

Bioinformatics algorithms

- Simon Frost, sdf22@cam.ac.uk
- No biology in the exam questions
- You need to know only the biology in the slides to understand the reason for the algorithms
- Partly based on book: Compeau and Pevzner Bioinformatics algorithms (chapters 3,5 in Vol I,7-10 in Vol II)
 - Also Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids Richard Durbin, Sean R. Eddy, Anders Krogh, Graeme Mitchison
- Color slides from the course website



Sequence Alignment Outline

- From Sequence Comparison to Biological Insights
- The Alignment Game and the Longest Common Subsequence
- The Manhattan Tourist Problem
- Dynamic Programming and Backtracking Pointers
- From Manhattan to the Alignment Graph
- From Global to Local Alignment
- Penalizing Insertions and Deletions in Sequence Alignment
- Space-Efficient Sequence Alignment
- Multiple Sequence Alignment

DNA: 4-letter alphabet, A (adenosine), T (thymine), C (cytosine) and G (guanine). In the double helix A pairs with T, C with G

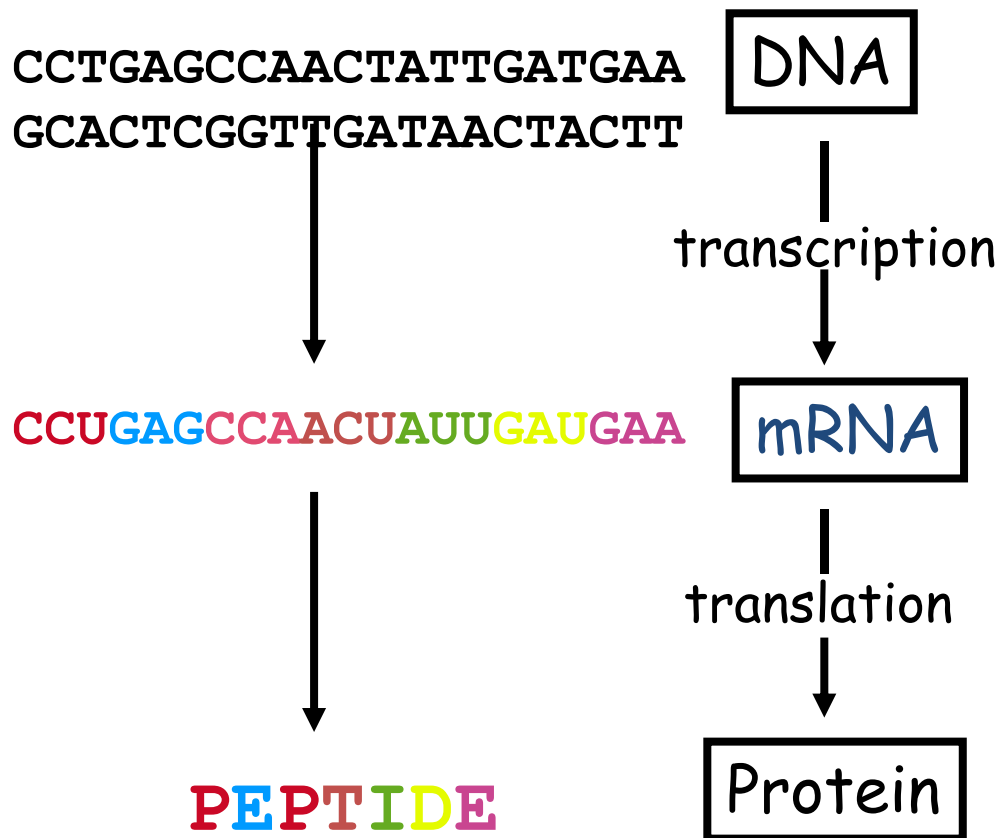
Gene: hereditary information located on the chromosomes and consisting of DNA.

RNA: same as DNA but T -> U (uracil)

3 letters (triplet – a codon) code for one amino acid in a protein.

Proteins: units are the 20 amino acids A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y.

Genome: an organism's genetic material



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

Why Align Sequences?

- Biological sequences can be represented as a vector of characters:
 - A, C, T and G for DNA
- We align sequences such that sites with the same ancestry are aligned
- This is a requirement e.g. for downstream phylogenetic analyses

The Alignment Game

A T G T T A T A
A T C G T C C

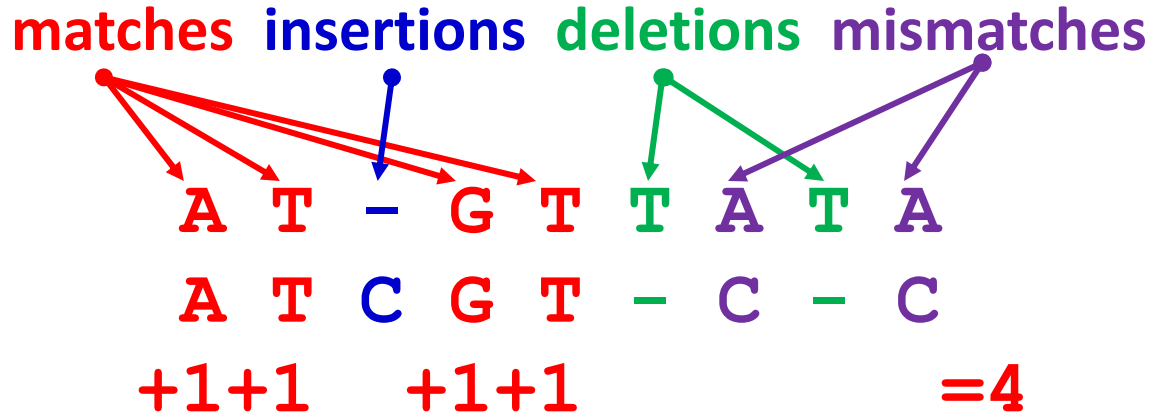
Alignment Game (maximizing the number of points):

- Remove the 1st symbol from each sequence
 - 1 point if the symbols match, 0 points if they don't match
- Remove the 1st symbol from one of the sequences
 - 0 points

The Alignment Game

A T - G T T A T A
A T C G T - C - C
+1+1 +1+1 =4

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

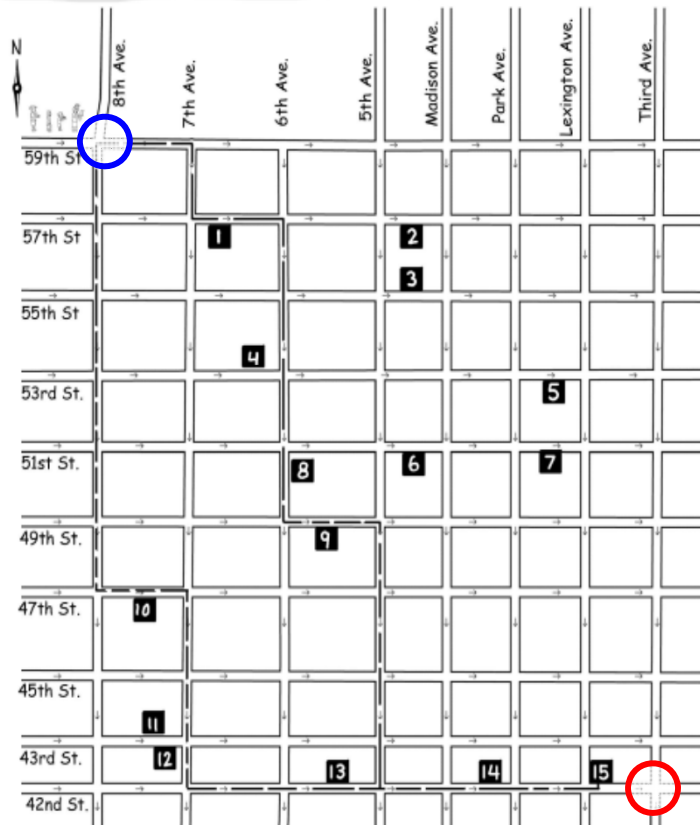
A T - G T T A T A
A T C G T - C - C

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

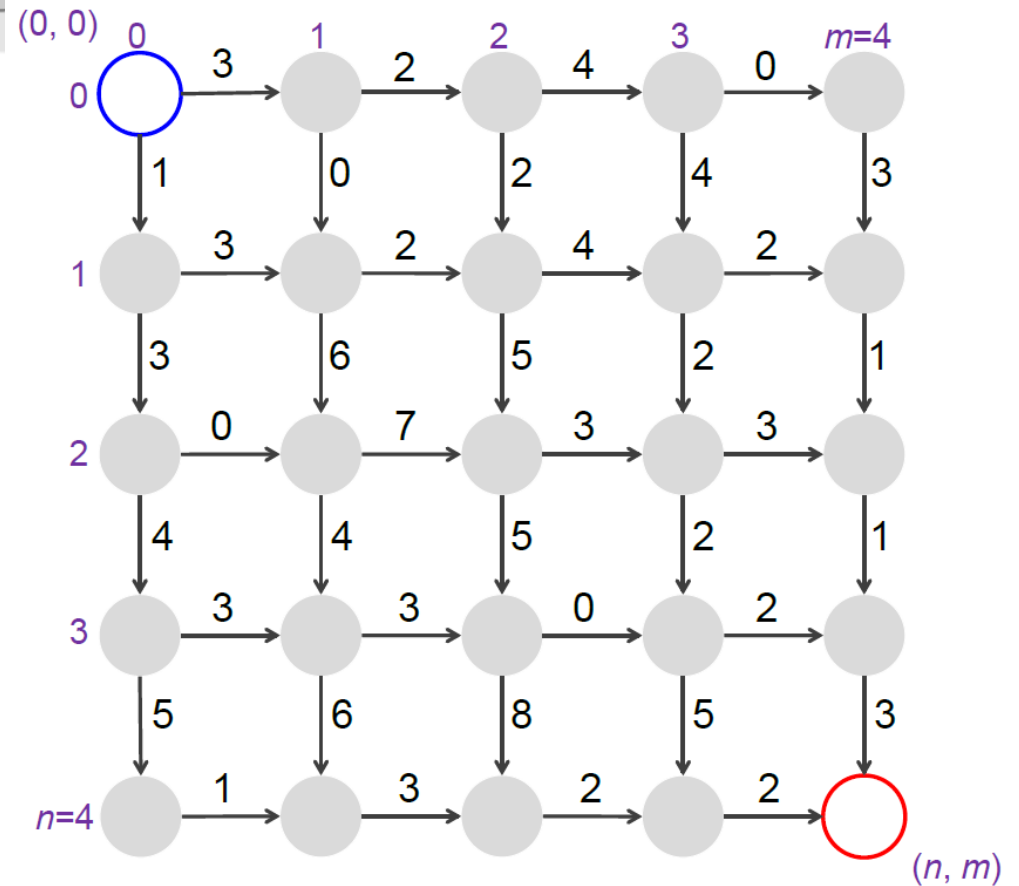
Longest Common Subsequence Problem: Find a longest common subsequence of two strings.

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

From Manhattan to a Grid Graph



- | | |
|-----------------------------|---|
| 1 Carnegie Hall | 9 The Today Show |
| 2 Tiffany & Co. | 10 Paramount Building |
| 3 Sony Building | 11 NY Times Building |
| 4 Museum of Modern Art | 12 Times Square |
| 5 Four Seasons | 13 General Society of Mechanics and Tradesmen (a must see!) |
| 6 St. Patrick's Cathedral | 14 Grand Central Terminal |
| 7 General Electric Building | 15 Chrysler Building |
| 8 Radio City Music Hall | |

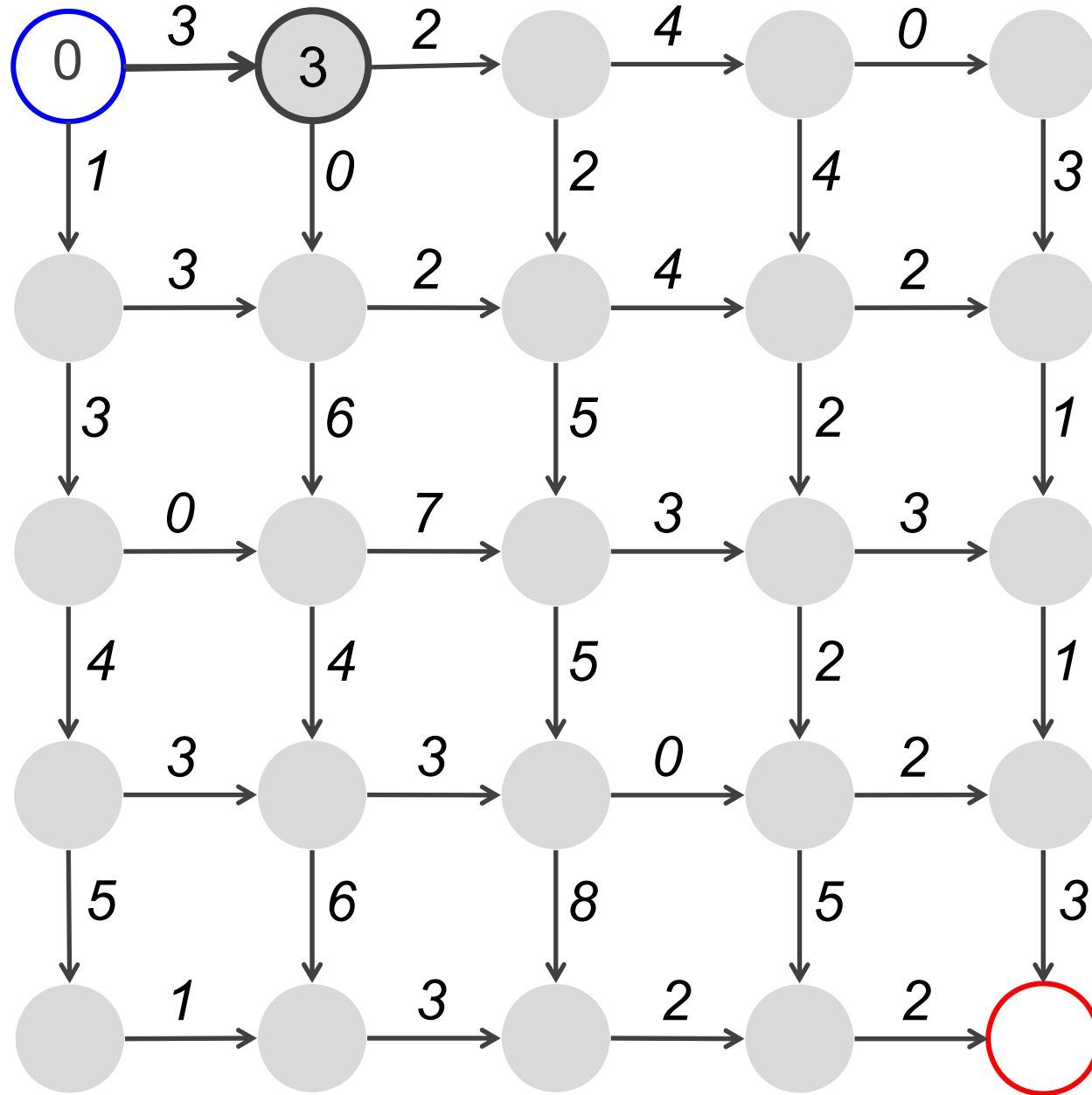


Manhattan Tourist Problem

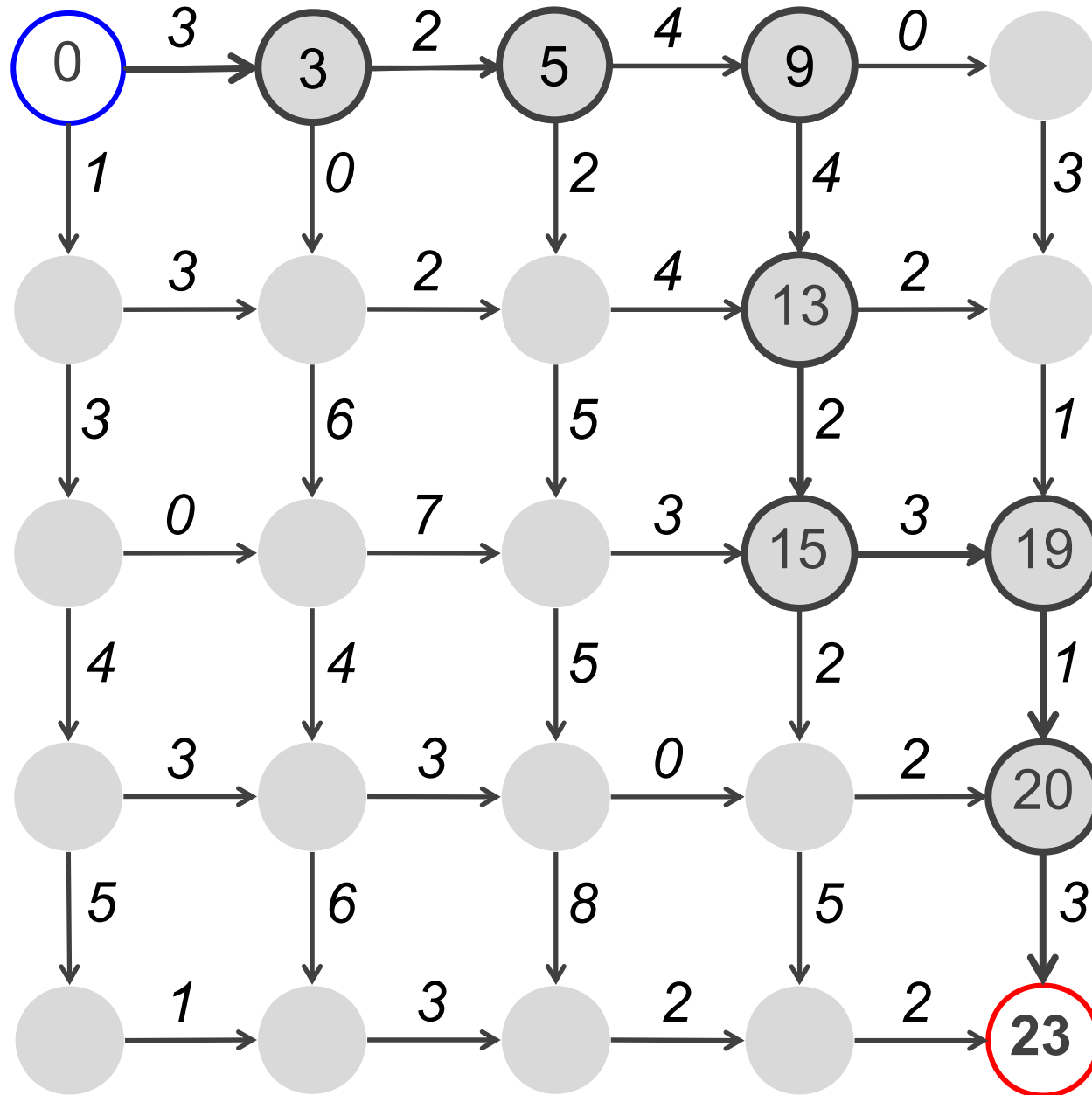
Manhattan Tourist Problem: Find a longest path in a rectangular city grid.

- **Input:** A weighted rectangular grid.
- **Output:** A longest path from the source to the sink in the grid.

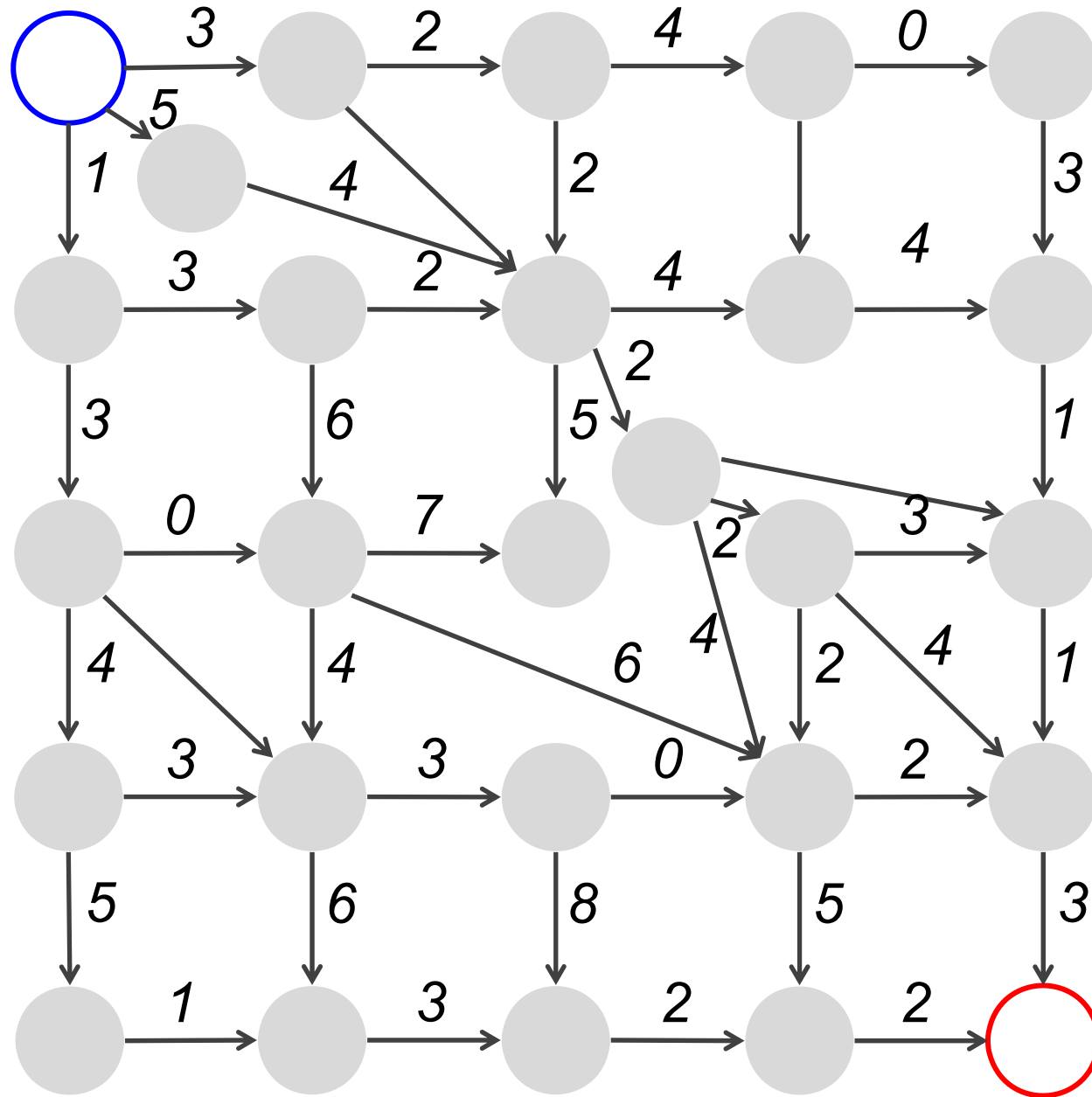
Greedy algorithm?



