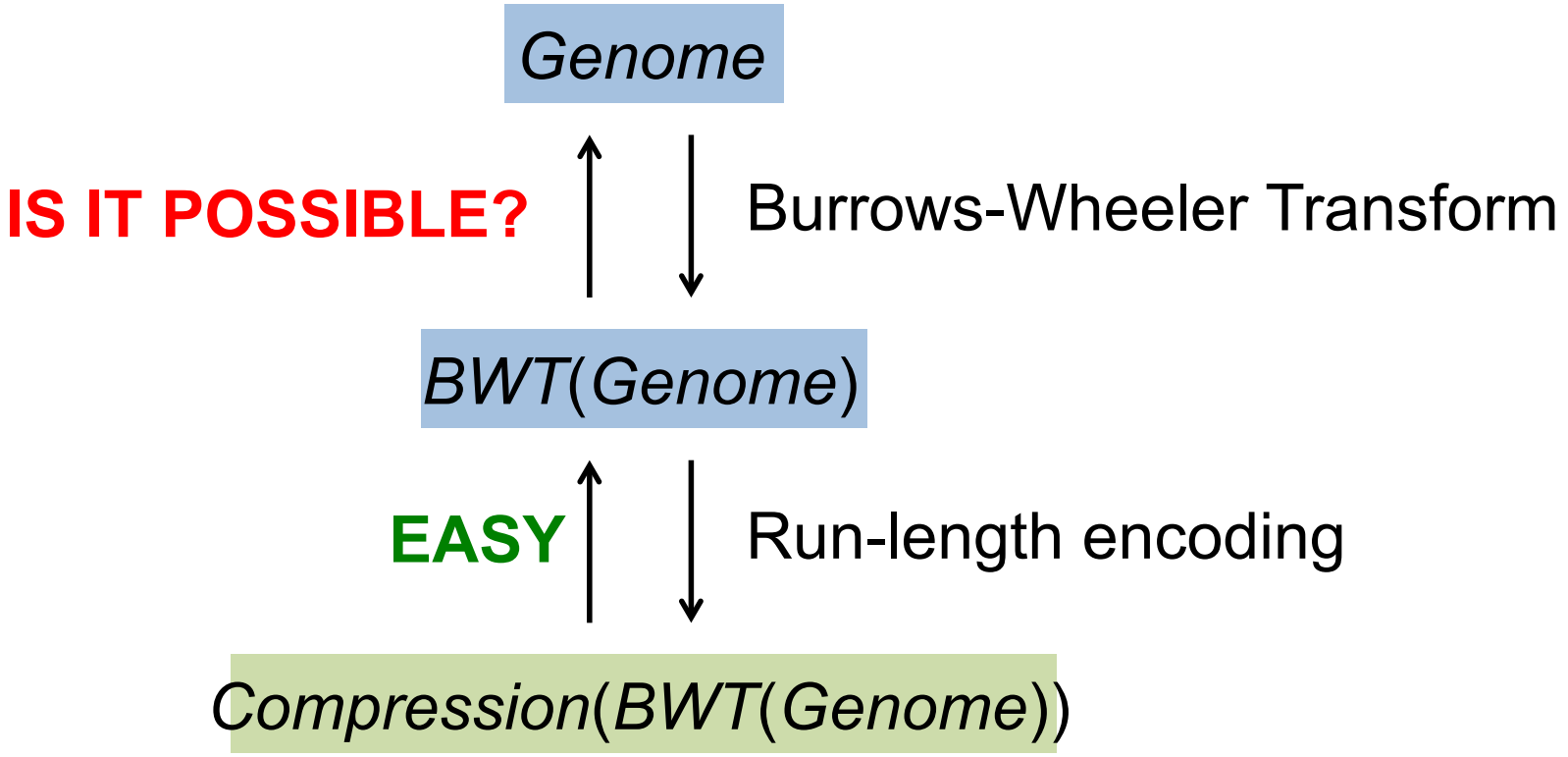
Form all cyclic rotations of "panamabananas$"

Burrows-Wheeler Transform: Last column = smnpbnnaaaaa$a

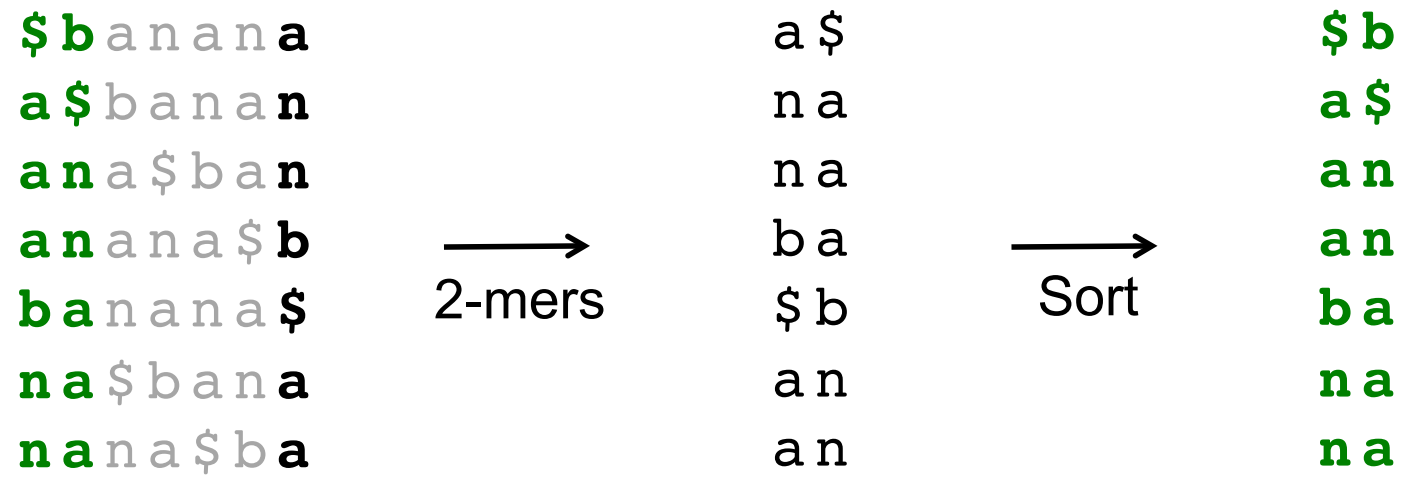
BWT: Converting Repeats to Runs



How Can We Decompress?

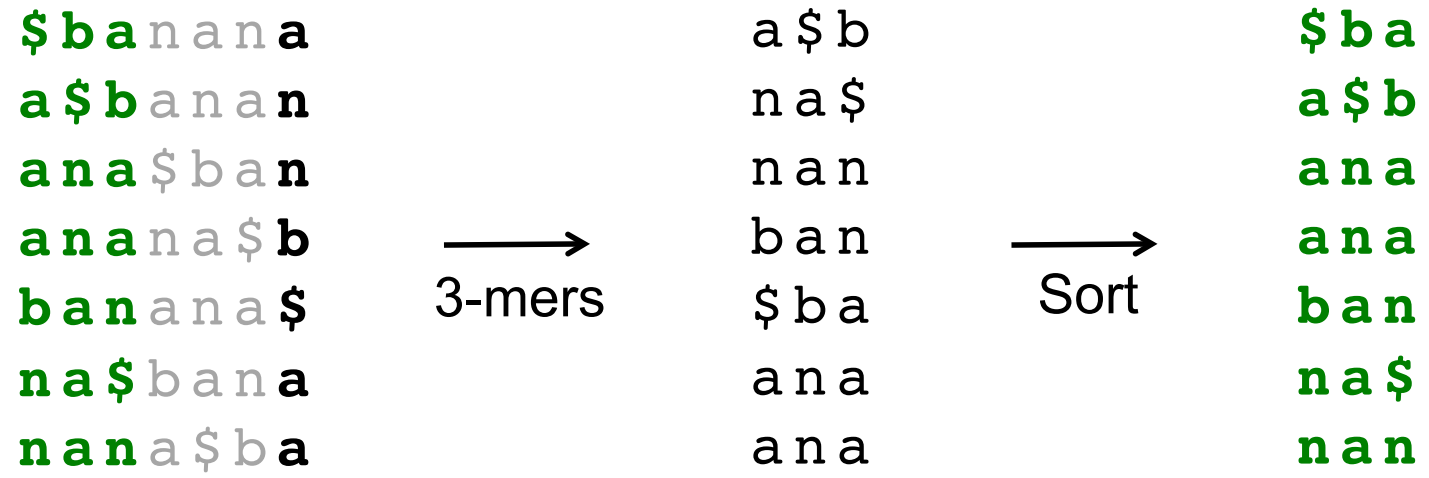


Reconstructing banana



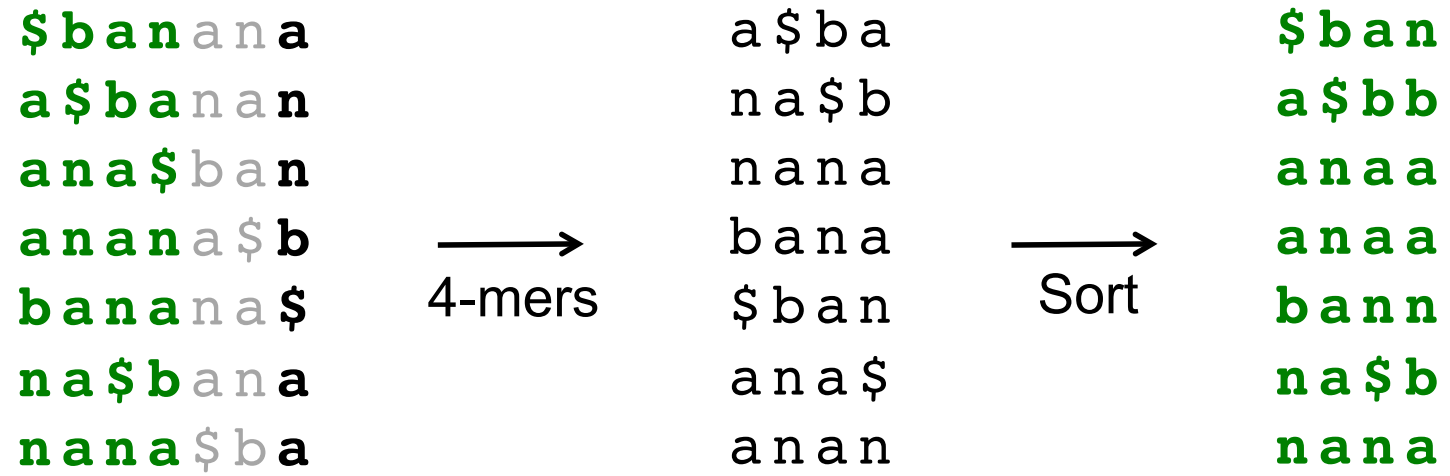
- We now know 2-mer composition of the circular string banana$
- Sorting gives us the first 2 columns of the matrix.

Reconstructing banana



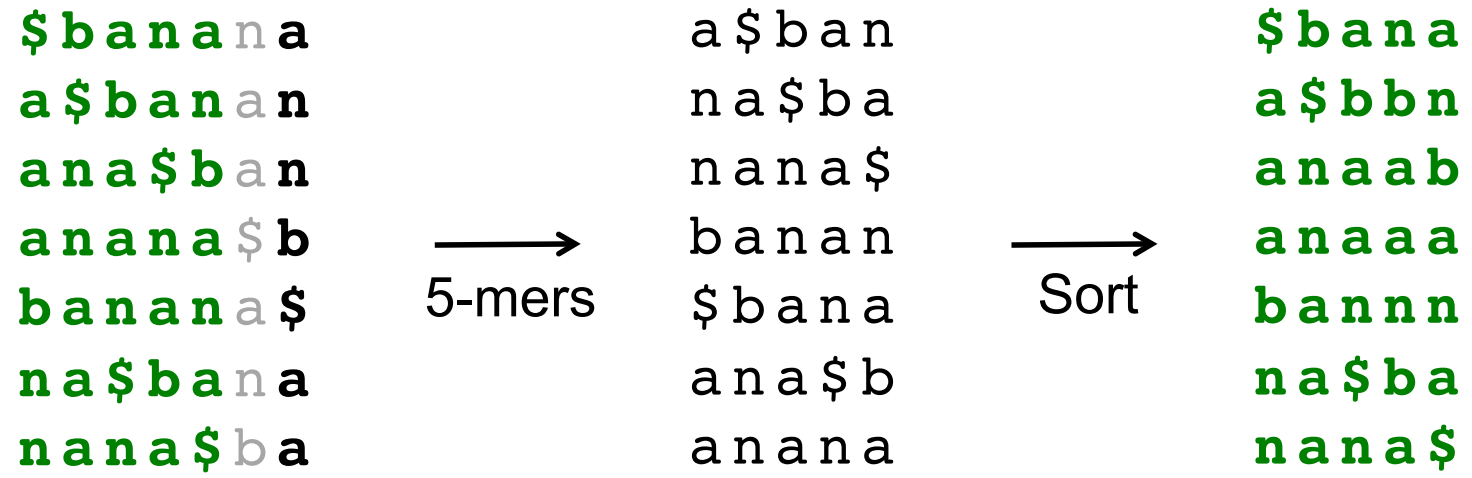
- We now know 3-mer composition of the circular string banana$
- Sorting gives us the first 3 columns of the matrix.

Reconstructing banana



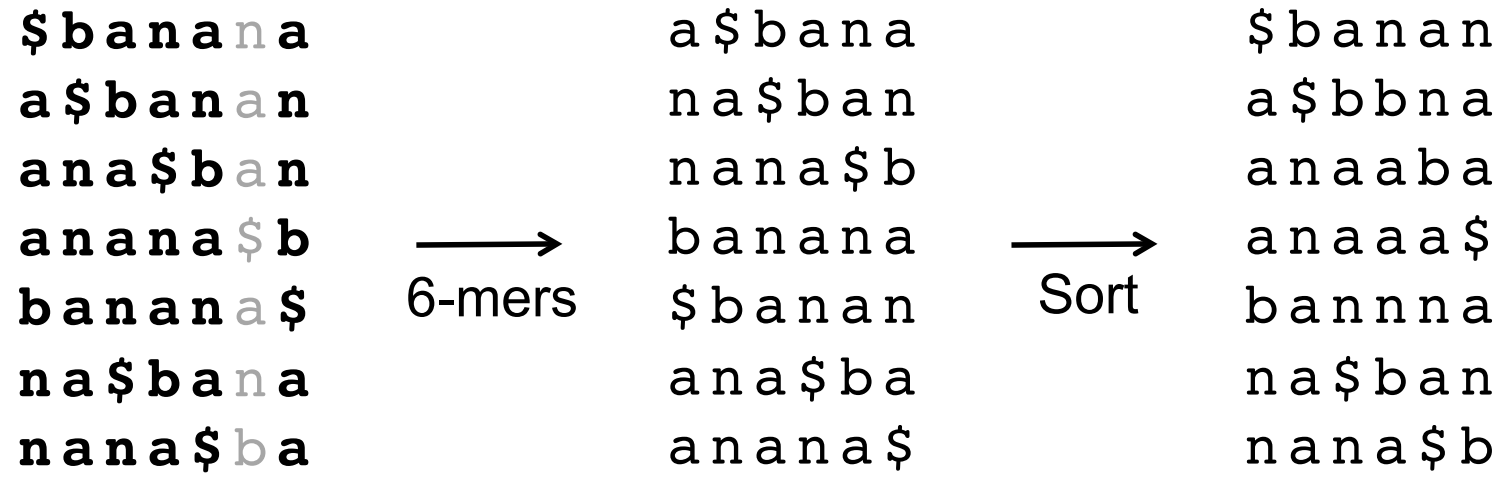
- We now know 4-mer composition of the circular string banana$
- Sorting gives us the first 4 columns of the matrix.

Reconstructing banana



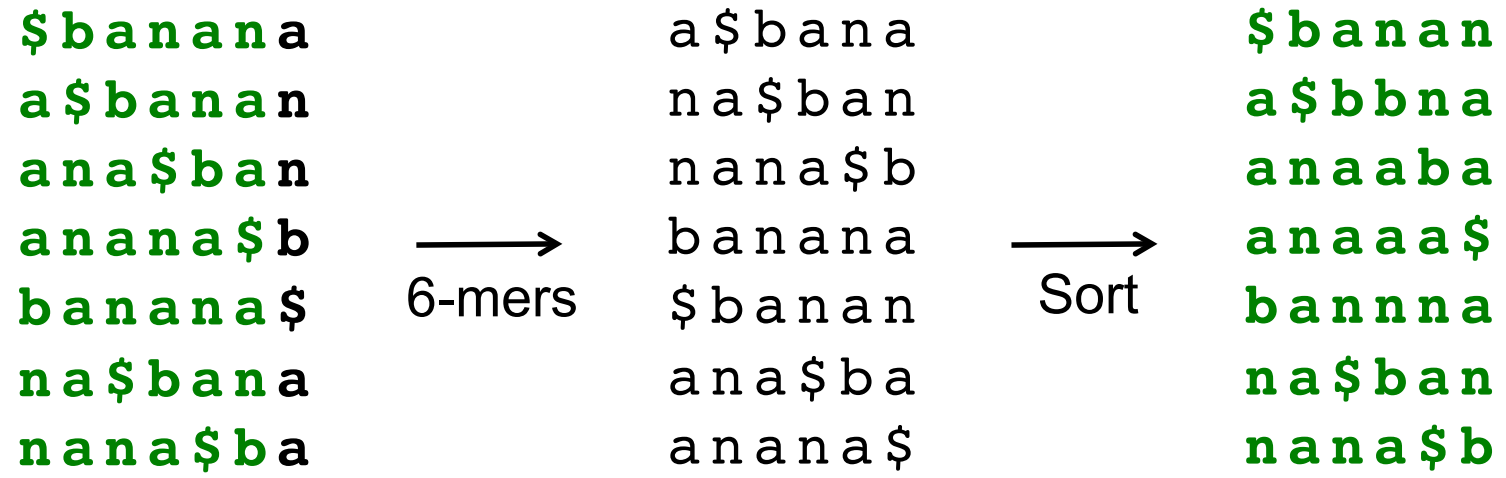
- We now know 5-mer composition of the circular string banana$
- Sorting gives us the first 5 columns of the matrix.

Reconstructing banana



- We now know 6-mer composition of the circular string banana$
- Sorting gives us the first 6 columns of the matrix.

Reconstructing banana



- We now know 6-mer composition of the circular string banana$
- Sorting gives us the first 6 columns of the matrix.

Reconstructing banana

```
$banana
a$banan
ana$ban
anana$b
banana$
na$bana
nana$ba
```

- We now know the entire matrix!

- Taking all elements in the first row (after $) produces banana.

More Memory Issues

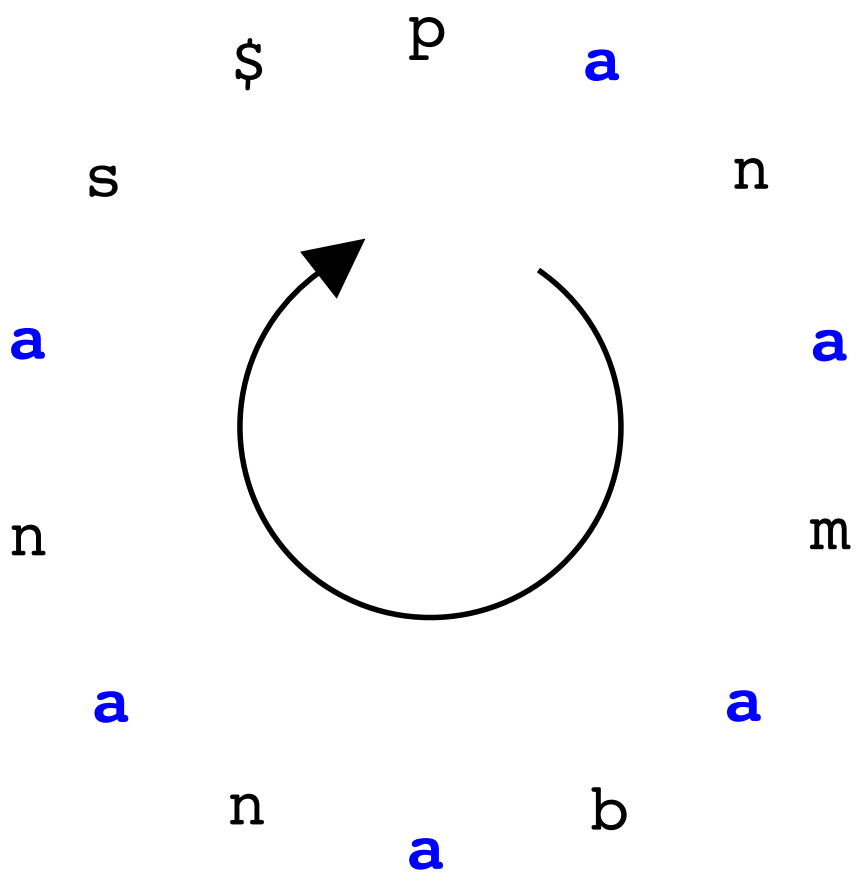
- Reconstructing *Genome* from *BWT*(*Genome*) required us to store |*Genome*| copies of |*Genome*|.

```
$banana
a$banan
ana$ban
anana$b
banana$
na$bana
nana$ba
```

- Can we invert BWT with less space?

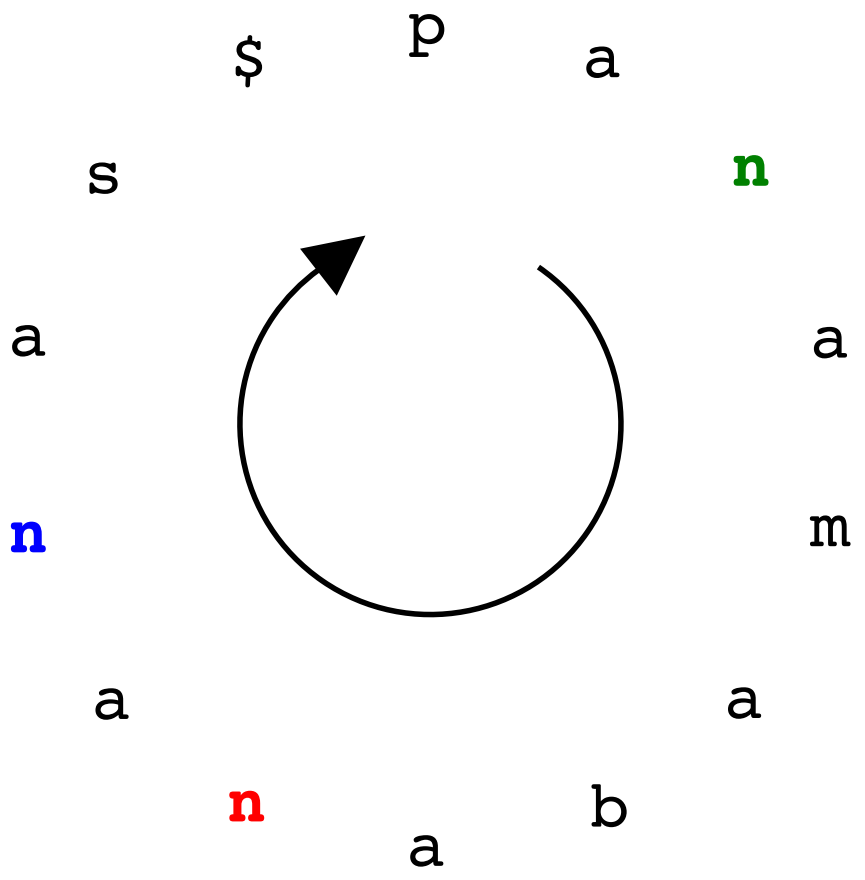
A Strange Observation

```
$panamabananas
abananas$panam
amabananas$pan
anamabananas$p
ananas$panamab
anas$panamaban
as$panamabanan
bananas$panama
mabananas$pana
namabananas$pa
nanas$panamaba
nas$panamabana
panamabananas$
s$panamabanana
```



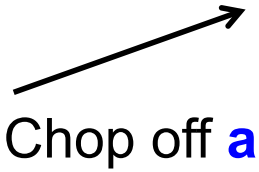
A Strange Observation

```
$panamabananas
abananas$panam
amabananas$pan
anamabananas$p
ananas$panamab
anas$panamaban
as$panamabanan
bananas$panama
mabananas$pana
namabananas$pa
nanas$panamaba
nas$panamabana
panamabananas$
s$panamabanana
```



Is It True in General?

	$panamabananas
1	abananas$panam
2	amabananas$pan
3	anamabananas$p
4	ananas$panamab
5	anas$panamaban
6	as$panamabanan
	bananas$panama
	mabananas$pana
	namabananas$pa
	nanas$panamaba
	nas$panamabana
	panamabananas$
	s$panamabanana

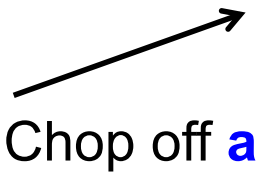


bananas$panam
mabananas$pan
namabananas$p
nanas$panamab
nas$panamaban
s$panamabanan

These strings are sorted

Is It True in General?

```
   $panamabananas
1  abananas$panam
2  amabananas$pan
3  anamabananas$p
4  ananas$panamab
5  anas$panamaban
6  as$panamabanan
   bananas$panama
   mabananas$pana
   namabananas$pa
   nanas$panamaba
   nas$panamabana
   panamabananas$
   s$panamabanana
```

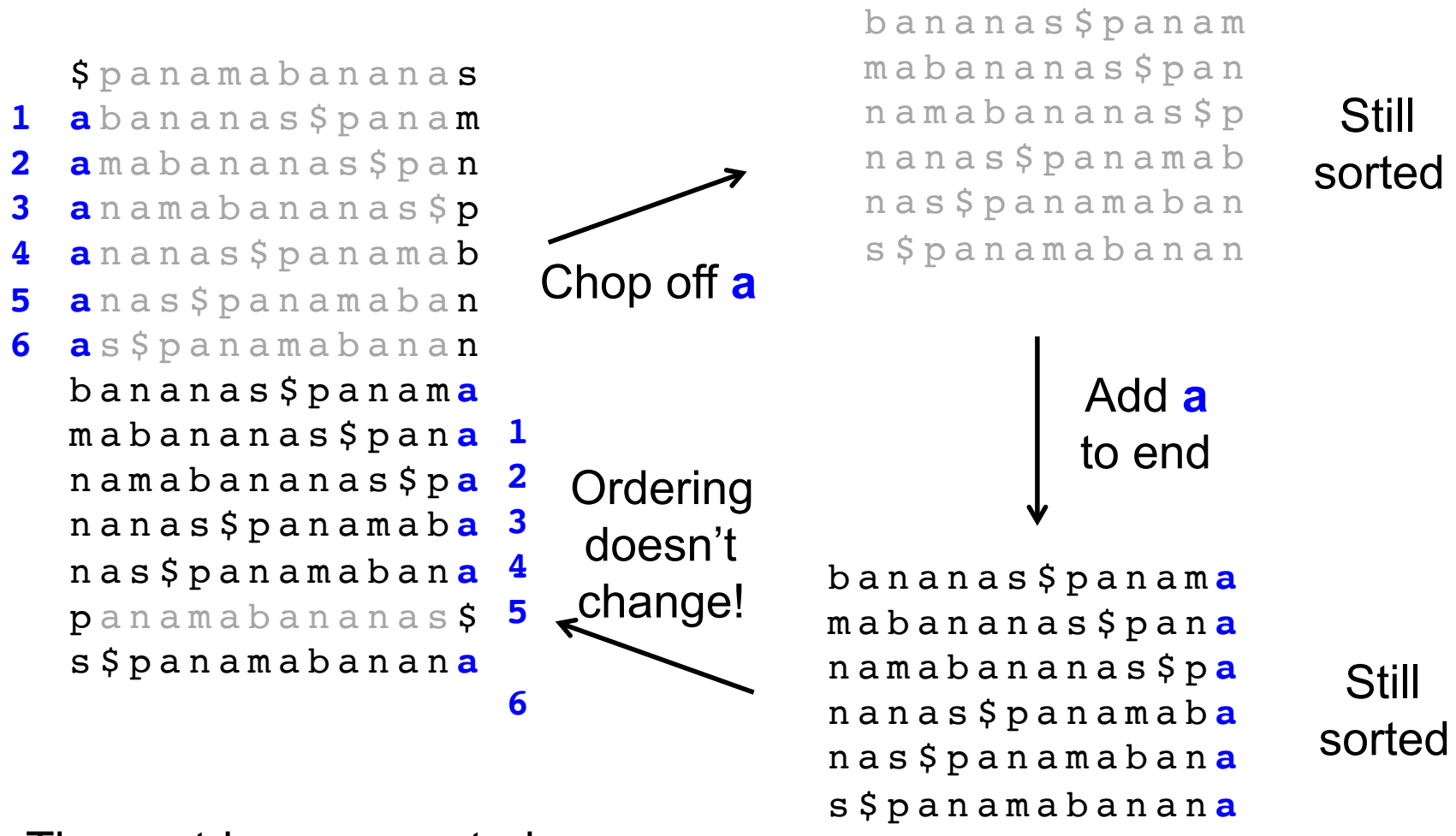



```
bananas$panam
mabananas$pan
namabananas$p
nanas$panamab
nas$panamaban
s$panamabanan
```

Still sorted

These strings are sorted

Is It True in General?



These strings are sorted

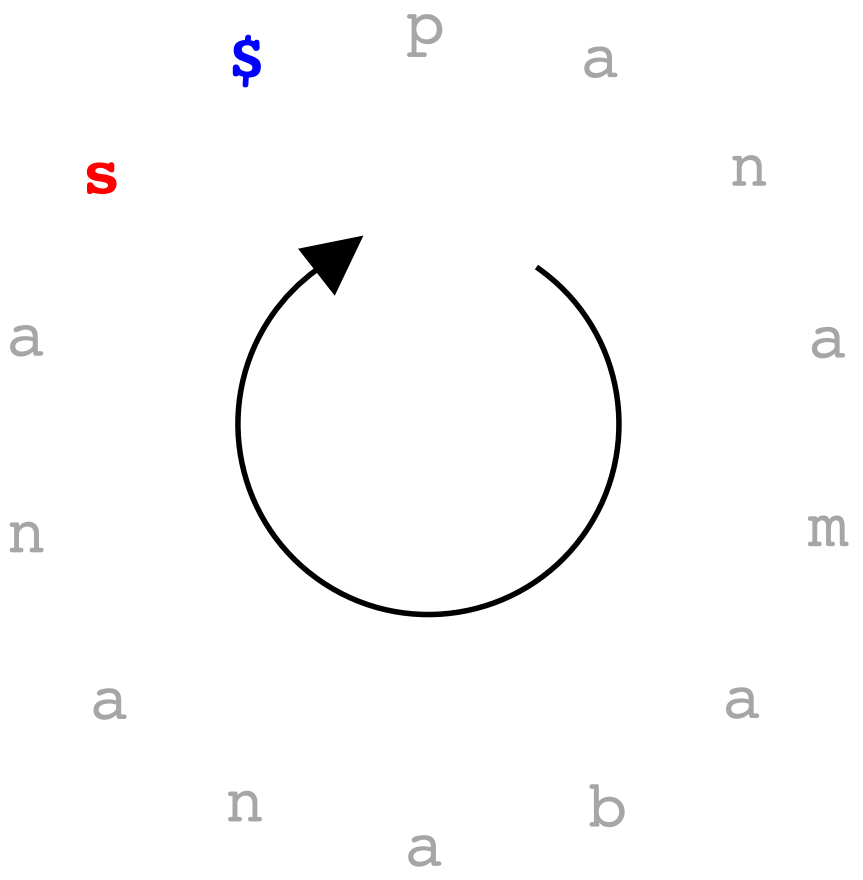
Is It True in General?