**Greedy
algorithm?**



From a
regular to an
irregular grid

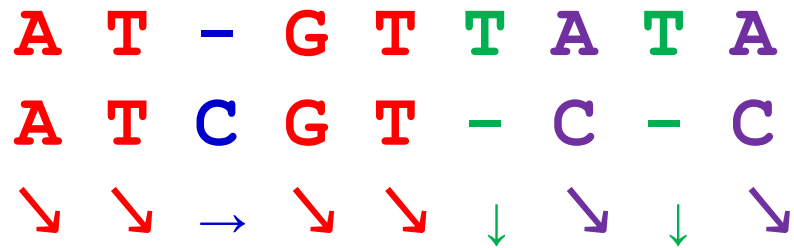


Search for Longest Paths in a Directed Graph

Longest Path in a Directed Graph Problem: Find a longest path between two nodes in an edge-weighted directed graph.

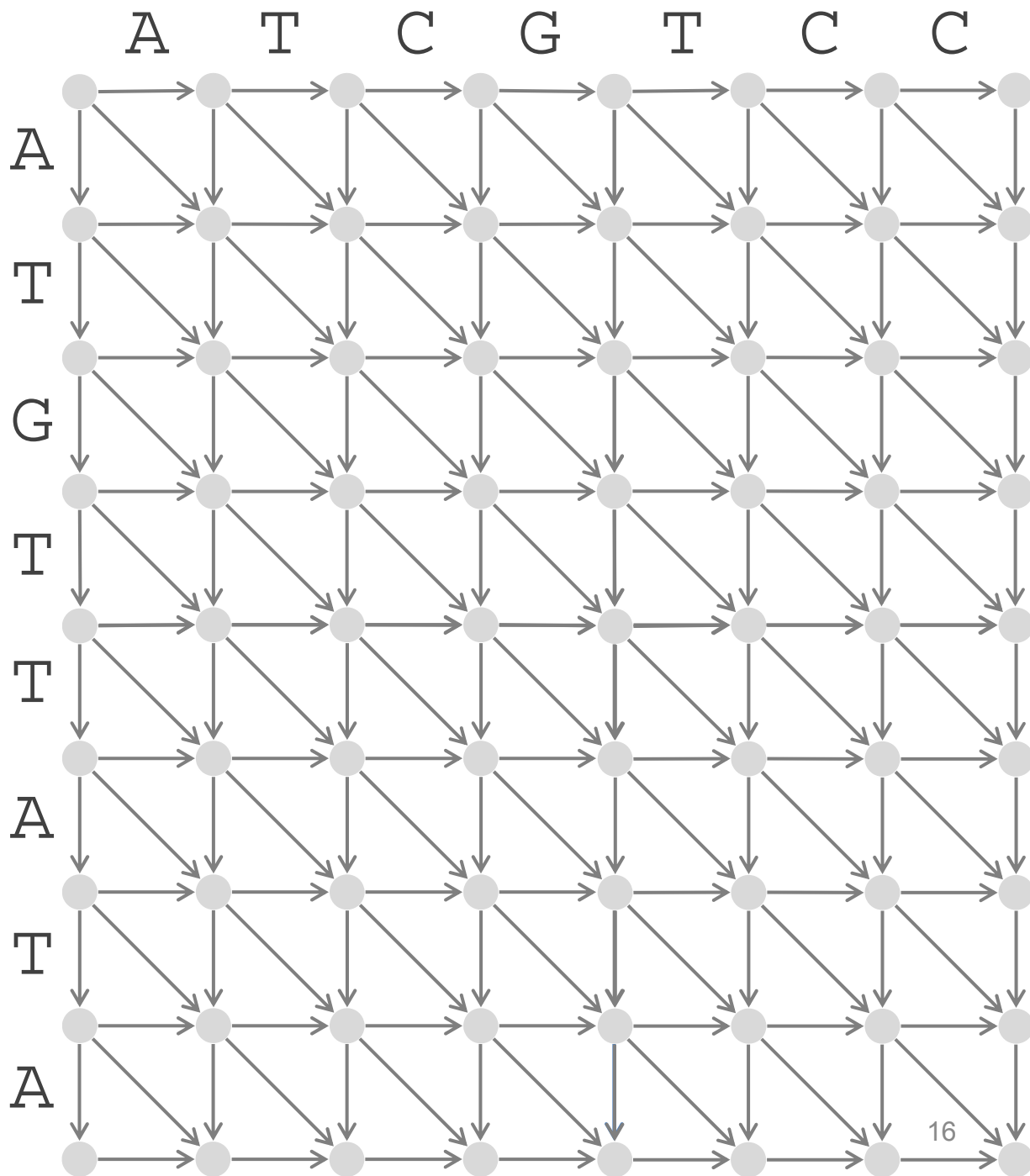
- **Input:** An edge-weighted directed graph with source and sink nodes.
- **Output:** A longest path from source to sink in the directed graph.

Do You See a Connection between the Manhattan Tourist and the Alignment Game?



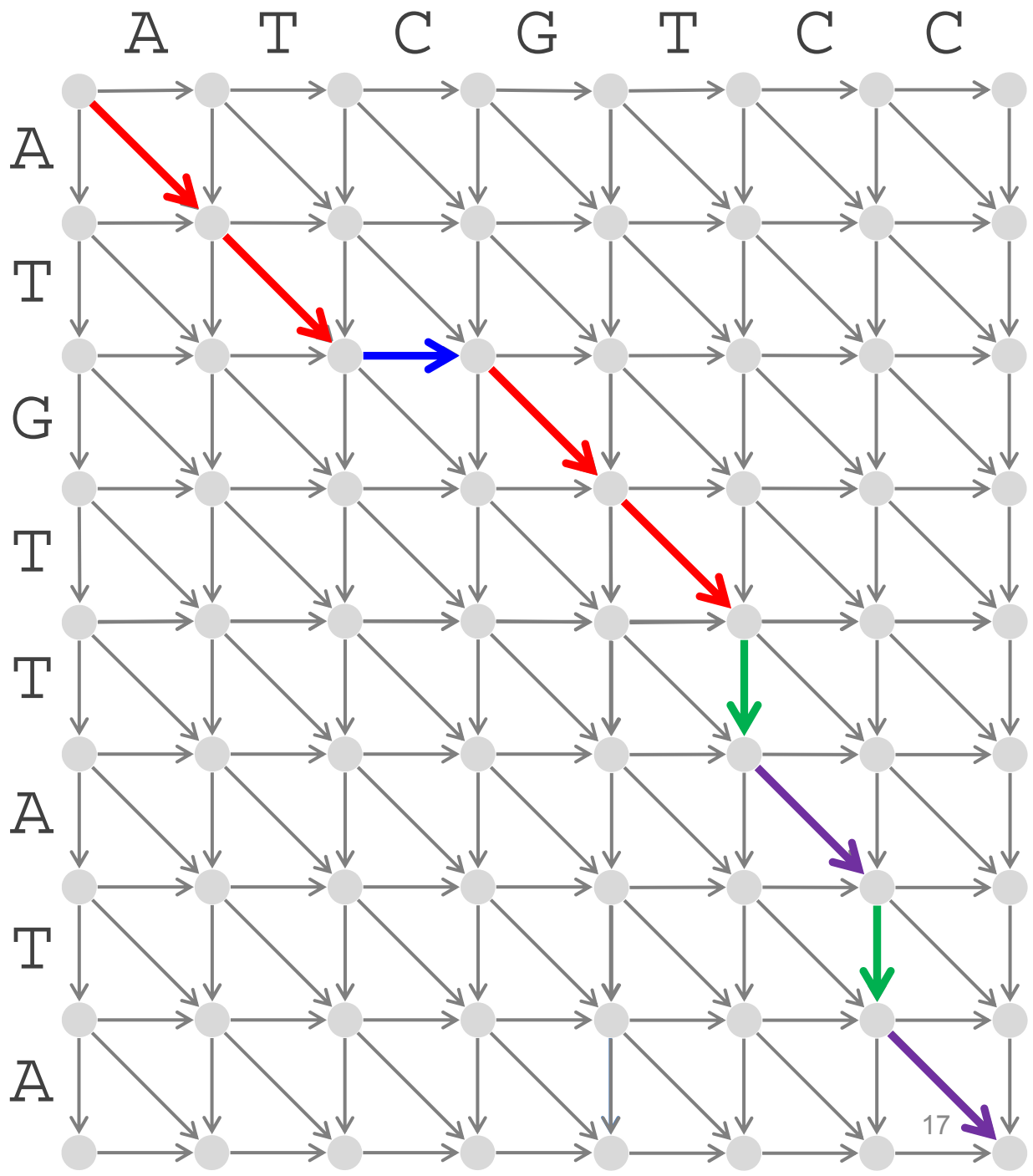
?
alignment → path

A T - G T T A T A
A T C G T - C - C
↘ ↘ → ↘ ↘ ↓ ↘ ↓ ↘



alignment → path

A T - G T T A T A
 A T C G T - C - C
 ↘ ↘ → ↘ ↘ ↓ ↘ ↓ ↘

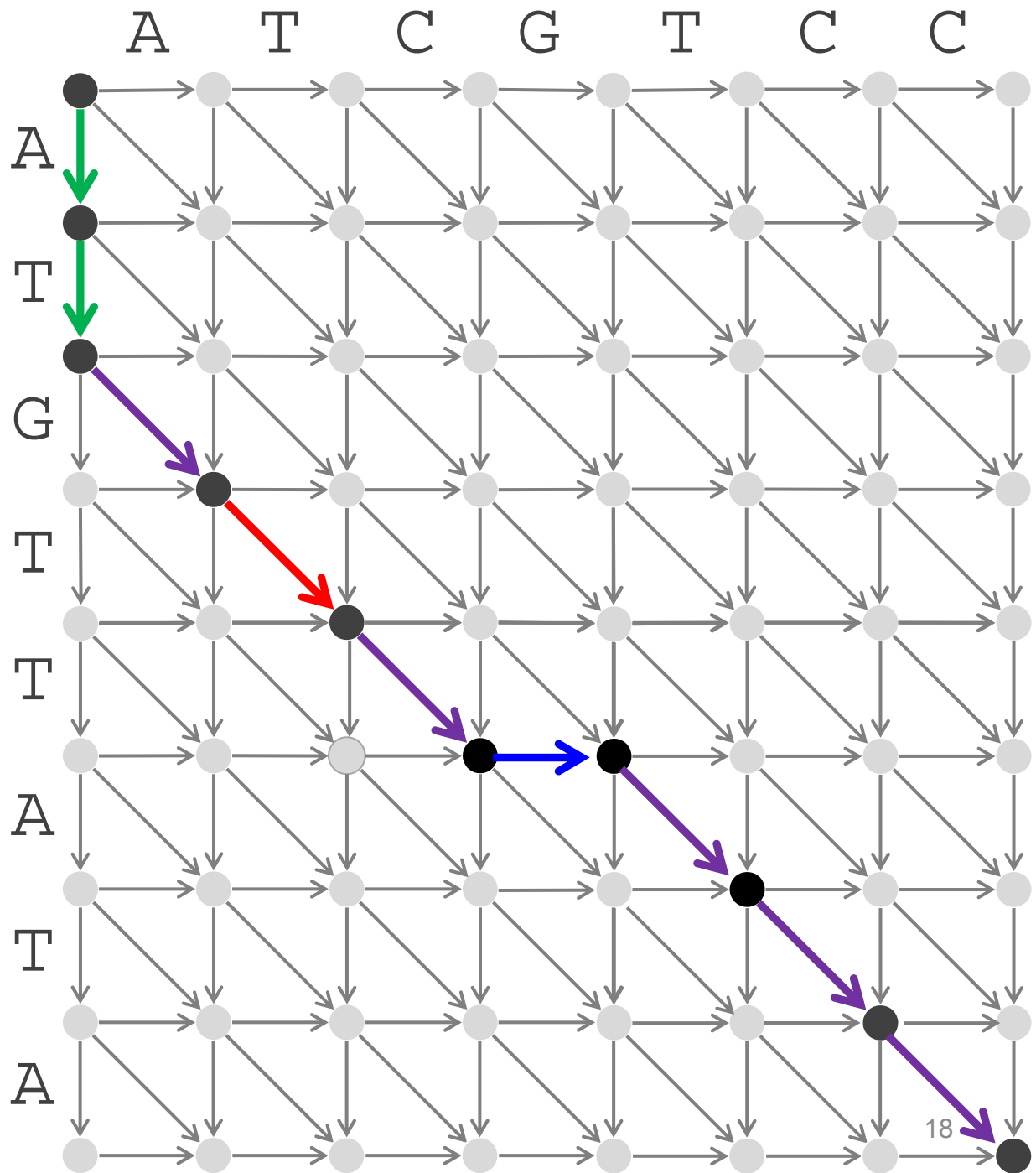


?

path → alignment

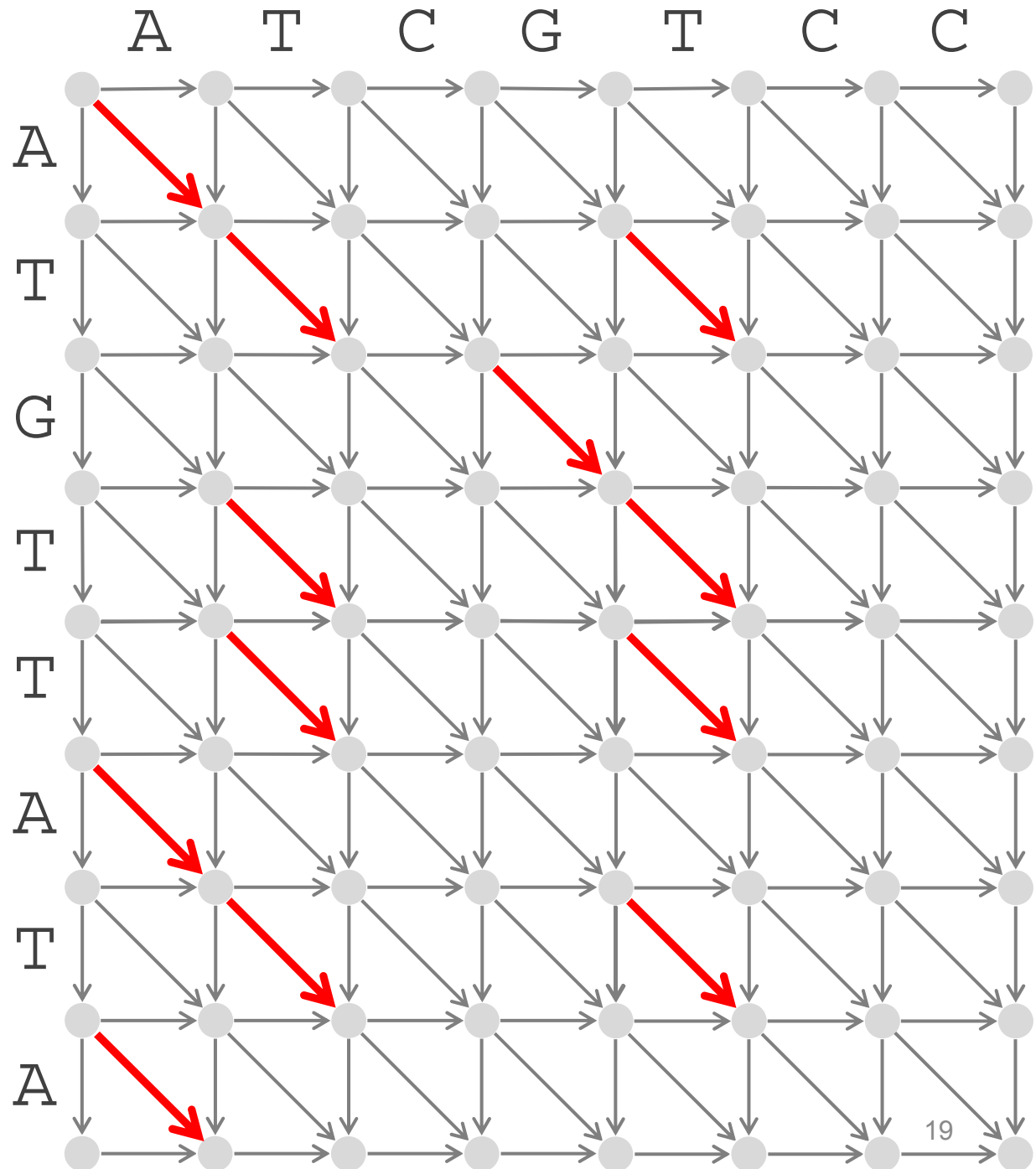
A T G T T - A T A
 - - A T C G T C C
 ↓ ↓ ↘ ↙ ↘ → ↘ ↘

highest-scoring alignment
 =
 longest path in a
 properly built
 Manhattan

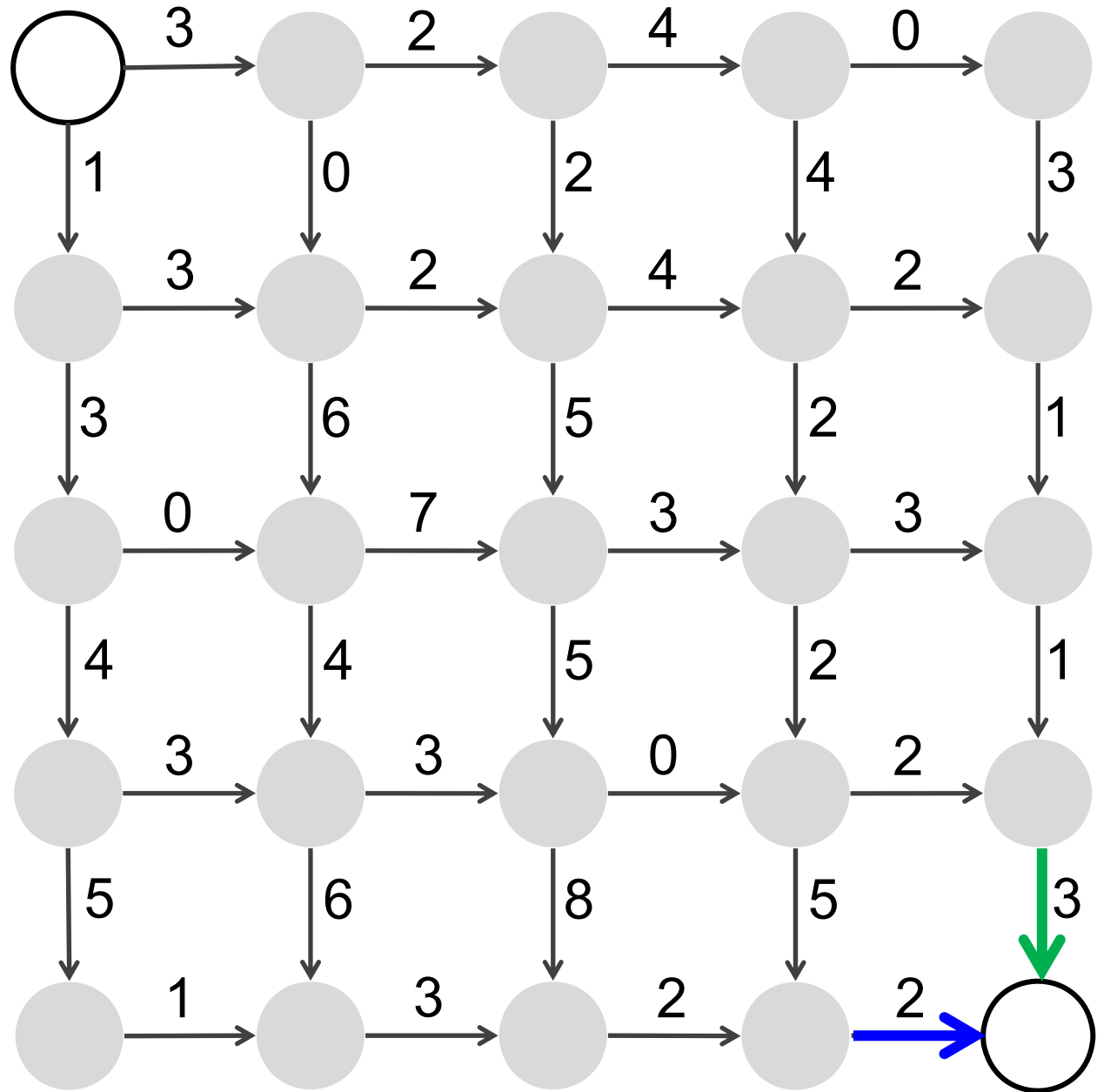


How to built a
Manhattan for the
Alignment Game
and the
Longest Common
Subsequence
Problem?

Diagonal red edges
correspond to
matching symbols
and have scores 1



There are only 2 ways to arrive to the sink: by moving **South** ↓ or by moving **East** →



South
or
East?

South or East?