- **First-Last Property**: The *k*-th occurrence of *symbol* in *FirstColumn* and the *k*-th occurrence of *symbol* in *LastColumn* correspond to the same position of *symbol* in *Genome*.

$_1panamabananas_1
a_1bananas$panam_1
a_2mabananas$pan_1
a_3namabananas$p_1
a_4nanas$panamab_1
a_5nas$panamaban_2
a_6s$panamabanan_3
b_1ananas$panama_1
m_1abananas$pana_2
n_1amabananas$pa_3
n_2anas$panamaba_4
n_3as$panamaban_a_5
p_1anamabananas$_1
s_1$panamabanana_6

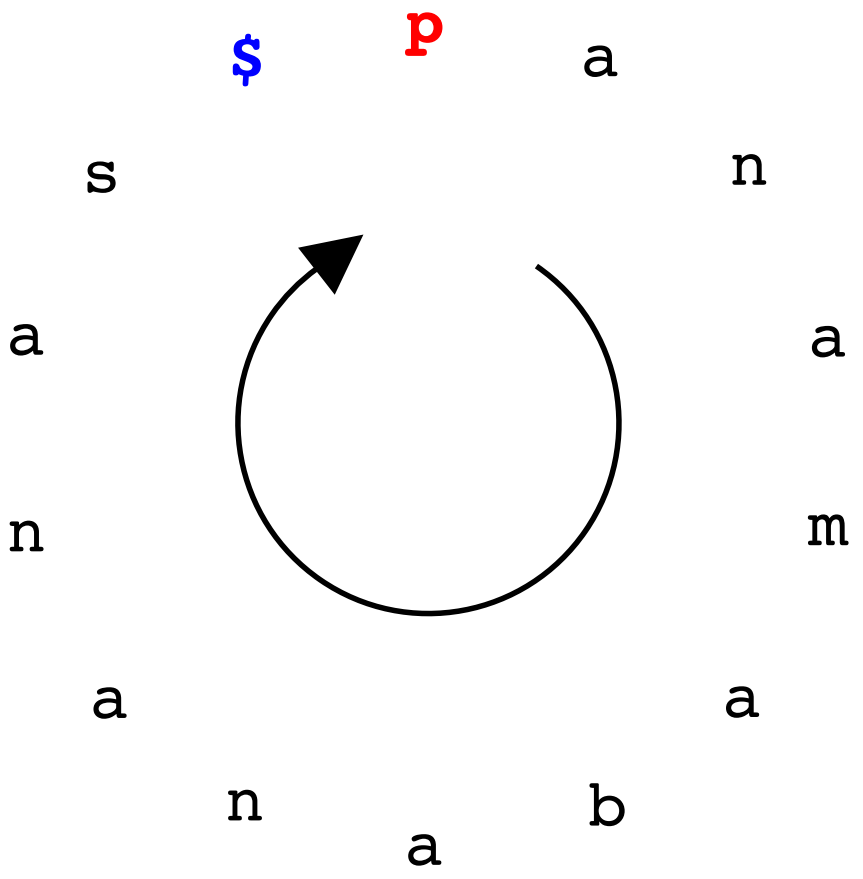
More Efficient BWT Decompression

$_1$ p a n a m a b a n a n a s$_1$
a$_1$ b a n a n a s $ p a n a m$_1$
a$_2$ m a b a n a n a s $ p a n$_1$
a$_3$ n a m a b a n a n a s $ p$_1$
a$_4$ n a n a s $ p a n a m a b$_1$
a$_5$ n a s $ p a n a m a b a n$_2$
a$_6$ s $ p a n a m a b a n a n$_3$
b$_1$ a n a n a s $ p a n a m a$_1$
m$_1$ a b a n a n a s $ p a n a$_2$
n$_1$ a m a b a n a n a s $ p a$_3$
n$_2$ a n a s $ p a n a m a b a$_4$
n$_3$ a s $ p a n a m a b a n a$_5$
p$_1$ a n a m a b a n a n a s $$_1$
s$_1$ $ p a n a m a b a n a n a$_6$



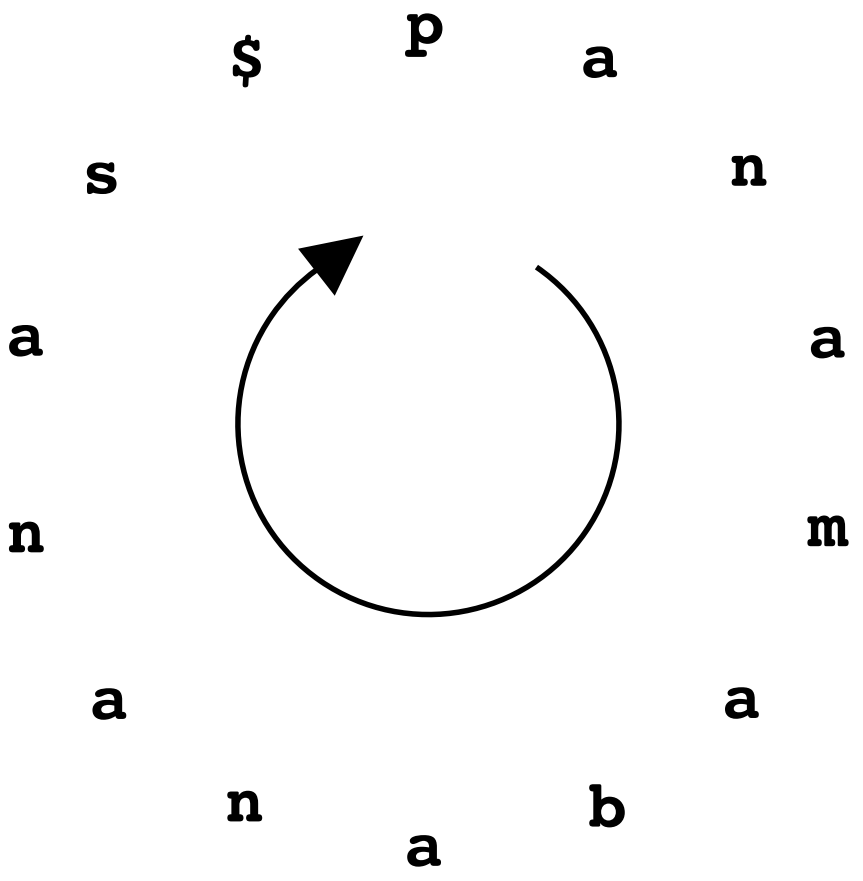
More Efficient BWT Decompression

$_1$ p a n a m a b a n a n a s $_1$
a $_1$ b a n a n a s $ p a n a m $_1$
a $_2$ m a b a n a n a s $ p a n $_1$
a $_3$ n a m a b a n a n a s $ p $_1$
a $_4$ n a n a s $ p a n a m a b $_1$
a $_5$ n a s $ p a n a m a b a n $_2$
a $_6$ s $ p a n a m a b a n a n $_3$
b $_1$ a n a n a s $ p a n a m a $_1$
m $_1$ a b a n a n a s $ p a n a $_2$
n $_1$ a m a b a n a n a s $ p a $_3$
n $_2$ a n a s $ p a n a m a b a $_4$
n $_3$ a s $ p a n a m a b a n a $_5$
p $_1$ a n a m a b a n a n a s $ $_1$
s $_1$ $ p a n a m a b a n a n a $_6$



More Efficient BWT Decompression

$_1$panamabananas$_1$
a$_1$bananas$panam$_1$
a$_2$mabananaspan_1$
a$_3$namabananas$p$_1$
a$_4$nanas$panamab$_1$
a$_5$nas$panamaban$_2$
a$_6$s$panamabanan$_3$
b$_1$ananas$panama$_1$
m$_1$abananas$pana$_2$
n$_1$amabananaspa_3$
n$_2$anas$panamaba$_4$
n$_3$as$panamaban$_5$
p$_1$anamabananas$$_1$
s$_1$$panamabanana$_6$



- Memory: 2|*Genome*| = O(|*Genome*|).

Recalling Our Goal

- Suffix Tree Pattern Matching:
  - Runtime: O(|*Genome*| + |*Patterns*|)
  - Memory: O(|*Genome*|)
  - Problem: suffix tree takes 20 x |*Genome*| space

- Can we use BWT(*Genome*) as our data structure instead?

Finding Pattern Matches Using BWT

- Searching for **ana** in p**ana**mab**anana**s

Finding Pattern Matches Using BWT

- Searching for ana in panamabananas

Finding Pattern Matches Using BWT

- Searching for a**na** in panamabananas

$_1$panamabananas$_1$
a$_1$bananas$panam**m**$_1$
a$_2$mabananas$pan**n**$_1$
a$_3$namabananas$**p**$_1$
a$_4$nanas$panama**b**$_1$
a$_5$nas$panamaban**n**$_2$
a$_6$s$panamabanan**n**$_3$
b$_1$ananas$panama$_1$
m$_1$abananas$pana$_2$
n$_1$amabananaspa_3$
n$_2$anas$panamaba$_4$
n$_3$as$panamaban$_5$
p$_1$anamabananas$$_1$
s$_1$$panamabanana$_6$

Finding Pattern Matches Using BWT

- Searching for **ana** in panamabananas

Where Are the Matches?

- **Multiple Pattern Matching Problem:**
  - **Input:** A collection of strings *Patterns* and a string *Genome.*
  - **Output:** All **positions** in *Genome* where one of *Patterns* appears as a substring.

- Where are the **positions**? BWT has not revealed them.

Where Are the Matches?

- Example: We know that **ana** occurs 3 times, but where?

$_1$panamabananas$_1$
a$_1$bananas$panam$_1$
a$_2$mabananaspan_1$
a$_3$namabananas$p$_1$
a$_4$nanas$panamab$_1$
a$_5$nas$panamaban$_2$
a$_6$s$panamabanan$_3$
b$_1$ananas$panama$_1$
m$_1$abananas$pana$_2$
n$_1$amabananaspa_3$
n$_2$anas$panamaba$_4$
n$_3$as$panamabana$_5$
p$_1$anamabananas$$_1$
s$_1$$panamabanana$_6$

Using the Suffix Array to Find Matches

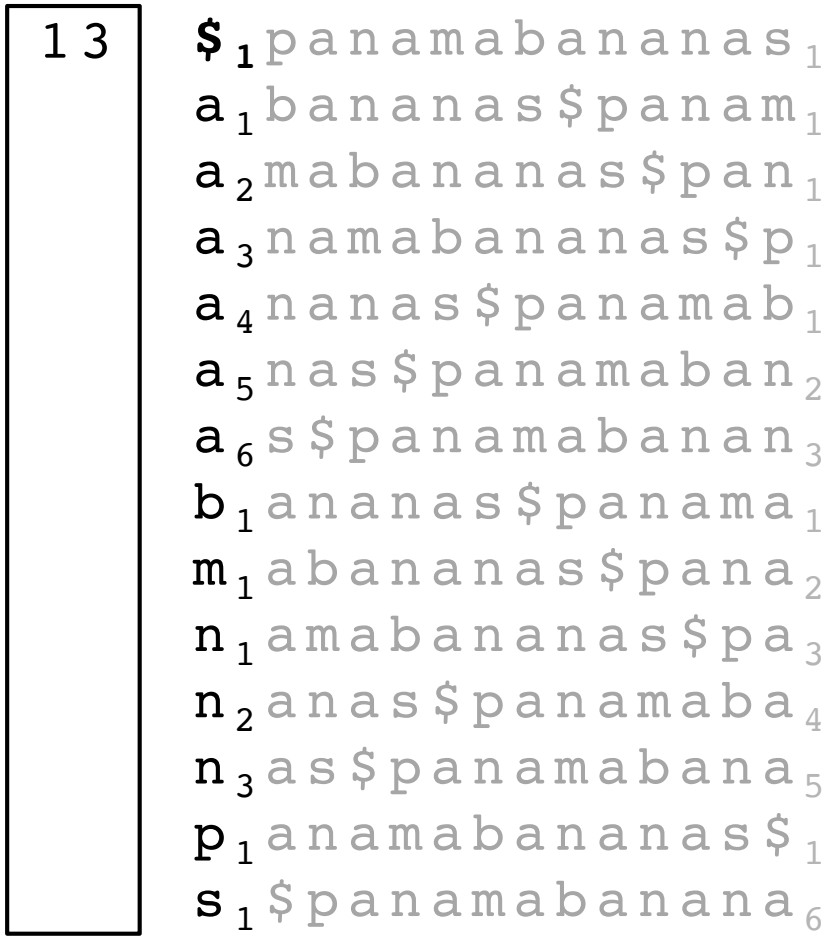
- **Suffix array:** holds starting position of each suffix beginning a row.

$_1$panamabananas$_1$
a$_1$bananas$panam$_1$
a$_2$mabananaspan_1$
a$_3$namabananas$p$_1$
a$_4$nanas$panamab$_1$
a$_5$nas$panamaban$_2$
a$_6$s$panamabanan$_3$
b$_1$ananas$panama$_1$
m$_1$abananas$pana$_2$
n$_1$amabananaspa_3$
n$_2$anas$panamaba$_4$
n$_3$as$panamabana$_5$
p$_1$anamabananas$$_1$
s$_1$$panamabanana$_6$

Using the Suffix Array to Find Matches

- **Suffix array:** holds starting position of each suffix beginning a row.

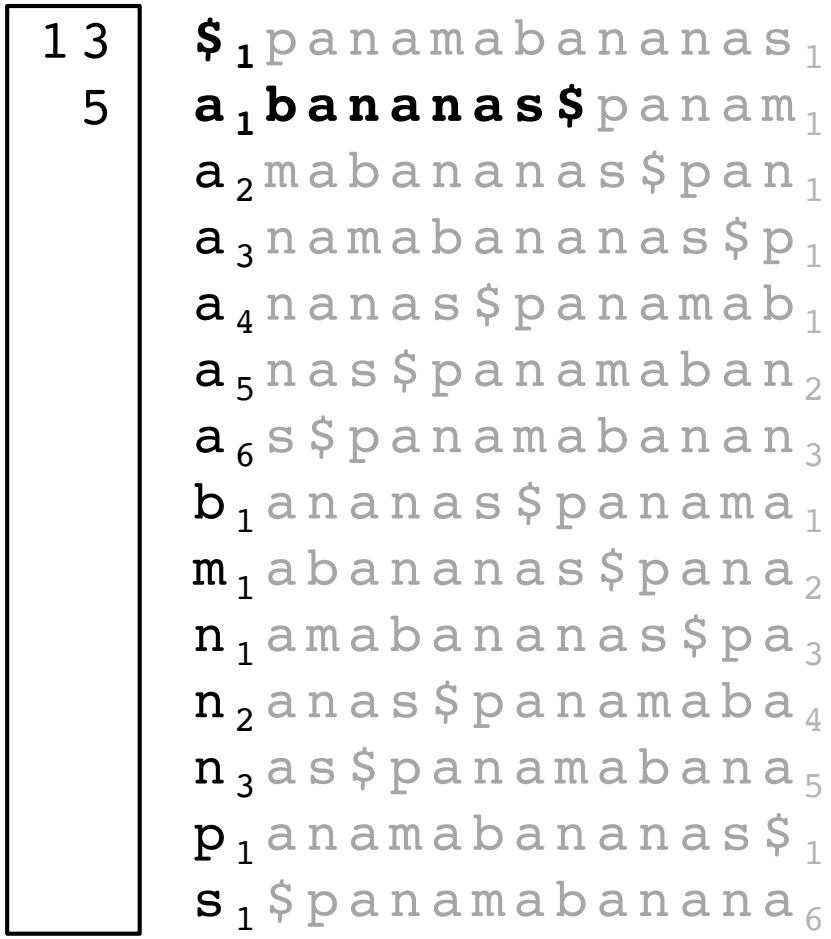
panamabananas**$**



Using the Suffix Array to Find Matches

- **Suffix array:** holds starting position of each suffix beginning a row.

panam**abananas$**



Using the Suffix Array to Find Matches

- **Suffix array:** holds starting position of each suffix beginning a row.

pan**amabananas$**

13
5
3

$_1$panamabananas$_1$
a$_1$bananas$panam$_1$
a$_2$mabananaspan_1$
a$_3$namabananas$p$_1$
a$_4$nanas$panamab$_1$
a$_5$nas$panamaban$_2$
a$_6$s$panamabanan$_3$
b$_1$ananas$panama$_1$
m$_1$abananas$pana$_2$
n$_1$amabananaspa_3$
n$_2$anas$panamaba$_4$
n$_3$as$panamabana$_5$
p$_1$anamabananas$$_1$
s$_1$$panamabanana$_6$

Using the Suffix Array to Find Matches

- **Suffix array:** holds starting position of each suffix beginning a row.

panamabananas$

13	$$_1$ panamabananas$_1$
5	a$_1$ **bananas$** panam$_1$
3	a$_2$ **mabananas$** pan$_1$
1	a$_3$ **namabananas$** p$_1$
	a$_4$ nanas$panamab$_1$
	a$_5$ nas$panamaban$_2$
	a$_6$ s$panamabanan$_3$
	b$_1$ ananas$panama$_1$
	m$_1$ abananas$pana$_2$
	n$_1$ amabananaspa_3$
	n$_2$ anas$panamaba$_4$
	n$_3$ as$panamabana$_5$
	p$_1$ anamabananas$$_1$
	s$_1$ $panamabanana$_6$

Using the Suffix Array to Find Matches

- **Suffix array:** holds starting position of each suffix beginning a row.

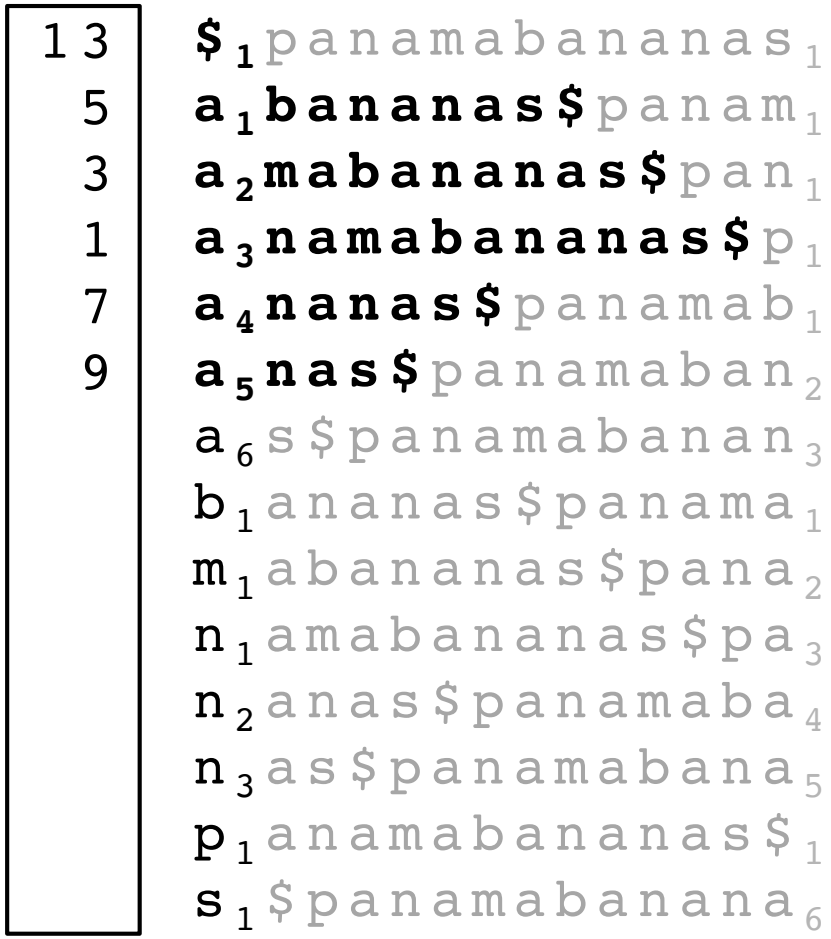
panamab**ananas$**

13	$$_1$ panamabananas$_1$
5	**a$_1$bananas$** panam$_1$
3	**a$_2$mabananas$** pan$_1$
1	**a$_3$namabananas$** p$_1$
7	**a$_4$nanas$** panamab$_1$
	a$_5$nas$panamaban$_2$
	a$_6$s$panamabanan$_3$
	b$_1$ananas$panama$_1$
	m$_1$abananas$pana$_2$
	n$_1$amabananaspa_3$
	n$_2$anas$panamaba$_4$
	n$_3$as$panamabana$_5$
	p$_1$anamabananas$$_1$
	s$_1$$panamabanana$_6$

Using the Suffix Array to Find Matches

- **Suffix array:** holds starting position of each suffix beginning a row.

panamaban**anas$**



Using the Suffix Array to Find Matches

- **Suffix array:** holds starting position of each suffix beginning a row.

panamabanan**as$**

13	**$**$_1$panamabananas$_1$
5	**a**$_1$**bananas$**panam$_1$
3	**a**$_2$**mabananas$**pan$_1$
1	**a**$_3$**namabananas$**p$_1$
7	**a**$_4$**nanas$**panamab$_1$
9	**a**$_5$**nas$**panamaban$_2$
11	**a**$_6$**s$**panamabanan$_3$
	b$_1$ananas$panama$_1$
	m$_1$abananas$pana$_2$
	n$_1$amabananaspa_3$
	n$_2$anas$panamaba$_4$
	n$_3$as$panamabana$_5$
	p$_1$anamabananas$$_1$
	s$_1$$panamabanana$_6$

Using the Suffix Array to Find Matches

- **Suffix array:** holds starting position of each suffix beginning a row.

panama**bananas$**

13	$$_1$ panamabananas$_1$
5	a$_1$**bananas$** panam$_1$
3	a$_2$**mabananas$** pan$_1$
1	a$_3$**namabananas$** p$_1$
7	a$_4$**nanas$** panamab$_1$
9	a$_5$**nas$** panamaban$_2$
11	a$_6$**s$** panamabanan$_3$
6	b$_1$**ananas$** panama$_1$
	m$_1$abananas$pana$_2$
	n$_1$amabananaspa_3$
	n$_2$anas$panamaba$_4$
	n$_3$as$panamabana$_5$
	p$_1$anamabananas$$_1$
	s$_1$$panamabanana$_6$

Using the Suffix Array to Find Matches

- **Suffix array:** holds starting position of each suffix beginning a row.

panamabana**nas$**

13	**$**$_1$panamabananas$_1$
5	**a**$_1$**bananas$**panam$_1$
3	**a**$_2$**mabananas$**pan$_1$
1	**a**$_3$**namabananas$**p$_1$
7	**a**$_4$**nanas$**panamab$_1$
9	**a**$_5$**nas$**panamaban$_2$
11	**a**$_6$**s$**panamabanan$_3$
6	**b**$_1$**ananas$**panama$_1$
4	**m**$_1$**abananas$**pana$_2$
2	**n**$_1$**amabananas$**pa$_3$
8	**n**$_2$**anas$**panamaba$_4$
10	**n**$_3$**as$**panamabana$_5$
	p$_1$anamabananas$_1$
	s$_1$$panamabanana$_6$

Using the Suffix Array to Find Matches

- **Suffix array:** holds starting position of each suffix beginning a row.

panamabananas$

13	$$_1$panamabananas$_1$
5	a$_1$bananas$panam$_1$
3	a$_2$mabananaspan_1$
1	a$_3$namabananas$p$_1$
7	a$_4$nanas$panamab$_1$
9	a$_5$nas$panamaban$_2$
11	a$_6$s$panamabanan$_3$
6	b$_1$ananas$panama$_1$
4	m$_1$abananas$pana$_2$
2	n$_1$amabananaspa_3$
8	n$_2$anas$panamaba$_4$
10	n$_3$as$panamabana$_5$
0	p$_1$anamabananas$_1$
	s$_1$$panamabanana$_6$

Using the Suffix Array to Find Matches

- **Suffix array:** holds starting position of each suffix beginning a row.

panamabanana**s$**

13	**$**$_1$panamabananas$_1$
5	**a**$_1$**bananas$**panam$_1$
3	**a**$_2$**mabananas$**pan$_1$
1	**a**$_3$**namabananas$**p$_1$
7	**a**$_4$**nanas$**panamab$_1$
9	**a**$_5$**nas$**panamaban$_2$
11	**a**$_6$**s$**panamabanan$_3$
6	**b**$_1$**ananas$**panama$_1$
4	**m**$_1$**abananas$**pana$_2$
2	**n**$_1$**amabananas$**pa$_3$
8	**n**$_2$**anas$**panamaba$_4$
10	**n**$_3$**as$**panamabana$_5$
0	**p**$_1$**anamabananas$**$_1$
12	**s**$_1$**$**panamabanana$_6$

Using the Suffix Array to Find Matches

- **Suffix array:** holds starting position of each suffix beginning a row.

`panamabananas$`

13	$$_1$panamabananas$_1$
5	a$_1$bananas$panam$_1$
3	a$_2$mabananaspan_1$
1	a$_3$namabananas$p$_1$
7	a$_4$nanas$panamab$_1$
9	a$_5$nas$panamaban$_2$
11	a$_6$s$panamabanan$_3$
6	b$_1$ananas$panama$_1$
4	m$_1$abananas$pana$_2$
2	n$_1$amabananaspa_3$
8	n$_2$anas$panamaba$_4$
10	n$_3$as$panamaban$a$_5$
0	p$_1$anamabananas$$_1$
12	s$_1$$panamabanana$_6$

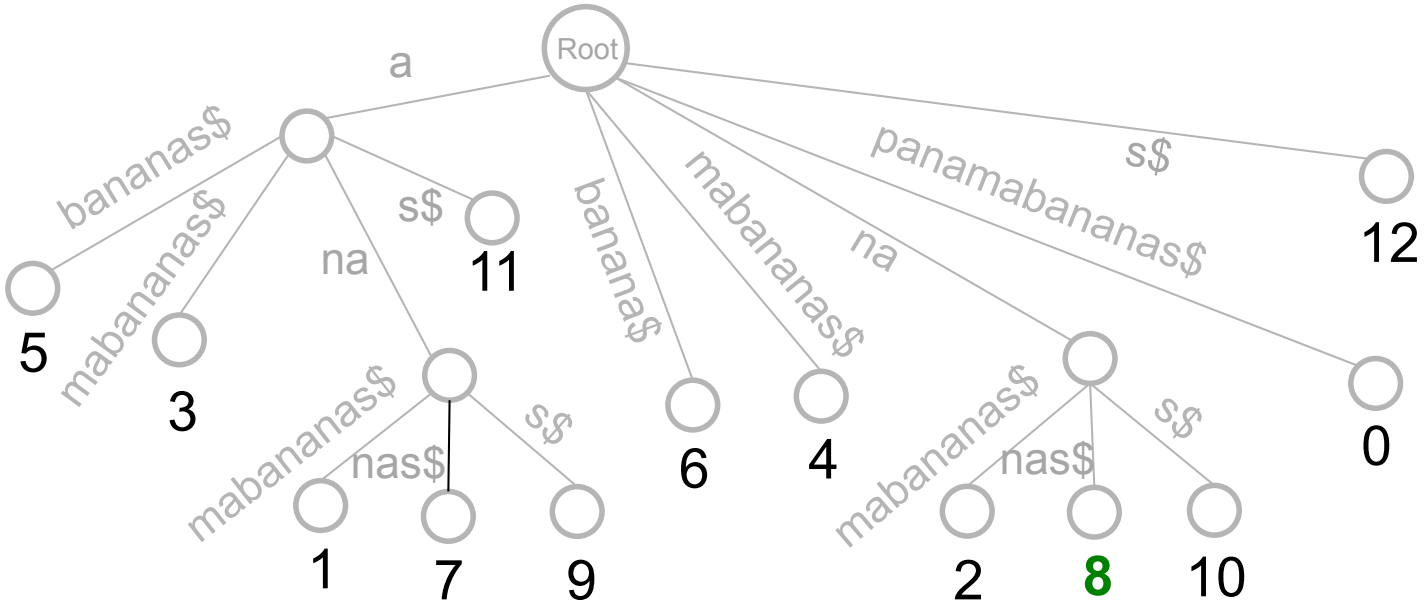
Using the Suffix Array to Find Matches

- **Suffix array:** holds starting position of each suffix beginning a row.
- Thus, **ana** occurs at positions **1**, **7**, **9** of p**ana**mab**anana**s$.

13	$
5	a bananas$panam
3	a mabananas$pan
1	**ana**mabananas$p
7	**ana**nas$panamab
9	**ana**s$panamaban
11	a s$panamabanan
6	b ananas$panama
4	m abananas$pana
2	n amabananas$pa
8	n anas$panamaba
10	n as$panamabana
0	p anamabananas$
12	s $panamabanana

The Suffix Array: Memory Once Again

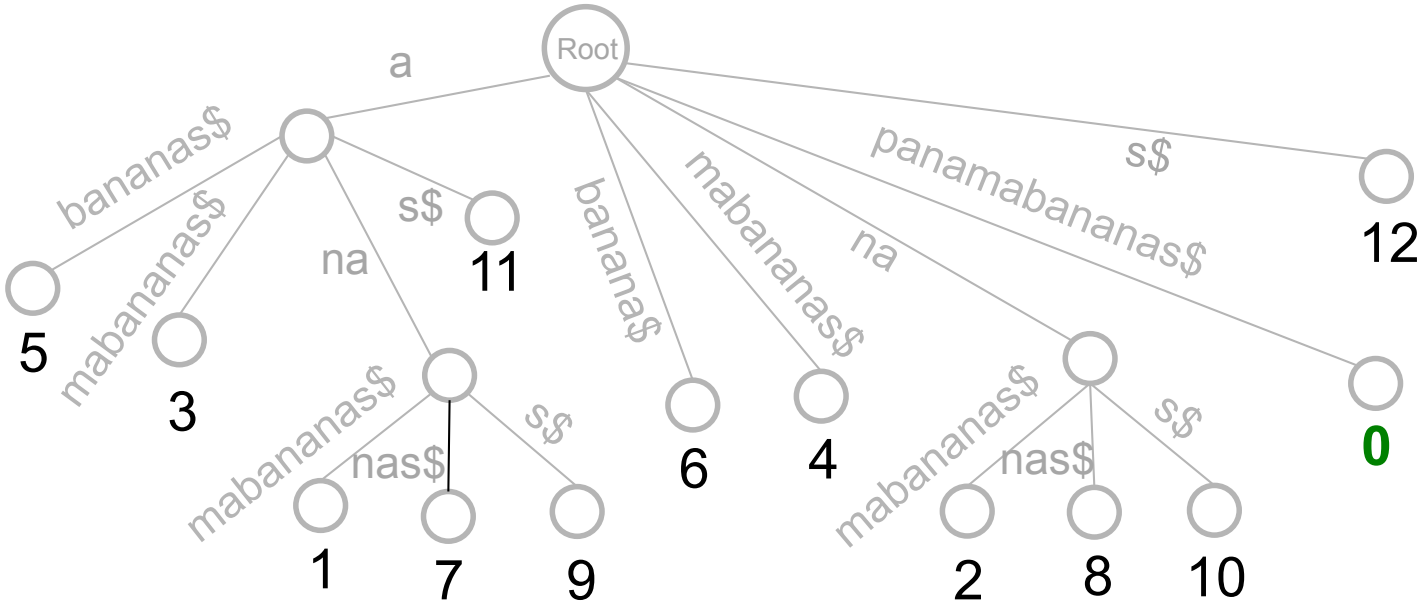
- Memory: ~ 4 x |*Genome*|.