SouthOrEast(n,m)

if $n=0$ and $m=0$

return 0

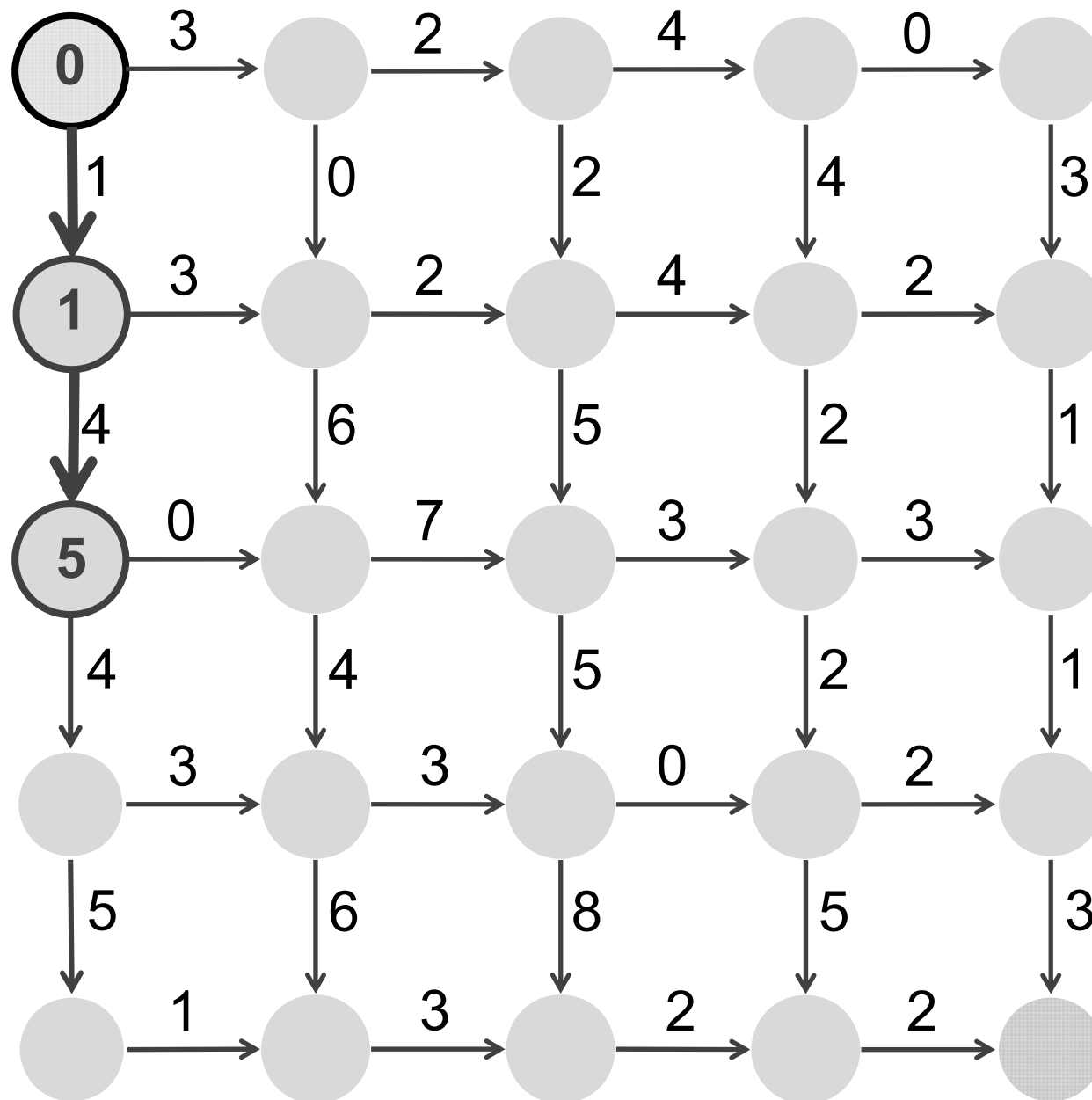
if $n>0$ and $m>0$

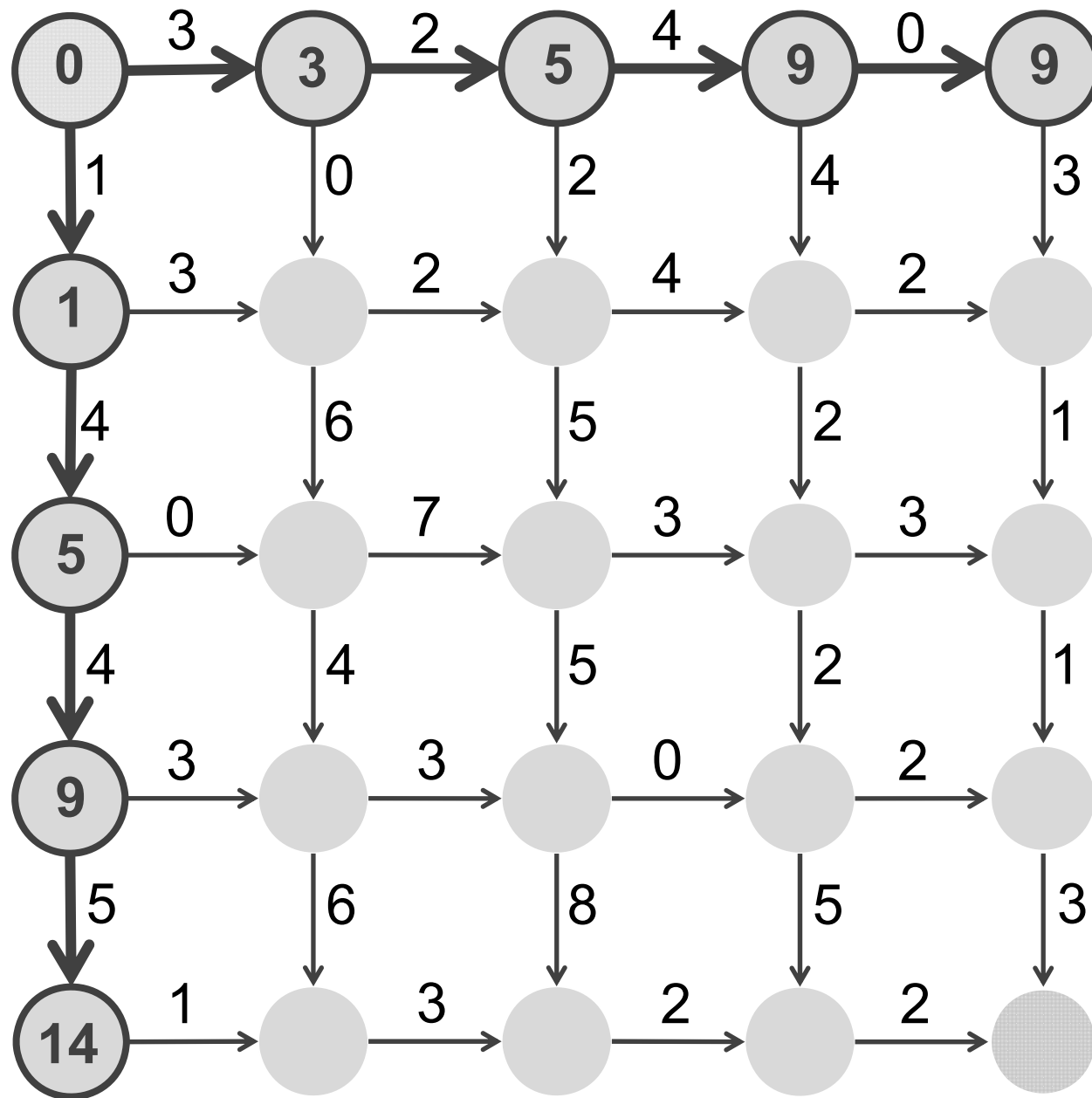
$x \leftarrow$ **SouthOrEast**($n-1,m$)+weight of edge “”into (n,m)

$y \leftarrow$ **SouthOrEast**($n,m-1$)+ weight of edge “”into (n,m)

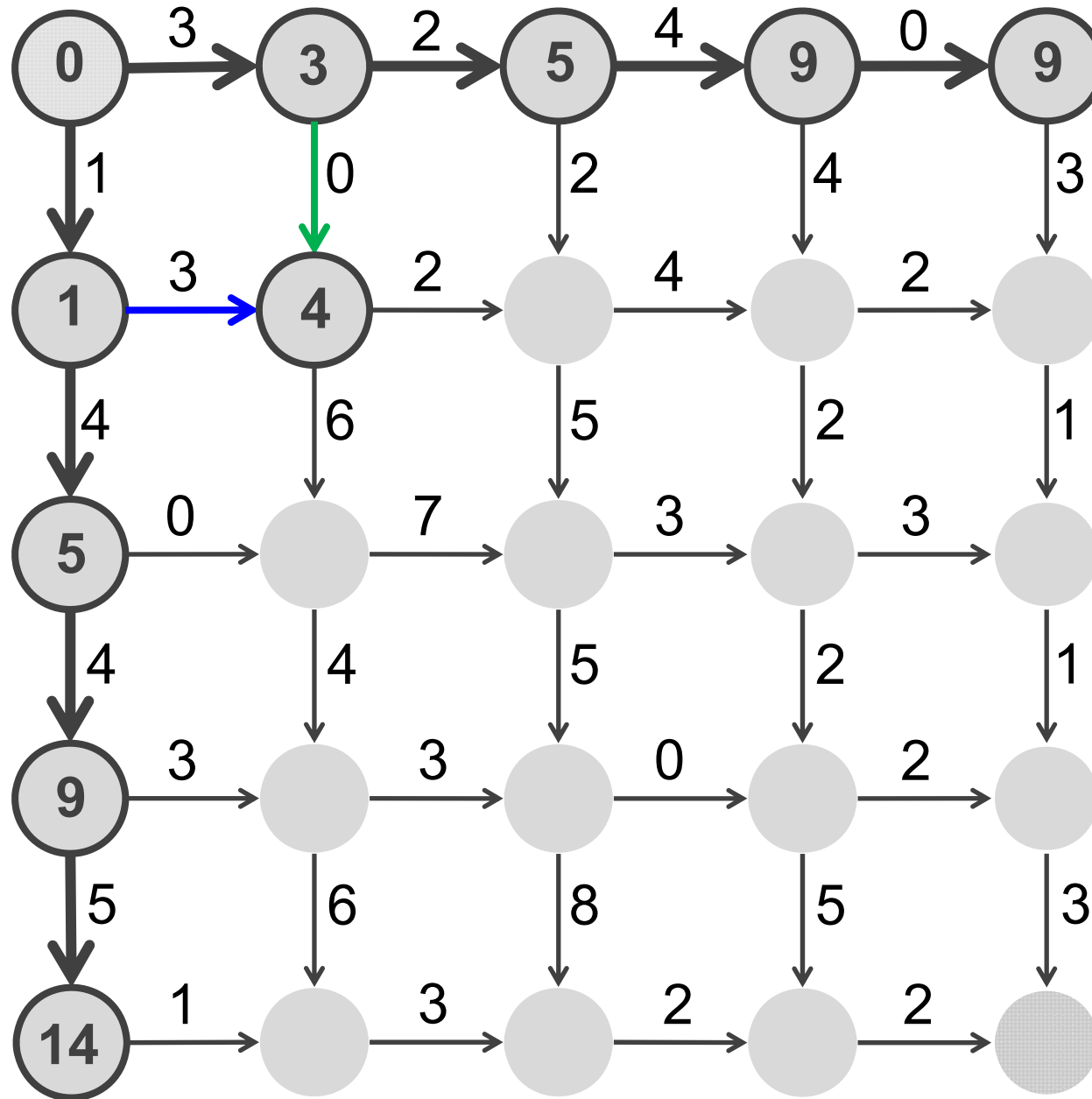
return $\max\{x,y\}$

return -infinity

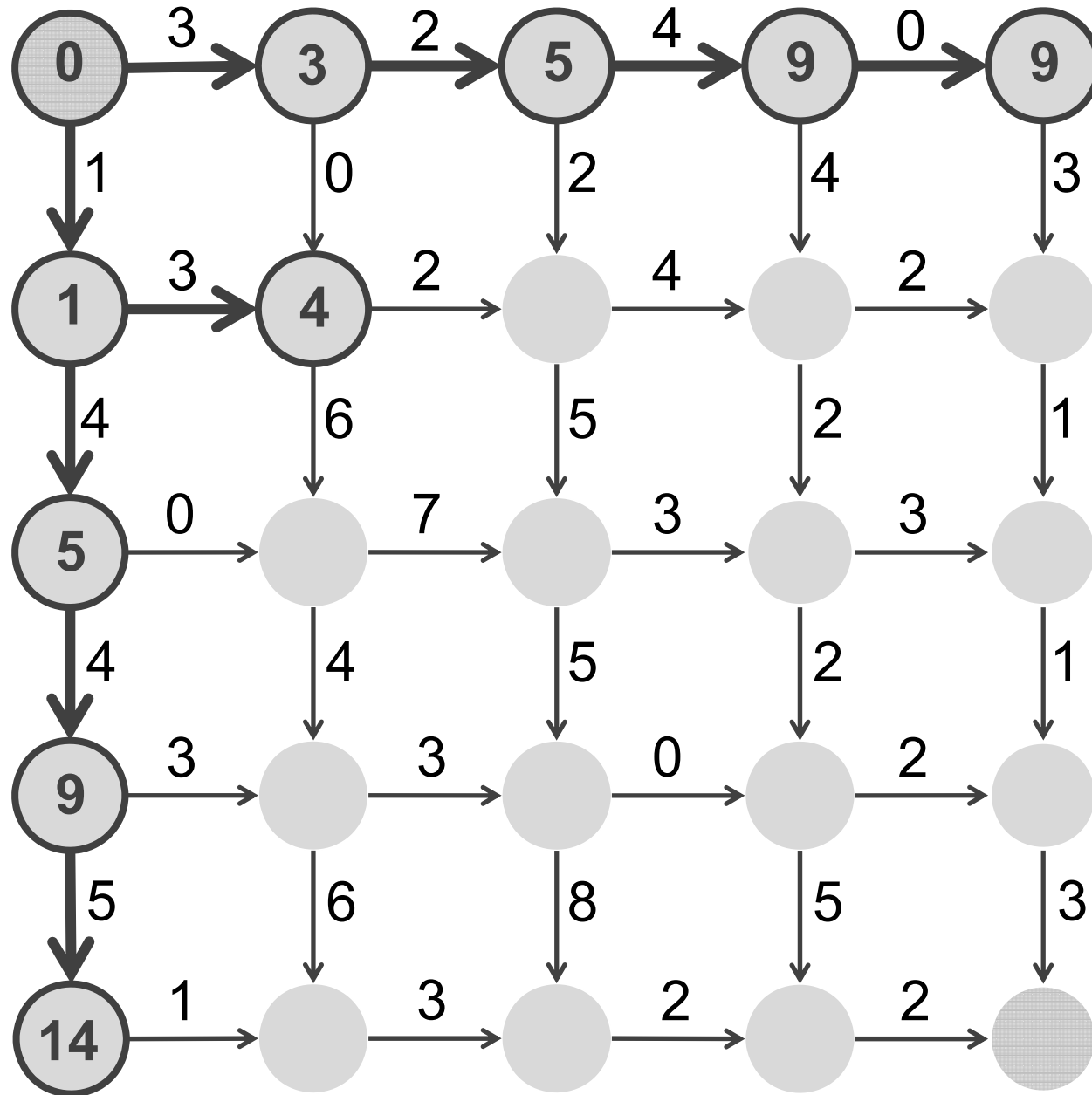
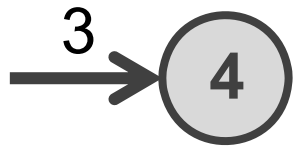




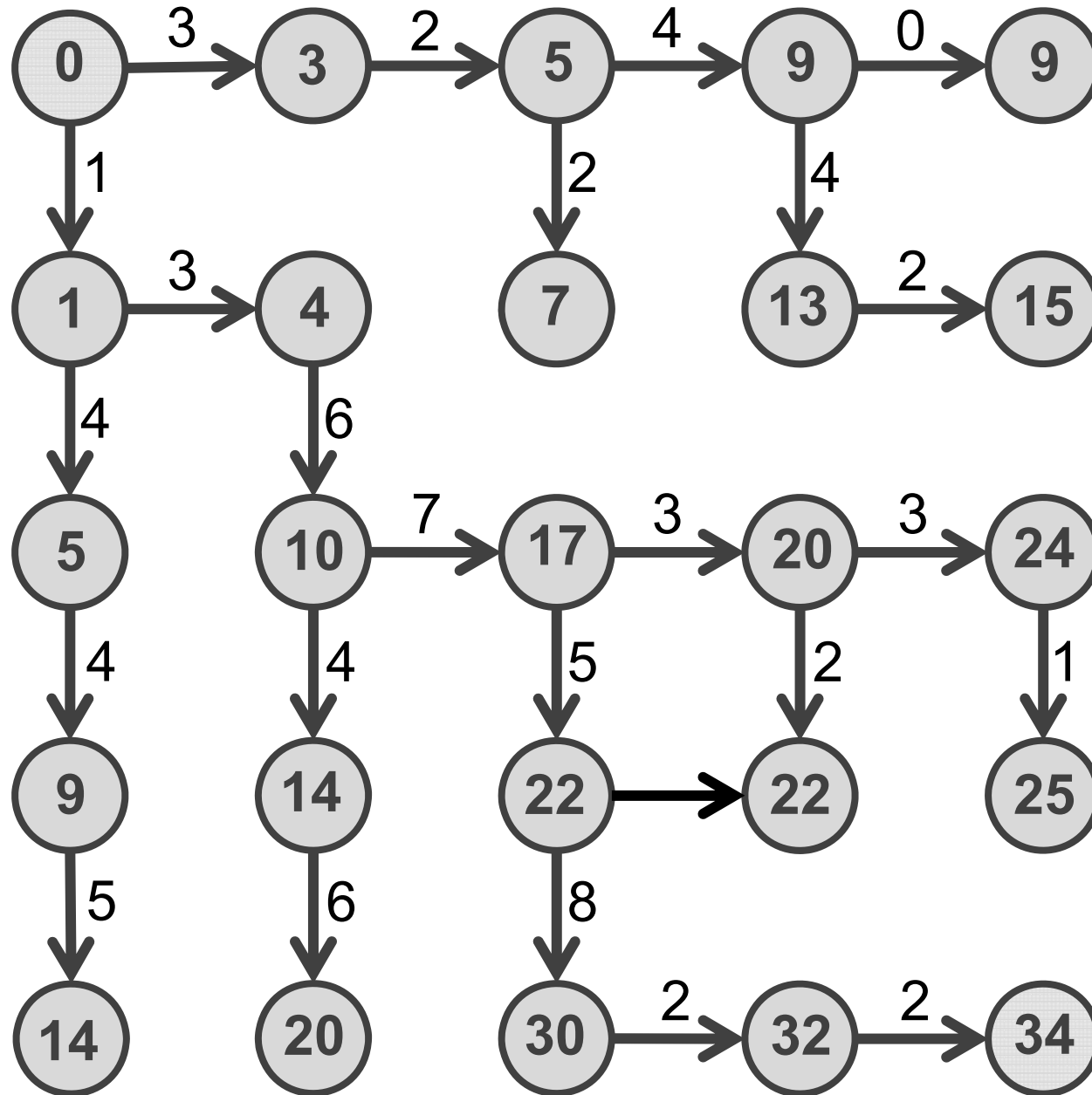
South
or
East?
 $1+3 > 3+0$



We arrived to (1,1) by the **bold** edge:



Backtracking pointers:
the best way
to get to
each node



Dynamic programming

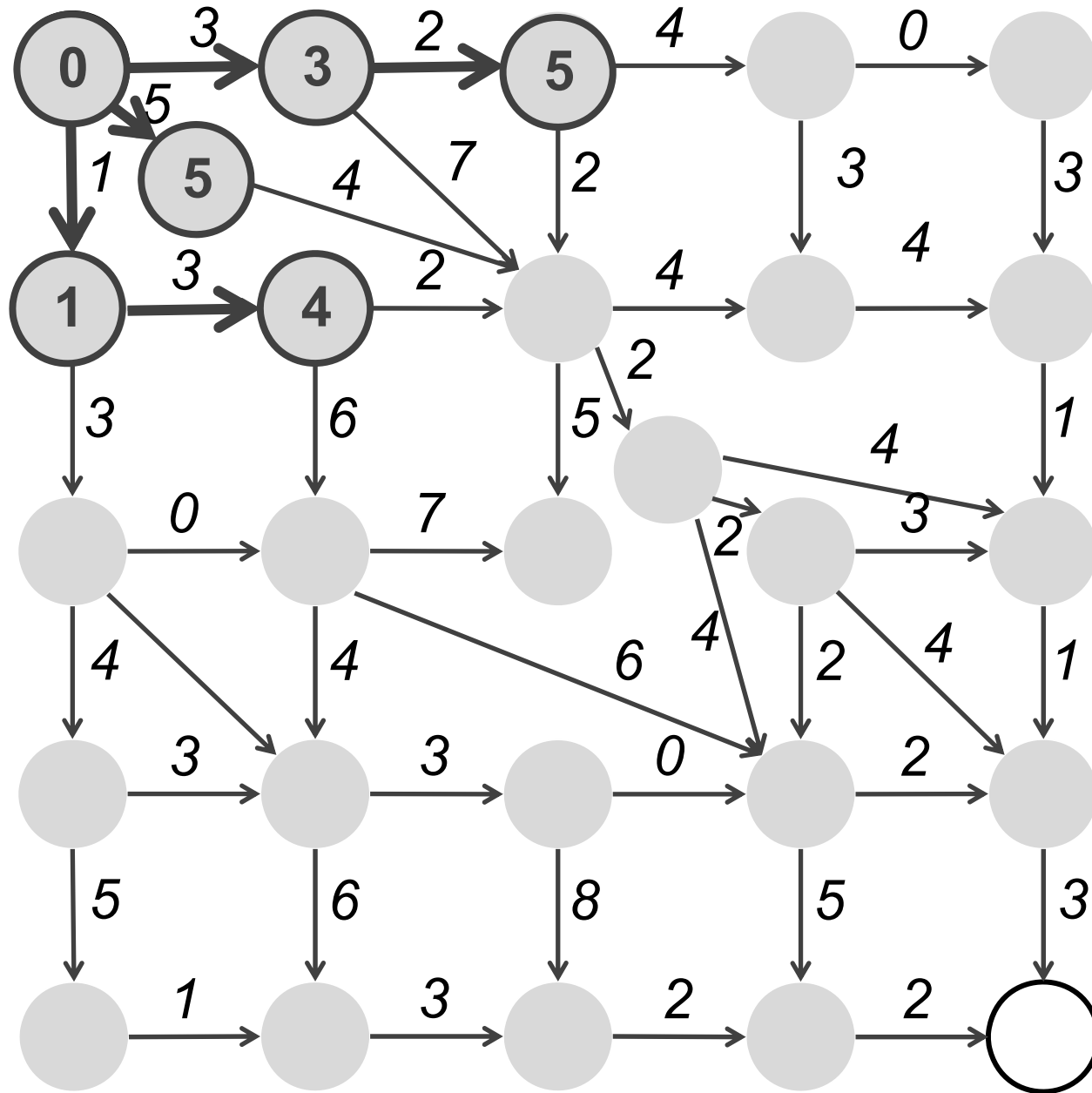
- Break down complex problem into simpler subproblems
- Solving subproblems once
- Store solutions
 - 'Memoization'