[13 5 3 1 7 9 11 6 4 2 **8** 10 0 1

The Suffix Array: Memory Once Again

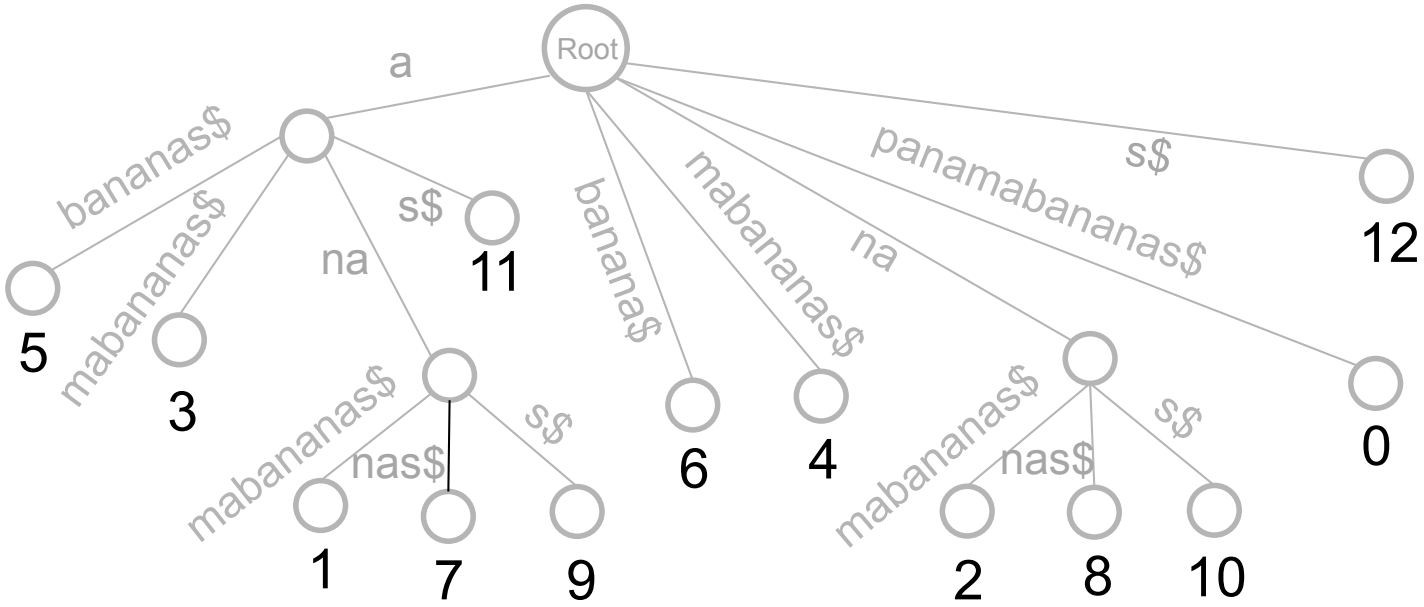
- Memory: ~ 4 x |*Genome*|.



[13 5 3 1 7 9 11 6 4 2 8 10 **0** 1

The Suffix Array: Memory Once Again

- Memory: ~ 4 x |*Genome*|.



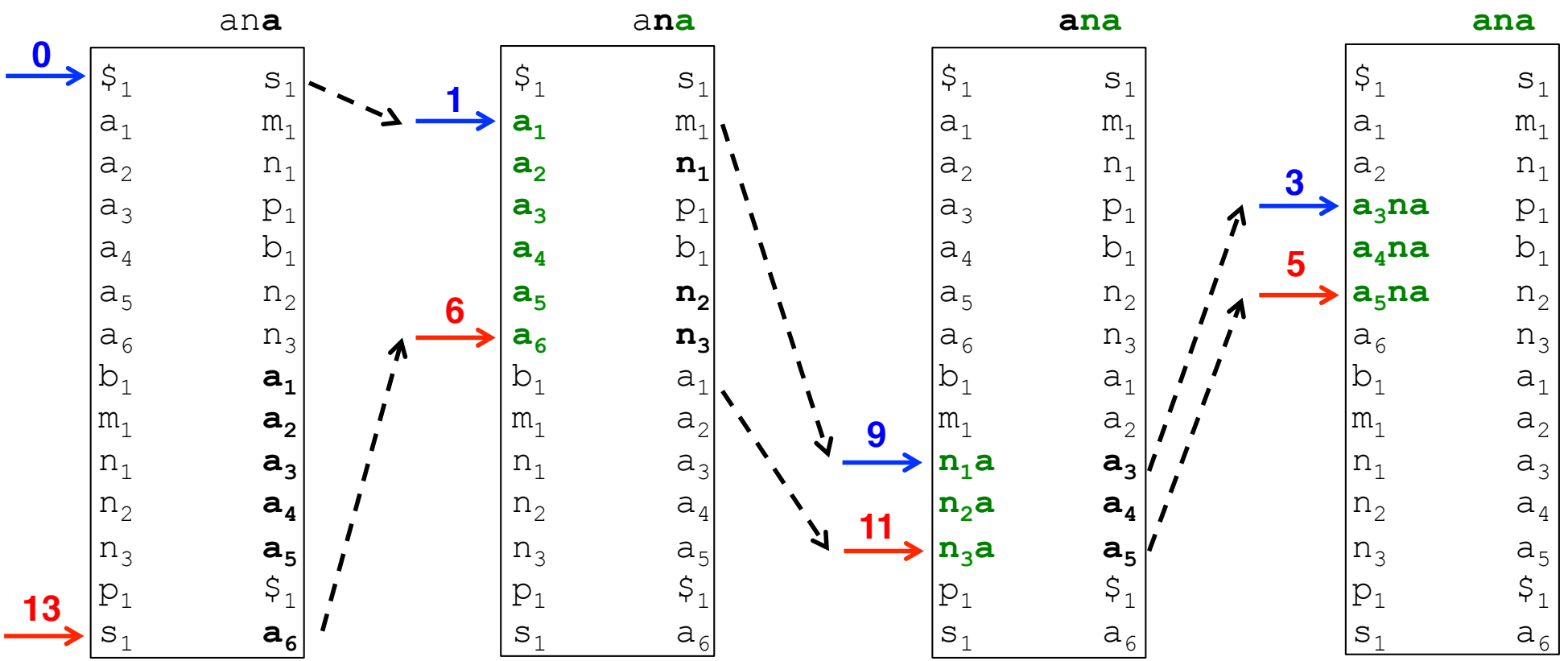
[13 5 3 1 7 9 11 6 4 2 8 10 0 1

Reducing Suffix Array Size

- We don't want to have to store all of the suffix array; can we store only part of it? Show how checkpointing can be used to store 1/100 the suffix array.

A Return to Constants

- Explain that using a checkpointed array increases runtime by a constant factor, but in practice it is a worthwhile trade-off.


ana
ana
ana
ana
0
1
6
13
9
11
3
5
$1 s1
a1 m1
a2 n1
a3 p1
a4 b1
a5 n2
a6 n3
b1 a1
m1 a2
n1 a3
n2 a4
n3 a5
p1 $1
s1 a6
$1 s1
a1 m1
a2 n1
a3 p1
a4 b1
a5 n2
a6 n3
b1 a1
m1 a2
n1 a3
n2 a4
n3 a5
p1 $1
s1 a6
$1 s1
a1 m1
a2 n1
a3 p1
a4 b1
a5 n2
a6 n3
b1 a1
m1 a2
n1a a3
n2a a4
n3a a5
p1 $1
s1 a6
$1 s1
a1 m1
a2 n1
a3na p1
a4na b1
a5na n2
a6 n3
b1 a1
m1 a2
n1 a3
n2 a4
n3 a5
p1 $1
s1 a6

Returning to Our Original Problem

- We need to look at INEXACT matching in order to find variants.

- **Approx. Pattern Matching Problem**:
  - **Input**: A string *Pattern*, a string *Genome*, and an integer *d*.
  - **Output:** All positions in *Genome* where *Pattern* appears as a substring with at most *d* mismatches.

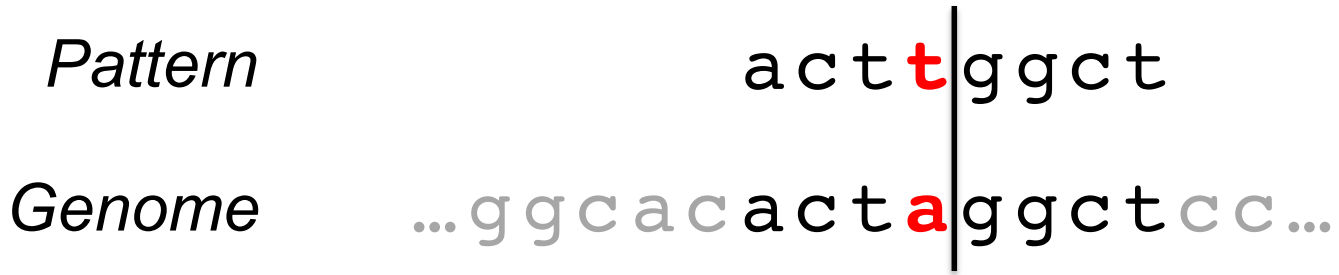
Returning to Our Original Problem

- We need to look at INEXACT matching in order to find variants.

- **Multiple Approx. Pattern Matching Problem**:
  - **Input**: A **collection** of strings *Patterns*, a string *Genome*, and an integer *d*.
  - **Output:** All positions in *Genome* where a string from *Patterns* appears as a substring with at most *d* mismatches.

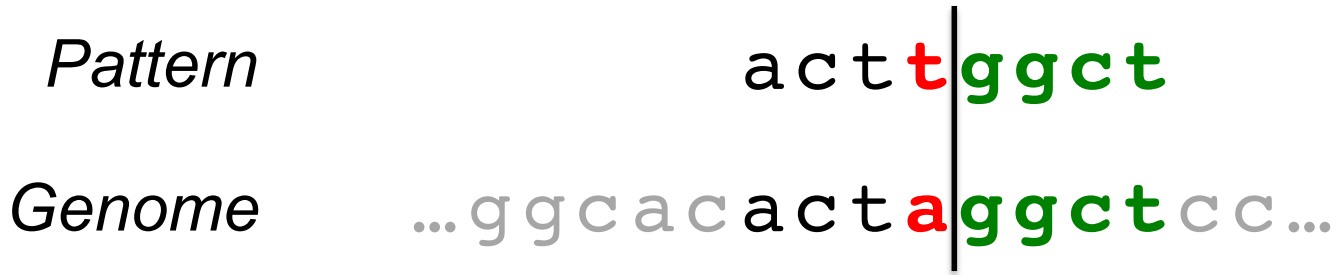
Method 1: Seeding

- Say that *Pattern* appears in *Genome* with 1 mismatch:



Method 1: Seeding

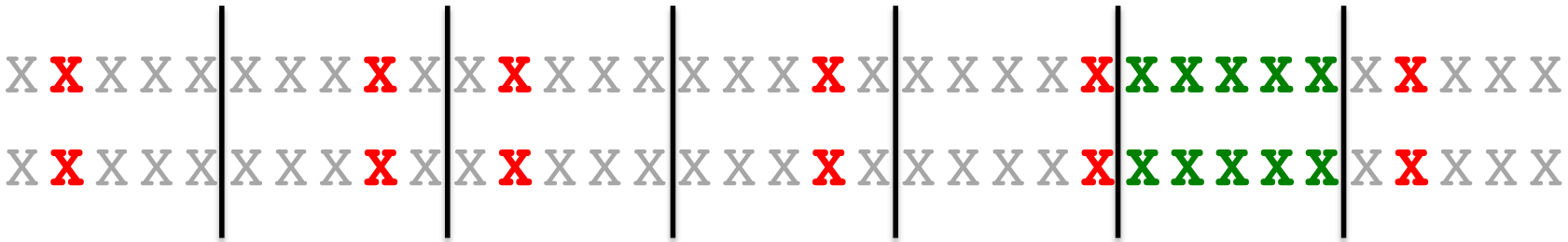
- Say that *Pattern* appears in *Genome* with 1 mismatch:



- One of the substrings must match!

Method 1: Seeding

- **Theorem:** If *Pattern* occurs in *Genome* with *d* mismatches, then we can divide *Pattern* into *d* + 1 "equal" pieces and find at least one exact match.



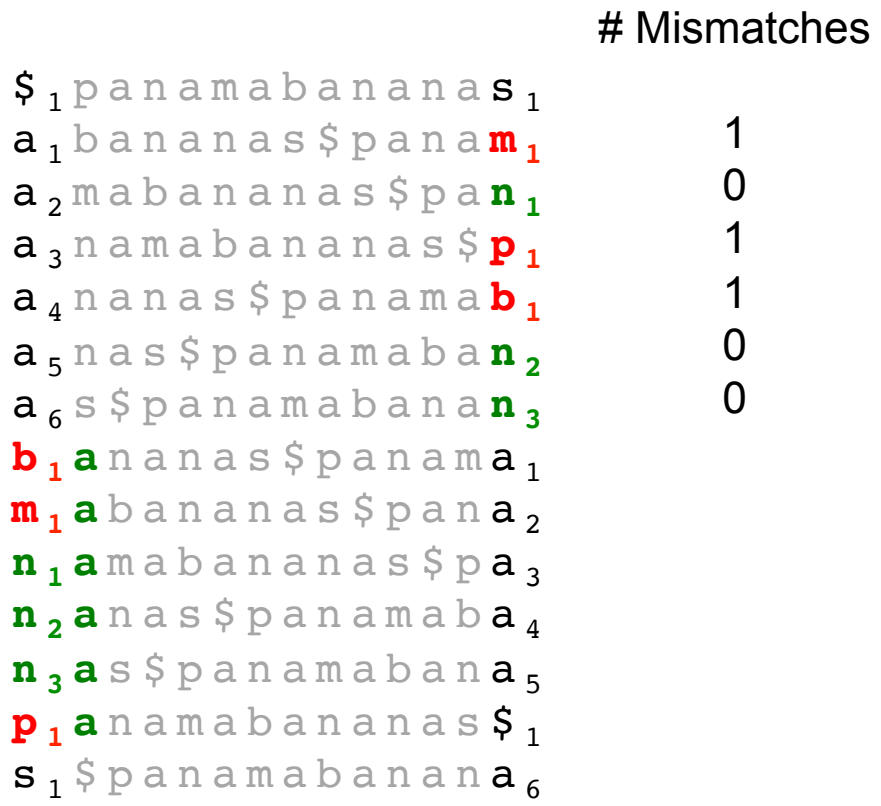
Method 1: Seeding

- Say we are looking for at most *d* mismatches.
- Divide each of our strings into *d* + 1 smaller pieces, called **seeds**.
- Check if each *Pattern* has a seed that matches *Genome* exactly.
- If so, check the entire *Pattern* against *Genome*.

Method 2: BWT Saves the Day Again

- Recall: searching for a**na** in panamabananas

Now we extend all strings with at most 1 mismatch.



Mismatches

1
0
1
1
0
0

Method 2: BWT Saves the Day Again

- Recall: searching for **ana** in panamabananas

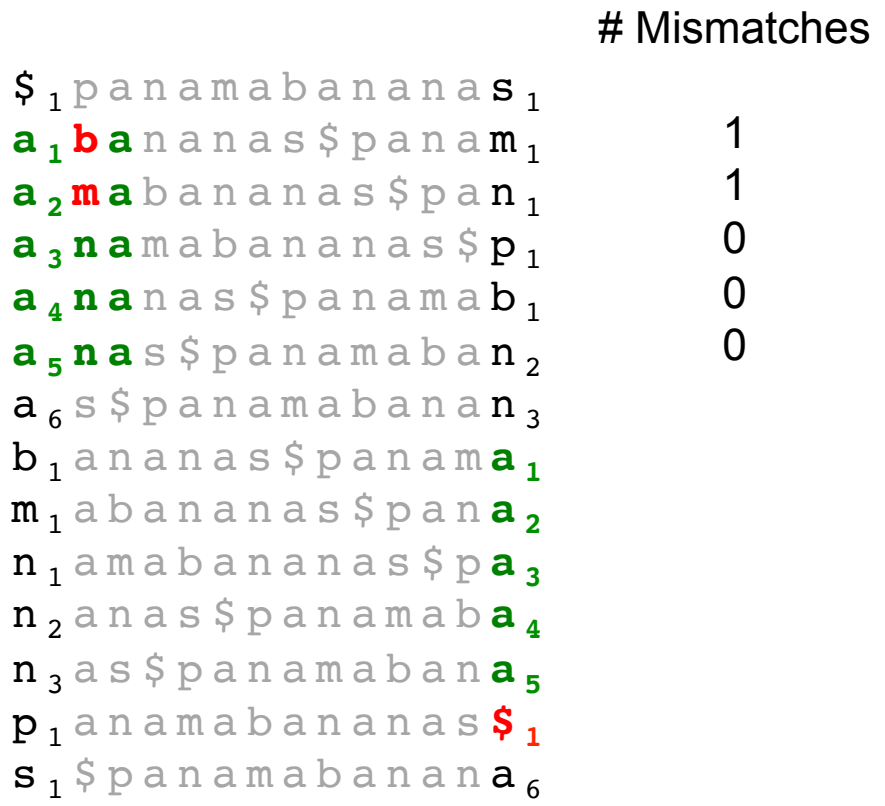
One string produces a second mismatch (the $), so we discard it.

	# Mismatches
$ 1 p a n a m a b a n a n a s 1	
a 1 b a n a n a s $ p a n a m 1	
a 2 m a b a n a n a s $ p a n 1	
a 3 n a m a b a n a n a s $ p 1	
a 4 n a n a s $ p a n a m a b 1	
a 5 n a s $ p a n a m a b a n 2	
a 6 s $ p a n a m a b a n a n 3	
b 1 a n a n a s $ p a n a m **a 1**	1
m 1 a b a n a n a s $ p a n **a 2**	1
n 1 a m a b a n a n a s $ p **a 3**	0
n 2 a n a s $ p a n a m a b **a 4**	0
n 3 a s $ p a n a m a b a n **a 5**	0
p 1 a n a m a b a n a n a s **$ 1**	2
s 1 $ p a n a m a b a n a n a 6	

Method 2: BWT Saves the Day Again

- Recall: searching for **ana** in panamabananas

In the end, we have five 3-mers with at most 1 mismatch.



Mismatches

1
1
0
0
0

Method 2: BWT Saves the Day Again

- Recall: searching for **ana** in panamab**ana**nas

In the end, we have five 3-mers with at most 1 mismatch.

	Suffix Array
$_1$panamabananas$_1$	
a$_1$bananas$panam$_1$	5
a$_2$mabananaspan_1$	3
a$_3$namabananas$p$_1$	1
a$_4$nanas$panamab$_1$	**7**
a$_5$nas$panamaban$_2$	9
a$_6$s$panamabanan$_3$	
b$_1$ananas$panama$_1$	
m$_1$abananas$pana$_2$	
n$_1$amabananaspa_3$	
n$_2$anas$panamaba$_4$	
n$_3$as$panamaban$_5$	
p$_1$anamabananas$$_1$	
s$_1$$panamabanana$_6$	

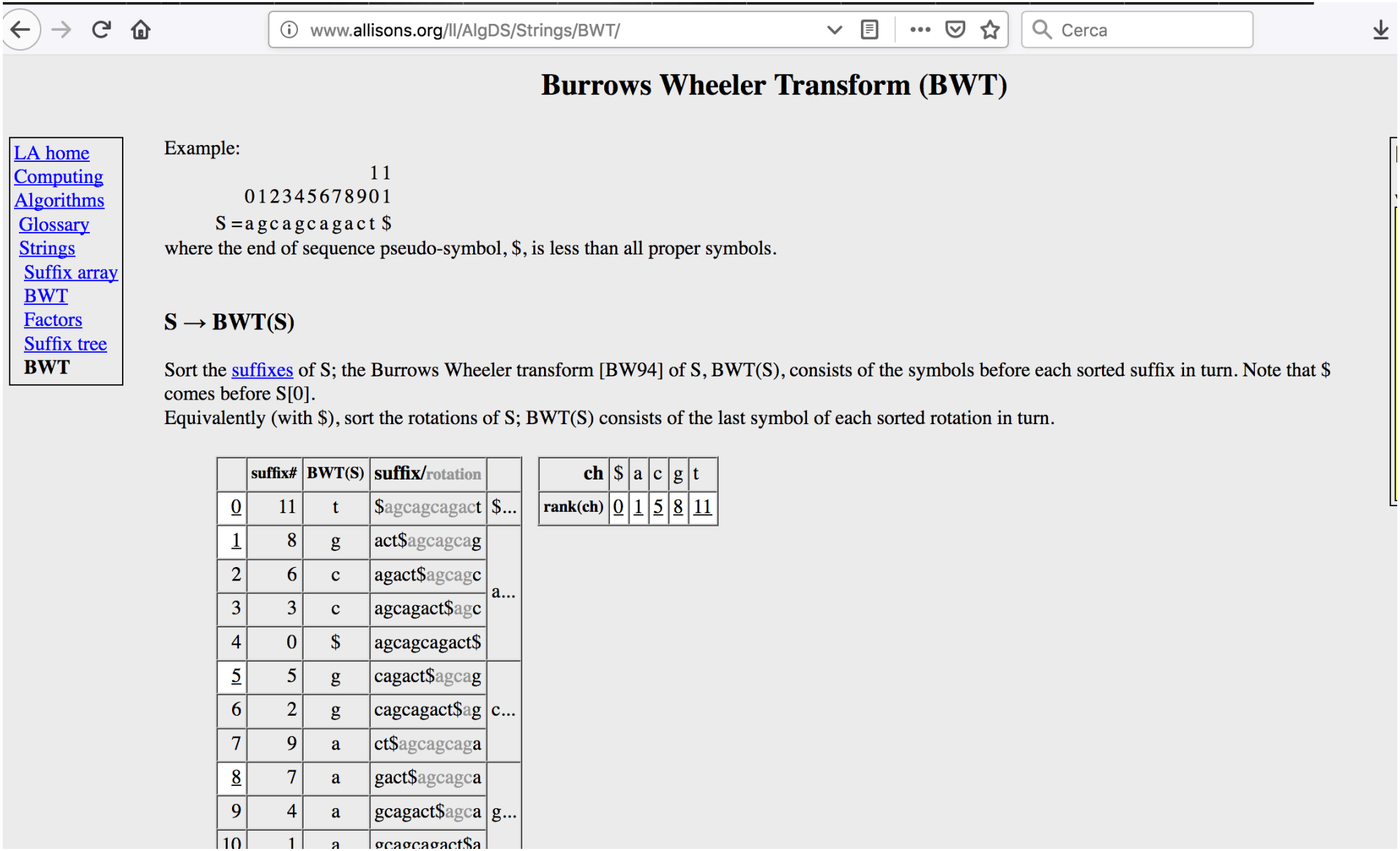
Method 2: BWT Saves the Day Again

- Recall: searching for **ana** in panamaban**ana**s

In the end, we have five 3-mers with at most 1 mismatch.

	Suffix Array
$_1$panamabananas$_1$	
a$_1$bananas$panam$_1$	5
a$_2$mabananaspan_1$	3
a$_3$namabananas$p$_1$	1
a$_4$nanas$panamab$_1$	7
a$_5$nas$panamaban$_2$	**9**
a$_6$s$panamabanan$_3$	
b$_1$ananas$panama$_1$	
m$_1$abananas$pana$_2$	
n$_1$amabananaspa_3$	
n$_2$anas$panamaba$_4$	
n$_3$as$panamaban$a$_5$	
p$_1$anamabananas$$_1$	
s$_1$$panamabanana$_6$	

http://www.allisons.org/ll/AlgDS/Strings/BWT/



Burrows Wheeler Transform (BWT)

LA home
Computing
Algorithms
Glossary
Strings
Suffix array
BWT
Factors
Suffix tree
BWT

Example:

```
                  11
        012345678901
     S =agcagcagact$
```

where the end of sequence pseudo-symbol, $, is less than all proper symbols.

S → BWT(S)

Sort the suffixes of S; the Burrows Wheeler transform [BW94] of S, BWT(S), consists of the symbols before each sorted suffix in turn. Note that $ comes before S[0].
Equivalently (with $), sort the rotations of S; BWT(S) consists of the last symbol of each sorted rotation in turn.

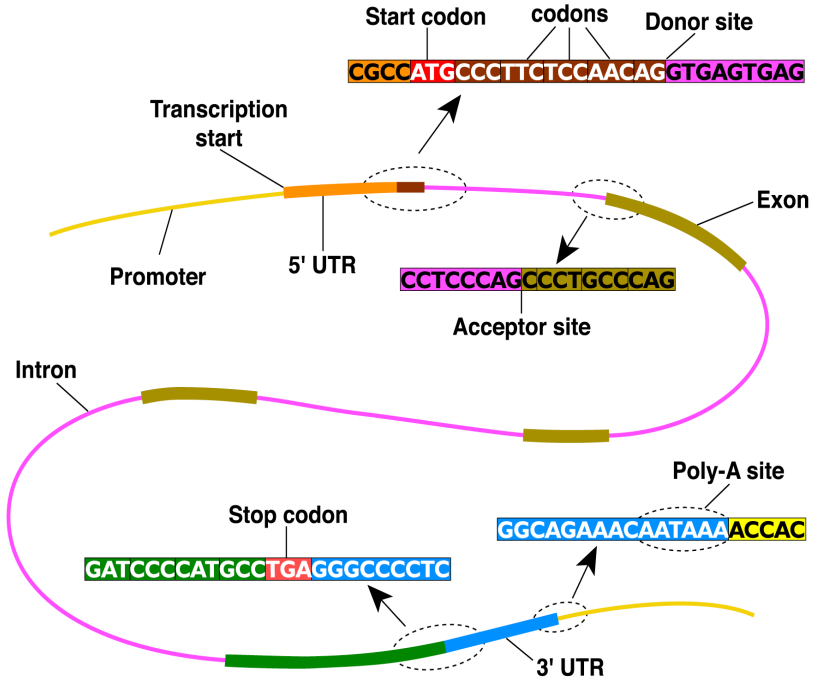
	suffix#	BWT(S)	suffix/rotation	
0	11	t	$agcagcagact	$...
1	8	g	act$agcagcag	a...
2	6	c	agact$agcagc	
3	3	c	agcagact$agc	
4	0	$	agcagcagact$	
5	5	g	cagact$agcag	c...
6	2	g	cagcagact$ag	
7	9	a	ct$agcagcaga	
8	7	a	gact$agcagca	g...
9	4	a	gcagact$agca	
10	1	a	gcagcagact$a	

ch	$	a	c	g	t
rank(ch)	0	1	5	8	11

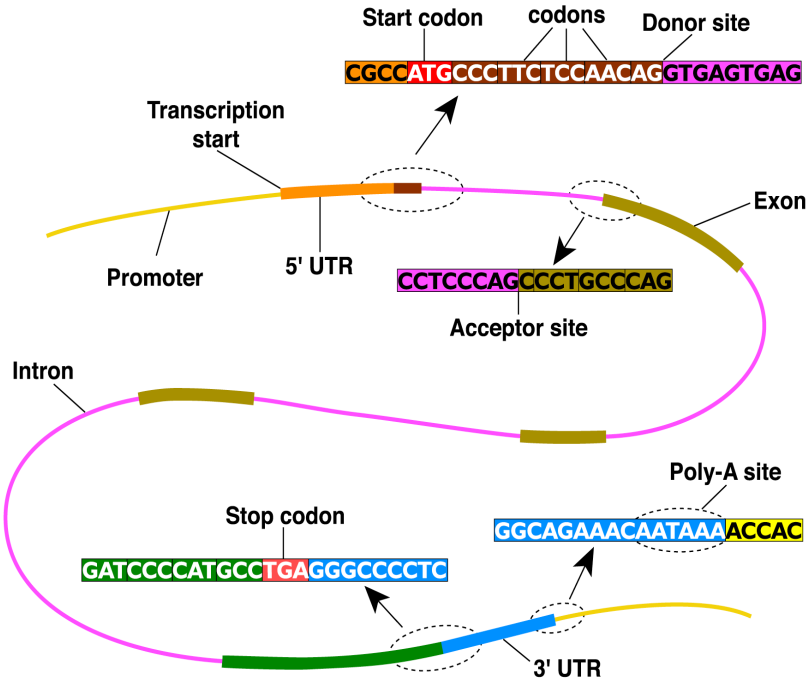
Hidden Markov models

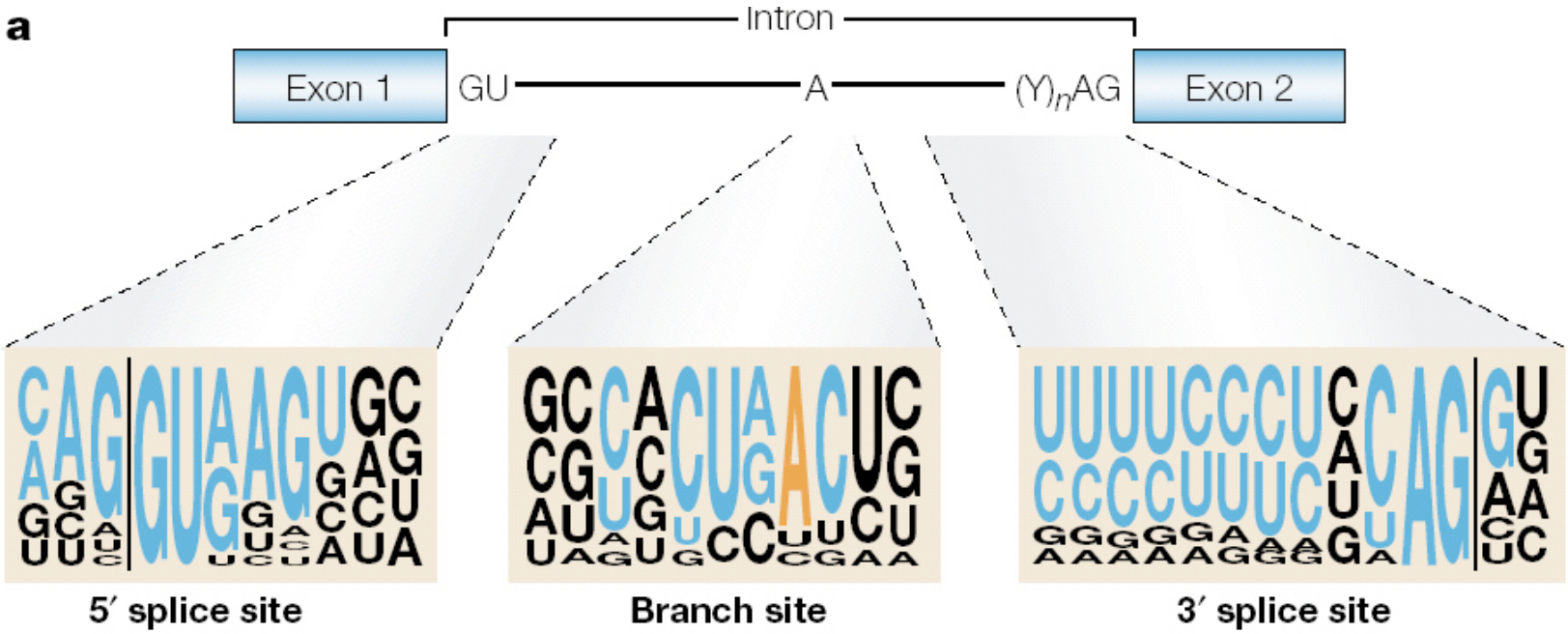
How to identify Genes and gene parts?

The gene information starts with the promoter, which is followed by a transcribed (i.e. RNA) but non-coding (i.e. not translated) region called 5' untranslated region (5' UTR). The initial exon contains the start codon which is usually ATG. There is an alternating series of introns and exons, followed by the terminating exon, which contains the stop codon. It is followed by another non-coding region called the 3' UTR; at the end there is a polyadenylation (polyA) signal, i.e. a repetition of the amino acid adenine. The intron/exon and exon/intron boundaries are conserved short sequences and called the acceptor and donor sites. For all these different parts we need to know their probability of occurrence in a large database.

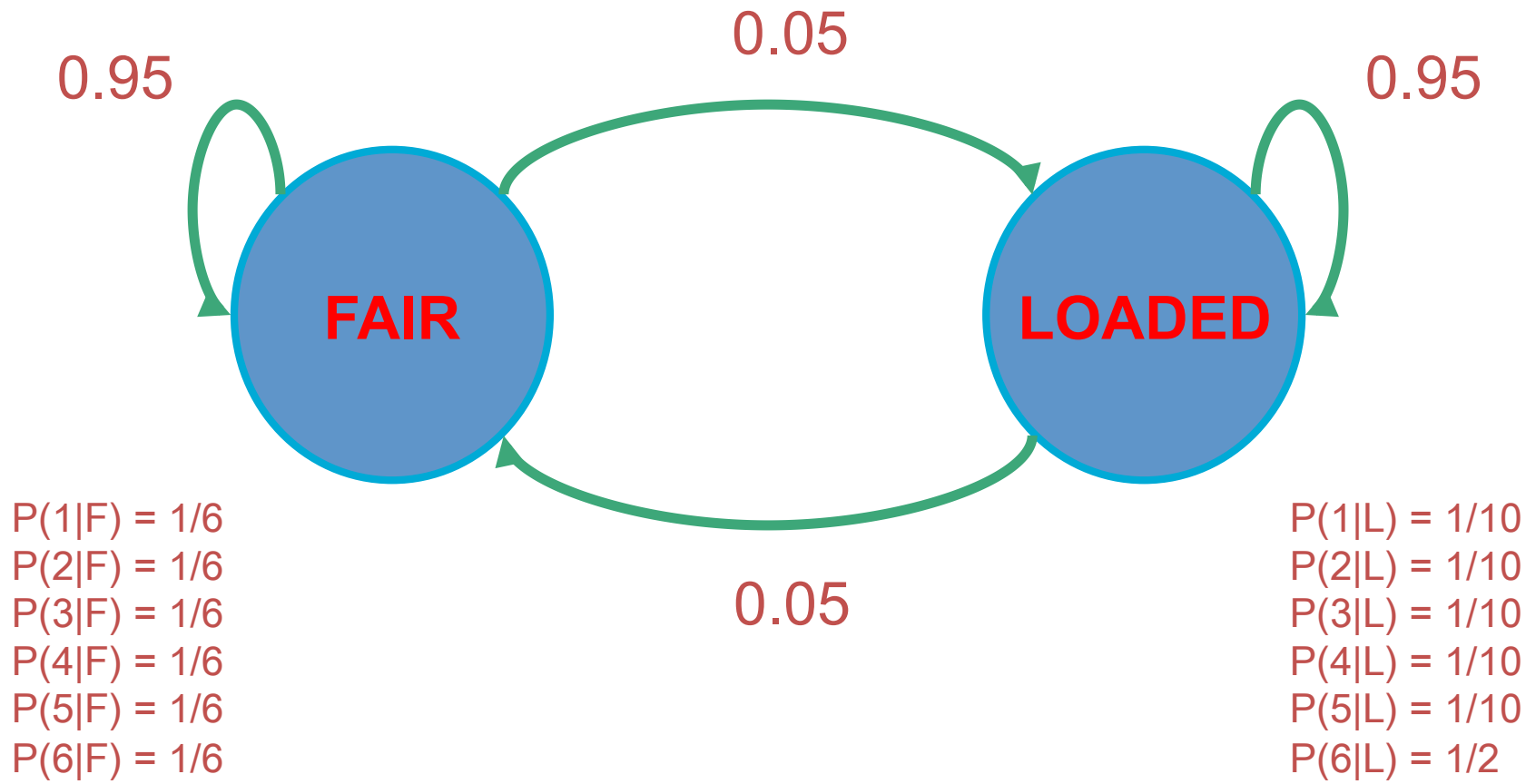


Splice Sites





The dishonest casino model



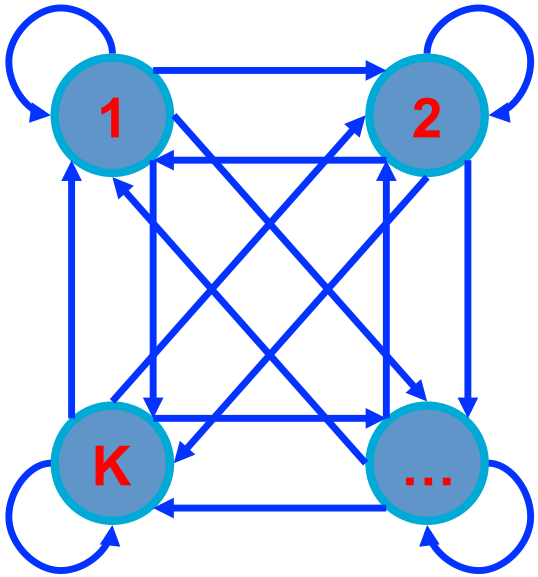
HMM

Definition: A hidden Markov model (HMM)

- Alphabet $\Sigma = \{ b_1, b_2, \ldots, b_M \}$
- Set of states $Q = \{ 1, \ldots, K \}$
- Transition probabilities between any two states

a_{ij} = transition prob from state i to state j

$a_{i1} + \ldots + a_{iK} = 1$, for all states i = 1…K



- Start probabilities a_{0i}

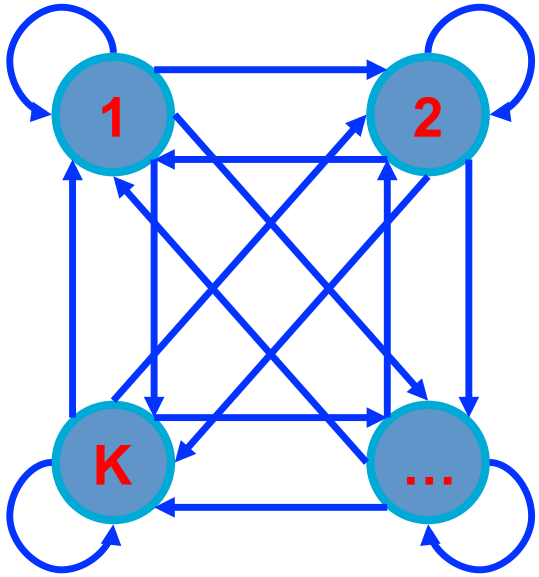
$a_{01} + \ldots + a_{0K} = 1$

- Emission probabilities within each state

$e_i(b) = P(x_i = b \mid \pi_i = k)$

$e_i(b_1) + \ldots + e_i(b_M) = 1$, for all states i = 1…K

A Hidden Markov Model is memory-less



At each time step t,
the only thing that affects future states
is the current state π_t

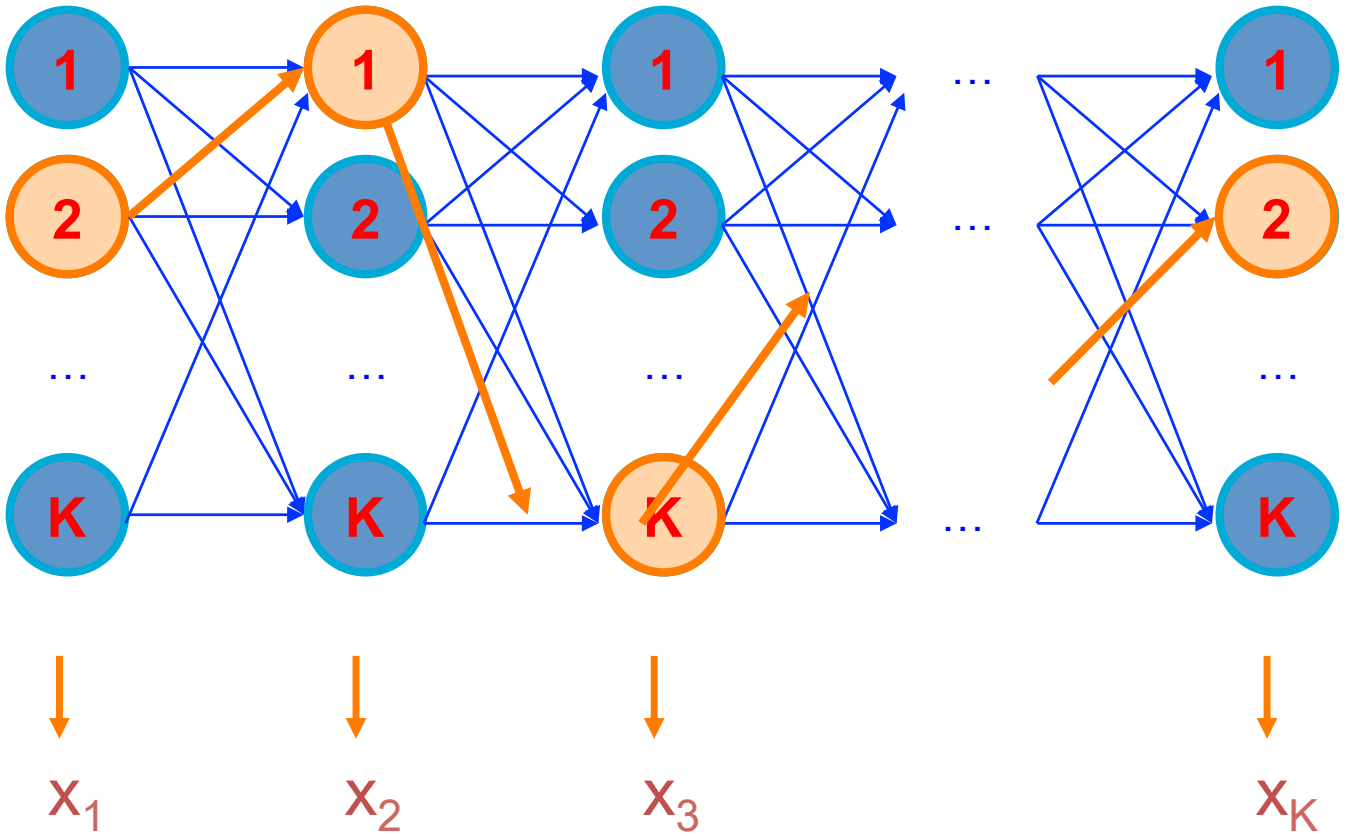
$P(\pi_{t+1} = k \mid$ "whatever happened so far"$) =$

$P(\pi_{t+1} = k \mid \pi_1, \pi_2, \ldots, \pi_t, x_1, x_2, \ldots, x_t) =$

$P(\pi_{t+1} = k \mid \pi_t)$

A parse of a sequence

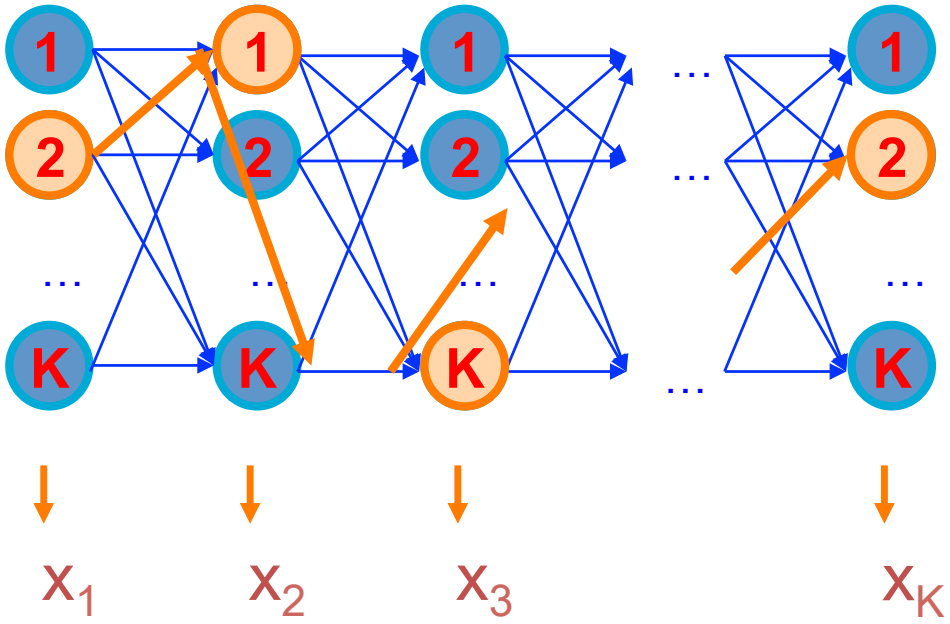
Given a sequence $x = x_1 \ldots\ldots x_N$,
A parse of x is a sequence of states $\pi = \pi_1, \ldots\ldots, \pi_N$



Likelihood of a parse

Given a sequence $x = x_1 \ldots\ldots x_N$

and a parse $\pi = \pi_1, \ldots\ldots, \pi_N$,



To find how likely is the parse:

(given our HMM)

$P(x, \pi) = P(x_1, \ldots, x_N, \pi_1, \ldots\ldots, \pi_N) =$

$P(x_N, \pi_N \mid \pi_{N-1})\ P(x_{N-1}, \pi_{N-1} \mid \pi_{N-2}) \ldots\ldots P(x_2, \pi_2 \mid \pi_1)$

$P(x_1, \pi_1) =$

$P(x_N \mid \pi_N)\ P(\pi_N \mid \pi_{N-1})\ \ldots\ldots P(x_2 \mid \pi_2)\ P(\pi_2 \mid \pi_1)\ P(x_1 \mid \pi_1)\ P(\pi_1) =$

$a_{0\pi 1}\ a_{\pi 1 \pi 2} \ldots\ldots a_{\pi N-1 \pi N}\ e_{\pi 1}(x_1) \ldots\ldots e_{\pi N}(x_N)$

Example: the dishonest casino

Let the sequence of rolls be:

x = 1, 2, 1, 5, 6, 2, 1, 6, 2, 4

Then, what is the likelihood of

π = Fair, Fair, Fair, Fair, Fair, Fair, Fair, Fair, Fair, Fair?

(say initial probs $a_{0Fair} = ½$, $a_{oLoaded} = ½$)

½ × P(1 | Fair) P(Fair | Fair) P(2 | Fair) P(Fair | Fair) … P(4 | Fair) =

$½ \times (1/6)^{10} \times (0.95)^{9} = .00000005211586472 11 = 0.5 \times 10^{-9}$

Example: the dishonest casino

So, the likelihood the die is fair in all this run

is just 0.521×10^{-9}

OK, but what is the likelihood of

= Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded?

$\frac{1}{2} \times P(1 \mid \text{Loaded})\, P(\text{Loaded}, \text{Loaded}) \ldots P(4 \mid \text{Loaded}) =$

$\frac{1}{2} \times (1/10)^8 \times (1/2)^2\, (0.95)^9 = .000000000781176215 = 7.9 \times 10^{-10}$

Therefore, it is after all 6.59 times more likely that the die is fair all the way, than that it is loaded all the way.

Example: the dishonest casino

Let the sequence of rolls be:

x = 1, 6, 6, 5, 6, 2, 6, 6, 3, 6

Now, what is the likelihood π = F, F, …, F?

$\frac{1}{2} \times (1/6)^{10} \times (0.95)^{9} = 0.5 \times 10^{-9}$, same as before

What is the likelihood

π = L, L, …, L?

$\frac{1}{2} \times (1/10)^{4} \times (1/2)^{6} (0.95)^{9} = .00000049238235134735 = 0.5 \times 10^{-7}$

So, it is 100 times more likely the die is loaded

The three main questions on HMMs

1. Evaluation

   GIVEN a HMM M, and a sequence x,

   FIND Prob[x | M]

2. Decoding

   GIVEN a HMM M, and a sequence x,

   FIND the sequence π of states that maximizes P[x, π | M]

3. Learning

   GIVEN a HMM M, with unspecified transition/emission probs., and a sequence x,

   FIND parameters $\theta = (e_i(.), a_{ij})$ that maximize P[x | θ]

Let's not be confused by notation

P[x | M]: The probability that sequence x was generated by the model

The model is: architecture (#states, etc)
+ parameters $\theta = a_{ij}, e_i(.)$

So, P[x | θ], and P[x] are the same, when the architecture, and the entire model, respectively, are implied

Similarly, P[x, π | M] and P[x, π] are the same

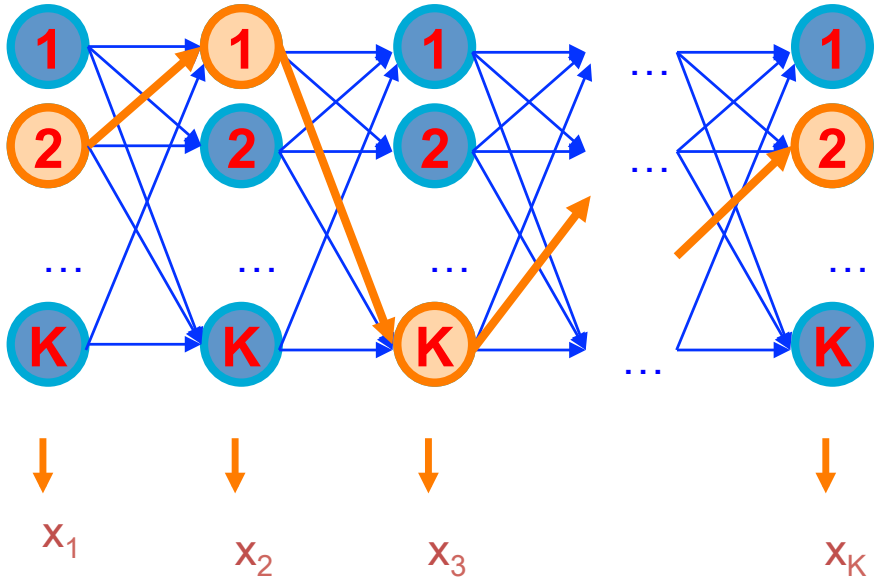
In the LEARNING problem we always write P[x | θ] to emphasize that we are seeking the θ that maximizes P[x | θ]

Decoding

GIVEN $x = x_1 x_2 \ldots\ldots x_N$

We want to find $\pi = \pi_1, \ldots\ldots, \pi_N$, such that $P[\, x, \pi \,]$ is maximized

$\pi^* = \text{argmax}_\pi \, P[\, x, \pi \,]$



We can use dynamic programming!

Let $V_k(i) = \max_{\{\pi 1,\ldots,i-1\}} P[x_1 \ldots x_{i-1}, \pi_1, \ldots, \pi_{i-1}, x_i, \pi_i = k]$

= Probability of most likely sequence of states ending at state $\pi_i = k$

Decoding – main idea

Given that for all states k, and for a fixed position i,

$V_k(i) = \max_{\{\pi 1,\ldots,i-1\}} P[x_1 \ldots x_{i-1}, \pi_1, \ldots, \pi_{i-1}, x_i, \pi_i = k]$

What is $V_k(i+1)$?

From definition,

$V_l(i+1) = \max_{\{\pi 1,\ldots,i\}} P[\, x_1 \ldots x_i, \pi_1, \ldots, \pi_i, x_{i+1}, \pi_{i+1} = l \,]$

$= \max_{\{\pi 1,\ldots,i\}} P(x_{i+1}, \pi_{i+1} = l \mid x_1 \ldots x_i, \pi_1, \ldots, \pi_i)\, P[x_1 \ldots x_i, \pi_1, \ldots, \pi_i]$

$= \max_{\{\pi 1,\ldots,i\}} P(x_{i+1}, \pi_{i+1} = l \mid \pi_i)\, P[x_1 \ldots x_{i-1}, \pi_1, \ldots, \pi_{i-1}, x_i, \pi_i]$

$= \max_k P(x_{i+1}, \pi_{i+1} = l \mid \pi_i = k) \max_{\{\pi 1,\ldots,i-1\}} P[x_1 \ldots x_{i-1}, \pi_1, \ldots, \pi_{i-1}, x_i, \pi_i = k] = e_l(x_{i+1}) \max_k a_{kl}\, V_k(i)$

The Viterbi Algorithm

Andrew Viterbi

Qualcomm

Input: $x = x_1 \ldots\ldots x_N$

Initialization:

$V_0(0) = 1$ (0 is the imaginary first position)

$V_k(0) = 0$, for all $k > 0$

Iteration:

$V_j(i) = e_j(x_i) \times \max_k a_{kj} V_k(i-1)$

$Ptr_j(i) = \text{argmax}_k a_{kj} V_k(i-1)$

Termination:

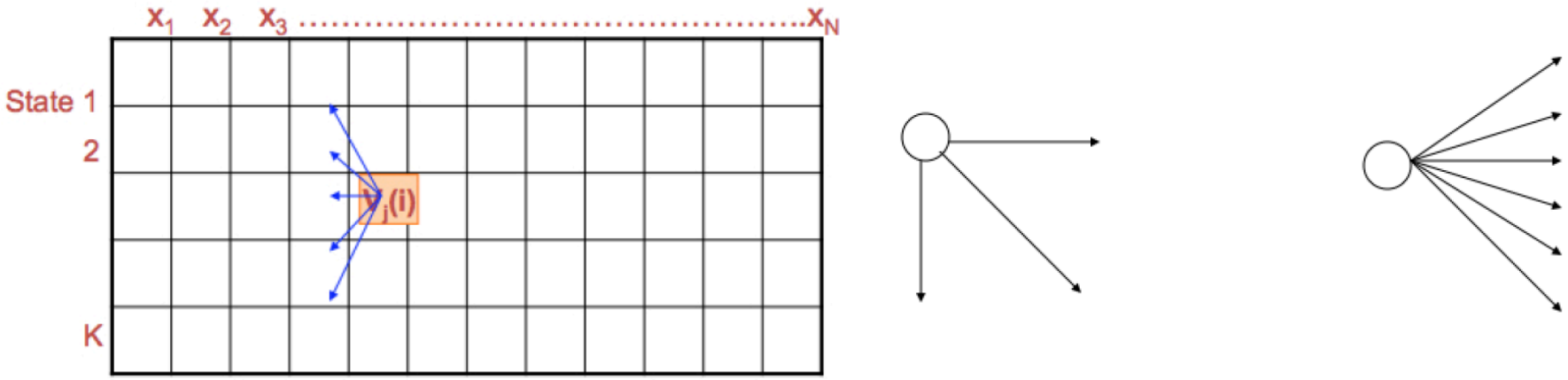
$P(x, \pi^*) = \max_k V_k(N)$

Traceback:

$\pi_N^* = \text{argmax}_k V_k(N)$

$\pi_{i-1}^* = Ptr_{\pi i}(i)$

The Viterbi Algorithm



left: Similar to "aligning" a set of states to a sequence,

Time: $O(K^2N)$; **Space:** $O(KN)$; right : comparison of valid directions in the alignment and decoding problem.

Viterbi Algorithm – a practical detail

Underflows are a significant problem

$$P[x_1,\ldots., x_i, \pi_1, \ldots, \pi_i] = a_{0\pi1} a_{\pi1\pi2}\ldots\ldots a_{\pi i} e_{\pi1}(x_1)\ldots\ldots e_{\pi i}(x_i)$$

These numbers become extremely small – underflow

Solution: Take the logs of all values

$$V_l(i) = \log e_k(x_i) + \max_k [V_k(i-1) + \log a_{kl}]$$

Example

Let x be a sequence with a portion of ~ 1/6 6's, followed by a portion of ~ ½ 6's…

x = 123456123456…12345 6626364656…1626364656

Then, it is not hard to show that optimal parse is (exercise):

FFF……F LLL……L

6 nucleotides "123456" parsed as F, contribute $.95^6 \times (1/6)^6 = 1.6 \times 10^{-5}$

parsed as L, contribute $.95^6 \times (1/2)^1 \times (1/10)^5 = 0.4 \times 10^{-5}$

"162636" parsed as F, contribute $.95^6 \times (1/6)^6 = 1.6 \times 10^{-5}$

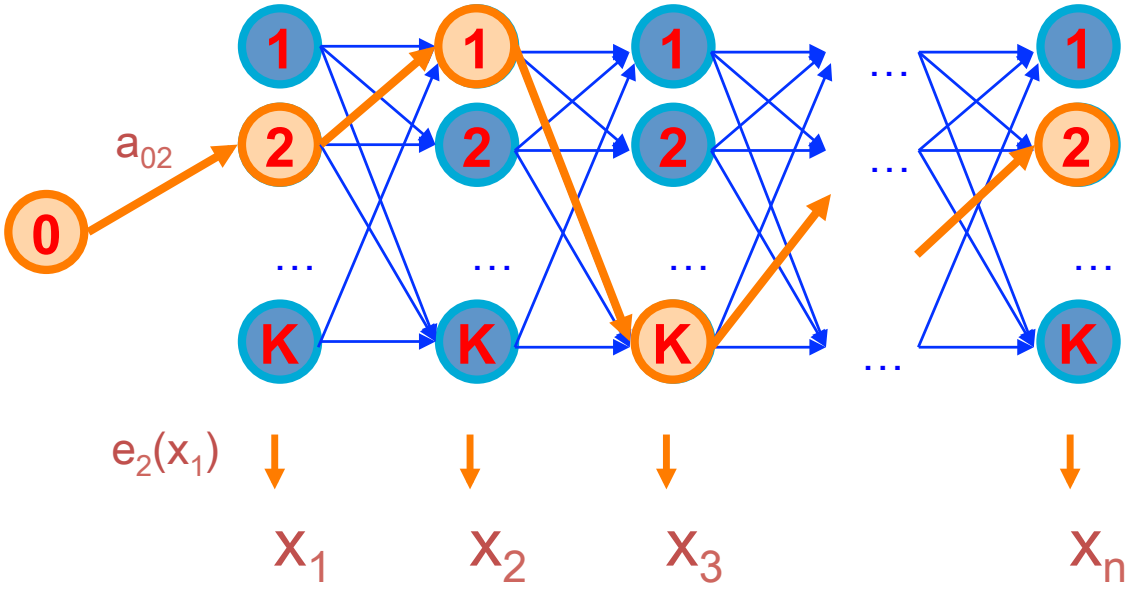
parsed as L, contribute $.95^6 \times (1/2)^3 \times (1/10)^3 = 9.0 \times 10^{-5}$

Generating a sequence by the model

Given a HMM, we can generate a sequence of length n as follows:

Start at state π_1 according to prob $a_{0\pi 1}$

1. Emit letter x_1 according to prob $e_{\pi 1}(x_1)$
2. Go to state π_2 according to prob $a_{\pi 1 \pi 2}$
3. … until emitting x_n



A couple of questions

Given a sequence x,

- What is the probability that x was generated by the model?
- Given a position i, what is the most likely state that emitted x_i?

Example: the dishonest casino

Say x = 12341623162616364616234161221341

Most likely path: π = FF……F

However: marked letters more likely to be L than unmarked letters

Evaluation

We will develop algorithms that allow us to compute:

$P(x)$ Probability of x given the model

$P(x_i \ldots x_j)$ Probability of a substring of x given the model

$P(\pi_i = k \mid x)$ Probability that the i^{th} state is k, given x

A more refined measure of which states x may be in

The Forward Algorithm

We want to calculate

P(x) = probability of x, given the HMM

Sum over all possible ways of generating x:

$$P(x) = \sum_{\pi} P(x, \pi) = \sum_{\pi} P(x \mid \pi)\, P(\pi)$$

To avoid summing over an exponential number of paths π, define

$$f_k(i) = P(x_1 \ldots x_i, \pi_i = k) \qquad \text{(the forward probability)}$$

The Forward Algorithm – derivation

Define the forward probability:

$$f_l(i) = P(x_1 \ldots x_i, \pi_i = l)$$

$$= \sum_{\pi 1 \ldots \pi i-1} P(x_1 \ldots x_{i-1}, \pi_1, \ldots, \pi_{i-1}, \pi_i = l)\, e_l(x_i)$$

$$= \sum_k \sum_{\pi 1 \ldots \pi i-2} P(x_1 \ldots x_{i-1}, \pi_1, \ldots, \pi_{i-2}, \pi_{i-1} = k)\, a_{kl}\, e_l(x_i)$$

$$= e_l(x_i) \sum_k f_k(i-1)\, a_{kl}$$

The Forward Algorithm

We can compute $f_k(i)$ for all k, i, using dynamic programming!

Initialization:

$f_0(0) = 1$

$f_k(0) = 0$, for all $k > 0$

Iteration:

$f_l(i) = e_l(x_i) \sum_k f_k(i-1)\, a_{kl}$

Termination:

$P(x) = \sum_k f_k(N)\, a_{k0}$

Where, a_{k0} is the probability that the terminating state is k (usually = a_{0k})

Relation between Forward and Viterbi

VITERBI

Initialization:

$V_0(0) = 1$

$V_k(0) = 0$, for all $k > 0$

Iteration:

$$V_j(i) = e_j(x_i) \ \max_k V_k(i-1)\, a_{kj}$$

Termination:

$$P(x, \pi^*) = \max_k V_k(N)$$

FORWARD

Initialization:

$f_0(0) = 1$

$f_k(0) = 0$, for all $k > 0$

Iteration:

$$f_l(i) = e_l(x_i) \sum_k f_k(i-1)\, a_{kl}$$

Termination:

$$P(x) = \sum_k f_k(N)\, a_{k0}$$

Motivation for the Backward Algorithm

We want to compute

$P(\pi_i = k \mid x)$,

the probability distribution on the i^{th} position, given x

We start by computing

$$P(\pi_i = k, x) = P(x_1 \ldots x_i, \pi_i = k, x_{i+1} \ldots x_N)$$

$$= P(x_1 \ldots x_i, \pi_i = k)\, P(x_{i+1} \ldots x_N \mid x_1 \ldots x_i, \pi_i = k)$$

$$= P(x_1 \ldots x_i, \pi_i = k)\, P(x_{i+1} \ldots x_N \mid \pi_i = k)$$

Forward, $f_k(i)$ Backward, $b_k(i)$

The Backward Algorithm – derivation

Define the backward probability:

$$b_k(i) = P(x_{i+1}\ldots x_N \mid \pi_i = k)$$

$$= \sum_{\pi i+1\ldots\pi N} P(x_{i+1}, x_{i+2}, \ldots, x_N, \pi_{i+1}, \ldots, \pi_N \mid \pi_i = k)$$

$$= \sum_l \sum_{\pi i+1\ldots\pi N} P(x_{i+1}, x_{i+2}, \ldots, x_N, \pi_{i+1} = l, \pi_{i+2}, \ldots, \pi_N \mid \pi_i = k)$$

$$= \sum_l e_l(x_{i+1})\, a_{kl} \sum_{\pi i+1\ldots\pi N} P(x_{i+2}, \ldots, x_N, \pi_{i+2}, \ldots, \pi_N \mid \pi_{i+1} = l)$$

$$= \sum_l e_l(x_{i+1})\, a_{kl}\, b_l(i+1)$$

The Backward Algorithm

We can compute $b_k(i)$ for all k, i, using dynamic programming

Initialization:

$b_k(N) = a_{k0}$, for all k

Iteration:

$$b_k(i) = \sum_l e_l(x_{i+1})\, a_{kl}\, b_l(i+1)$$

Termination:

$$P(x) = \sum_l a_{0l}\, e_l(x_1)\, b_l(1)$$

Computational Complexity

What is the running time, and space required, for Forward, and Backward?