Dynamic Programming Recurrence

$s_{i,j}$: the length of a longest path from $(0,0)$ to (i,j)

$$s_{i,j} = \max \left\{ \begin{array}{l} s_{i-1,j} + \text{weight of edge } \downarrow \text{ into } (i,j) \\ s_{i,j-1} + \text{weight of edge } \rightarrow \text{ into } (i,j) \end{array} \right.$$

How does the recurrence change for this graph?



$$s_a = \max_{\text{all predecessors } b \text{ of node } a} \{s_b + \text{weight of edge from } b \text{ to } a\}$$

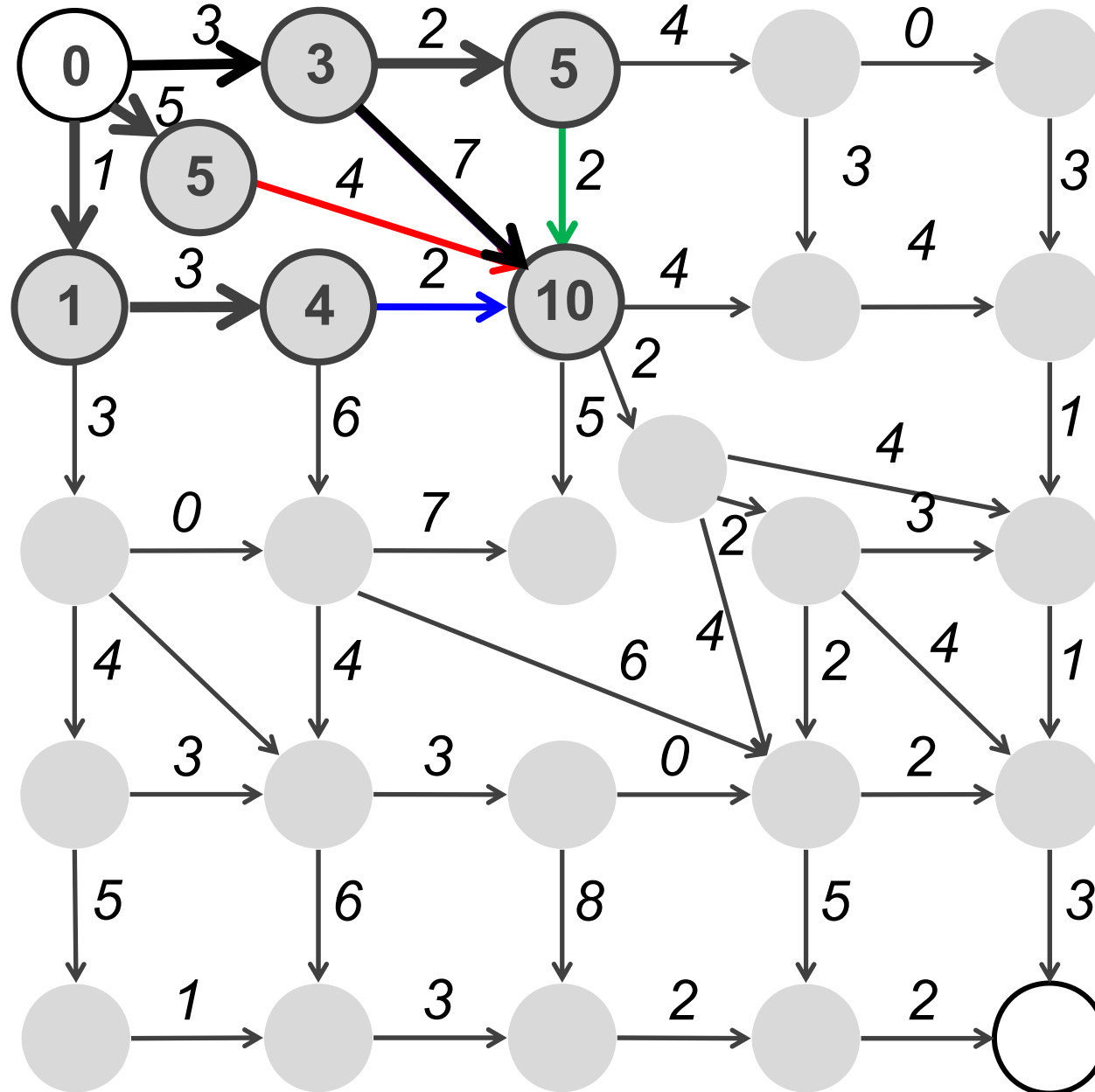
4 choices:

$$5 + 2$$

$$3 + 7$$

$$5 + 4$$

$$4 + 2$$



$$s_a = \max_{\text{all predecessors } b \text{ of node } a} \{s_b + \text{weight of edge from } b \text{ to } a\}$$

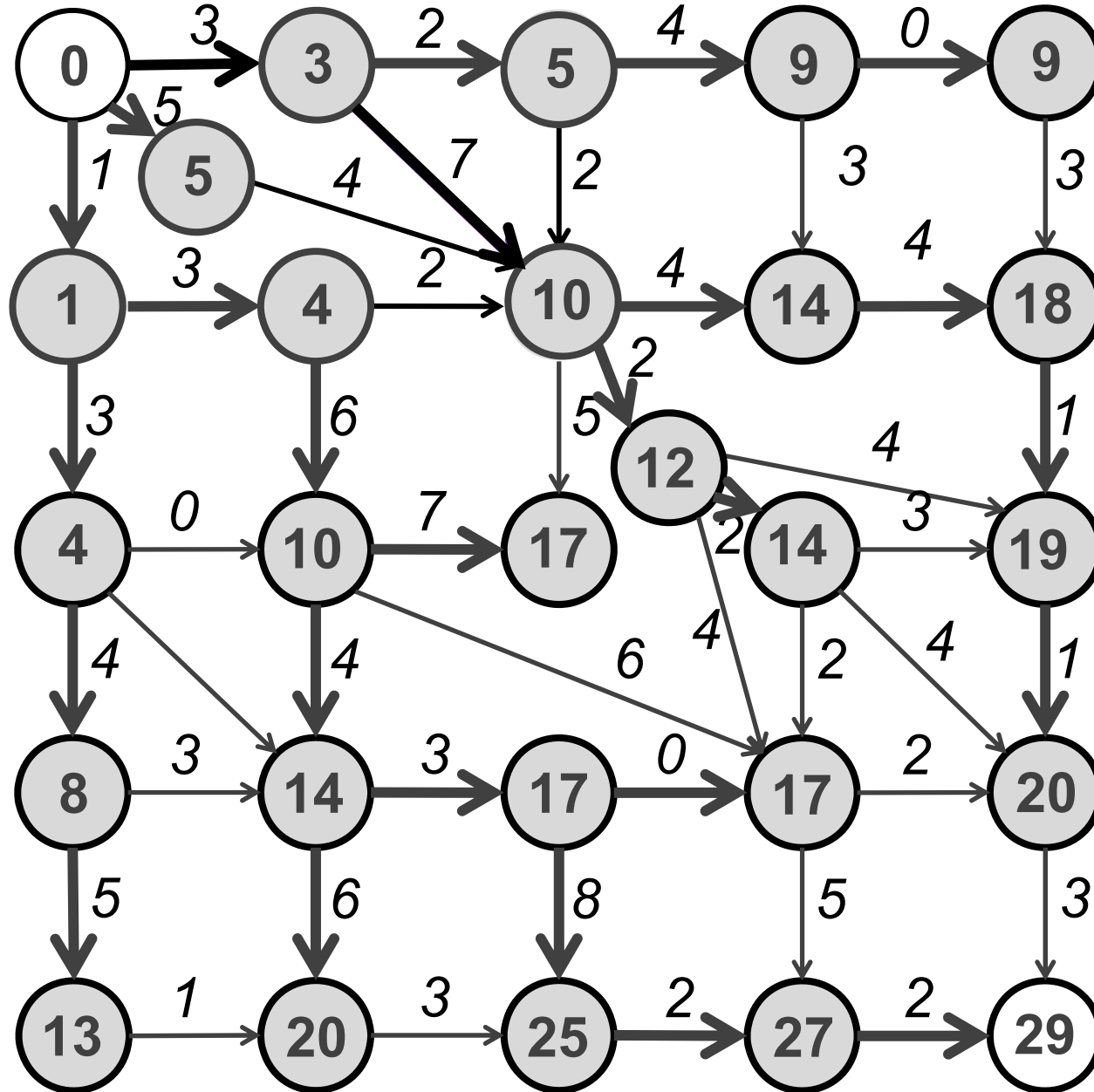
4 choices:

$$5 + 2$$

$$3 + 7$$

$$5 + 4$$

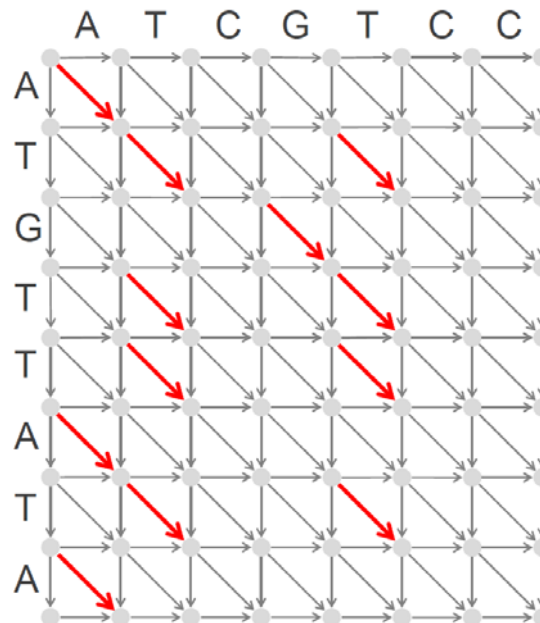
$$4 + 2$$



Dynamic Programming Recurrence for the Alignment Graph

$s_{i,j}$: the length of a longest path from (0,0) to (i,j)

$$s_{i,j} = \max \begin{cases} s_{i-1,j} + \text{weight of edge } \downarrow \text{ into } (i,j) \\ s_{i,j-1} + \text{weight of edge } \rightarrow \text{ into } (i,j) \\ s_{i-1,j-1} + \text{weight of edge } \searrow \text{ into } (i,j) \end{cases}$$

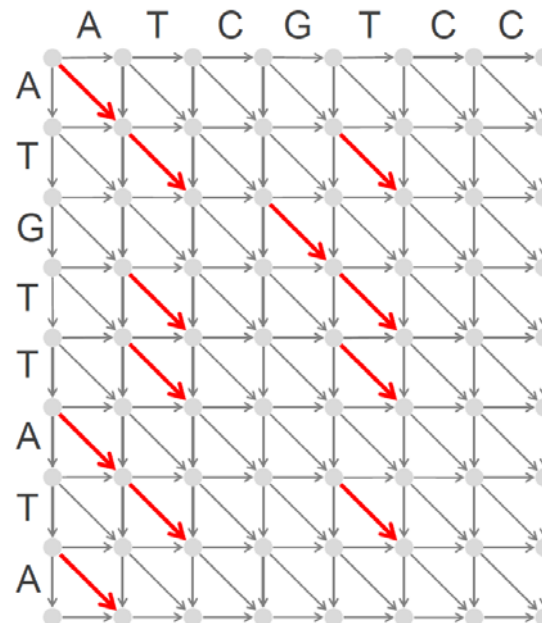


red edges \searrow – weight 1
other edges – weight 0

Dynamic Programming Recurrence for the Longest Common Subsequence Problem

$s_{i,j}$: the length of a longest path from $(0,0)$ to (i,j)

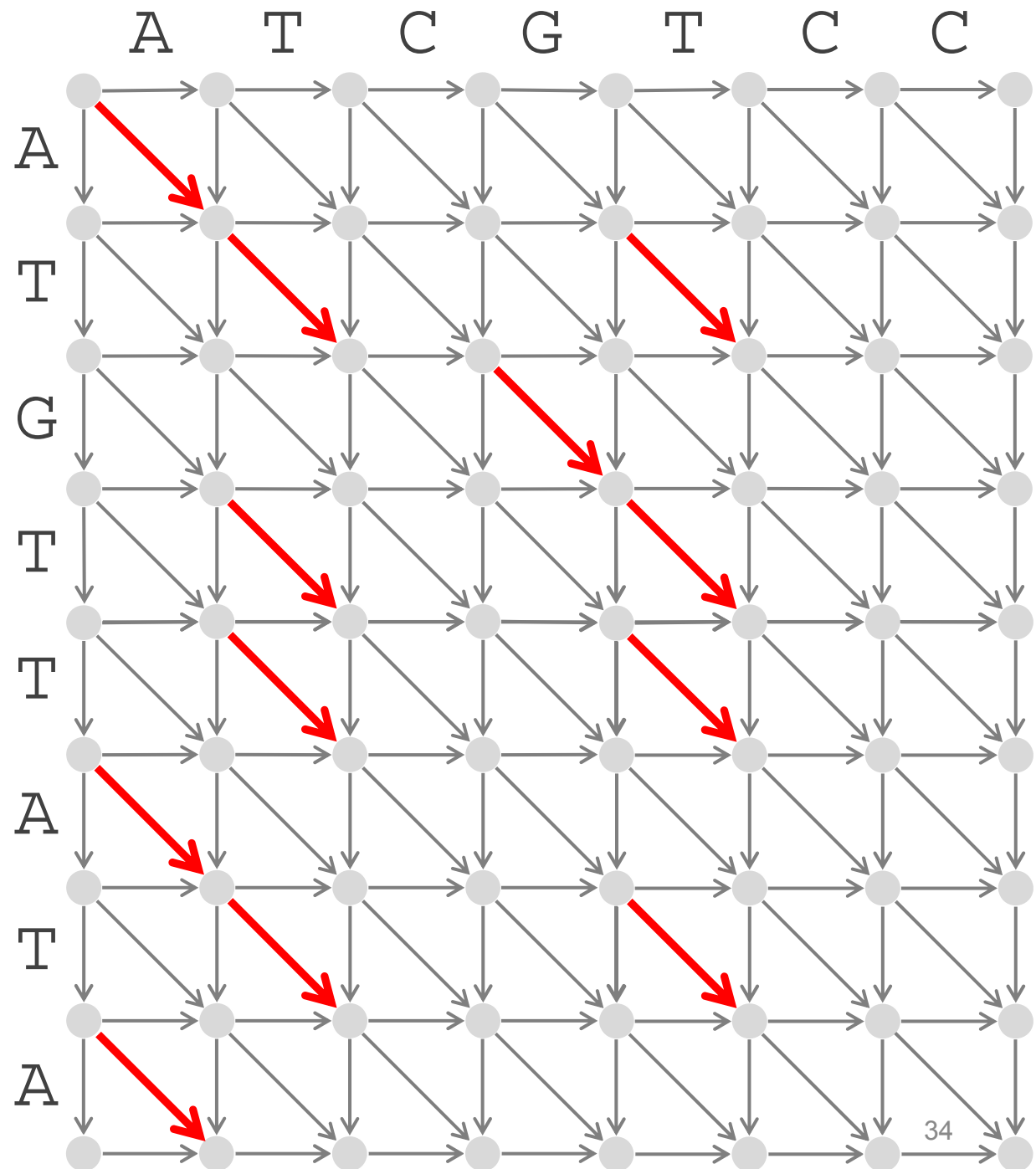
$$s_{i,j} = \max \begin{cases} s_{i-1,j} + 0 \\ s_{i,j-1} + 0 \\ s_{i-1,j-1} + 1, \text{ if } v_i = w_j \end{cases}$$



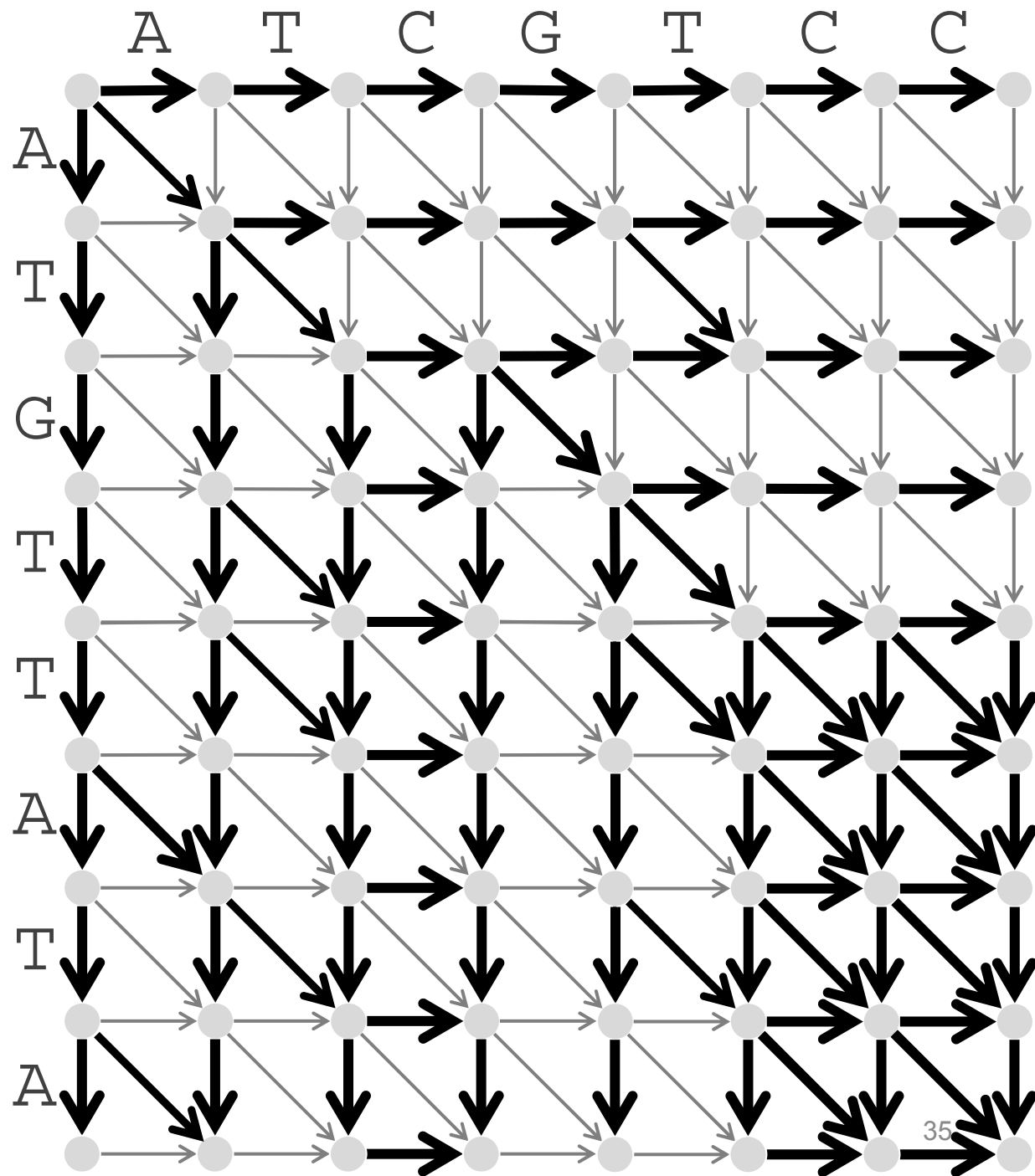
red edges \searrow – weight 1
 other edges – weight 0