Time: $O(K^2N)$

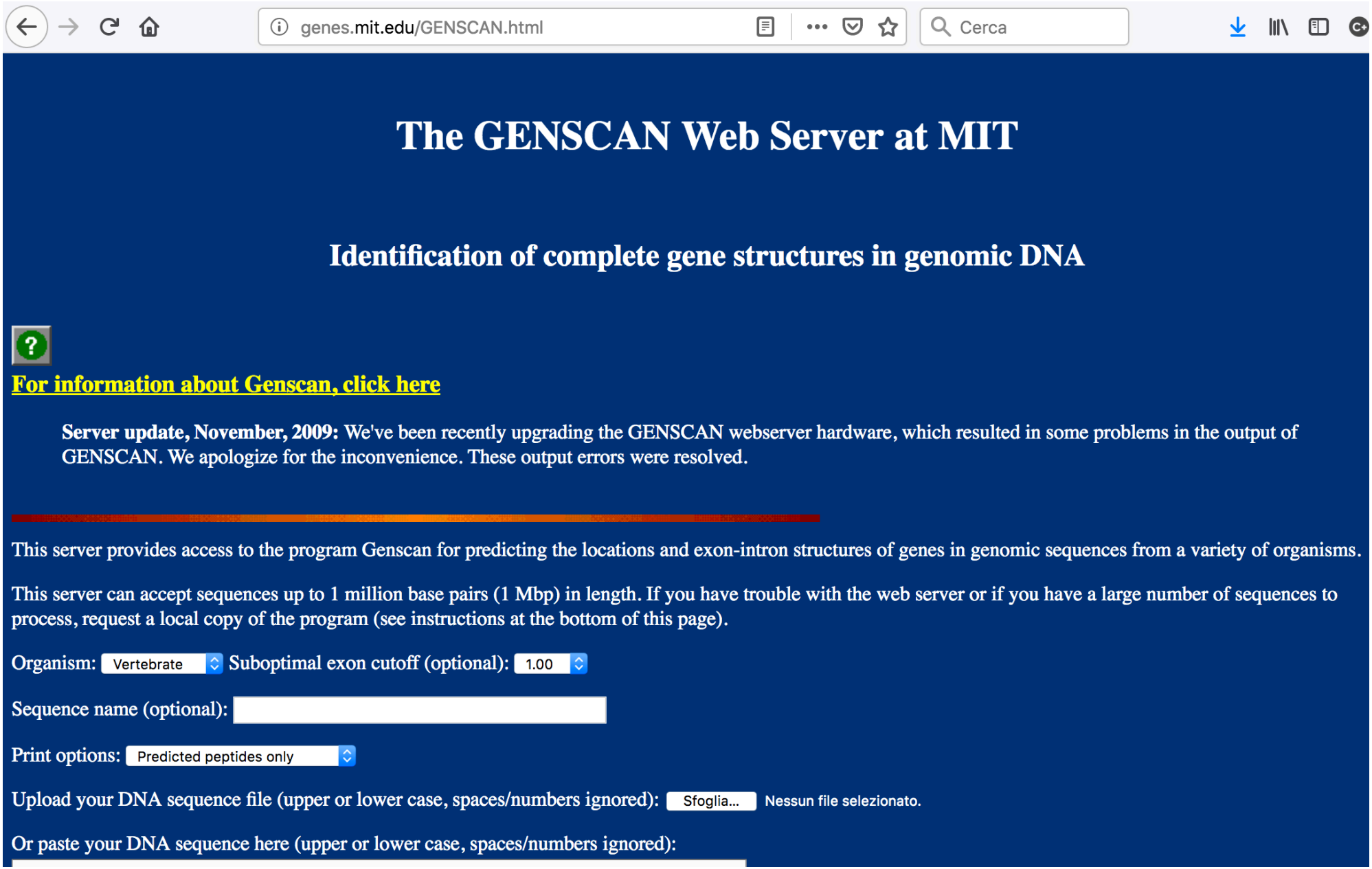
Space: $O(KN)$

Useful implementation technique to avoid underflows

Viterbi: sum of logs

Forward/Backward: rescaling at each position by multiplying by a constant

Genscan


genes.mit.edu/GENSCAN.html
Cerca
The GENSCAN Web Server at MIT
Identification of complete gene structures in genomic DNA
For information about Genscan, click here
Server update, November, 2009: We've been recently upgrading the GENSCAN webserver hardware, which resulted in some problems in the output of GENSCAN. We apologize for the inconvenience. These output errors were resolved.
This server provides access to the program Genscan for predicting the locations and exon-intron structures of genes in genomic sequences from a variety of organisms.
This server can accept sequences up to 1 million base pairs (1 Mbp) in length. If you have trouble with the web server or if you have a large number of sequences to process, request a local copy of the program (see instructions at the bottom of this page).
Organism: Vertebrate Suboptimal exon cutoff (optional): 1.00
Sequence name (optional):
Print options: Predicted peptides only
Upload your DNA sequence file (upper or lower case, spaces/numbers ignored): Sfoglia... Nessun file selezionato.
Or paste your DNA sequence here (upper or lower case, spaces/numbers ignored):

This server provides access to the program GenomeScan for predicting the locations and exon-intron structures of genes in genomic sequences from a variety of organisms.

GenomeScan incorporates protein homology information when predicting genes. This server allows you to input proteins suspected to be similar to regions of your DNA sequence. You can find such proteins by doing a BLASTX comparison of your sequence to all known proteins, or by running GENSCAN and then comparing the results to known proteins using BLASTP. Please input the proteins in FastA format; the file may contain multiple proteins so long as each is separated by a header on its own line. Files should contain less than one million bases.

If you would like to test the program, feel free to use this DNA testfile and the corresponding protein file.

More information on GenomeScan: GenomeScan homepage

You may also wish to use or read about the GENSCAN server, GenomeScan's predecessor.

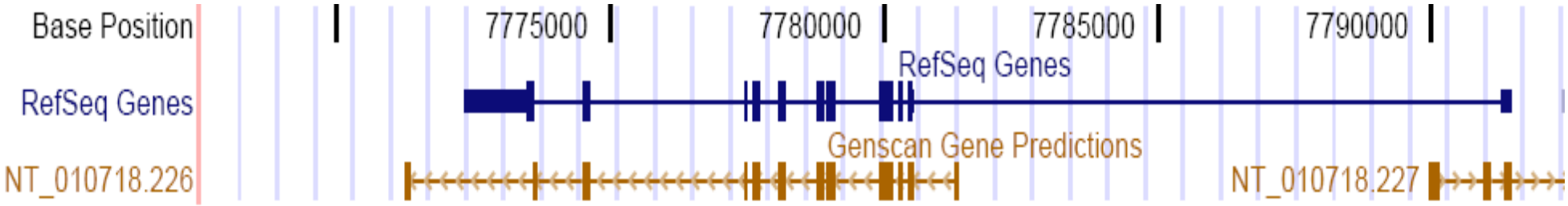
Run GenomeScan:

Organism: Vertebrate

Sequence name (optional):

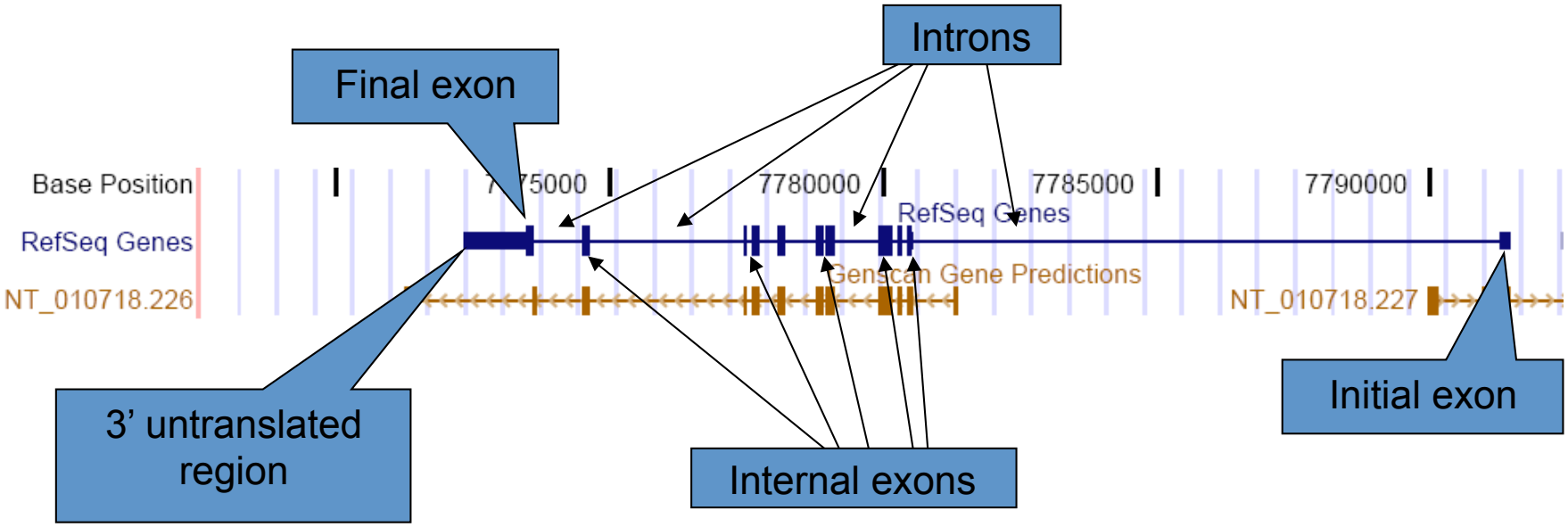
Print options: Predicted peptides only

A eukaryotic gene



- This is the human p53 tumor suppressor gene on chromosome 17.
- Genscan is one of the most popular gene prediction algorithms.

A eukaryotic gene



This particular gene lies on the reverse strand.

An Intron

revcomp(CT)=AG

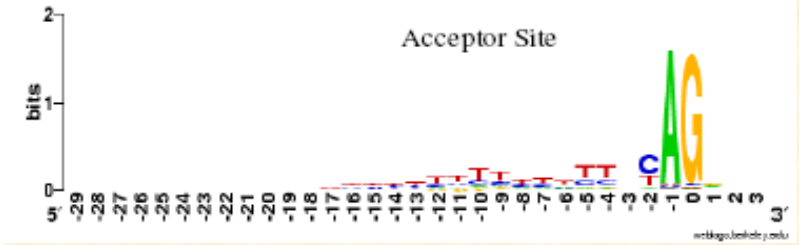
GT: signals start of intron

AG: signals end of intron

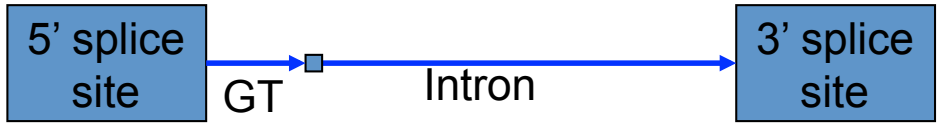
revcomp(AC)=GT

3’ splice site

5’ splice site

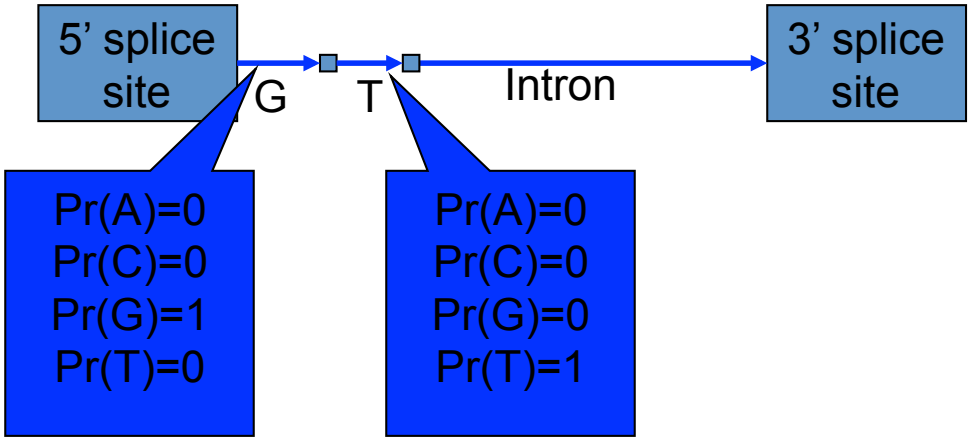


Modeling the 5’ splice site



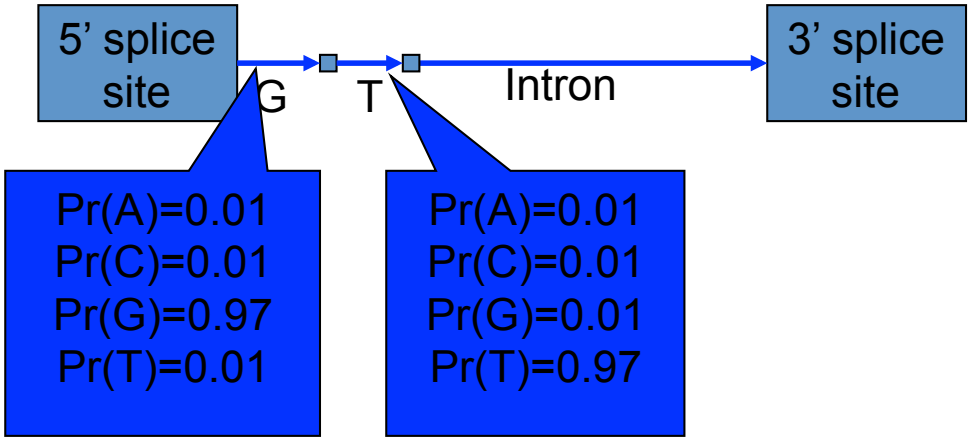
- Most introns begin with the letters “GT.”
- We can add this signal to the model.

Modeling the 5’ splice site



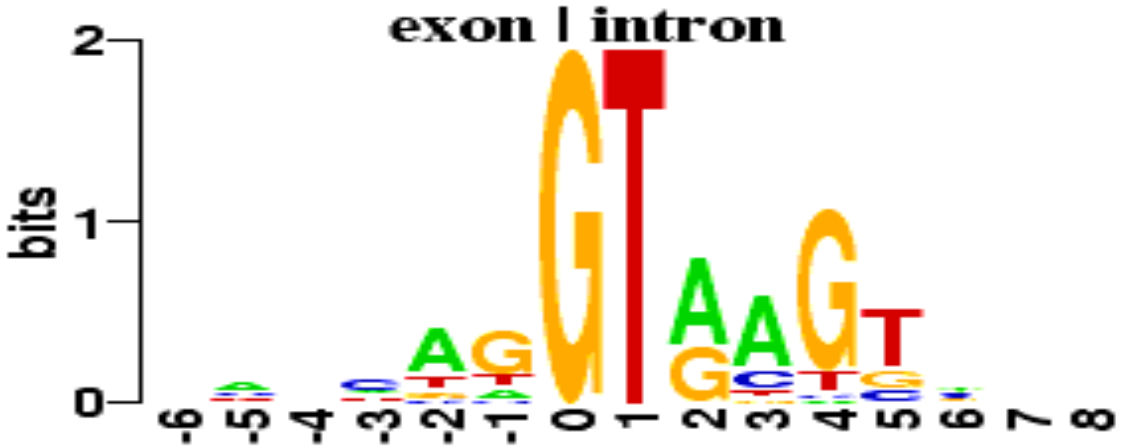
- Most introns begin with the letters “GT.”
- We can add this signal to the model.
- Indeed, we can model each nucleotide with its own arrow.

Modeling the 5’ splice site



- Like most biological phenomenon, the splice site signal admits exceptions.
- The resulting model of the 5’ splice site is a length-2 PSSM.

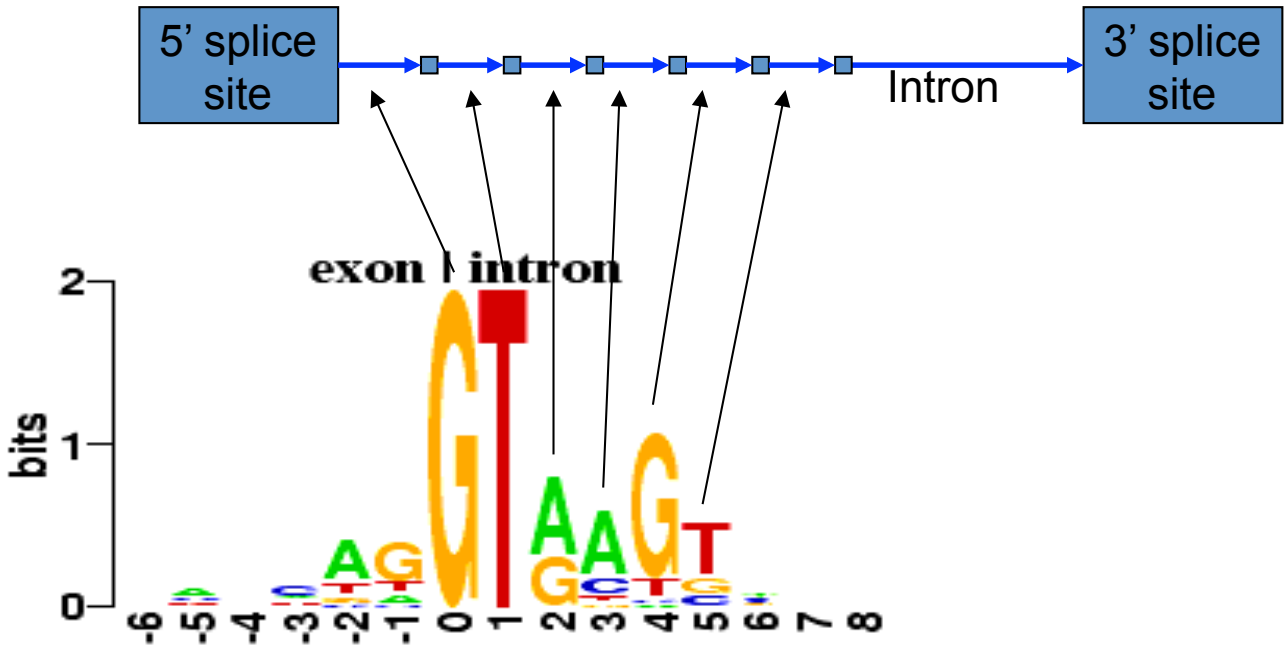
Real splice sites



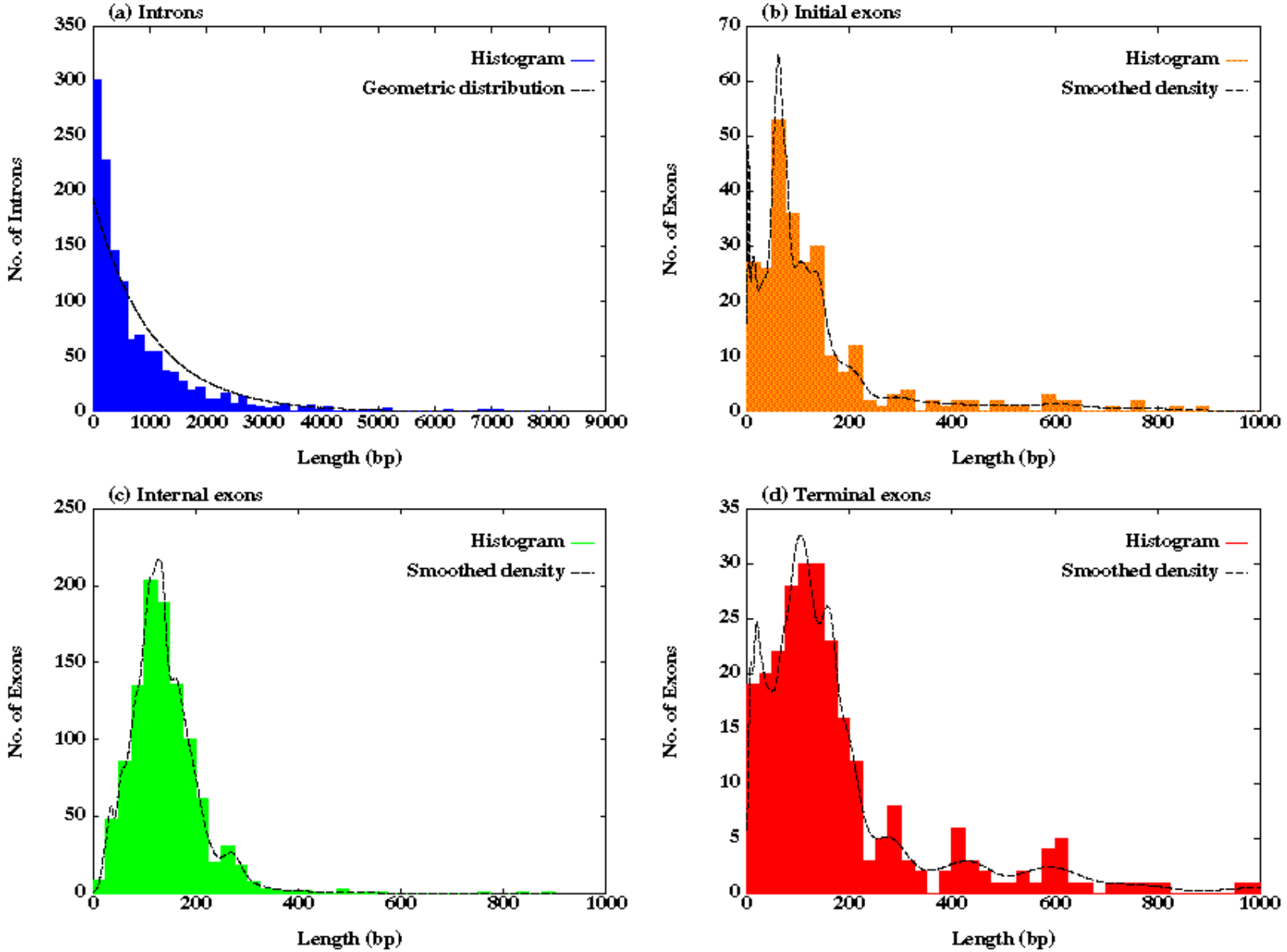
- Real splice sites show some conservation at positions beyond the first two.
- We can add additional arrows to model these states.

weblogo.berkeley.edu

Modeling the 5’ splice site

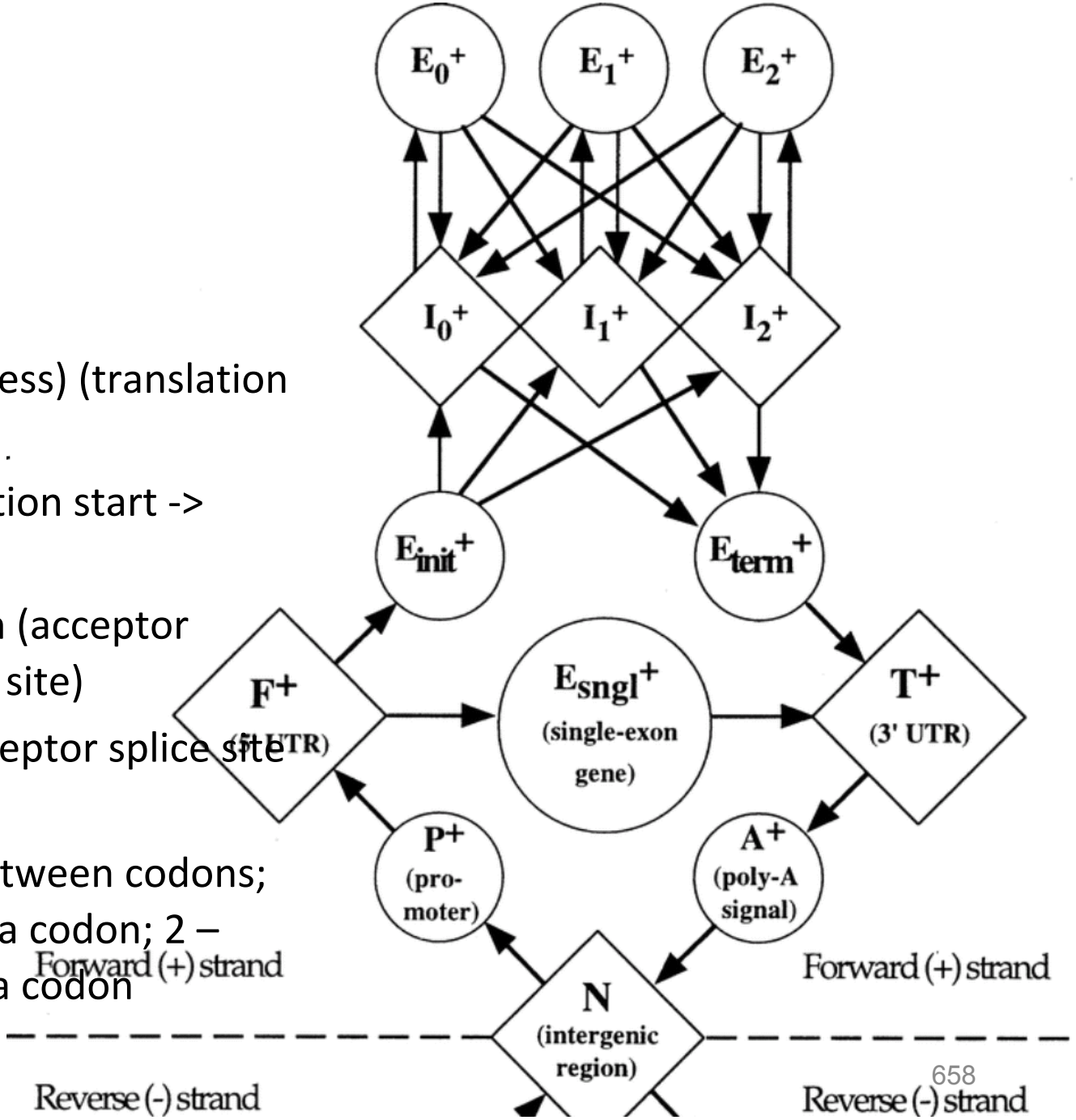


Length distributions of human introns and initial, internal and terminal exons

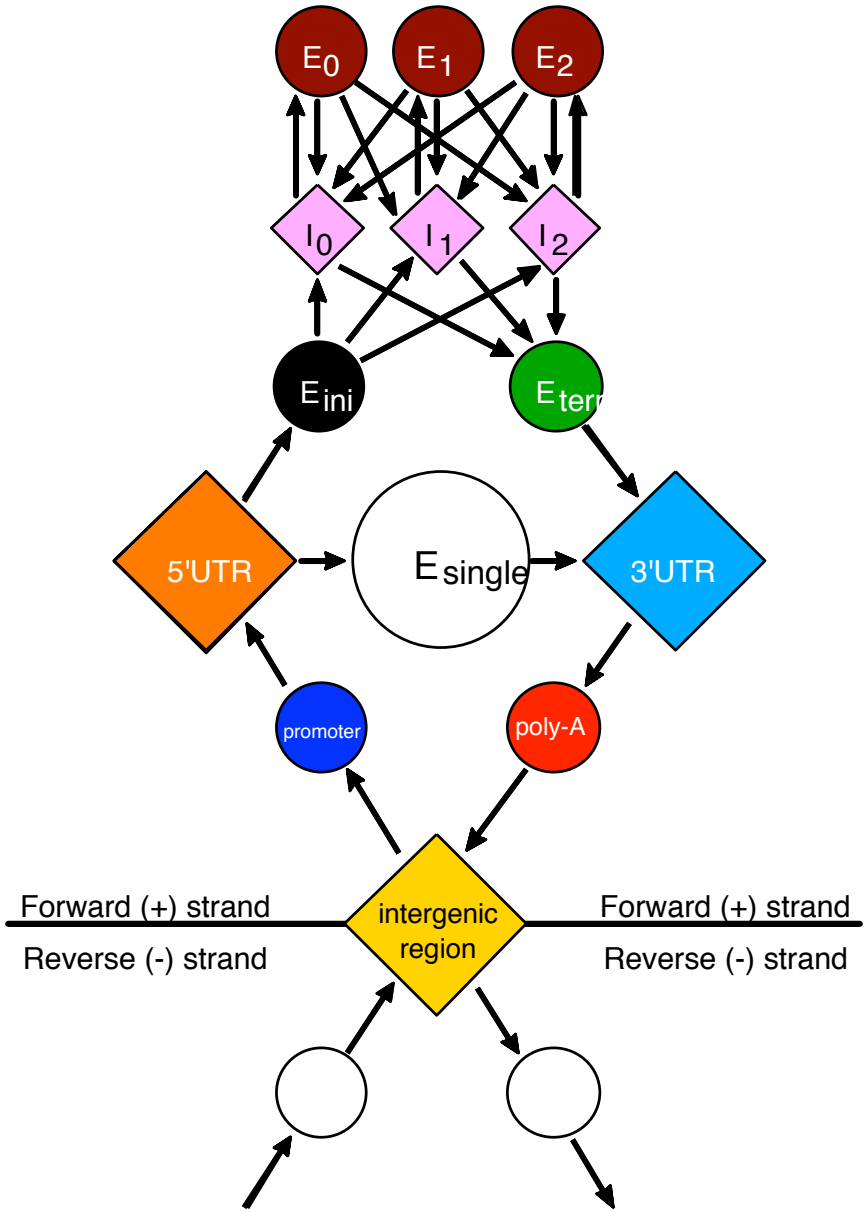


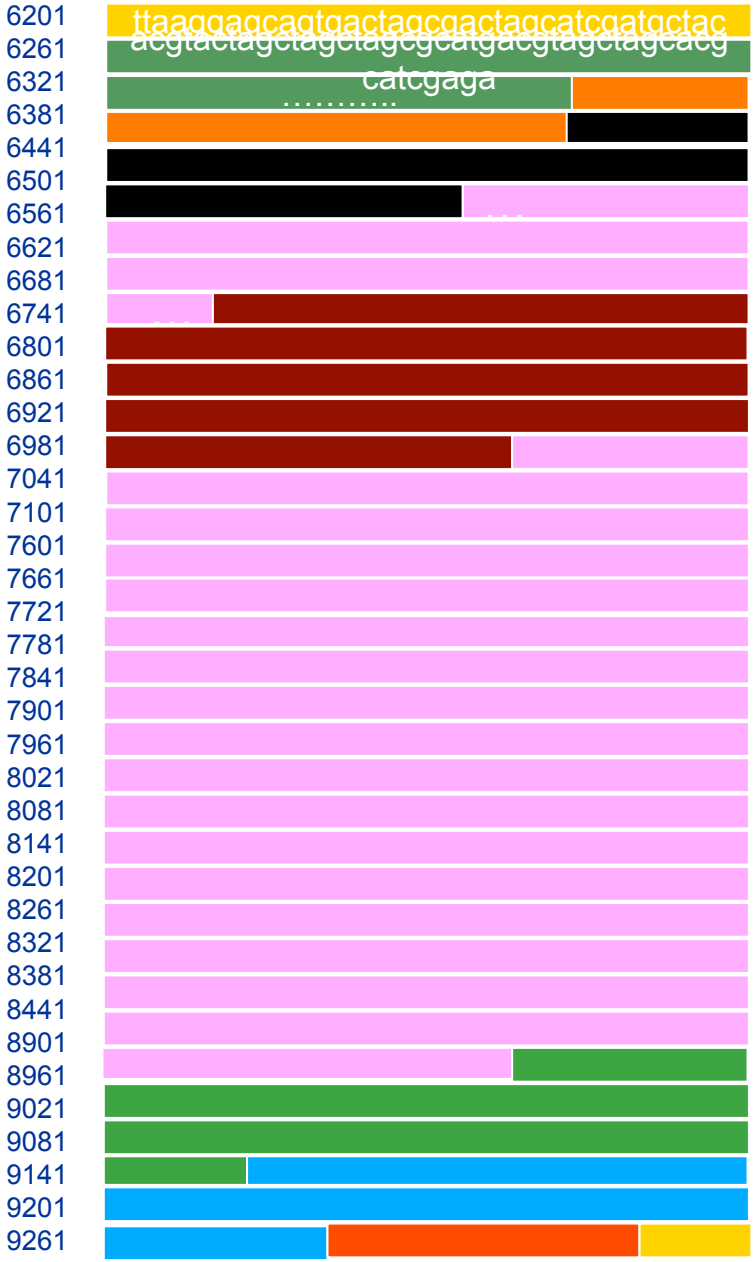
GenScan

- N - intergenic region
- P - promoter
- F - 5’ untranslated region
- E_{sngl} – single exon (intronless) (translation start -> stop codon)
- E_{init} – initial exon (translation start -> donor splice site)
- E_k – phase k internal exon (acceptor splice site -> donor splice site)
- E_{term} – terminal exon (acceptor splice site -> stop codon)
- I_k – phase k intron: 0 – between codons; 1 – after the first base of a codon; 2 – after the second base of a codon