backtracking pointers
for the Longest
Common Subsequence

red edges ↘ – weight 1
other edges – weight 0



backtracking pointers
for the Longest
Common Subsequence

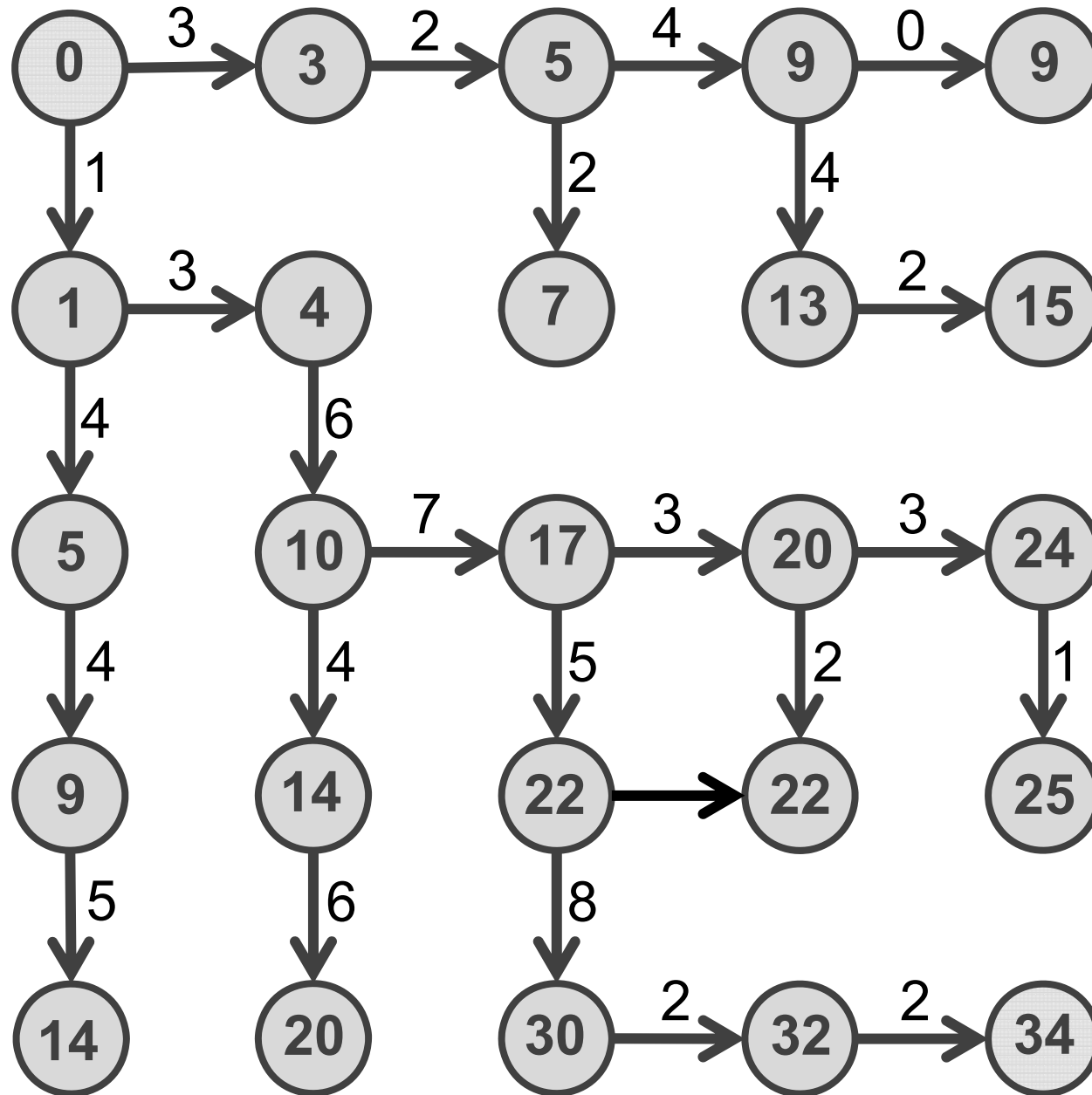


Computing Backtracking Pointers

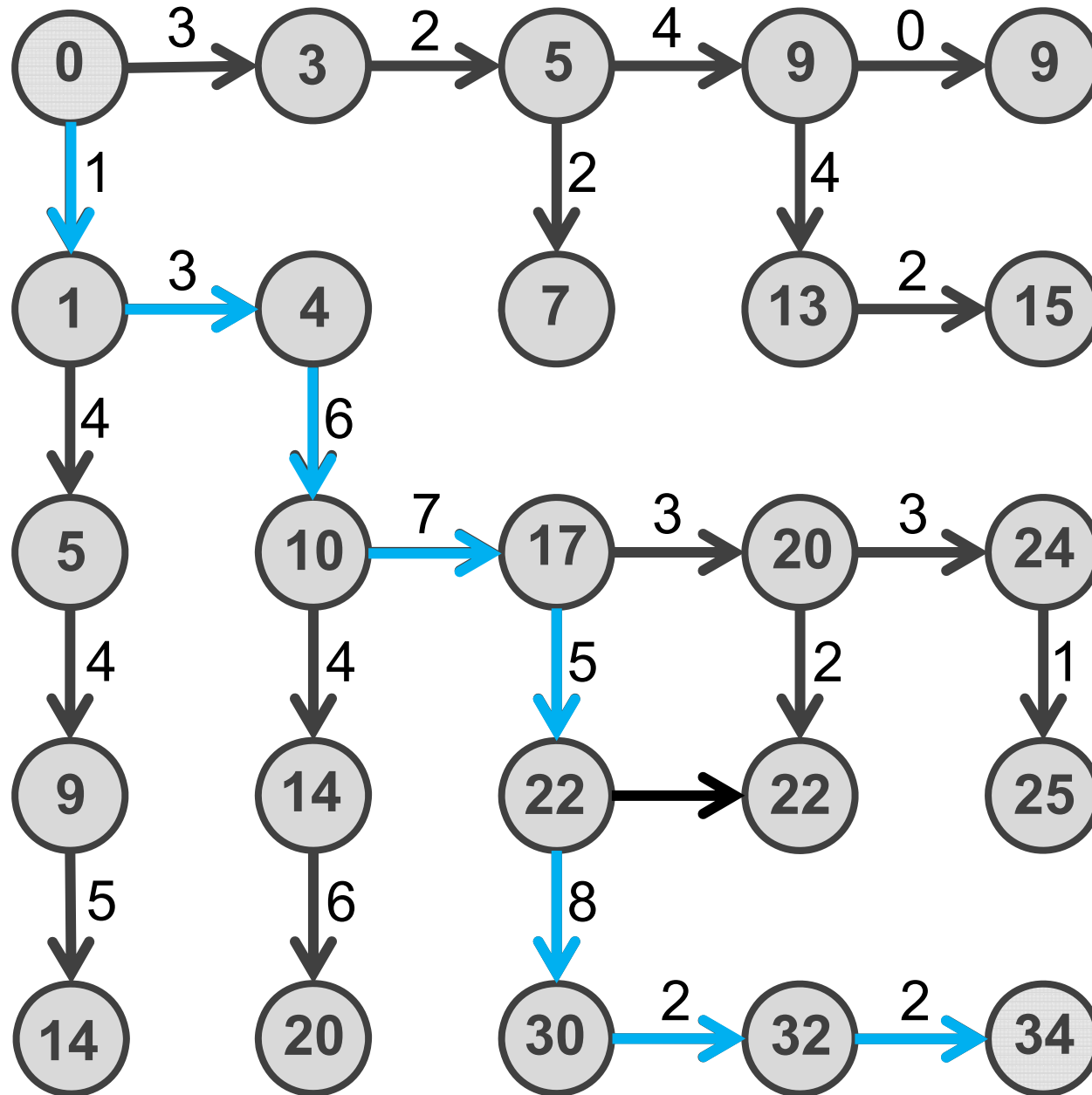
$$s_{i,j} \leftarrow \max \begin{cases} s_{i,j-1} + 0 \\ s_{i-1,j} + 0 \\ s_{i-1,j-1} + 1, \text{ if } v_i = w_j \end{cases}$$

$$\text{backtrack}_{i,j} \leftarrow \begin{cases} \text{"}\rightarrow\text{"}, \text{ if } s_{i,j} = s_{i,j-1} \\ \text{"}\downarrow\text{"}, \text{ if } s_{i,j} = s_{i-1,j} \\ \text{"}\searrow\text{"}, \text{ if } s_{i,j} = s_{i-1,j-1} + 1 \end{cases}$$

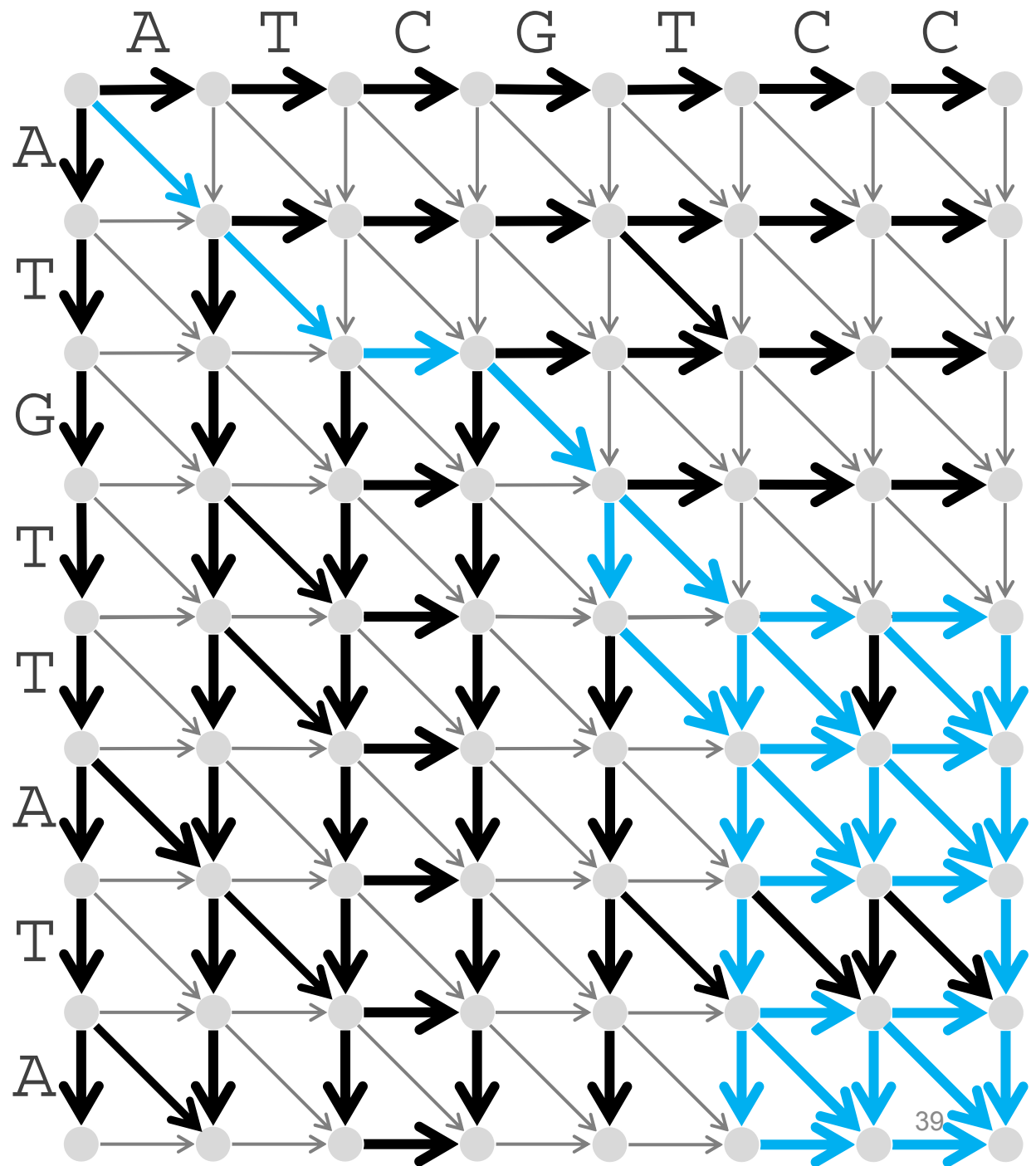
Why did we store the **backtracking pointers**?



What is the optimal alignment path?



backtracking pointers
for the Longest
Common Subsequence



Using Backtracking Pointers to Compute LCS

OutputLCS (*backtrack*, *v*, *i*, *j*)

if $i = 0$ or $j = 0$

return

if $backtrack_{i,j} = \rightarrow$

OutputLCS (*backtrack*, *v*, *i*, *j*-1)

else if $backtrack_{i,j} = \downarrow$

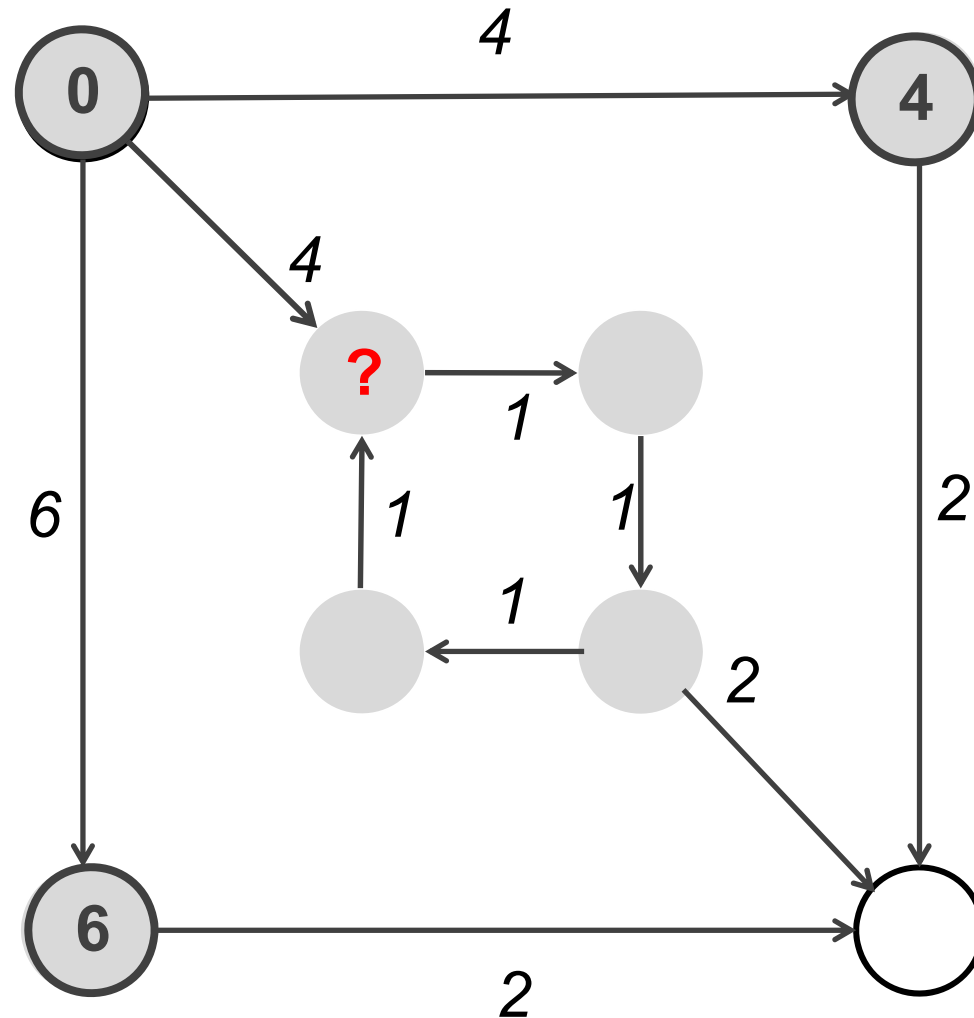
OutputLCS (*backtrack*, *v*, *i*-1, *j*)

else

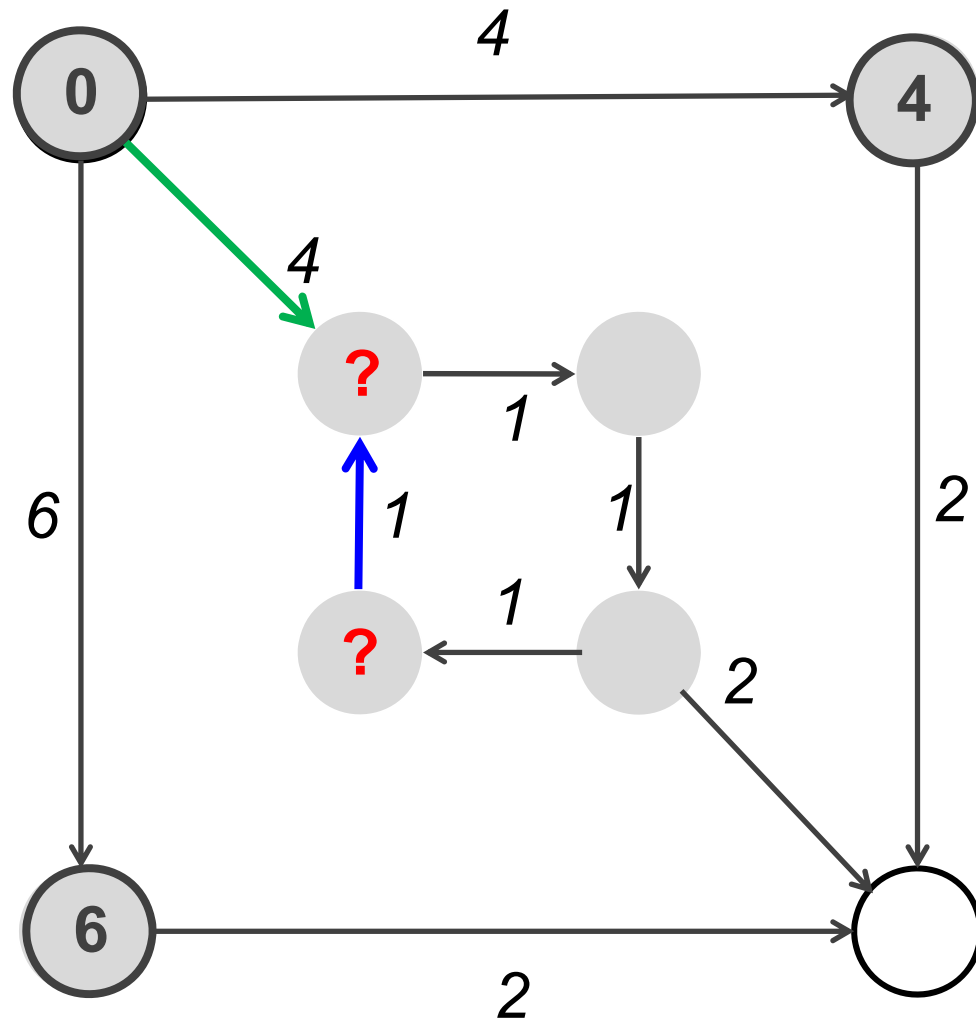
OutputLCS (*backtrack*, *v*, *i*-1, *j*-1)

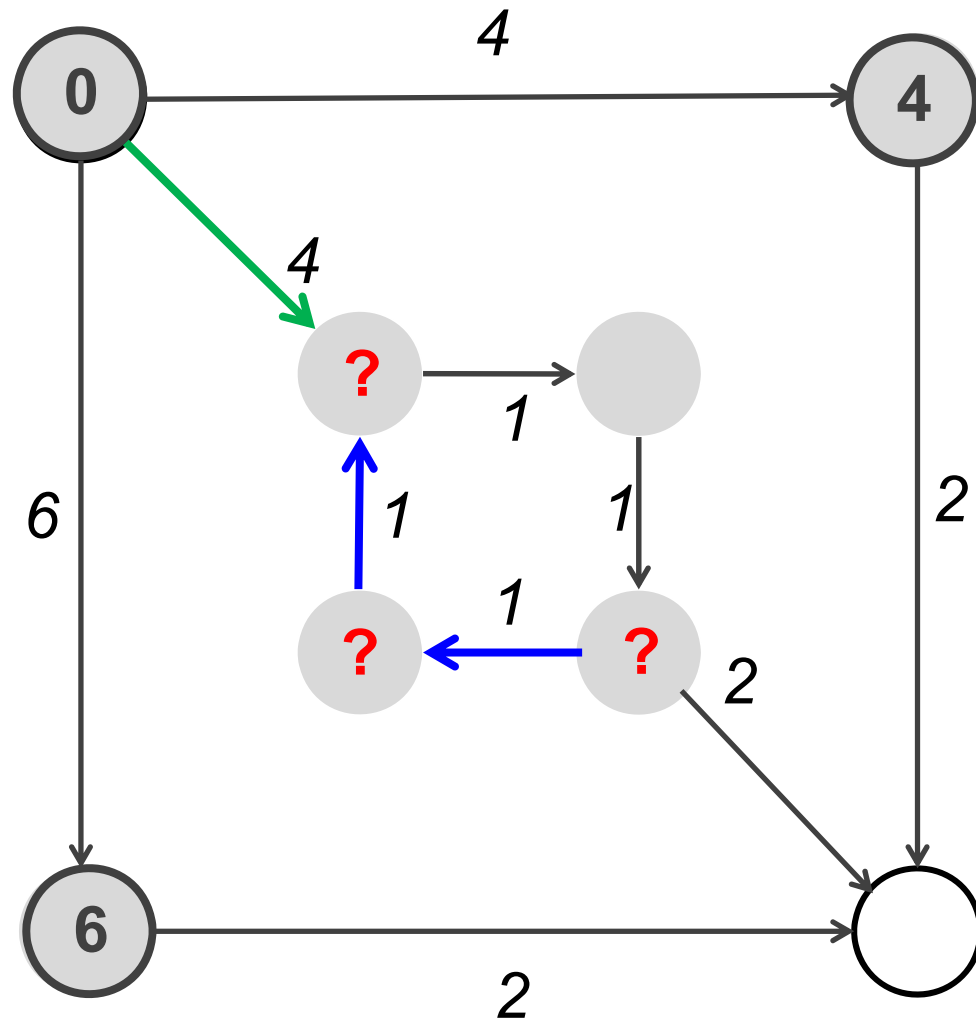
output v_i

Computing Scores of **ALL** Predecessors

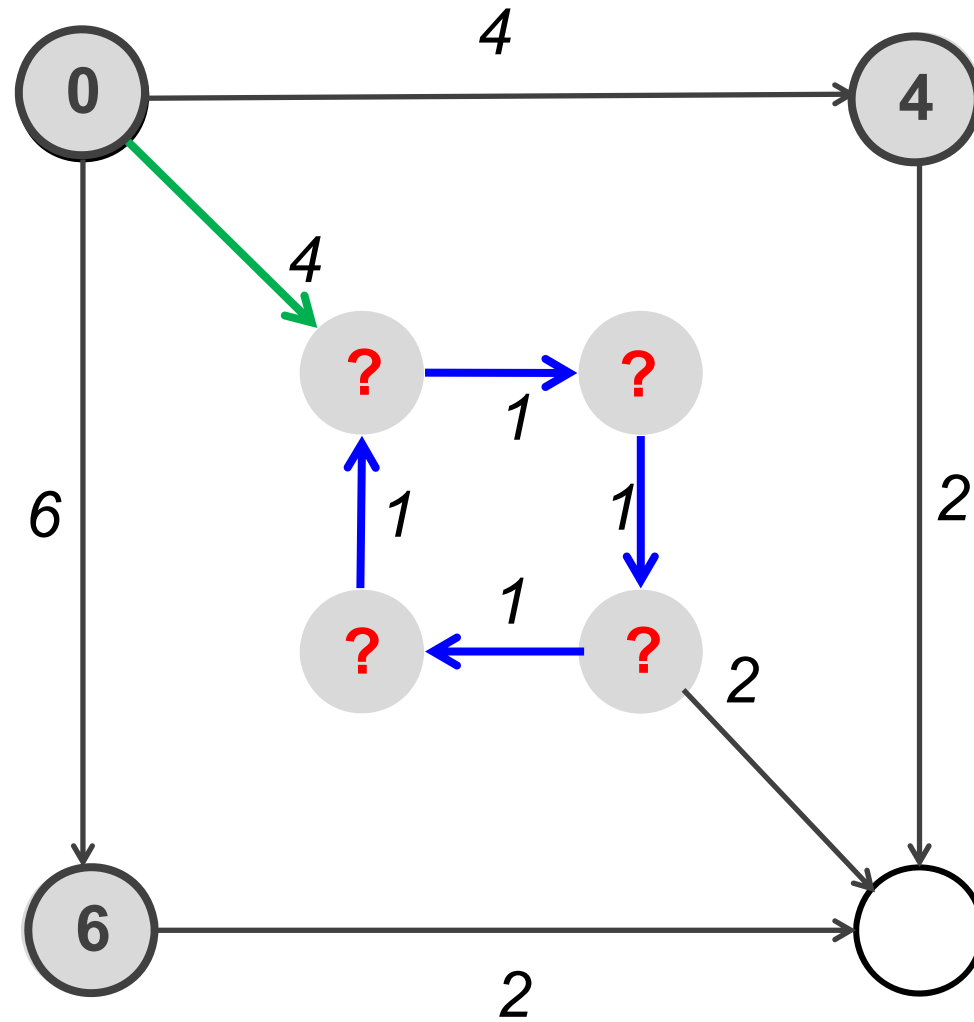


$$s_a = \max_{\text{ALL predecessors } b \text{ of node } a} \{s_b + \text{weight of edge from } b \text{ to } a\}$$



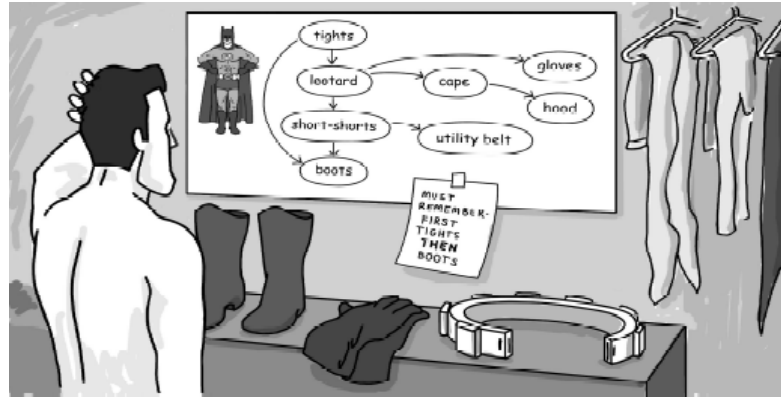


A Vicious Cycle



In What Order Should We Explore Nodes of the Graph?

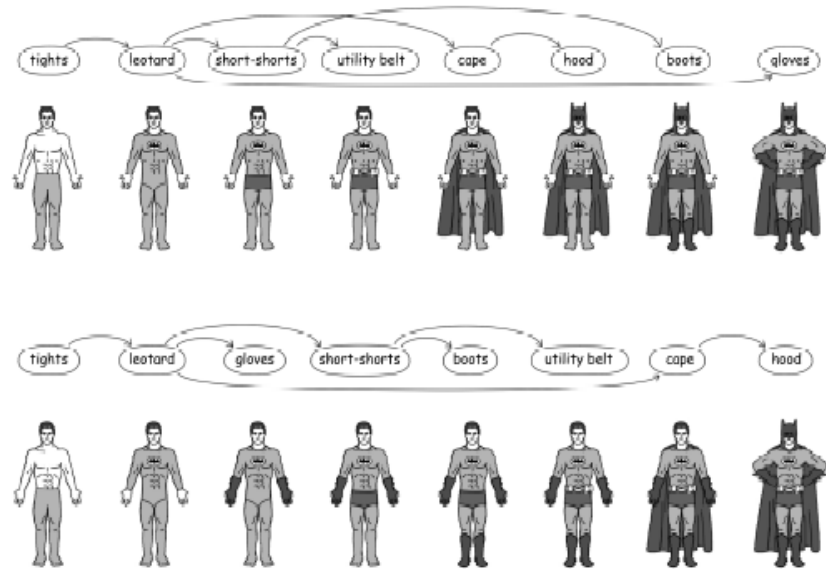
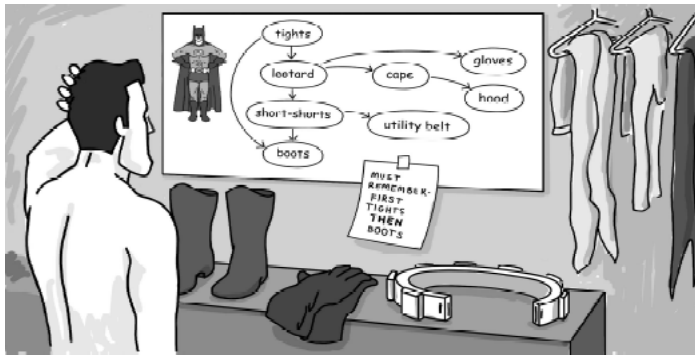
$$s_a = \max_{\text{ALL predecessors } b \text{ of node } a} \{s_b + \text{weight of edge from } b \text{ to } a\}$$



- By the time a node is analyzed, the scores of all its predecessors should already be computed.
- If the graph has a **directed cycle**, this condition is impossible to satisfy.
- **Directed Acyclic Graph (DAG)**: a graph without directed cycles.

Topological Ordering

- **Topological Ordering:** Ordering of nodes of a DAG on a line such that all edges go from left to right.



- **Theorem:** Every DAG has a topological ordering.

LongestPath

LongestPath(*Graph*, *source*, *sink*)

for each node *a* in *Graph*

$s_a \leftarrow -\text{infinity}$

$s_{\text{source}} \leftarrow 0$

topologically order *Graph*

for each node *a* (from *source* to *sink* in topological order)

$s_a \leftarrow \max_{\text{all predecessors } b \text{ of node } a} \{s_b + \text{weight of edge from } b \text{ to } a\}$

return s_{sink}

Mismatches and Indel Penalties

$$\# \text{matches} - \mu \cdot \# \text{mismatches} - \sigma \cdot \# \text{indels}$$

$$\begin{array}{cccccccccc}
 \text{A} & \text{T} & - & \text{G} & \text{T} & \text{T} & \text{A} & \text{T} & \text{A} & \\
 \text{A} & \text{T} & \text{C} & \text{G} & \text{T} & - & \text{C} & - & \text{C} & \\
 +1 & +1 & -2 & +1 & +1 & -2 & -3 & -2 & -3 & = -7
 \end{array}$$

	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	$-\sigma$
-	$-\sigma$	$-\sigma$	$-\sigma$	$-\sigma$	

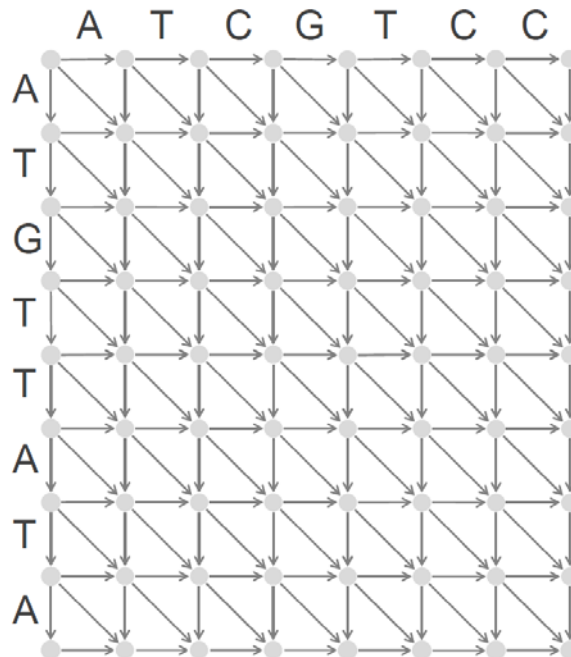
Scoring matrix

	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

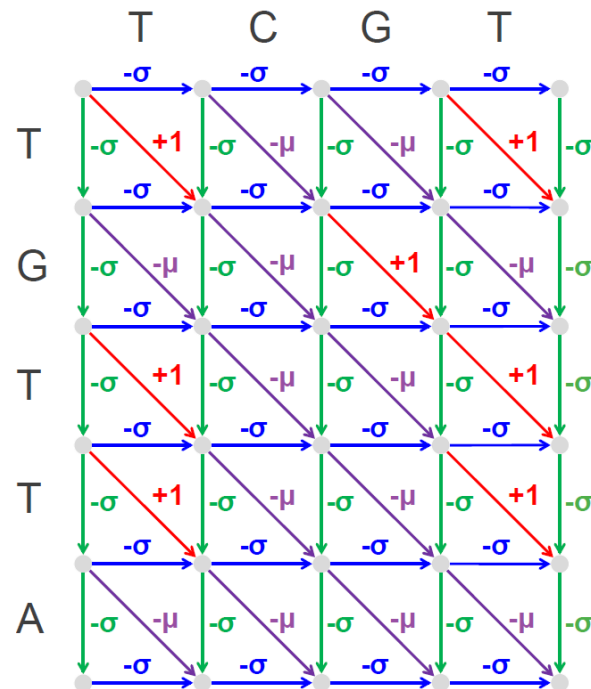
Dynamic Programming Recurrence for the Alignment Graph

$$s_{i,j} = \max \begin{cases} s_{i-1,j} + \text{weight of edge } \downarrow \text{ into } (i,j) \\ s_{i,j-1} + \text{weight of edge } \rightarrow \text{ into } (i,j) \\ s_{i-1,j-1} + \text{weight of edge } \swarrow \text{ into } (i,j) \end{cases}$$



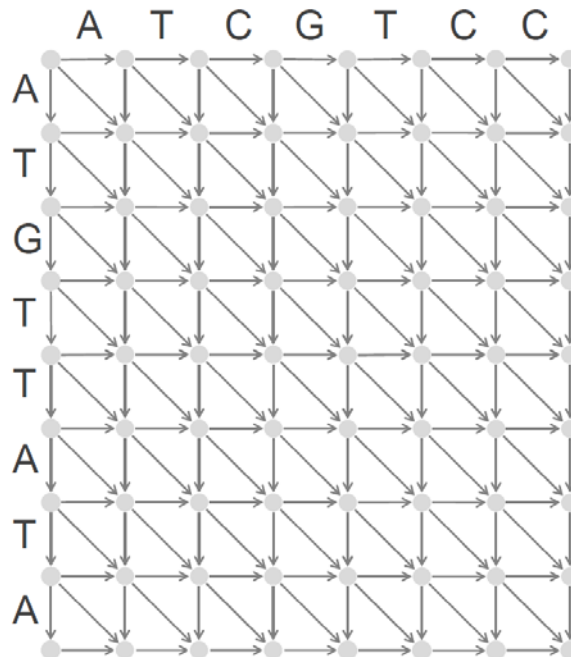
Dynamic Programming Recurrence for the Alignment Graph

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



Dynamic Programming Recurrence for the Alignment Graph

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



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 .

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
GCCGCCGTCTTTTCAGCAGTTATGTCAG-----A-----T-----

Which Alignment is Better?

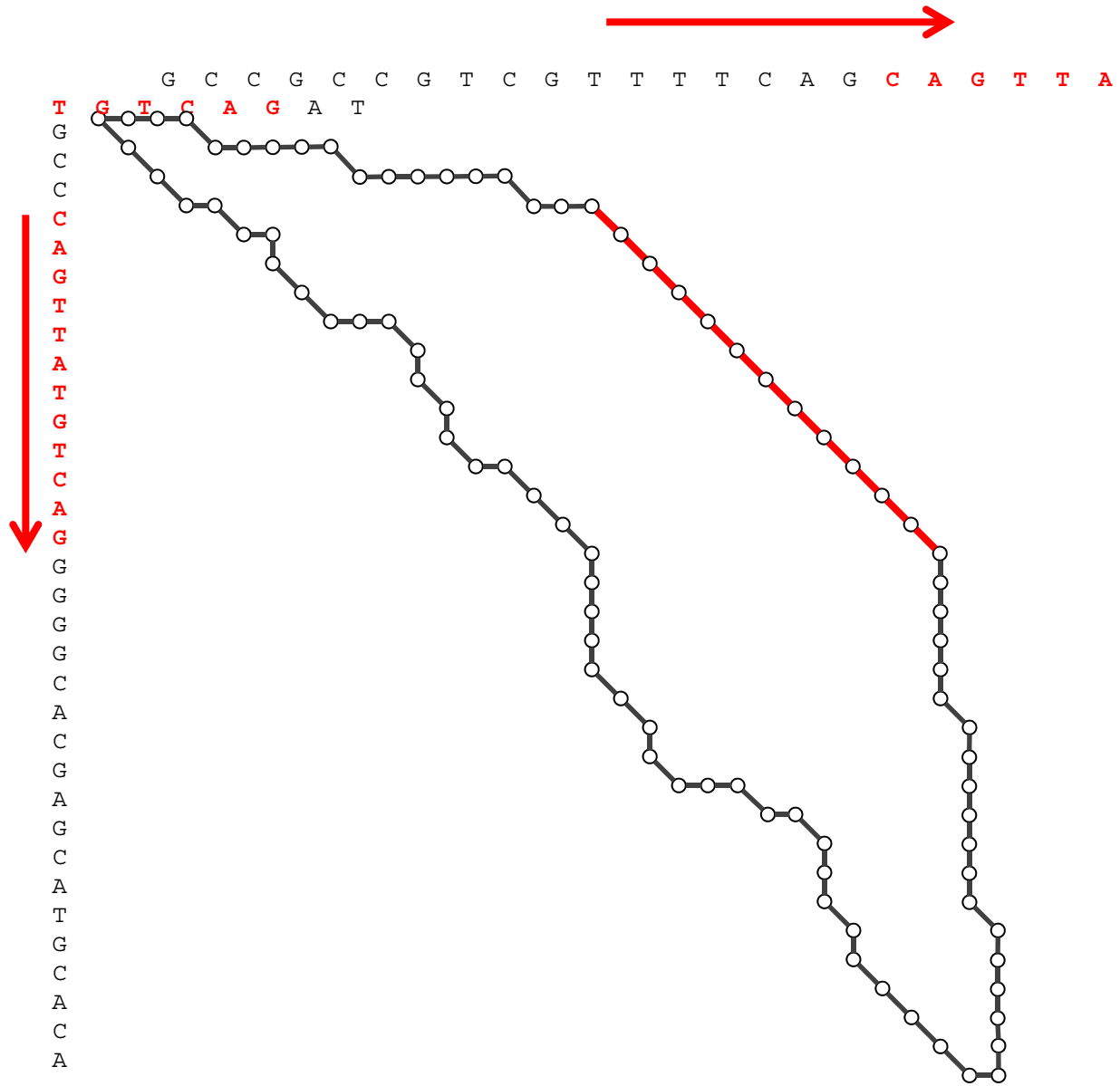
- Alignment 1: score = 22 (matches) - 20 (indels)=2.

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

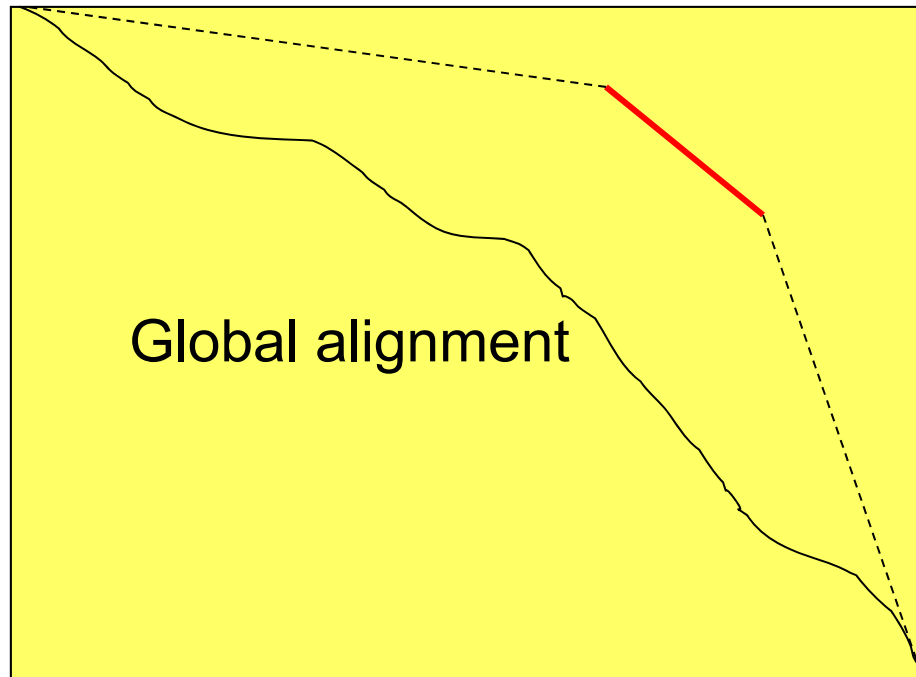
- Alignment 2: score = 17 (matches) - 30 (indels)=-13.

---G---C-----C--**CAGTTATGTCAG**GGGGGCACGAGCATGCAGA
GCCGCCGTCGTTTTCAG**CAGTTATGTCAG**-----A-----T-----
local alignment

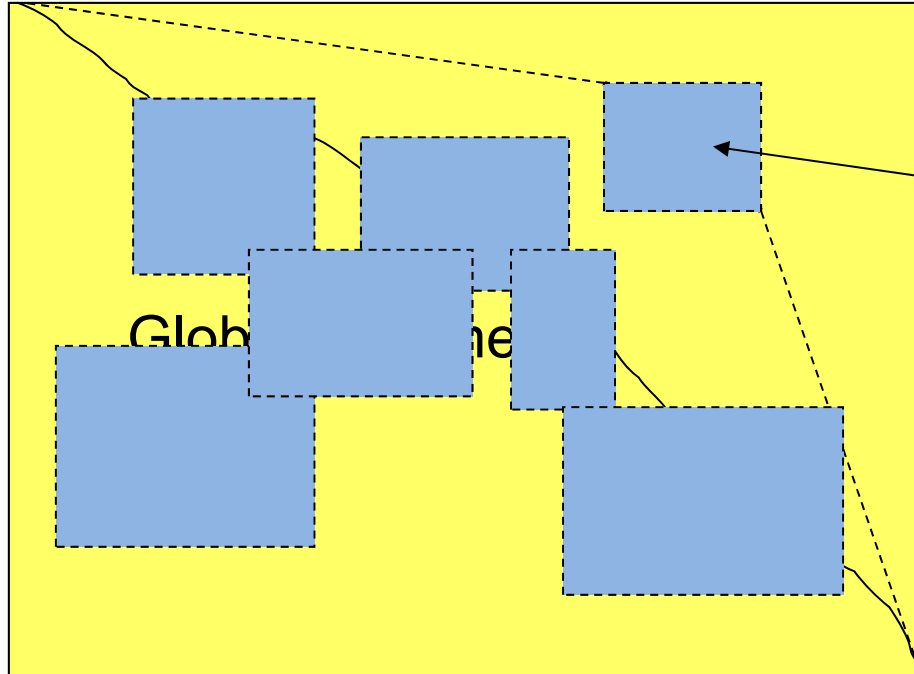




Local Alignment



Local Alignment= Global Alignment in a Subrectangle



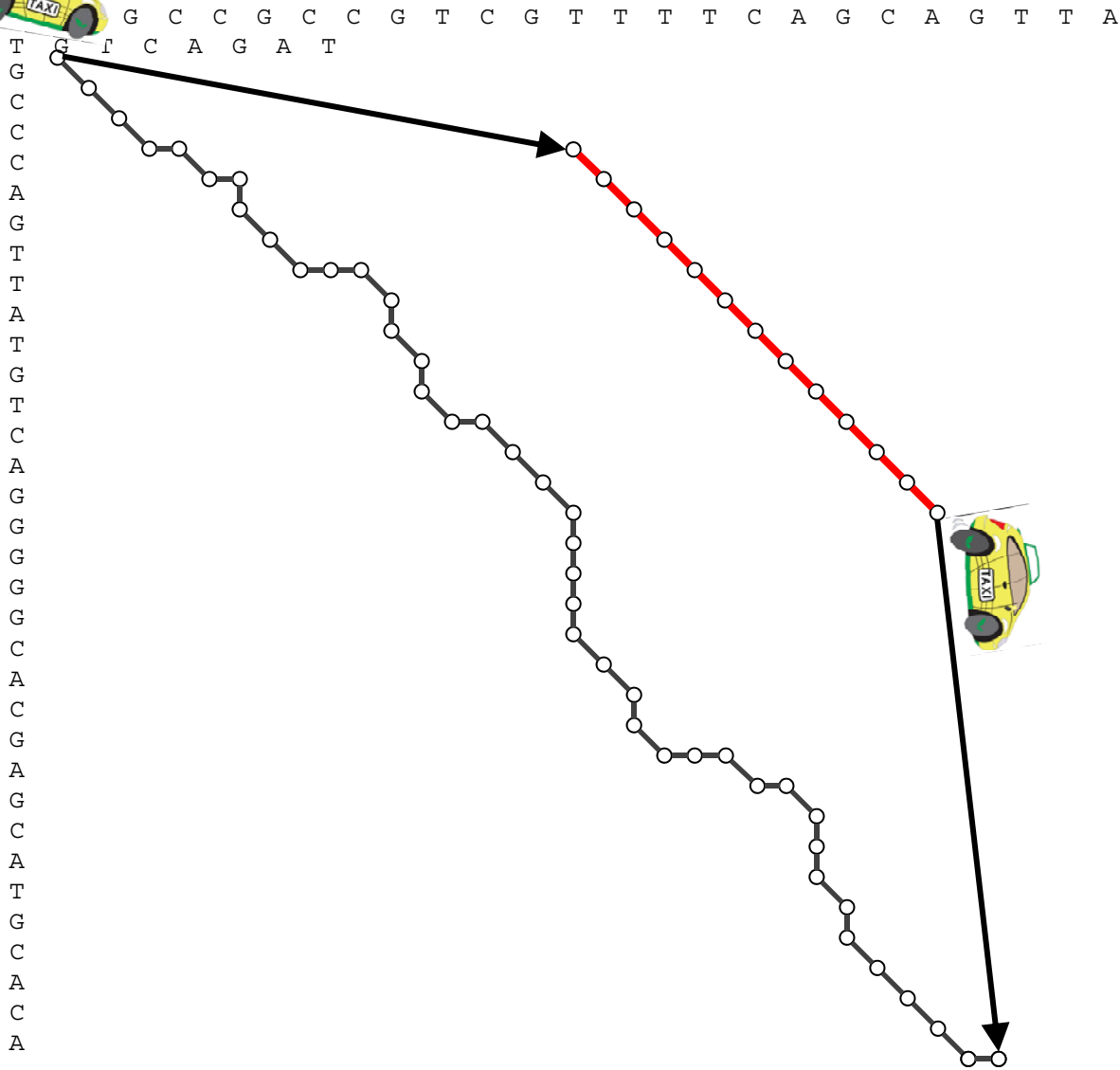
Compute a Global Alignment within each rectangle to get a Local Alignment

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 .

Free Taxi Rides!