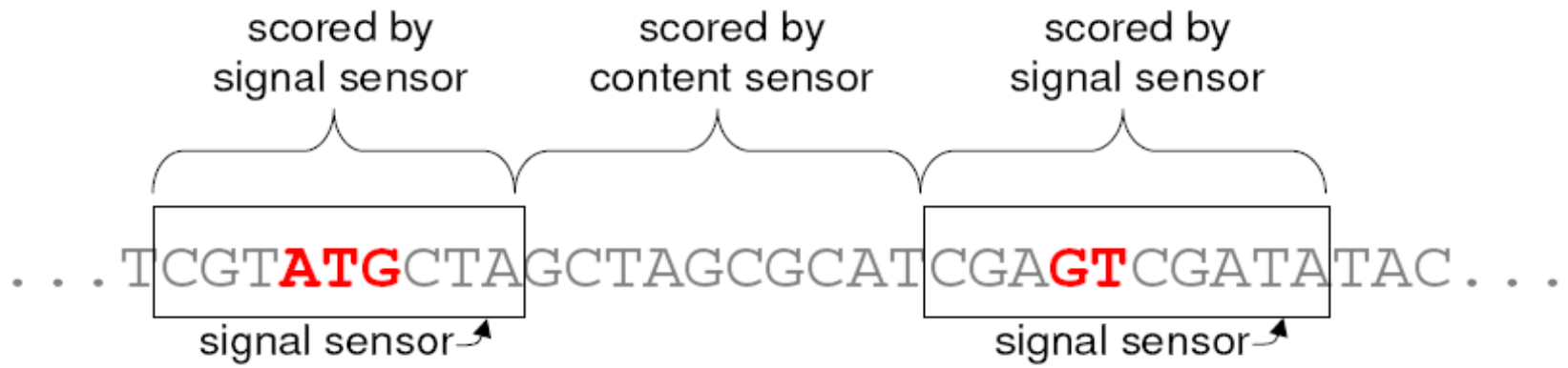
GENSCAN (Burge & Karlin)





Genscan model

- Duration of states – length distributions of
  - Exons (coding)
  - Introns (non coding)
- Signals at state transitions
  - ATG
  - Stop Codon TAG/TGA/TAA
  - Exon/Intron and Intron/Exon Splice Sites
- Emissions
  - Coding potential and frame at exons
  - Intron emissions



GenScan features

- Model both strands at once
- Each state may output a string of symbols (according to some probability distribution).
- Explicit intron/exon length modeling
- Advanced splice site modeling
- Complete intron/exon annotation for sequence
- Able to predict multiple genes and partial/whole genes
- Parameters learned from annotated genes
- Separate parameter training for different CpG content groups (< 43%, 43-51%, 51-57%,>57% CG content)

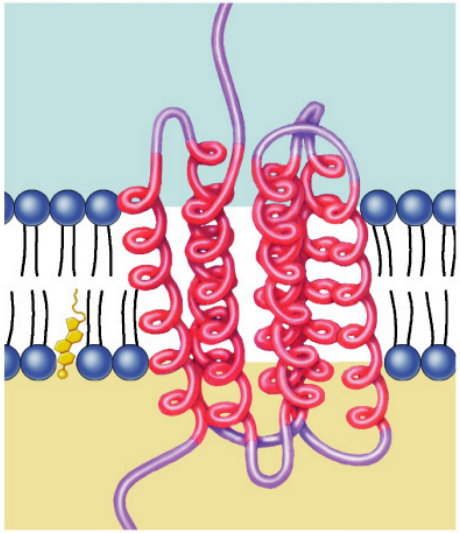
Performance

- > 80% correct exon predictions, and > 90% correct coding/non coding predictions by bp.
- BUT - the ability to predict the whole gene correctly is much lower

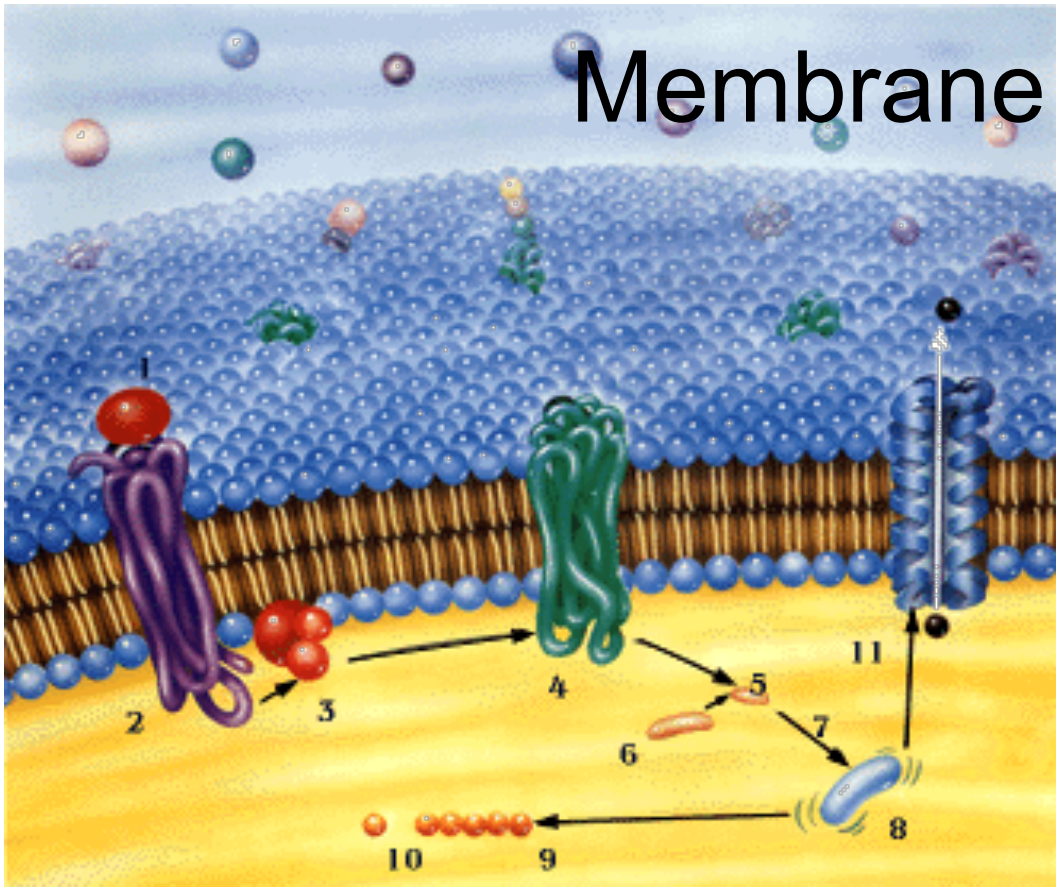
Hidden Markov models

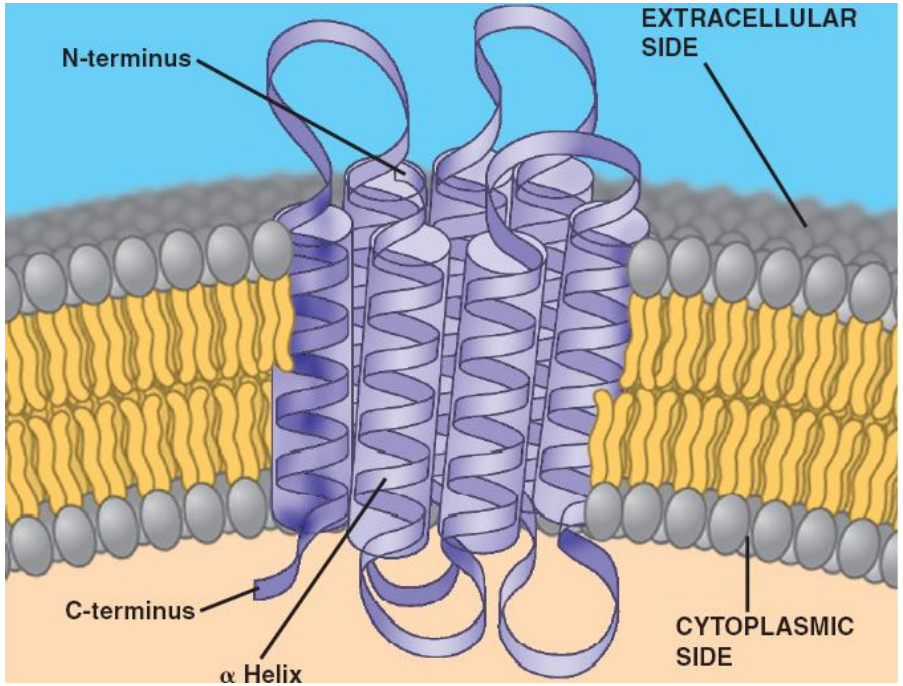
How to identify protein structural parts?

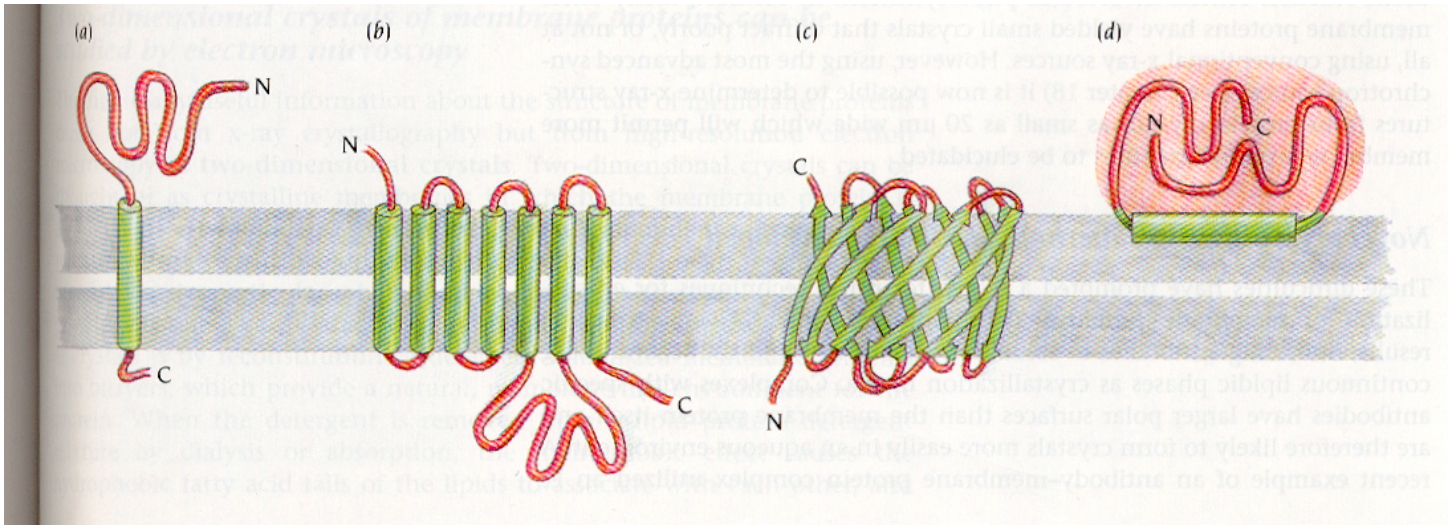
Membrane proteins that are important for ce
import/export. We would like to predict the
position in the amino acids with respect to th
membrane. The prediction of gene parts and
the membrane protein topology (i.e. which
parts are outside, inside and buried in the
membrane) will require to train the model wi
a dataset of experimentally determined
genes / transmembrane helices and to valida
the model with another dataset. The figure
below describes a 7 helix membrane protein
forming a sort of a cylinder (porus) across the
cell membrane



Membrane proteins

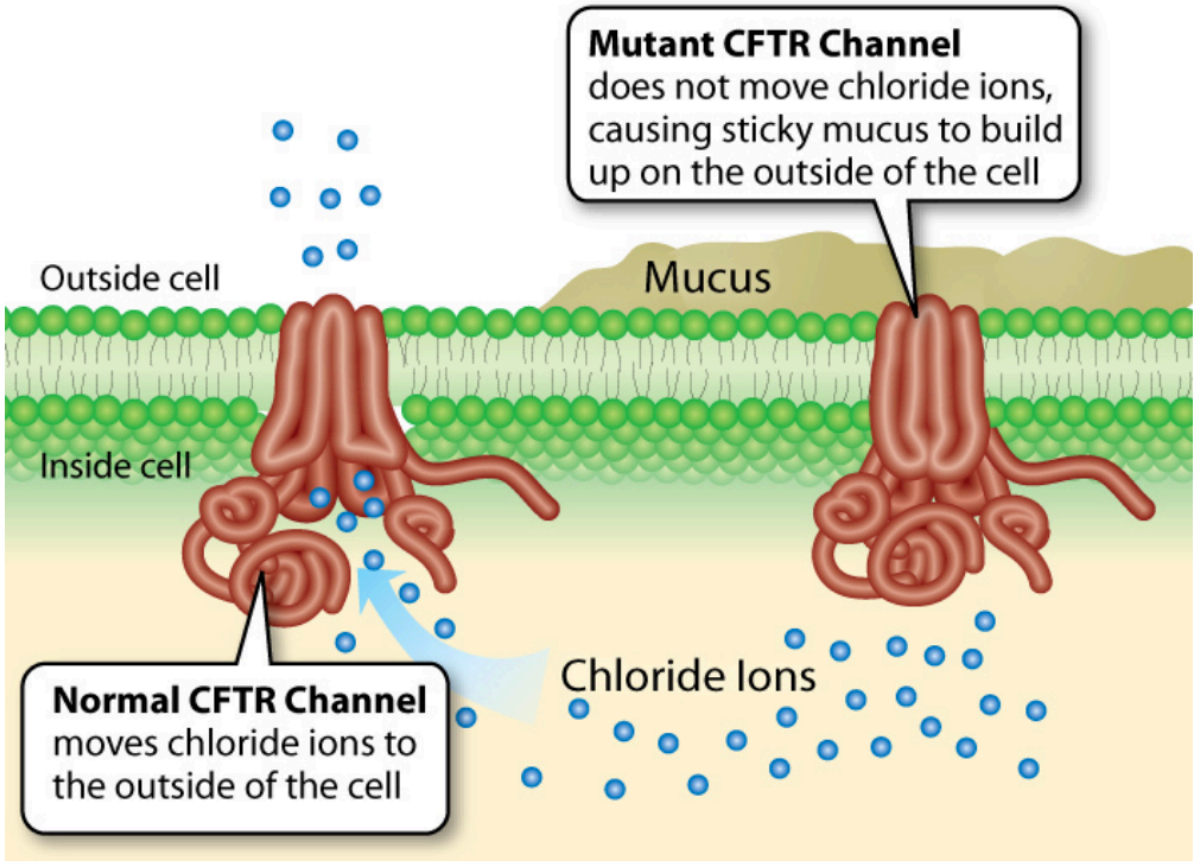






Cystic fibrosis

The gene affected by CF controls the movement of salt and water in and out of cells. People with cystic fibrosis experience a build-up of thick sticky mucus in the lungs, digestive system and other organs, causing a wide range of challenging symptoms affecting the entire body.

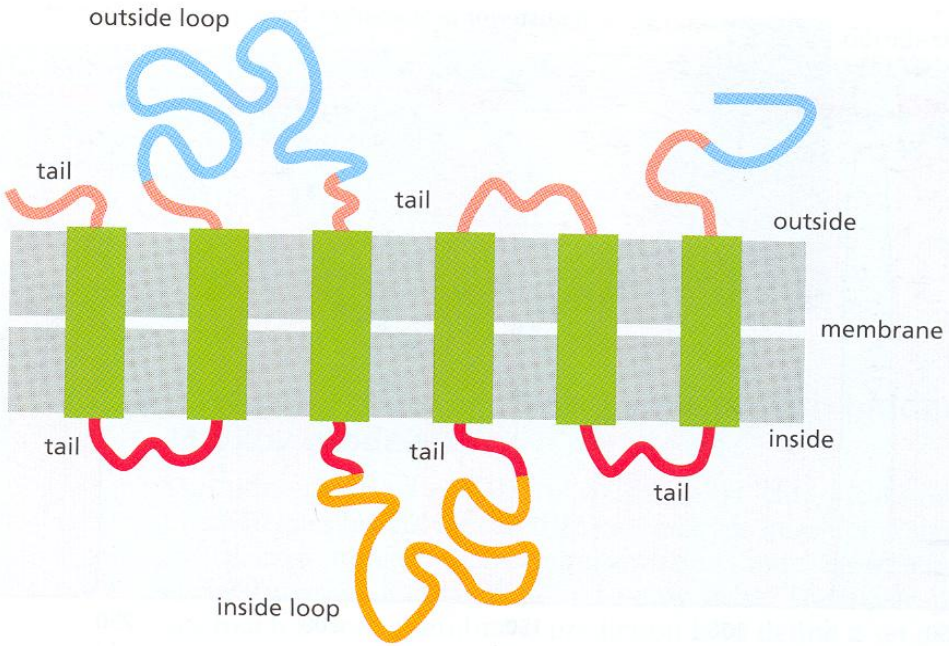


TMHMM: Prediction of transmembrane topology of protein sequence

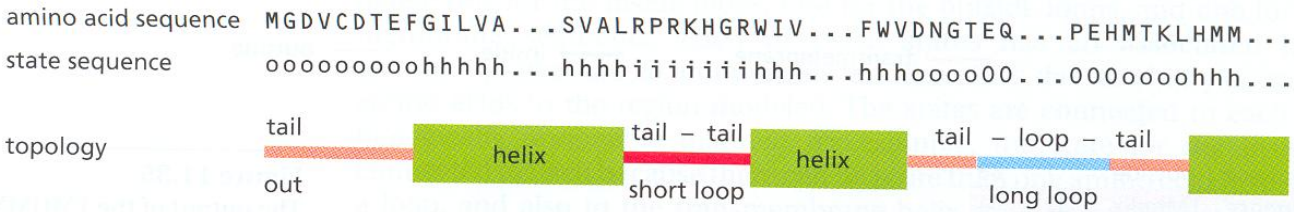
Model consists of submodels for:

- helix core and cap regions (cytoplasmic and extracellular)
- cytoplasmic and extracellular loop regions
- globular domain regions

Trained form 160 proteins with experimentally determined transmembrane



Prediction method: Posterior decoding, the program computes for each residue of the sequence the probability of being part if a transmembrane helix, an intracellular loop or globular domain region, or an extracellular loop or domain region.



Assessing performance: Sensitivity and Specificity

- Testing of predictions is performed on sequences where the gene structure is known
- ***Sensitivity*** is the fraction of known genes (or bases or exons) correctly predicted: $\mathbf{Sn = N_{True\ Positives} / N_{All\ True}}$
  - "Am I finding the things that I'm supposed to find?
- ***Specificity*** is the fraction of predicted genes (or bases or exons) that correspond to true genes: $\mathbf{Sp = N_{True\ Positives} / N_{All\ Positives}}$
  - "What fraction of my predictions are true?
- In general, increasing one decreases the other

Validation

1. be predicted to occur: Predicted Positive (PP)
2. be predicted not to occur: Predicted Negative (PN)
3. actually occur: Actual Positive (AP)
4. actually not occur: Actual Negative (AN)
5. True Positive $TP = PP \bigcap AP$
6. True Negative $TN = PN \bigcap AN$
7. False Negative $FN = PN \bigcap AP$
8. False Positive $FP = PP \bigcap AN$
9. Sensitivity: probability of correctly predicting a positive example Sn = TP/(TP + FN)
10. Specificity: probability of correctly predicting a negative example Sp = TN/(TN + FP) or
11. Probability that positive prediction is correct Sp = TP/(TP + FP).

Assessing performance: Sensitivity and Specificity

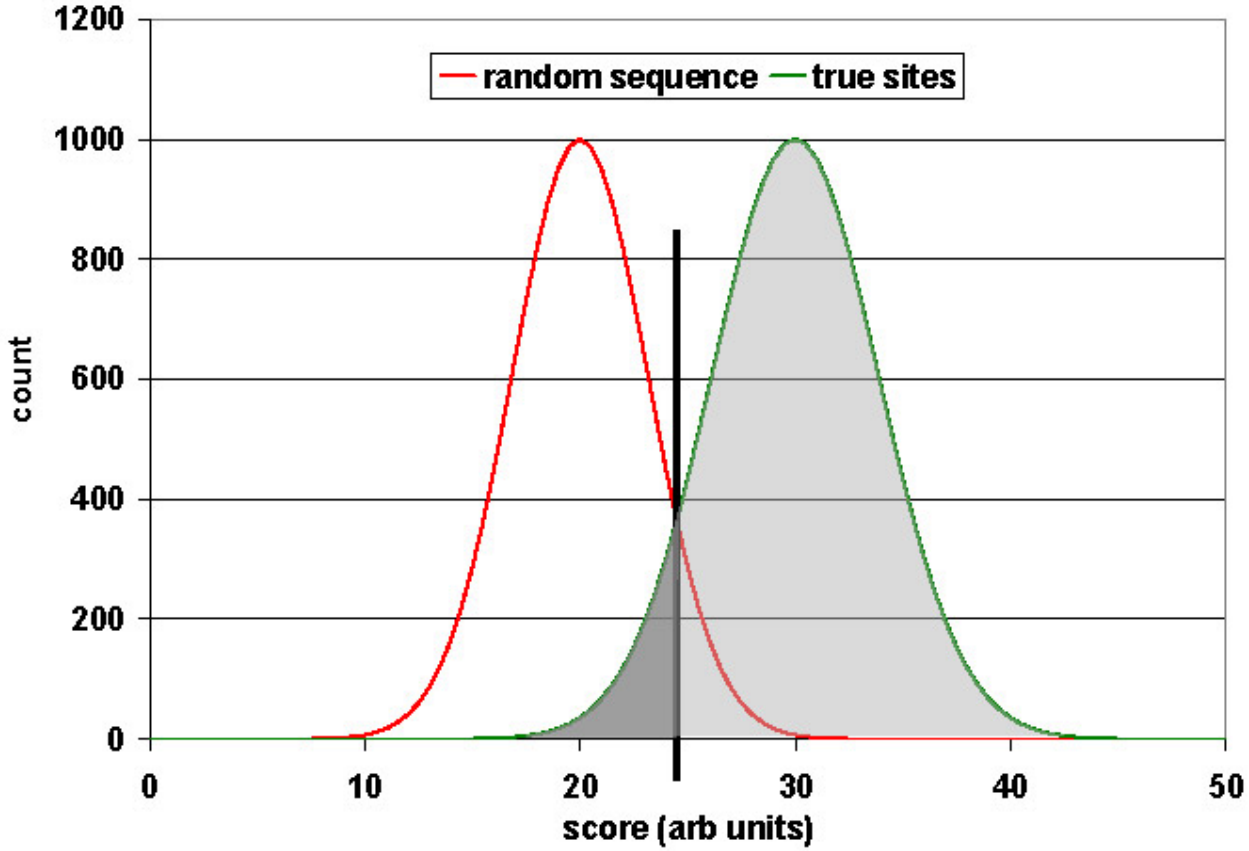
- Testing of predictions is performed on sequences where the gene structure is known
- ***Sensitivity*** is the fraction of known genes (or bases or exons) correctly predicted: **Sn=$N_{\text{True Positives}}$ /$N_{\text{All True}}$**
  - "Am I finding the things that I'm supposed to find?
- ***Specificity*** is the fraction of predicted genes (or bases or exons) that correspond to true genes: **Sp=$N_{\text{True Positives}}$ /$N_{\text{All Positives}}$**
  - "What fraction of
- In general, increas

$$CC = \frac{\left[(TP)(TN) - (FP)(FN)\right]}{\sqrt{(AN)(PP)(AP)(PN)}}$$

$$AN = TN + FP; AP = TP + FN;$$

$$PP = TP + FP; PN = TN + FN$$

Graphic View of Specificity and Sensitivity



$$Sn = \frac{TruePositive}{AllTrue} = \frac{TruePositive}{TruePositive + FalseNegative}$$

$$Sp = \frac{TruePositive}{AllPositive} = \frac{TruePositive}{TruePositive + FalsePositive}$$

Correlation Coefficient

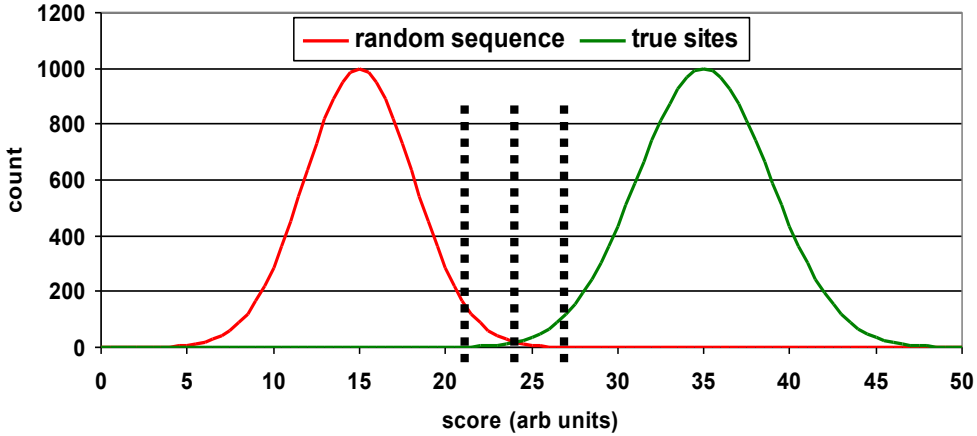
$$CC = \frac{[(TP)(TN) - (FP)(FN)]}{\sqrt{(AN)(PP)(AP)(PN)}}$$

$$AN = TN + FP; AP = TP + FN;$$

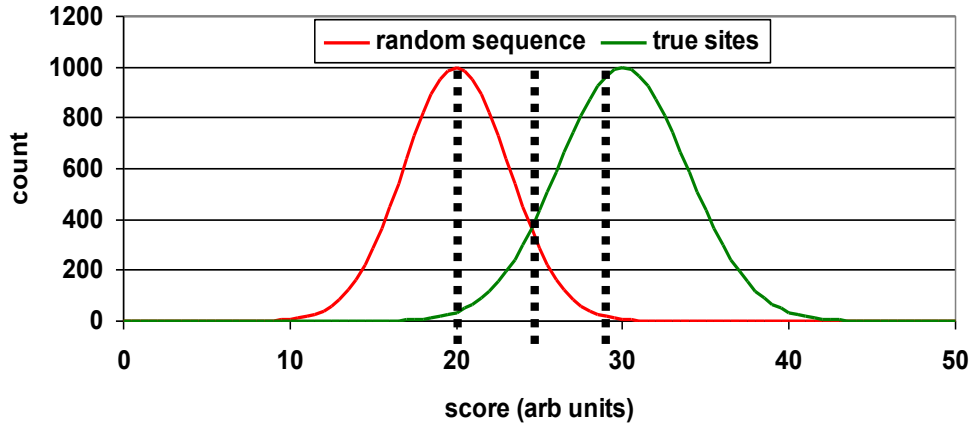
$$PP = TP + FP; PN = TN + FN$$

Specificity/Sensitivity Tradeoffs

- Ideal Distribution of Scores



- More Realistically…

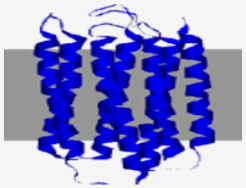


DTU Bioinformatics
Department of Bio and Health Informatics

Home

TMHMM Server v. 2.0

Prediction of transmembrane helices in proteins



Instructions

SUBMISSION

Submission of a local file in FASTA format (HTML 3.0 or higher)

Sfoglia... Nessun file selezionato.

OR by pasting sequence(s) in FASTA format:

```
>AAA39861.1 opsin [Mus musculus]
MNGTEGPNFYVPFSNVTGVGRSPFEQPQYYLAEPWQFSMLAAYMFLLIVLGFPINFLTLYVTVQHKKLRT
PLNYILLNLAVADLFMVFGGFTTTLYTSLHGYFVFGPTGCNLEGFFATLGGEIALWSLVVLAIERYVVVC
KPMSNFRFGENHAIMGVVFTWIMALACAAPPLVGWSRYIPEGMQCSCGIDYYTLKPEVNNESFVIYMFVV
HFTIPMIVIFFCYGQLVFTVKEAAAQQQESATTQKAEKEVTRMVIIMVIFFLICWLPYASVAFYIFTHQG
SNFGPIFMTLPAFFAKSSSIYNPVIYIMLNKQFRNCMLTTLCCGKNPLGDDDASATASKTETSQVAPA
```

Output format:

- (•) Extensive, with graphics
- ( ) Extensive, no graphics
- ( ) One line per protein

Other options:

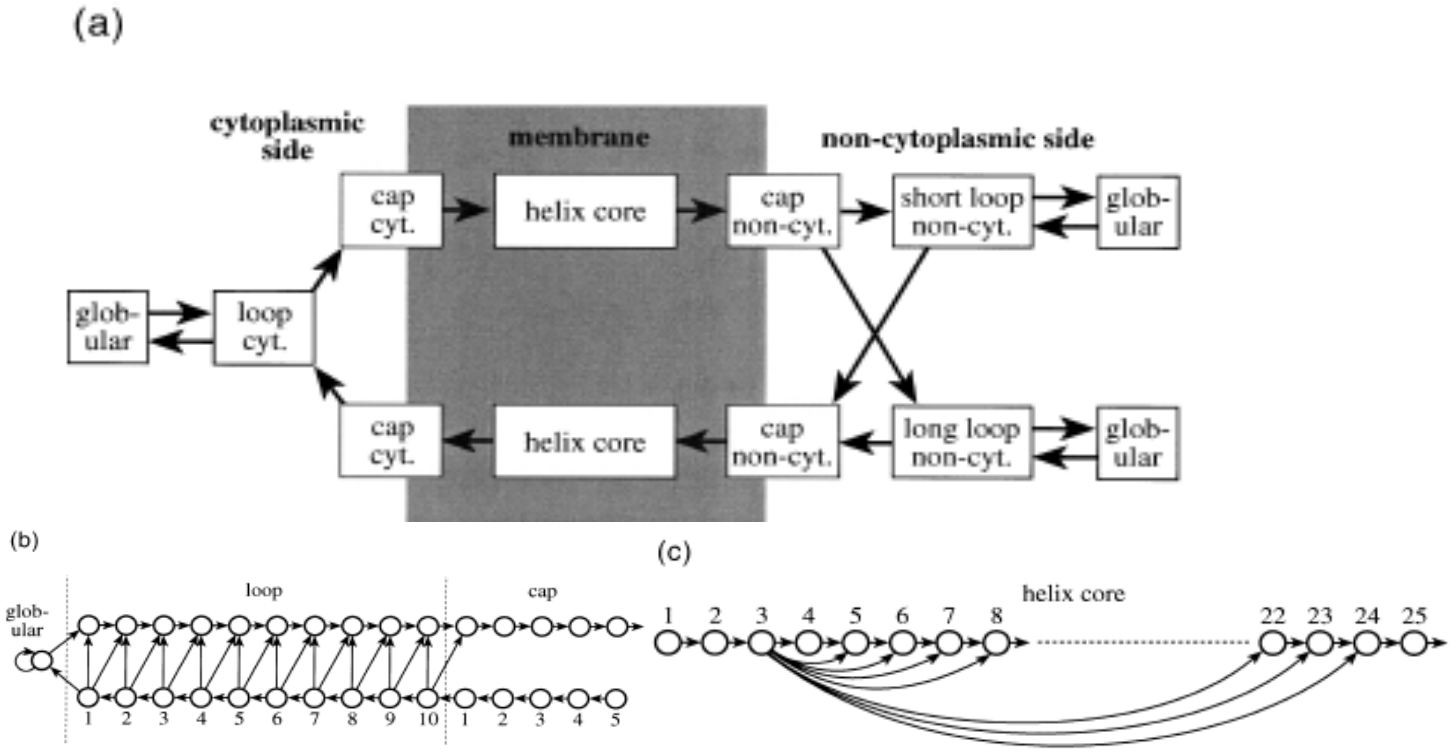
- [x] Use old model (version 1)

Submit Clear

Restrictions:
At most 10,000 sequences and 4,000,000 amino acids per submission; each sequence not more than 8,000 amino acids.