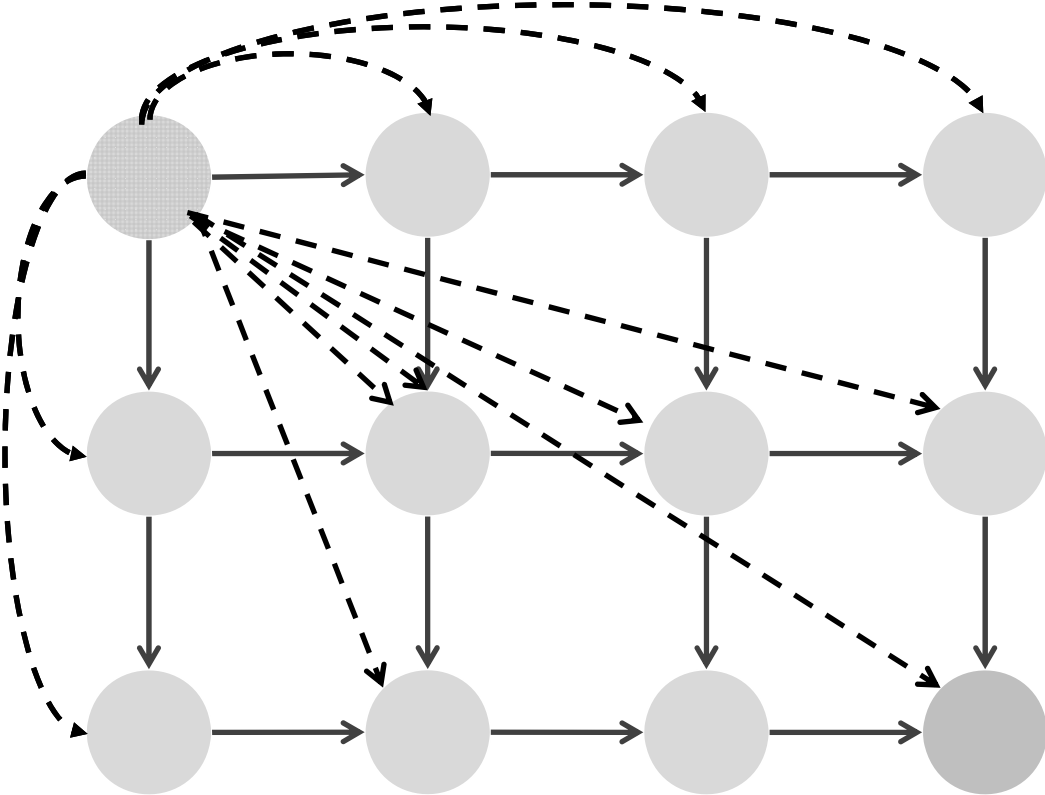
GCC-C-AGT-TATGT-CAGGGGGCACG--A-GCATGCACA-
 GCCGCC-GTCGT-T-TTCAG----CA-GTTATG-T-CAGAT

Global alignment

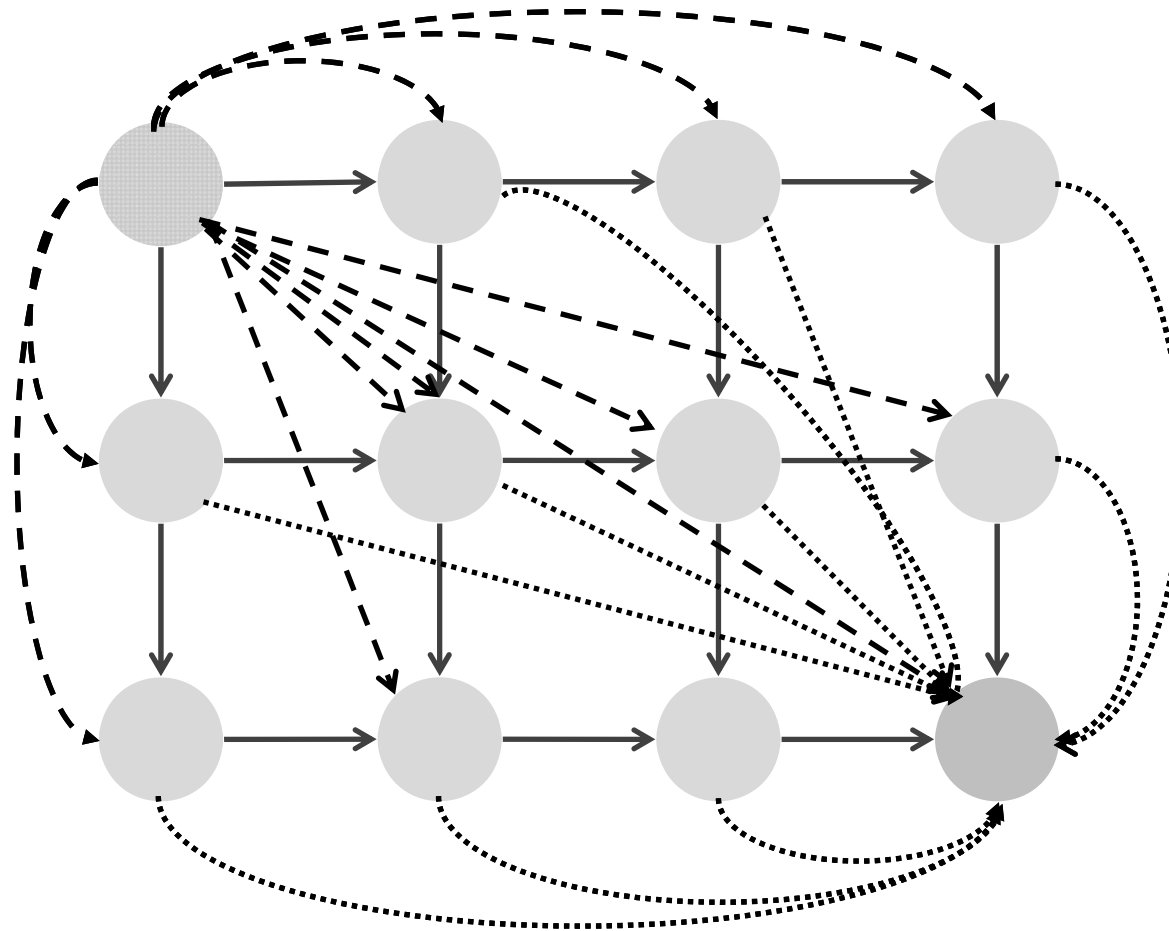
---G---C-----C--**CAGTTATGTCAG**GGGGCACGAGCATGCACA
 GCCGCCGTCGTTTTTCAG**CAGTTATGTCAG**-----A-----T-----

Local alignment

What Do Free Taxi Rides Mean in the Terms of the Alignment Graph?



Building Manhattan for the Local Alignment Problem

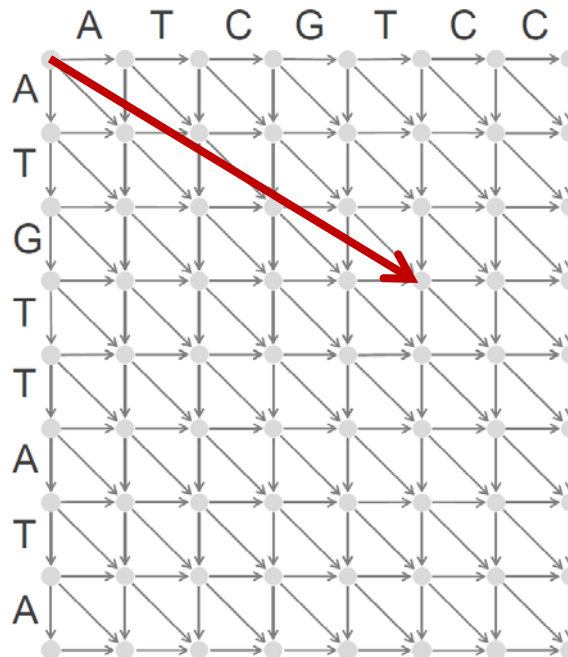


How many edges have we added?

Dynamic Programming for the Local Alignment

weight of edge from (0,0) to (i,j)

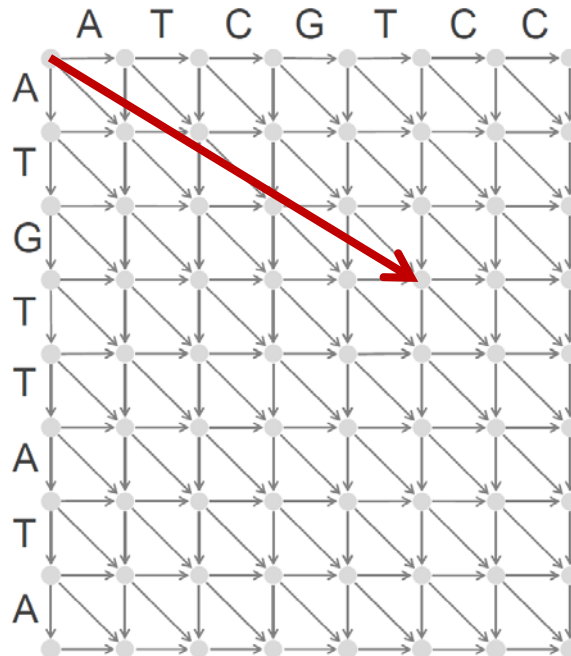
$$s_{i,j} = \max \begin{cases} s_{i-1,j} + \text{weight of edge } \downarrow \text{ into } (i,j) \\ s_{i,j-1} + \text{weight of edge } \rightarrow \text{ into } (i,j) \\ s_{i-1,j-1} + \text{weight of edge } \swarrow \text{ into } (i,j) \end{cases}$$



Dynamic Programming for the Local Alignment

0

$$s_{i,j} = \max \begin{cases} s_{i-1,j} + \text{weight of edge } \downarrow \text{ into } (i,j) \\ s_{i,j-1} + \text{weight of edge } \rightarrow \text{ into } (i,j) \\ s_{i-1,j-1} + \text{weight of edge } \swarrow \text{ into } (i,j) \end{cases}$$



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)

More Adequate Gap Penalties

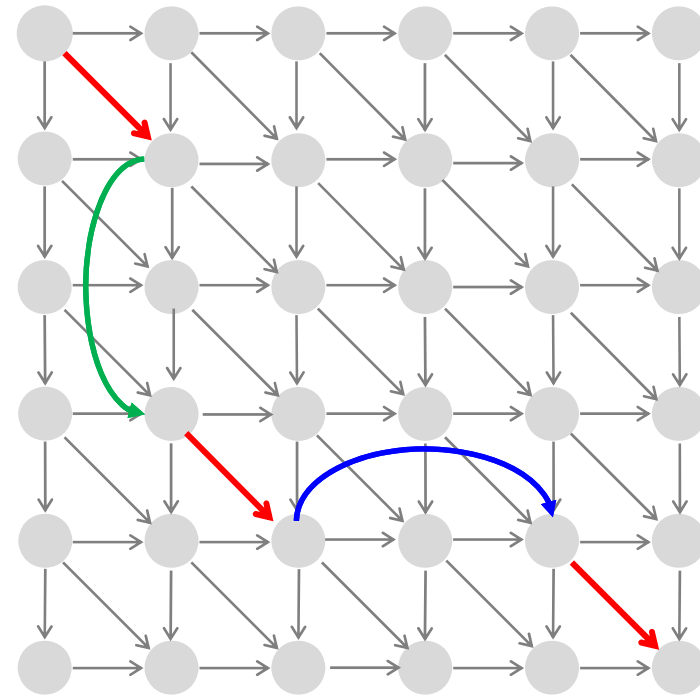
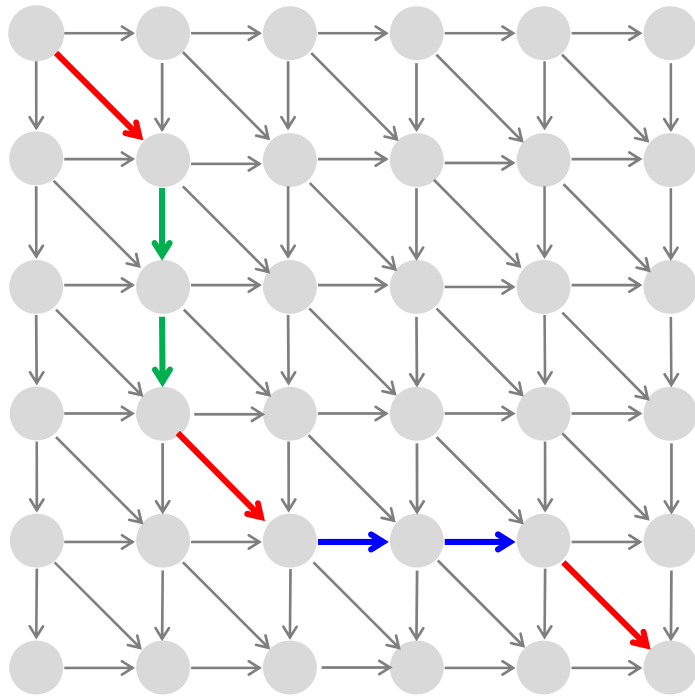
Affine gap penalty for a gap of length k : $\sigma + \varepsilon \cdot (k-1)$

σ - the **gap opening penalty**

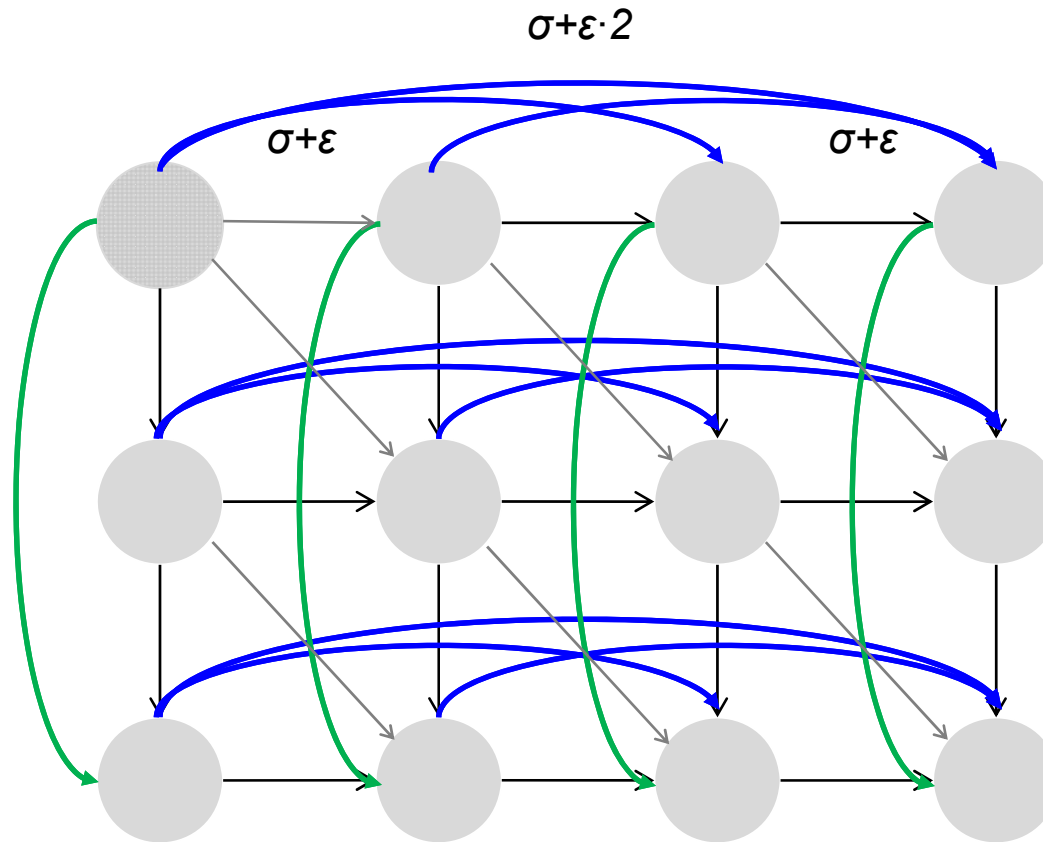
ε - the **gap extension penalty**

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

Modelling Affine Gap Penalties by Long Edges



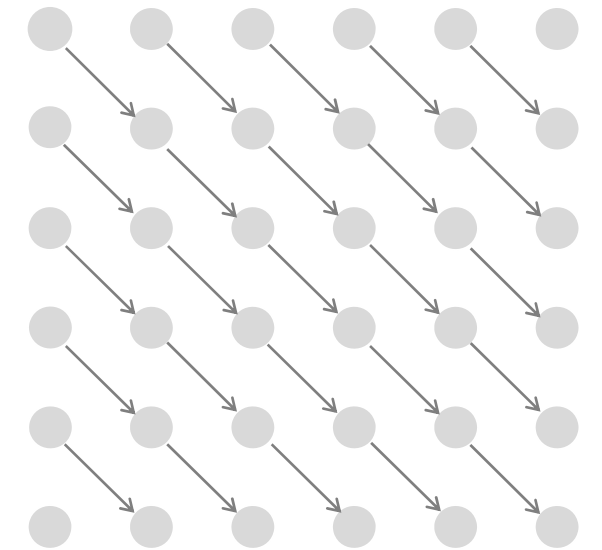
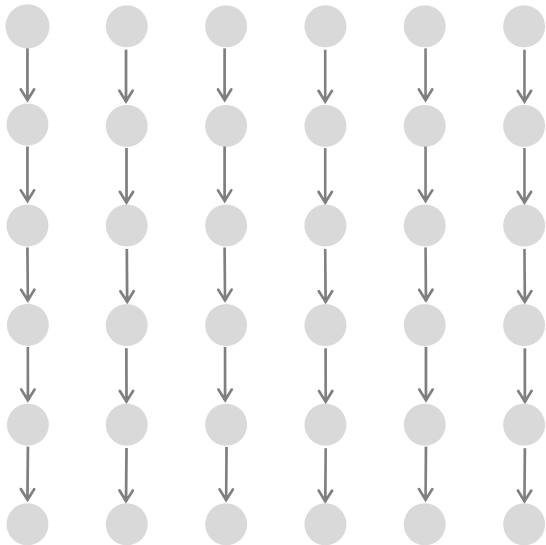
Building Manhattan with Affine Gap Penalties



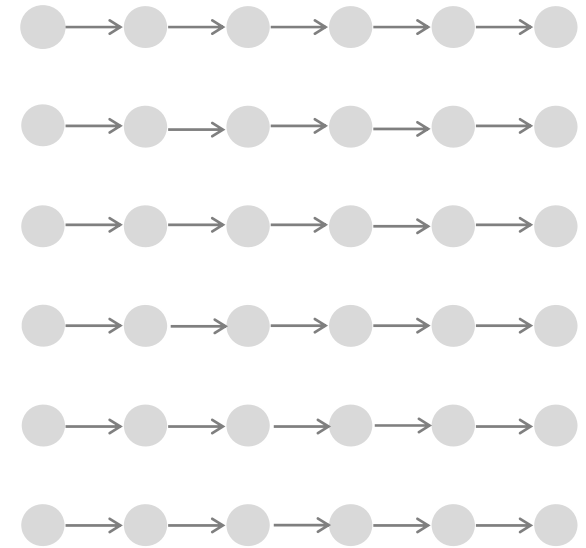
We have just added $O(n^3)$ edges to the graph...

Building Manhattan on 3 levels

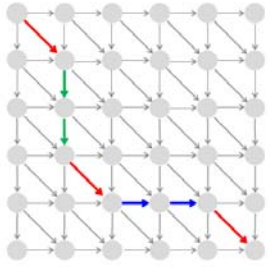
bottom level
(insertions)



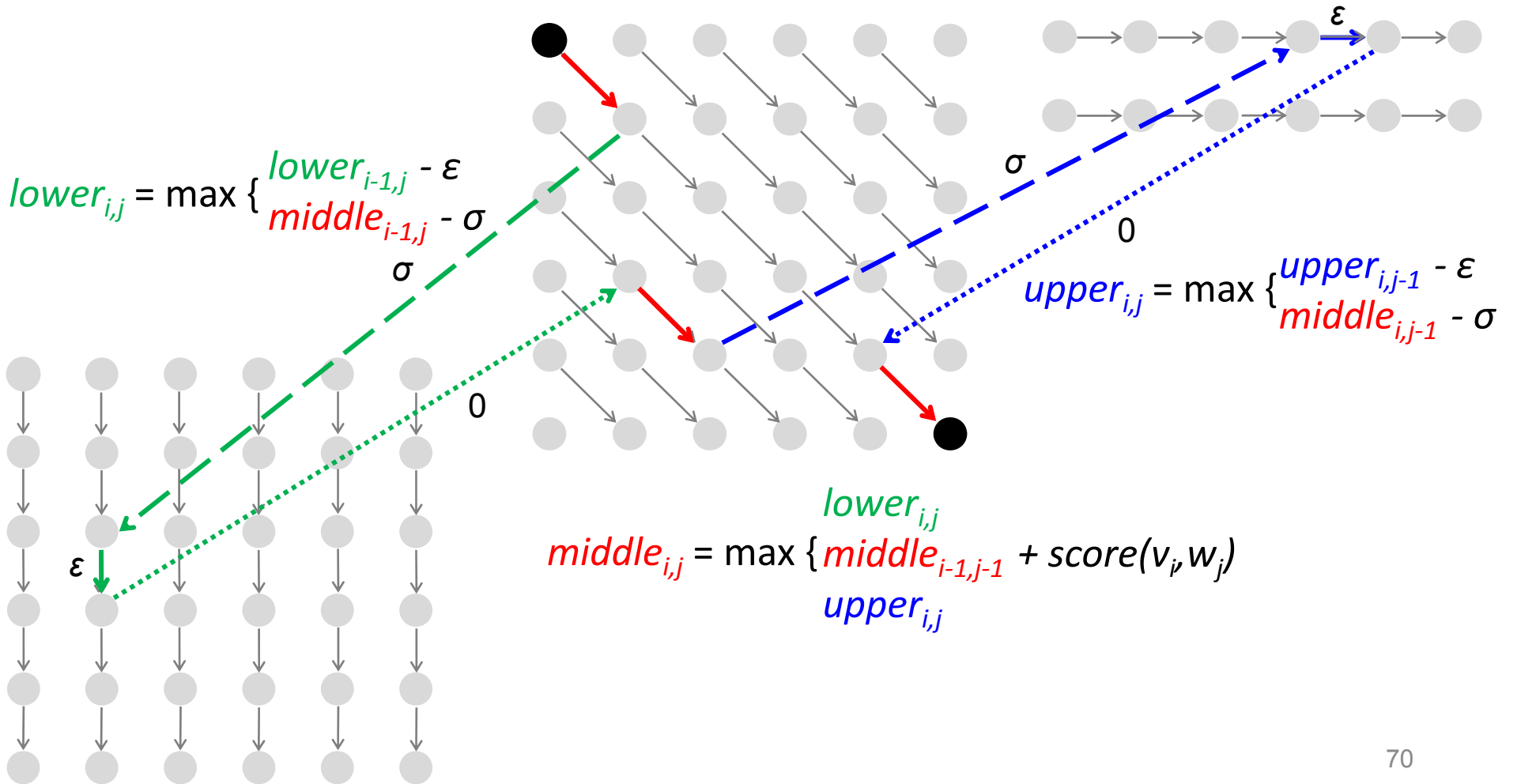
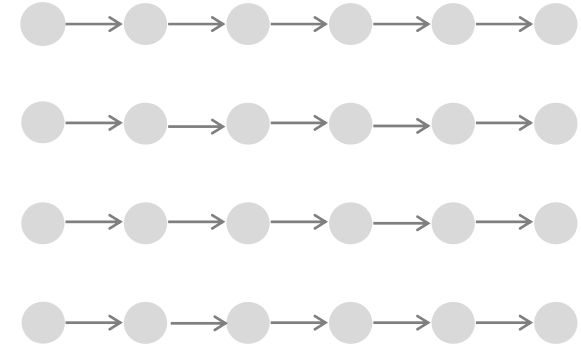
middle level
(matches/mismatches)