Confidentiality:
The sequences are kept confidential and will be deleted after processing.

Model architecture of TMHMM



TMHMM: uses cyclic model with 7 states for
- TM helix core
- TM helix caps on the N- and C-terminal side
- non-membrane region on the cytoplasmic side
- 2 non-membrane regions on the non-cytoplasmic side (for short and long loops to account for different membrane insertion mechanism)
- a globular domain state in the middle of each non-membrane region

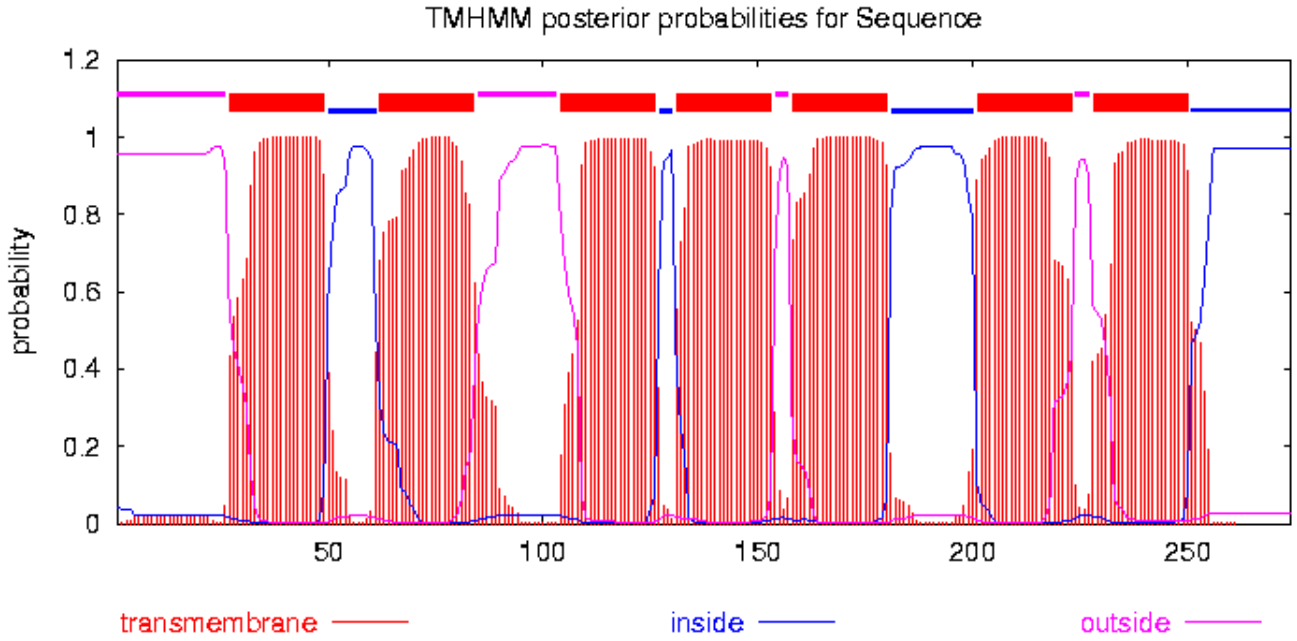
Example for TMHMM
www.cbs.dtu.dk/services/TMHMM/

>gi|218694017|ref|YP_002401684.1| membrane protein; channel [Escherichia coli 55989]
MQDLISQVEDLAGIEIDHTTSMVMIFGIIFLTAVVVHIILHWVVLRTFEKRAIASS
RLWLQIITQNKLFH
RLAFTLQGIIVNIQAVFWLQKGTEAADILTTCAQLWIMMYALLSVFSLLDVILNL
AQKFPAASQLPLKGI
FQGIKLIGAILVGILMISLLIGQSPAILISGLGAMAAVLMLVFKDPILGLVAGIQLS
ANDMLKLGDWLEM
PKYGADGAVIDIGLTTVKVRNWDNTITTIPTWSLVSDSFKNWSGMSASGGRR
IKRSISIDVTSIRFLDED
EMQRLNKAHLLKPYLTSRHQEINEWNRQQGSTESILNLRRMTNIGTFRAYLN
EYLRNHPRIRKDMTLMVR
QLAPGDNGLPLEIYAFTNTVVWLEYESIQADIFDHIFAIVEEFGLRLHQSPTGN
DIRSLAGAFKQ

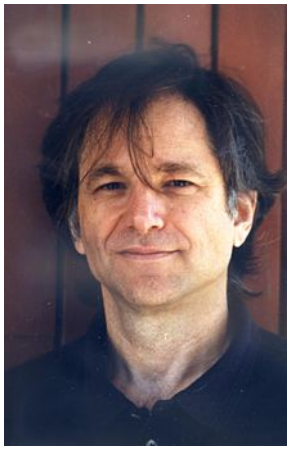
TMHMM-Output

```
# Sequence Length: 274
# Sequence Number of predicted TMHs:  7
# Sequence Exp number of AAs in TMHs: 153.74681
# Sequence Exp number, first 60 AAs:  22.08833
# Sequence Total prob of N-in:        0.04171
# Sequence POSSIBLE N-term signal sequence
Sequence        TMHMM2.0        outside      1    26
Sequence        TMHMM2.0        TMhelix     27    49
Sequence        TMHMM2.0        inside      50    61
Sequence        TMHMM2.0        TMhelix     62    84
Sequence        TMHMM2.0        outside     85   103
Sequence        TMHMM2.0        TMhelix    104   126
Sequence        TMHMM2.0        inside     127   130
Sequence        TMHMM2.0        TMhelix    131   153
Sequence        TMHMM2.0        outside    154   157
Sequence        TMHMM2.0        TMhelix    158   180
Sequence        TMHMM2.0        inside     181   200
Sequence        TMHMM2.0        TMhelix    201   223
Sequence        TMHMM2.0        outside    224   227
Sequence        TMHMM2.0        TMhelix    228   250
Sequence        TMHMM2.0        inside     251   274
```

http://www.cbs.dtu.dk/services/TMHMM-2.0/



DNA for computing:



Adleman, L. M. (1994). “Molecular computation of solutions to combinatorial problems”. Science 266 (5187): 1021-1024. doi:10.1126/science.7973651.

Adleman's first DNA computation solved a traveling salesman problem of seven cities. He used DNA techniques to assemble itineraries at random; Select itineraries from initial city to final city. The correct number of cities must be visited. No city can be left out.

Each city is represented by a unique sequence of bases. Connections between two cities are created from a combination of the complement of the first half of the sequence of one city, and the complement of the second half of the sequence of a connected city. In this way DNA representing the trip will be created with one strand representing a sequence of cities and the complementing strand representing a series of connections.

The next step is filtering out trips that start and end in the correct cities, then filtering trips with the correct number of cities, and finally filtering out trips that contain each city only once. Pros: 1 gram of DNA can hold about $1x10^{14}$ MB of data. A test tube of DNA can contain trillions of strands. Each operation on a test tube of DNA is carried out on all strands in the tube in parallel; Adleman figured his computer was running 2×10^{19} operations per joule. Adleman’s process to solve the traveling salesman problem for 200 cities would require an amount of DNA that weighed more than the Earth.

DNA for computing:

Represent Each City By A DNA Strand of 20 Bases

City1 ATGCTCAGCTACTATAGCGA

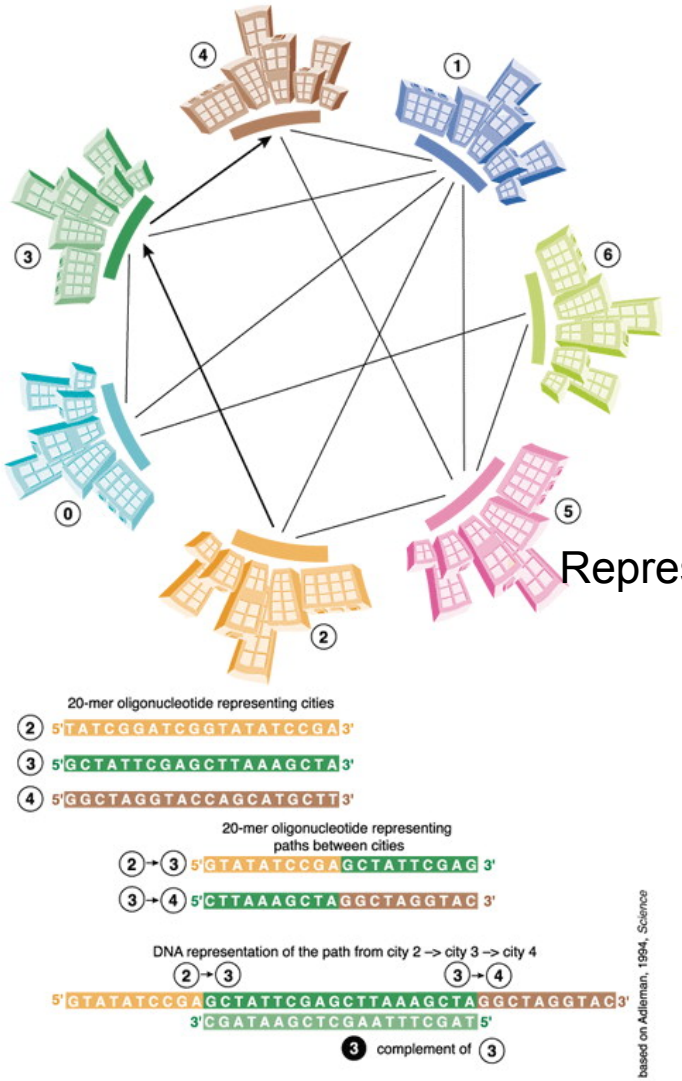
City2 TGCGATGTACTAGCATATAT

City3 GCATATGGTACACTGTACAA

City4 TTATTAGCGTGCGGCCTATG

City5 CCGCGATAGTCTAGATTTCC

Etc.



Represent Each Air Route By Mixed Complementary Strands

City 1->2 TGATATCGCTACGCTACATG

City 2->3 ATCGTATATACGTATACCAT

City 3->4 GTGACATGTTAATAATCGCA

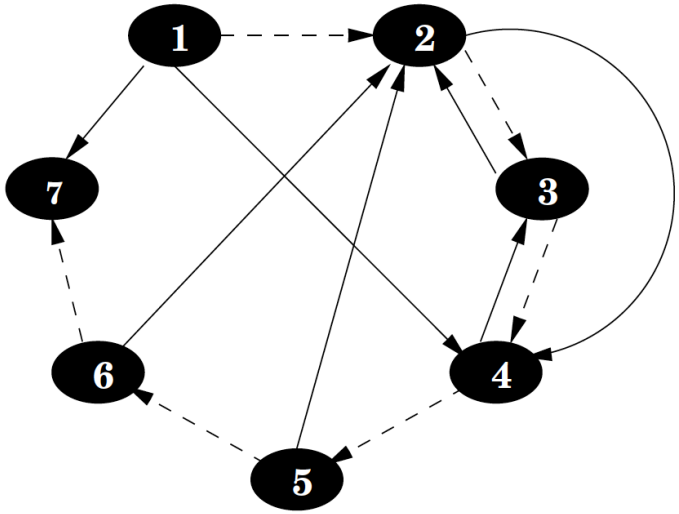
City 4->5 CGCCGGATACGGCGCTATCA

City 5->6 GATCTAAAGGTATGCATACG

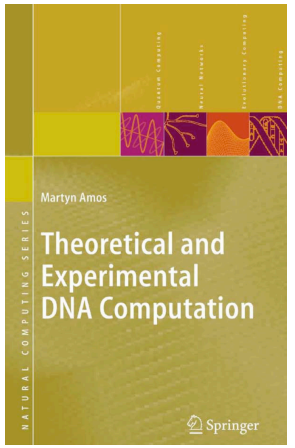
Etc.

L. Adelman, *Scientific American*, pp. 54-61 (Aug 1998);

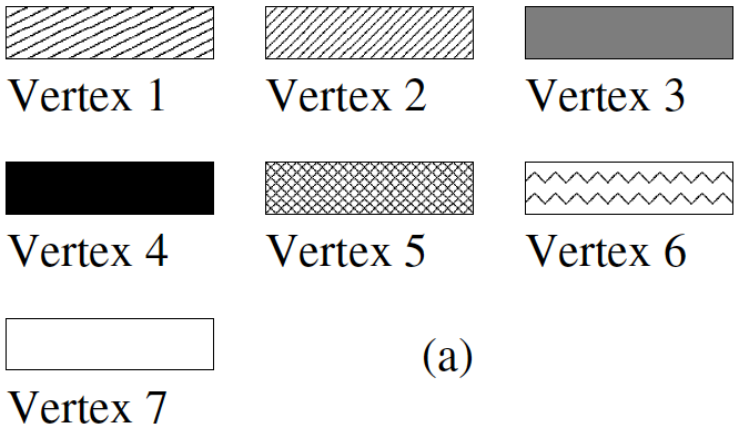
DNA for computing



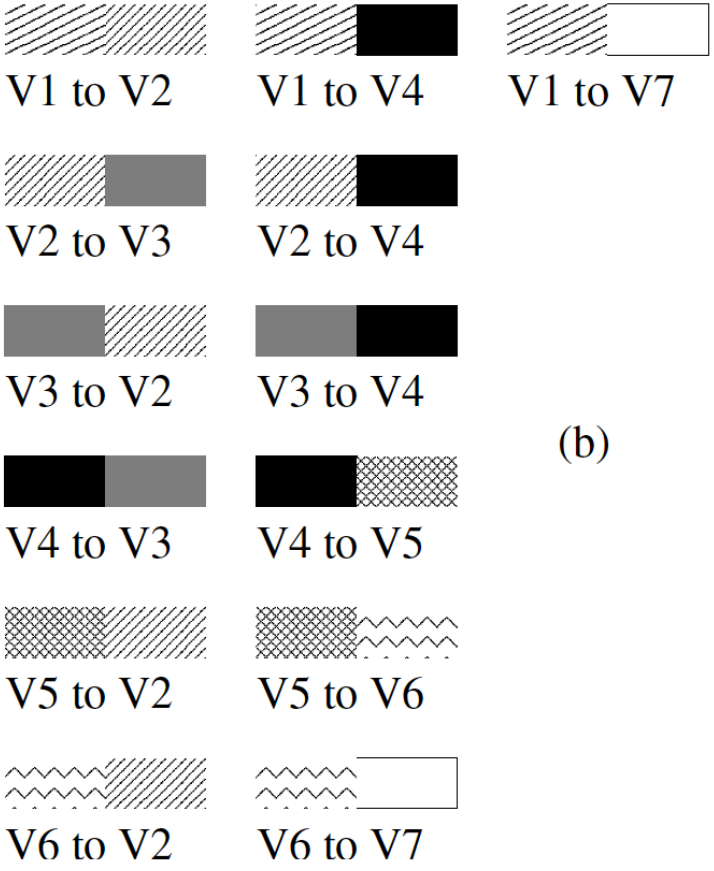
figures from Martyn Amos



cities



routes



selection for length and initial/end points

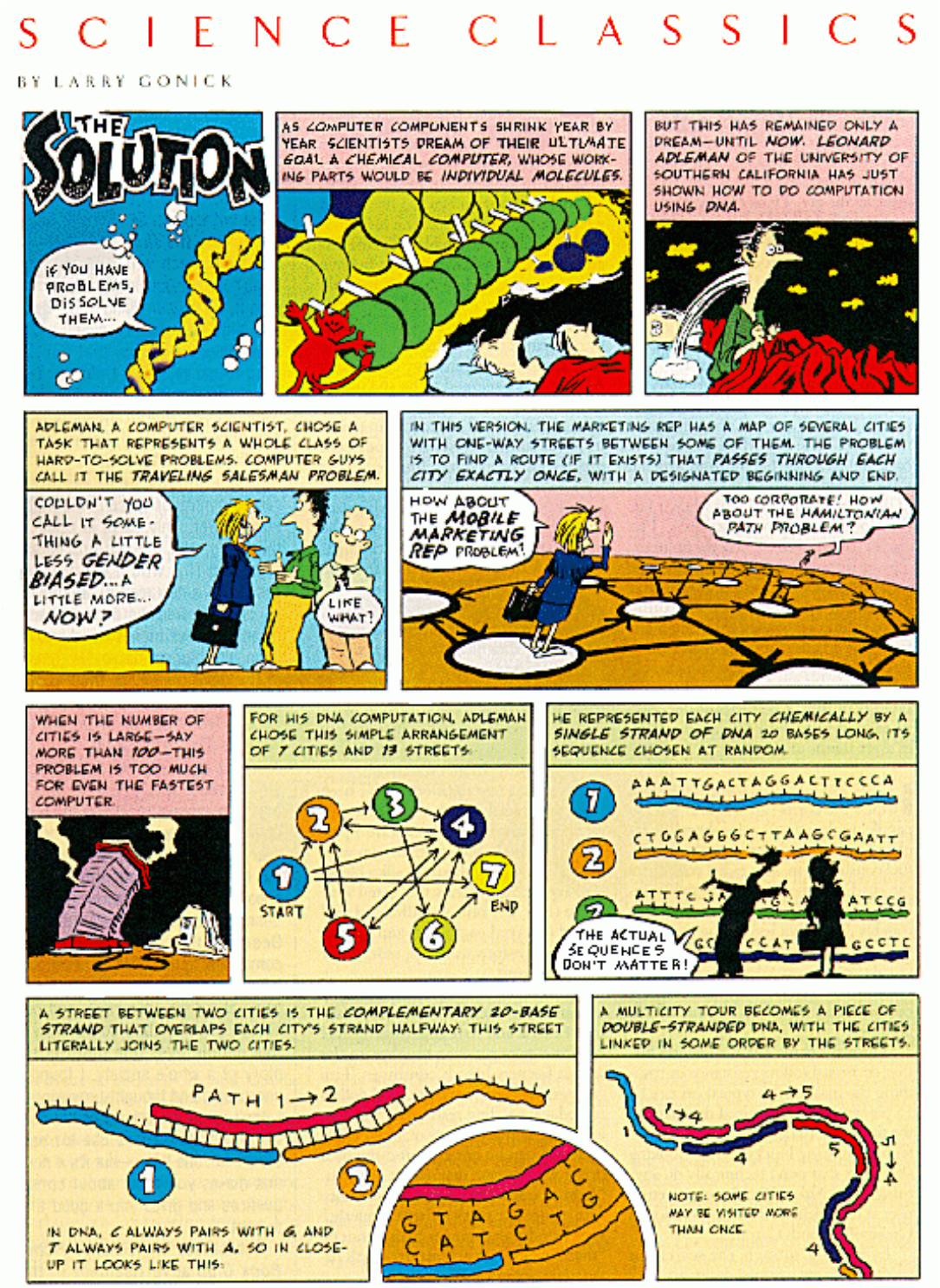
‘travelling salesman’ problem

The challenge is finding a route between various cities, passing through each only once.

Adleman first generated all the possible itineraries and then selected the correct itinerary.

Since the enzymes (enzymes are proteins catalyzing a reaction) work on many DNA molecules at once, the selection process is massively parallel. Specifically, the method based on Adleman’s experiment would be as follows:

- Generate all possible routes.
- Select itineraries that start with the proper city and end with the final city.
- Select itineraries with the correct number of cities.
- Select itineraries that contain each city only once.
- All of the above steps can be accomplished with standard molecular biology techniques.



Discover magazine published an article in comic strip format about Leonard Adleman's DNA computation.

Sort the DNA by length and select the DNA whose length corresponds to 7 cities

A test tube is now filled with DNA encoded itineraries that start with LA and end with NY, where the number of cities in between LA and NY varies.

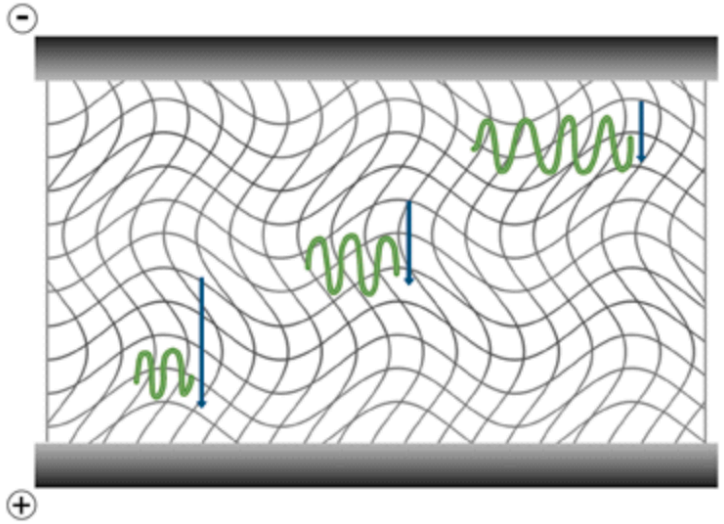
We now want to select those itineraries that are seven cities long. To accomplish this we can use a technique called Gel Electrophoresis, which is a common procedure used to resolve the size of DNA. The basic principle behind Gel Electrophoresis is to force DNA through a gel matrix by using an electric field.

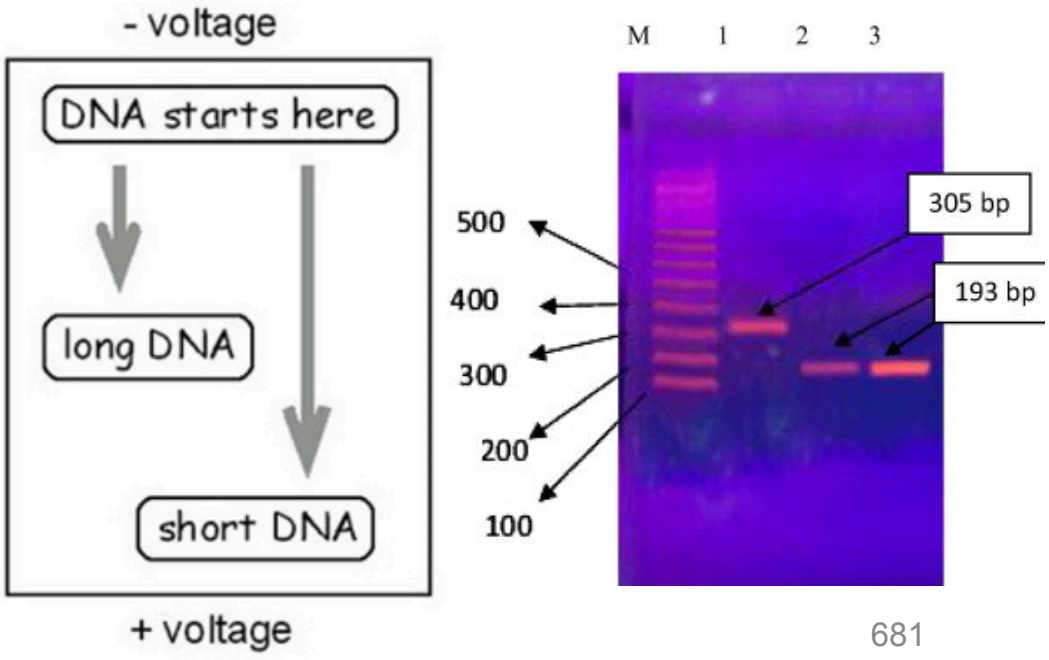
DNA is a negatively charged molecule under most conditions, so if placed in an electric field it will be attracted to the positive potential.

The gel is made up of a polymer that forms a meshwork of linked strands. The DNA now is forced to thread its way through the tiny spaces between these strands, which slows down the DNA at different rates depending on its length.

What we typically end up with after running a gel is a series of DNA bands, with each band corresponding to a certain length.

We can then simply cut out the band of interest to isolate DNA of a specific length. Since we know that each city is encoded with a certain number of base pairs of DNA, knowing the length of the itinerary gives us the number of cities.





Technique for Generating Routes Strategy:

Encode city names in short DNA sequences. Encode itineraries by connecting the city sequences for which routes exist.

Synthesizing short single stranded DNA is now a routine process, so encoding the city strings is straightforward. Itineraries can then be produced from the city encodings by linking them together in proper order.

To accomplish this you can take advantage of the fact that DNA hybridizes (=binds) with its complimentary sequence (complementary strands of DNA bind each other).

For example, you can encode the routes between cities by encoding the compliment of the second half (last n letters) of the departure city and the first half (first n letters) of the arrival city.

For example the route between Miami (CTACGG) and NY (ATGCCG) can be made by taking the second half of the coding for Miami (CGG) and the first half of the coding for NY (ATG). This gives CGGATG.

By taking the complement of this you get, GCCTAC, which not only uniquely represents the route from Miami to NY, but will connect the DNA representing Miami and NY by hybridizing itself to the second half of the code representing Miami (...CGG) and the first half of the code representing NY (ATG…).

Random itineraries can be made by mixing city encodings with the route encodings. Finally, the DNA strands can be connected together by an enzyme called ligase (ligases are enzymes, i.e. proteins connecting strings). What we are left with are strands of DNA representing itineraries with a random number of cities and random set of routes.

Itineraries Selection:
Start and End with Correct Cities

Strategy: Selectively copy and amplify only the section of the DNA that starts with LA and ends with NY by using the Polymerase Chain Reaction (PCR). See next slide.

After generating the routes, we now have a test tube full of various lengths of DNA that encode possible routes between cities.

What we want are routes that start with LA and end with NY. To accomplish this we can use a technique called Polymerase Chain Reaction (PCR), which allows you to produce many copies of a specific sequence of DNA.

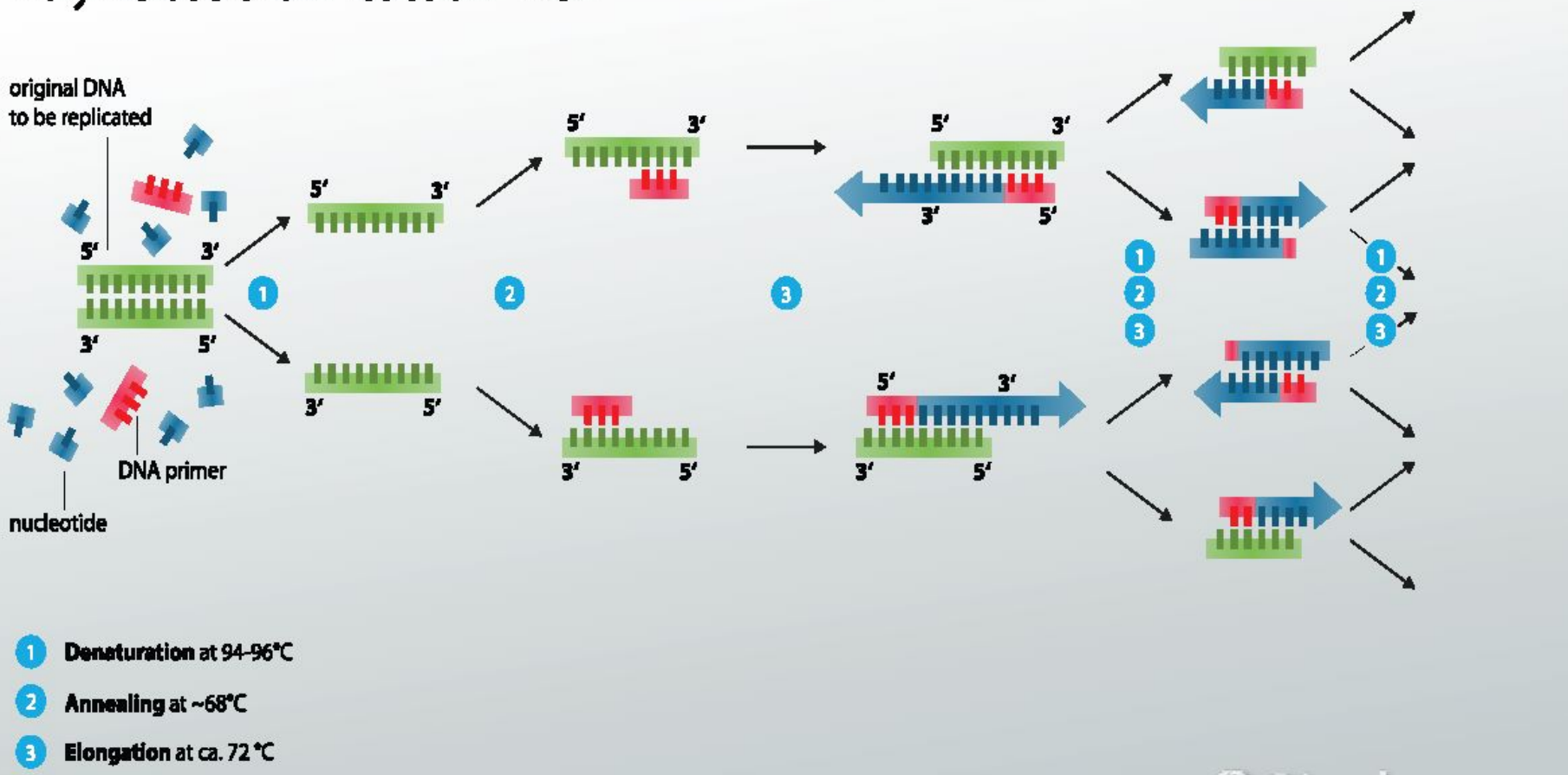
After many iterations of PCR, the DNA you're working on is amplified exponentially.

So to selectively amplify the itineraries that start and stop with our cities of interest, we use primers that are complimentary to LA and NY.

What we end up with after PCR is a test tube full of double stranded DNA of various lengths, encoding itineraries that start with LA and end with NY.

WHAT IS PCR?

from wikipedia

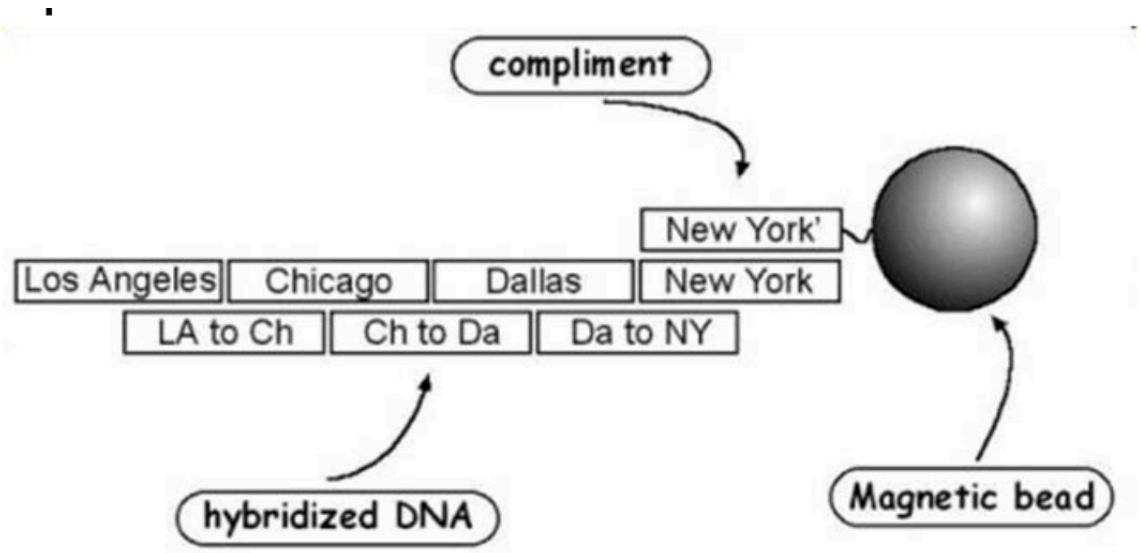


PCR is an iterative process that cycle through a series of copying events using an enzyme called polymerase. Polymerase will copy a section of single stranded DNA starting at the position of a primer, a short piece of DNA complimentary to one end of a section of the DNA that you're interested in.

By selecting primers that flank the section of DNA you want to amplify, the polymerase preferentially amplifies the DNA between these primers, doubling the amount of DNA containing this sequence.

Itineraries Selection: Have a Complete Set of Cities

DNA containing a specific sequence can be purified from a sample of mixed DNA by a technique called affinity purification, as shown below. This is accomplished by attaching the compliment of the sequence in question to a substrate like a magnetic bead. The beads are then mixed with the DNA. DNA, which contains the sequence you're after then hybridizes with the complement sequence on the beads. These beads can then be retrieved and the DNA isolated.



Select itineraries that have a complete set of cities. Sequentially affinity-purify n times, using a different city complement for each run. We are left with itineraries that start in LA, visit each city once, and end in NY.

- Adleman's experiment solved a seven city problem, but there are two major shortcomings preventing a large scaling up of his computation.
- The complexity of the traveling salesman problem simply doesn’t disappear when applying a different method of solution - it still increases exponentially.
- For Adleman’s method, what scales exponentially is not the computing time, but rather the amount of DNA. Unfortunately this places some hard restrictions on the number of cities that can be solved; after the Adleman article was published, more than a few people have pointed out that using his method to solve a 200 city problem would take an amount of DNA that weighed more than the earth.

Adleman's pros & cons

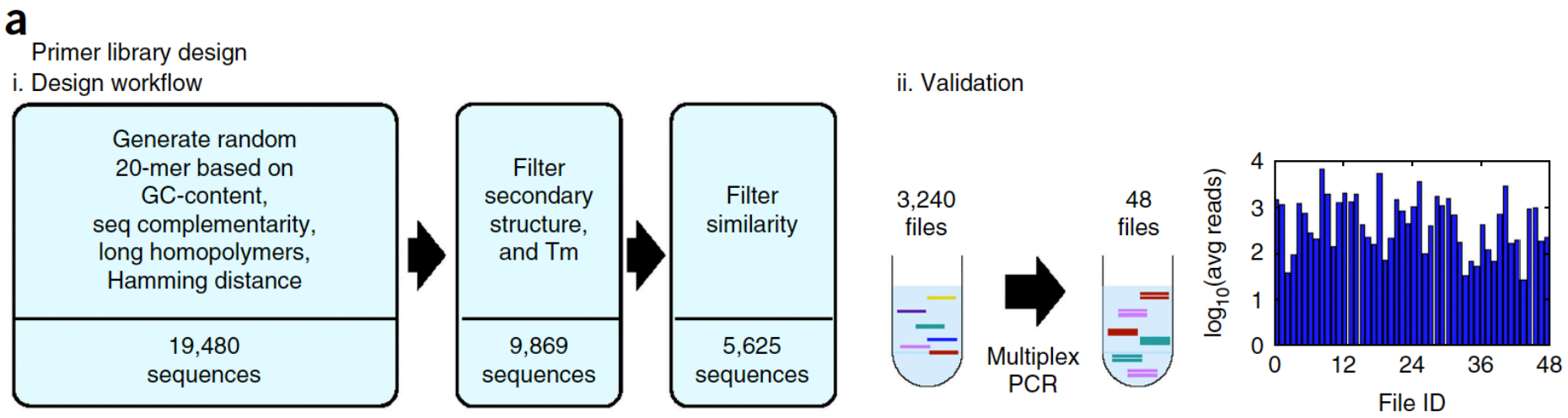
Pros: 1 gram of DNA can hold about $1x10^{14}$ MB of data. A test tube of DNA can contain trillions of strands.

5 grams of DNA contain 10^{21} bases (Zetta Bytes)

Each operation on a test tube of DNA is carried out on all strands in the tube in parallel; Adleman figured his computer was running 2×10^{19} operations per joule.

Adleman's process to solve the traveling salesman problem for 200 cities would require an amount of DNA that weighed more than the Earth.

Speed: 500-5000 base pairs a second.



Design of random access primers and coding algorithm. (a, i) They designed a primer library. The primer sequence set is then filtered that has low similarity between the sequences. (a, ii) The resulting set of candidate primers is then validated experimentally by synthesizing a pool of about 100,000 strands containing sets of size 1 to 200 DNA sequences each, surrounded by one of the candidate primer pairs, and then randomly selecting 48 of those pairs for amplification. The product is sequenced, and sequences with each of the 48 primer pairs appear among sequencing reads, albeit at different relative proportions when normalized to the number of sequences in each set.

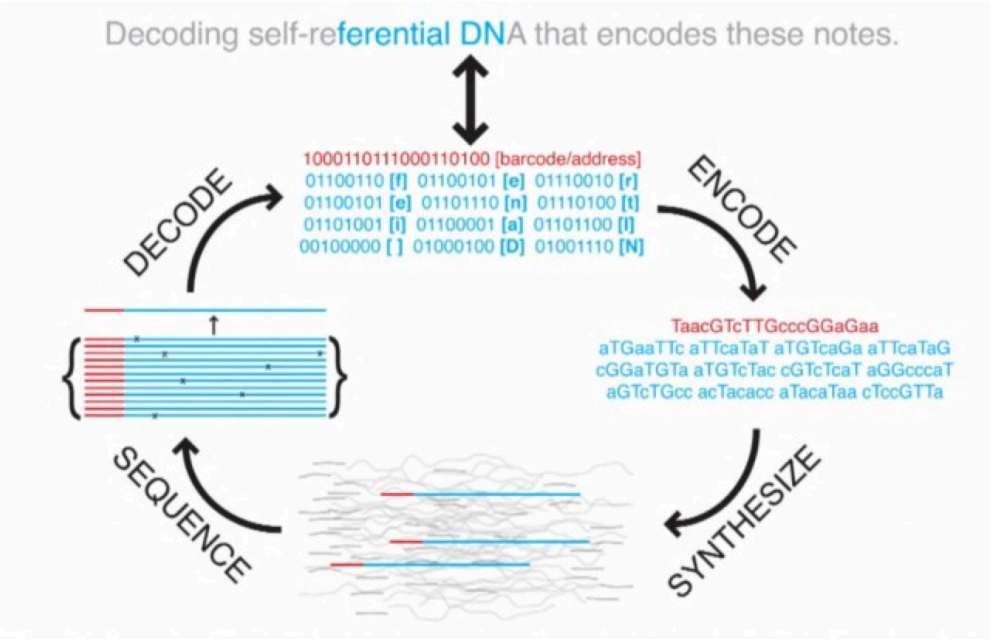
References

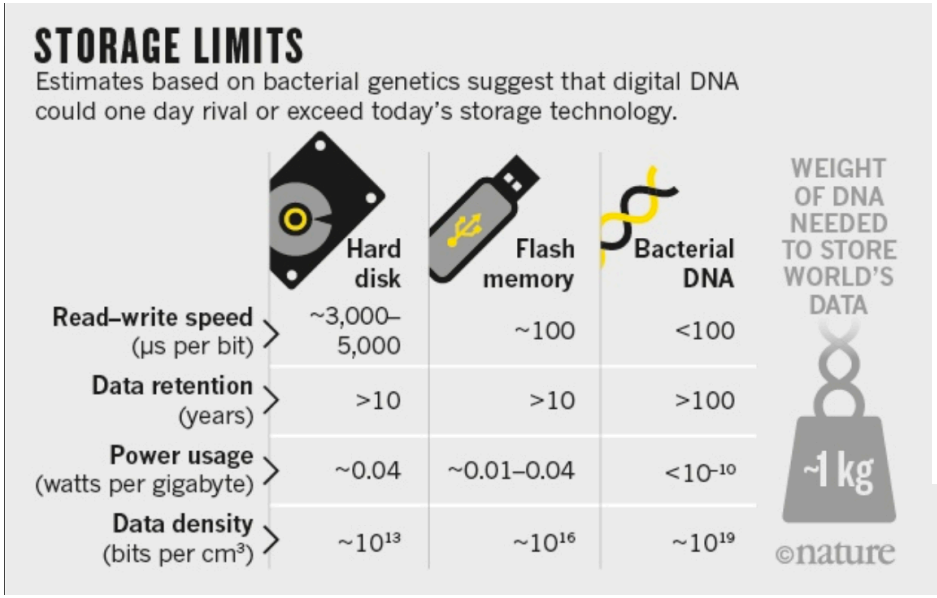
- Adleman, L. M. (1994). "*Molecular computation of solutions to combinatorial problems*". Science 266 (5187): 1021-1024. doi:10.1126/science. 7973651. PMID 7973651.
- Boneh, D.; Dunworth, C.; Lipton, R. J.; Sgall, J. Í. (1996). "*On the computational power of DNA*". Discrete Applied Mathematics 71: 79-94
- Shapiro, Ehud (1999-12-07)., "*A Mechanical Turing Machine: Blueprint for a Biomolecular Computer*," Weizmann Institute of Science. Retrieved 2009-08-13.

DNA as information storage

The work, carried out by George Church and Sri Kosuri, basically treats DNA as just another digital storage device. Instead of binary data being encoded as magnetic regions on a hard drive platter, strands of DNA that store 96 bits are synthesized, with each of the bases (TGAC) representing a binary value (T and G = 1, A and C = 0).

To read the data stored in DNA, you simply sequence it — just as if you were sequencing the human genome — and convert each of the TGAC bases back into binary. To aid with sequencing, each strand of DNA has a 19-bit address block at the start (the red bits in the image below) — so a whole vat of DNA can be sequenced out of order, and then sorted into usable data using the addresses.





more at the end of the course

https://www.nature.com/articles/nbt.4079

ARTICLES

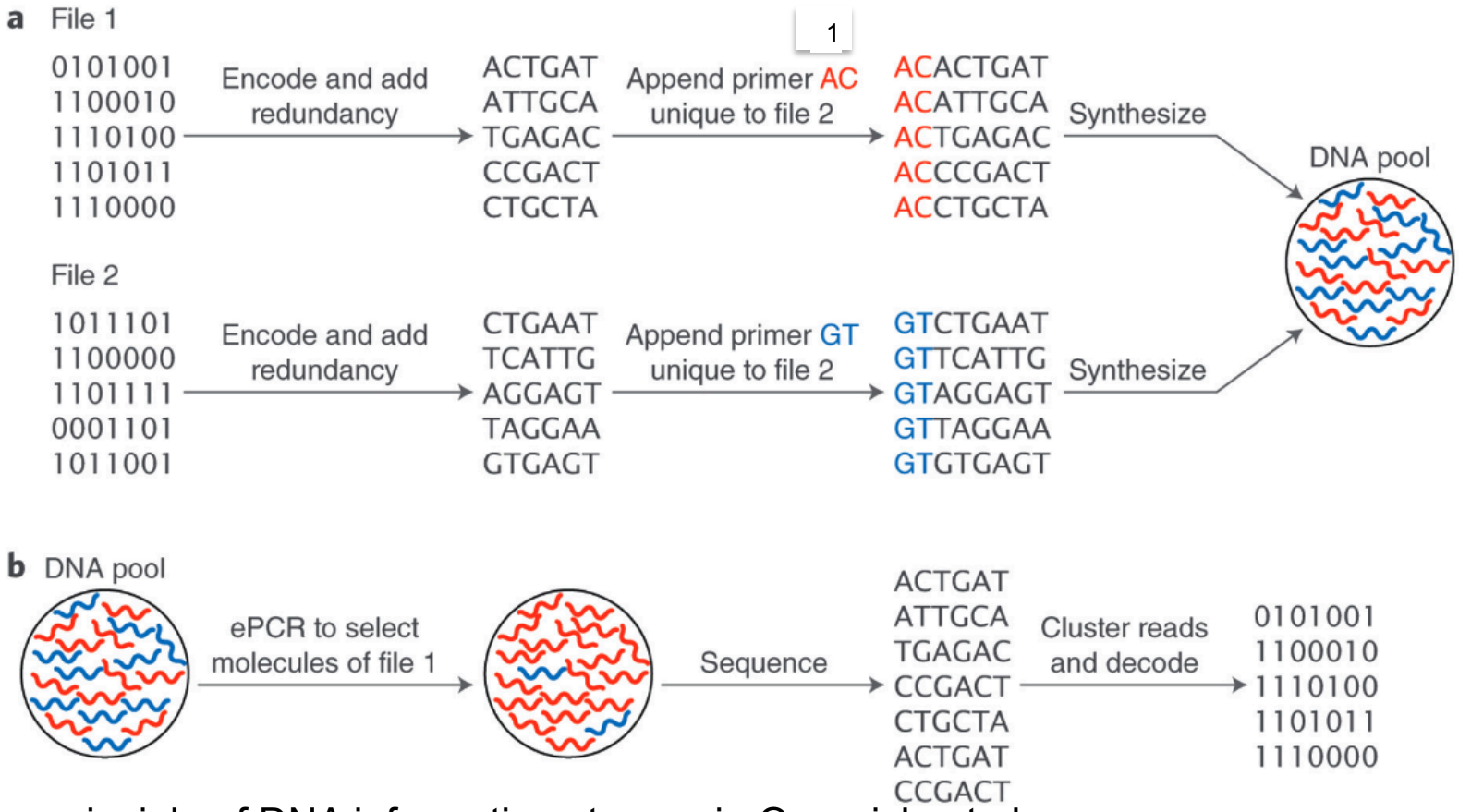
nature biotechnology

Random access in large-scale DNA data storage

Lee Organick[1], Siena Dumas Ang[2], Yuan-Jyue Chen[2], Randolph Lopez[3], Sergey Yekhanin[2], Konstantin Makarychev[2,5], Miklos Z Racz[2,5], Govinda Kamath[2,5], Parikshit Gopalan[2,5], Bichlien Nguyen[2], Christopher N Takahashi[1], Sharon Newman[1,5], Hsing-Yeh Parker[2], Cyrus Rashtchian[2], Kendall Stewart[1], Gagan Gupta[2], Robert Carlson[2], John Mulligan[2], Douglas Carmean[2], Georg Seelig[1,4], Luis Ceze[1] & Karin Strauss[2]

Synthetic DNA is durable and can encode digital data with high density, making it an attractive medium for data storage. However, recovering stored data on a large-scale currently requires all the DNA in a pool to be sequenced, even if only a subset of the information needs to be extracted. Here, we encode and store 35 distinct files (over 200 MB of data), in more than 13 million DNA oligonucleotides, and show that we can recover each file individually and with no errors, using a random access approach. We design and validate a large library of primers that enable individual recovery of all files stored within the DNA. We also develop an algorithm that greatly reduces the sequencing read coverage required for error-free decoding by maximizing information from all sequence reads. These advances demonstrate a viable, large-scale system for DNA data storage and retrieval.

july 2018



The principle of DNA information storage in Organick et al.
(a) Two files are stored by encoding each file in a set of different DNA sequences. Redundant information is added to enable error recovery at retrieval, and a distinct primer is appended to each set of sequences corresponding to a file.
The resulting strings are synthesized and stored as a pool of different DNA molecules.

(b) A specific file is retrieved by amplifying the molecules corresponding to the file by ePCR, sequencing the PCR products, and algorithmically reconstructing the data from the reads.

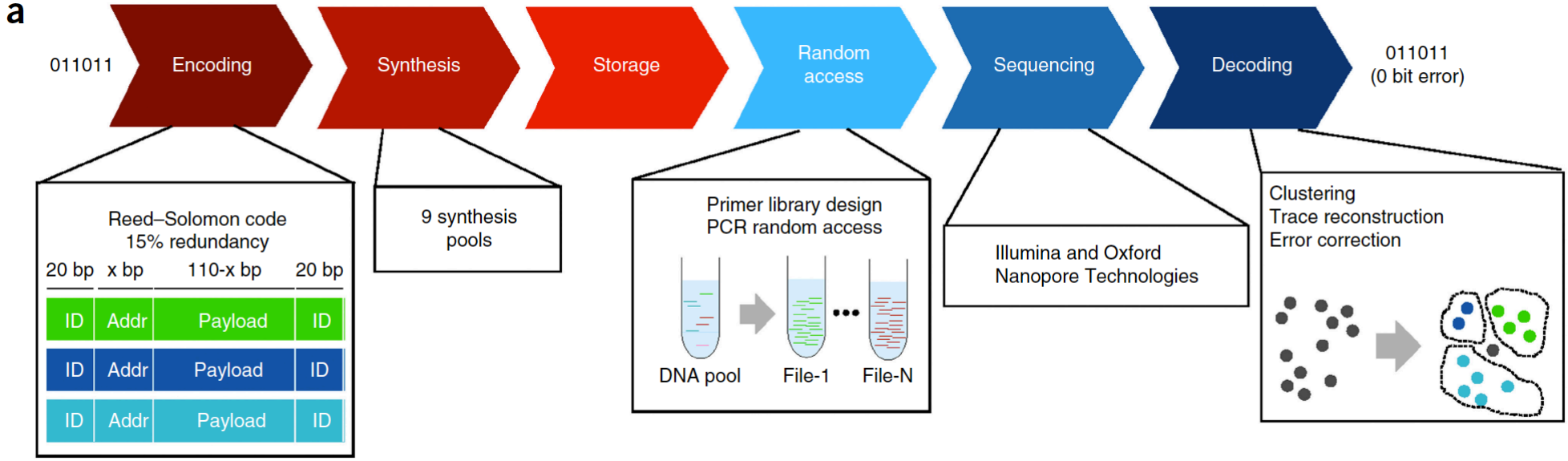
Organick et al. stored and retrieved more than 200 megabytes of data.

Specifically, they attach distinct primers to each set of DNA molecules carrying information about a file. This allows them to retrieve a given file by selectively amplifying and sequencing only the molecules with the primer marking the desired file.

To test their scheme, they designed a primer library that allowed them to uniquely tag data stored in DNA. They encoded 35 digital files into 13,448,372 DNA sequences, each 150-nucleotides long. Redundant information using error detection codes is also included to increase robustness to missing sequences and errors.

To improve recovery of the information, Organick et al. develop a clustering and consensus algorithm that aligns and filters reads before error correction.

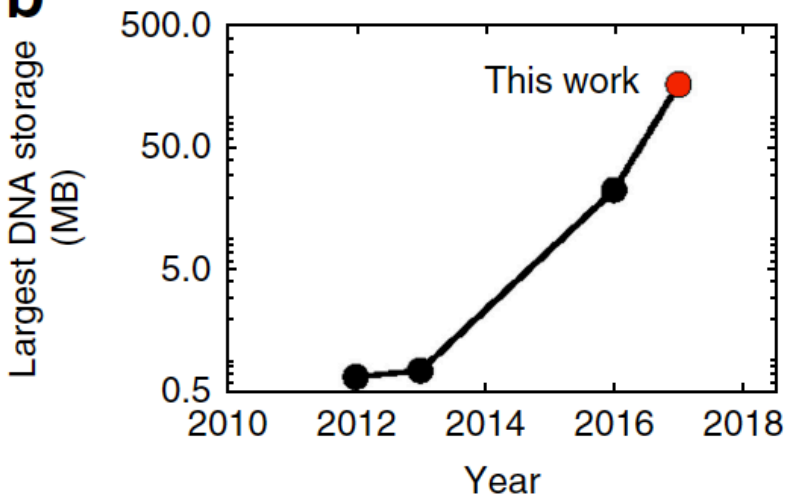
This algorithm also takes into account reads that differ from the correct length.



This work describes large-scale random access, low redundancy, and robust encoding and decoding of information stored in DNA, as well as a notable increase in the volume of data stored (200 MB, the largest synthetic DNA pool available to date).Overview of the DNA data storage workflow and stored data.

(a) The encoding process maps digital files into a large set of 150-nucleotide DNA sequences, including Reed–Solomon code redundancy to overcome errors in synthesis and sequencing. The resulting collection of sequences is synthesized. The random access process starts with amplifying a subset of the sequences corresponding to one of the files using PCR. The amplified pools are sequenced. Finally, sequencing reads are decoded using clustering, consensus and error correction algorithms.

b



Example files encoded within the 200 MB of data.

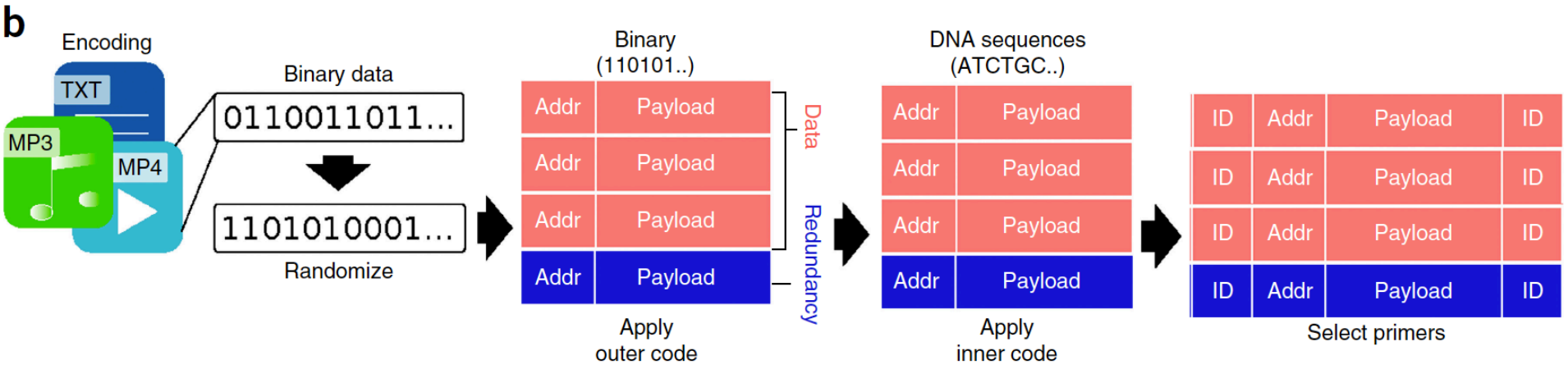
a comparison to research achievements shows that our coding scheme has similar logical redundancy, but requires lower sequencing coverage to recover files

c

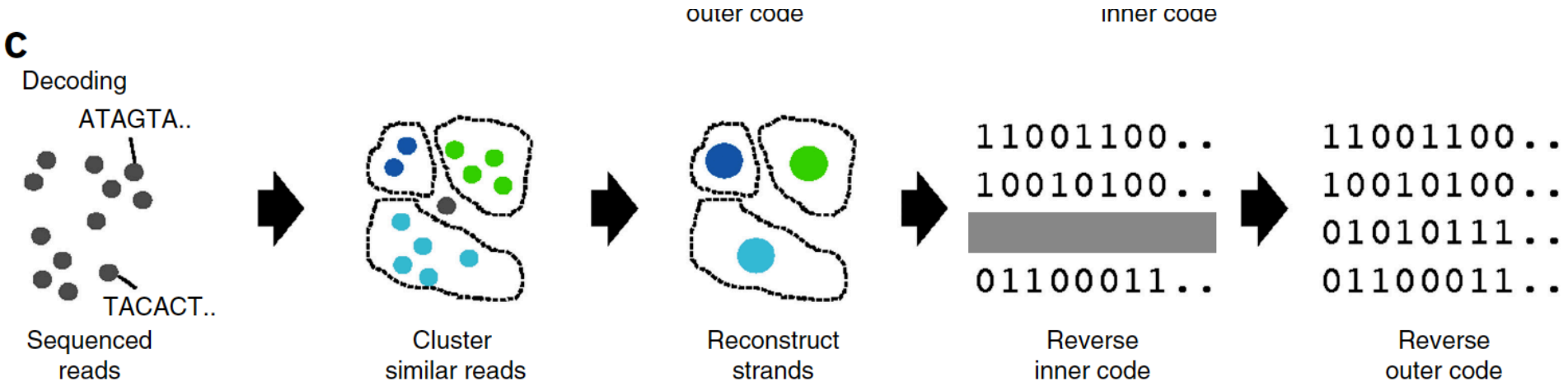
Data	File size	Number of DNA sequences
OK GO (HD video)	44.2 MB	3.2 million
Classical music collection (Music)	13.9 MB	890,000

d

	Data size	Sequencing technology	Subsampled to low coverage	Coverage	Bits per base including primers	Bits per base excluding primers	Random access
Church *et al.*[3]	0.65 MB	Illumina	No	3,000x	0.60	0.83	No
Goldman *et al.*[4]	0.63 MB	Illumina	Yes	51x	0.19	0.29	No
Grass *et al.*[5]	0.08 MB	Illumina	No	372x	0.86	1.16	No
Bornholt *et al.*[9]	0.15 MB	Illumina	Yes	40x	0.57	0.85	Yes
Erlich and Zielinski[7]	2.11 MB	Illumina	Yes	10.5x	1.18	1.55	No
Blawat *et al.*[6]	22 MB	Illumina	No	160x	0.89	1.08	No
This work	200.2 MB	Illumina	Yes	5x	0.81	1.10	Yes
Yadzi *et al.*[10]	0.003 MB	Nanopore	No	200x	1.71	1.74	Yes
This work	0.033 MB	Nanopore	Yes	36x / 80x	0.81	1.10	Yes



The encoding process starts by randomizing data to reduce chances of secondary structures, primer–payload non-specific binding, and improved properties during decoding. It then breaks the data into fixed-size payloads, adds addressing information (Addr), and applies outer coding, which adds redundant sequences using a Reed–Solomon code to increase robustness to missing sequences and errors. The level of redundancy is determined by expected errors in sequencing and synthesis, as well as DNA degradation. Next, it applies inner coding, which ultimately converts the bits to DNA sequences. The resulting set of sequences is surrounded by a primer pair chosen from the library based on (low) level of overlap with payloads.



The decoding process starts by clustering reads based on similarity, and finding a consensus between the sequences in each cluster to reconstruct the original sequences, which are then decoded back to digital data.

The data longevity and information density of current DNA data storage systems already surpass those of traditional storage systems, but the cost and the read and write speeds do not.

Storing one megabyte of data in DNA with existing technology costs hundreds of dollars, compared with less than $0.0001 per year using tape, the standard for archival data storage.

The price of DNA storage will undoubtedly drop substantially as the costs of DNA synthesis and sequencing fall.

The more pressing challenge is that DNA synthesis and sequencing are inherently slow.

DNA synthesis and sequencing DNA can be extensively parallelized, their slow speeds limit the amount of data that can be written and read in a given time interval. The bottleneck for both cost and speed is synthesis.

A fully automated DNA drive would include synthesis and sequencing technology, components to store and handle the DNA, as well as a supply of chemicals.

Exam questions

1 Bioinformatics (PL)

(*a*) What are the usage and the limitations of the Bootstrap technique in phylogeny? [6 marks]

Answer: This is a procedure of resampling of the sites in an alignment and tree reconstructions of all the pseudo alignments; it depends on the size of the alignment (length of the sequences and their number). The percentage of times each interior branch is given a value of 1 is noted. This is known as the bootstrap value. As a general rule, if the bootstrap value for a given interior branch is 95% or higher, then the topology at that branch is considered correct.The presence of several repeated columns decreases the amount of information in each pseudoalignment.

(*c*) How can you evaluate the results obtained (number of clusters and their relative position) using the K means algorithm for clustering? [5 marks]

Answer: The quality of cluster could be assessed by ratio of distance to nearest cluster and cluster diameter. A cluster can be formed even when there is no similarity between clustered patterns. This occurs because the algorithm forces k clusters to be created. Linear relationship with the number of data points; Complexity is O(nKI) where n = number of points, K = number of clusters, I = number of iterations.

Exam questions

Bioinformatics

(*a*) Discuss the space–time complexity of dynamic programming algorithms in sequence alignment. [7 marks]

(*b*) Discuss with one example how to score a multiple sequence alignment. [5 marks]

Exam questions

1. Give the alignment matrix of the sequences `AATCGCGCGGT' and `ATGCGCCGT' assuming the following costs: Cost(a,a)=0; Cost(a,b)=3 when a ≠ b, Cost(a,-)=Cost(-,a)=2.
2. How would you set the function Cost in order to compute the longest subsequence common to x and y?
3. Describe the differences between the algorithms for global and local alignments
4. Which of the following reasons would lead you to use the Smith-Waterman local alignment algorithm instead of the Needleman-Wunsch global alignment algorithm?

Select all appropriate answers.

(a) Computer memory is too limited to compute the optimal global alignment.
(b) One wants to identify common protein domains in the two sequences.
(c) The sequences have very different lengths.
(d) Smith-Waterman is faster than Needleman-Wunsch on long sequences.

5. Describe the notion of a parsimonious phylogeny for a finite set of sequences and the hypothesis assumed on them

8 Bioinformatics (PL)

Given the two DNA sequences: GCACTT and CCCAAT

(*a*) Compute the alignment (using the edit graph) and the final score with the following rules: match score $= +1$, mismatch $= -1$, gap penalty $= -1$. [4 marks]

(*b*) Discuss how the alignment score and the quality of the result depend on the match score, mismatch, and gap penalty. [6 marks]

(*c*) Generate four, short DNA sequences (a,b,c,d) such that their relations as a tree are approximately the following: ((a,b),(c,d)). [5 marks]

(*d*) How is the score matrix used in phylogenetic tree building techniques? [5 marks]

COMPUTER SCIENCE TRIPOS Part II – 2013 – Paper 9

11 Bioinformatics (PL)

(*a*) What are the usage and the limitations of the Bootstrap technique in phylogeny? [6 marks]

(*b*) We often use Hidden Markov Models (HMM) to predict a pattern (for instance the exons). How can you compute the number of True Positives, True Negatives, False Positives and False Negatives and use them to evaluate your HMM? [6 marks]

(*c*) How can you evaluate the results obtained (number of clusters and their relative position) using the K means algorithm for clustering? [5 marks]

HMM

(*b*) We often use Hidden Markov Models (HMM) to predict a pattern (for instance the exons). How can you compute the number of True Positives, True Negatives, False Positives and False Negatives and use them to evaluate your HMM?

[6 marks]

Answer:

(*i*) be predicted to occur: Predicted Positive (PP)

(*ii*) be predicted not to occur: Predicted Negative (PN)

(*iii*) actually occur: Actual Positive (AP)

(*iv*) actually not occur: Actual Negative (AN)

(*v*) True Positive $TP = PP \bigcap AP$

(*vi*) True Negative $TN = PN \bigcap AN$

(*vii*) False Negative $FN = PN \bigcap AP$

(*viii*) False Positive $FP = PP \bigcap AN$

(*ix*) Sensitivity: probability of correctly predicting a positive example Sn = TP/(TP + FN)

(*x*) Specificity: probability of correctly predicting a negative example Sp = TN/(TN + FP)
or

(*xi*) probability that positive prediction is correct Sp = TP/(TP + FP)