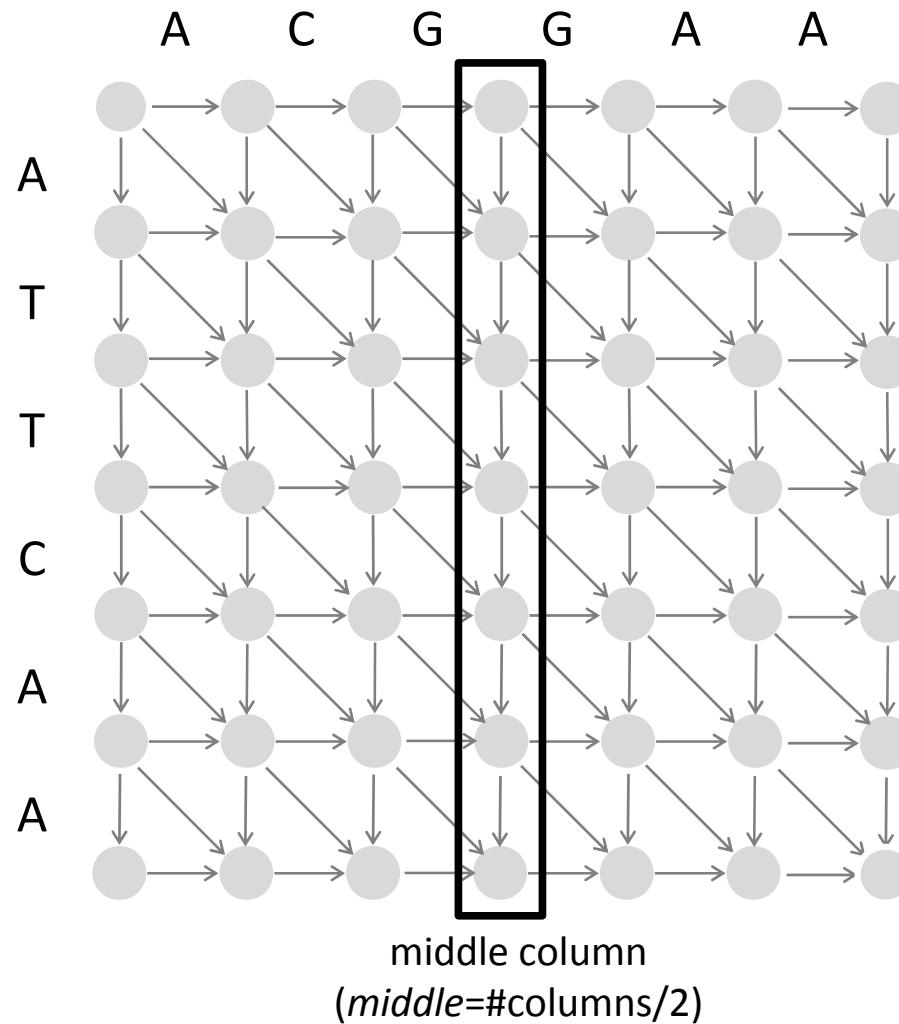
upper level
(deletions)



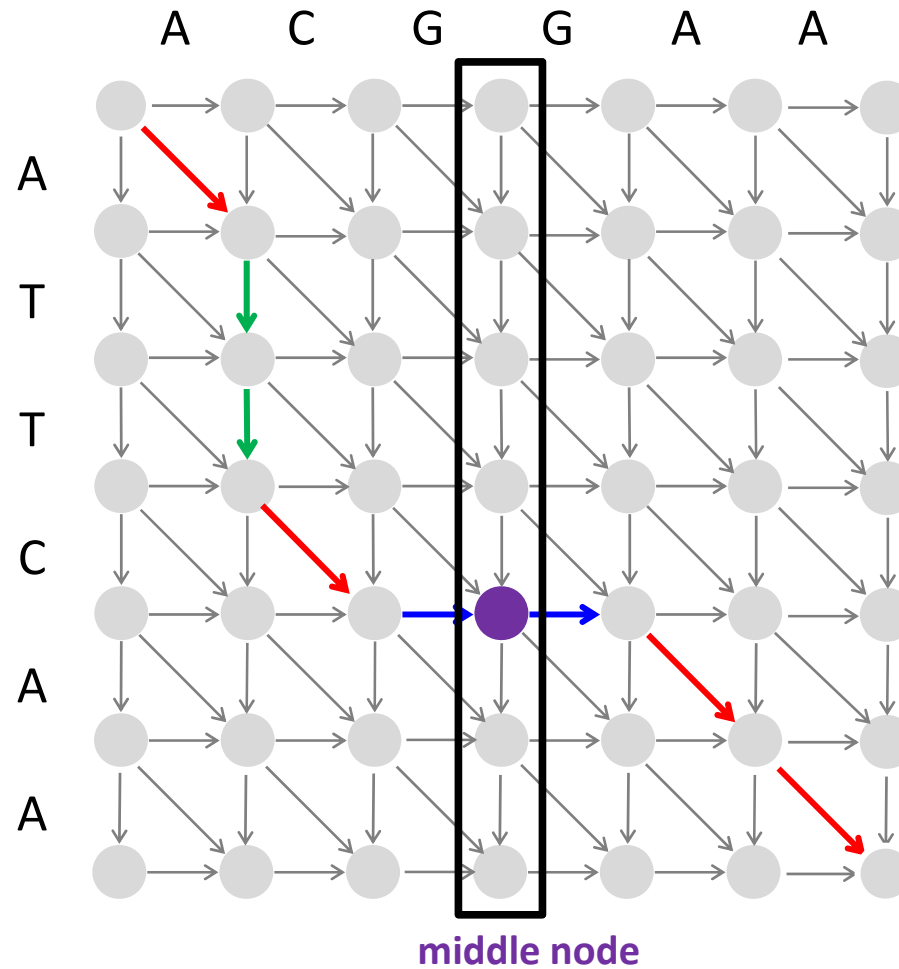
How can we emulate
this path in the 3-level
Manhattan?



Middle Column of the Alignment



Middle Node of the Alignment

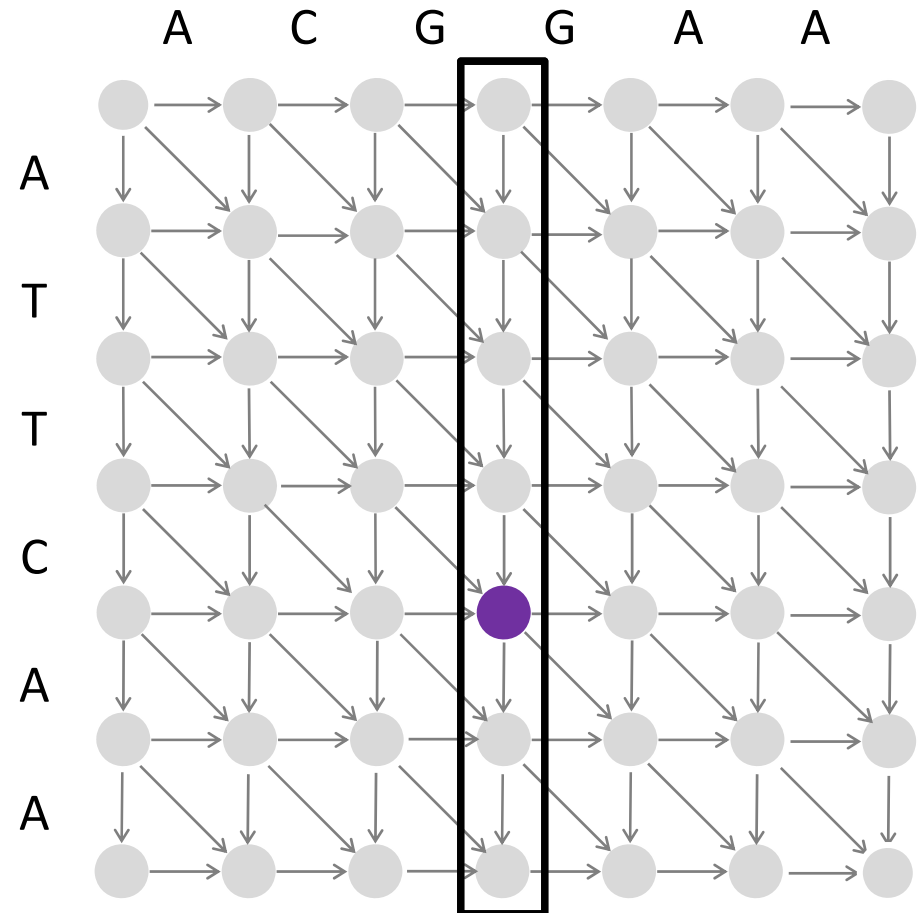


(a node where an optimal alignment path crosses the middle column)

Divide and Conquer Approach to Sequence Alignment

AlignmentPath(*source*, *sink*)

find *MiddleNode*

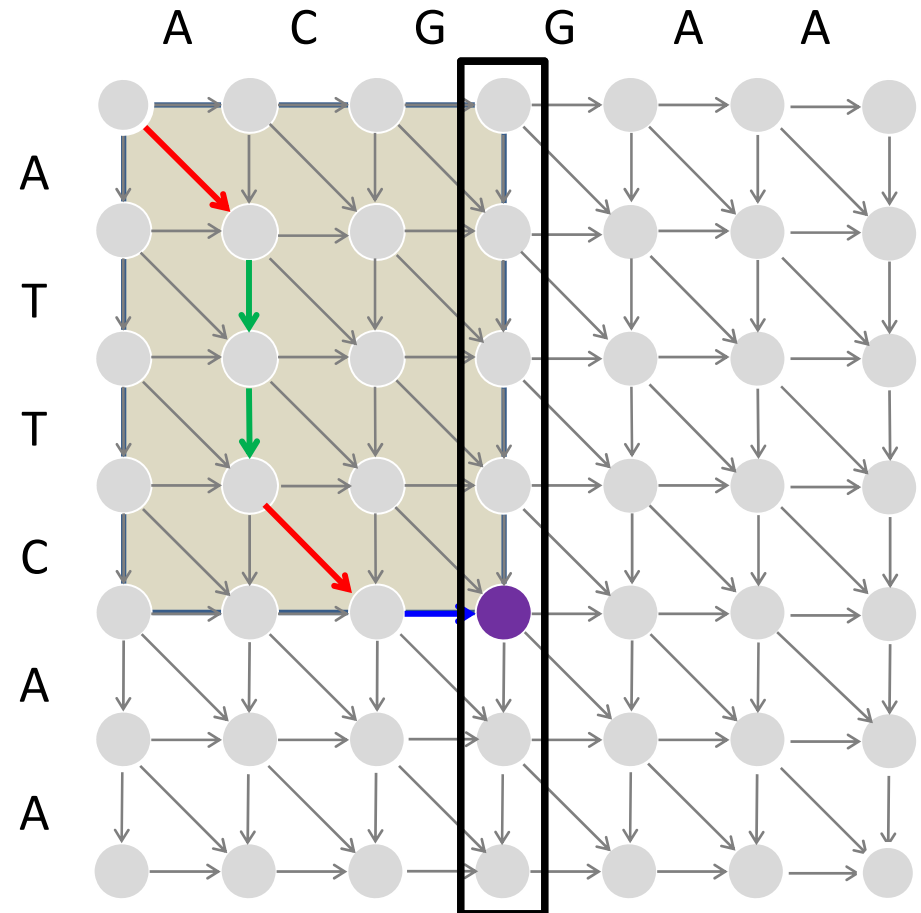


Divide and Conquer Approach to Sequence Alignment

AlignmentPath(source, sink)

find *MiddleNode*

AlignmentPath(source, *MiddleNode*)



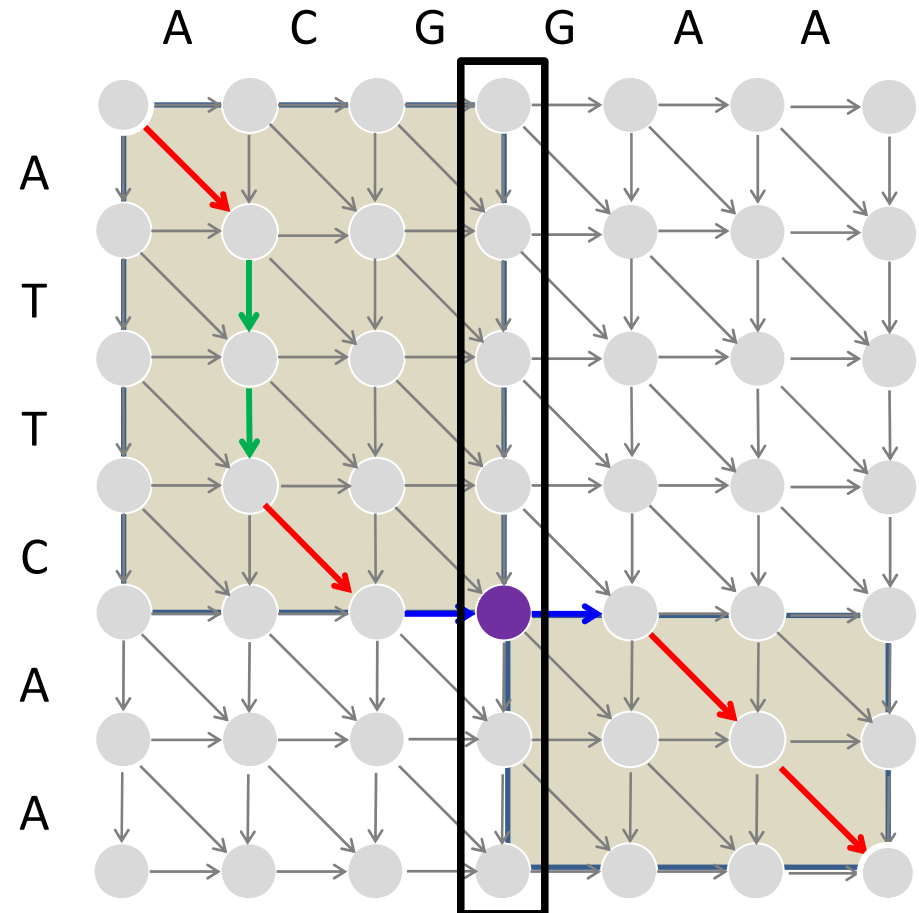
Divide and Conquer Approach to Sequence Alignment

AlignmentPath(*source*, *sink*)

find *MiddleNode*

AlignmentPath(*source*, *MiddleNode*)

AlignmentPath(*MiddleNode*, *sink*)



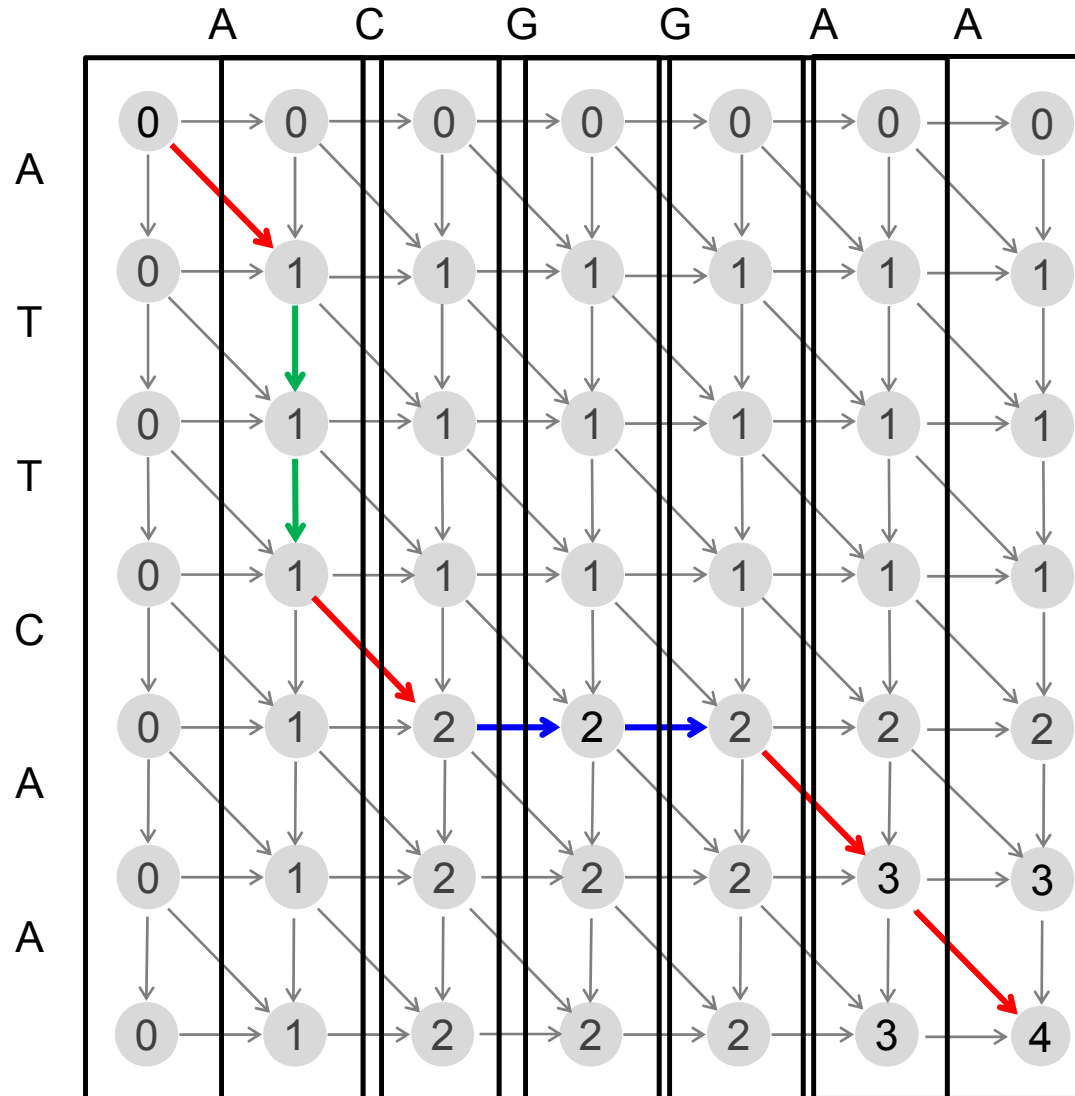
The only problem left is how to find this middle node in **linear space!**

Computing Alignment Score in Linear Space

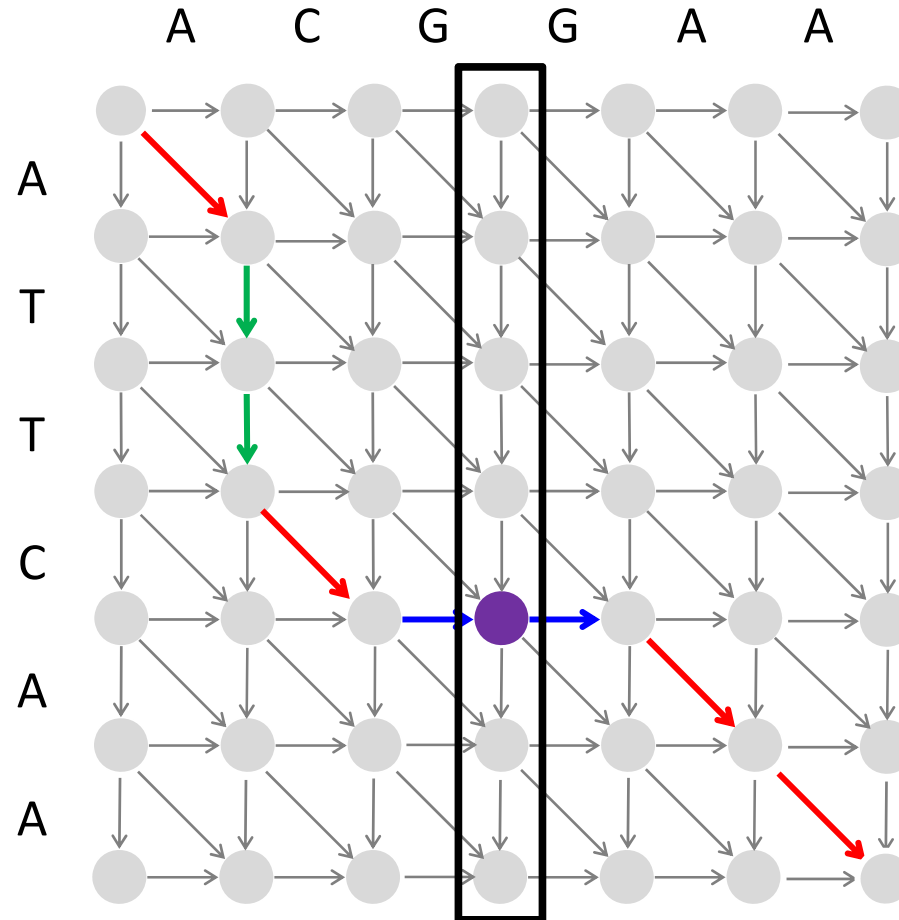
Finding the **longest path** in the alignment graph **requires** storing all backtracking pointers – $O(nm)$ memory.

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



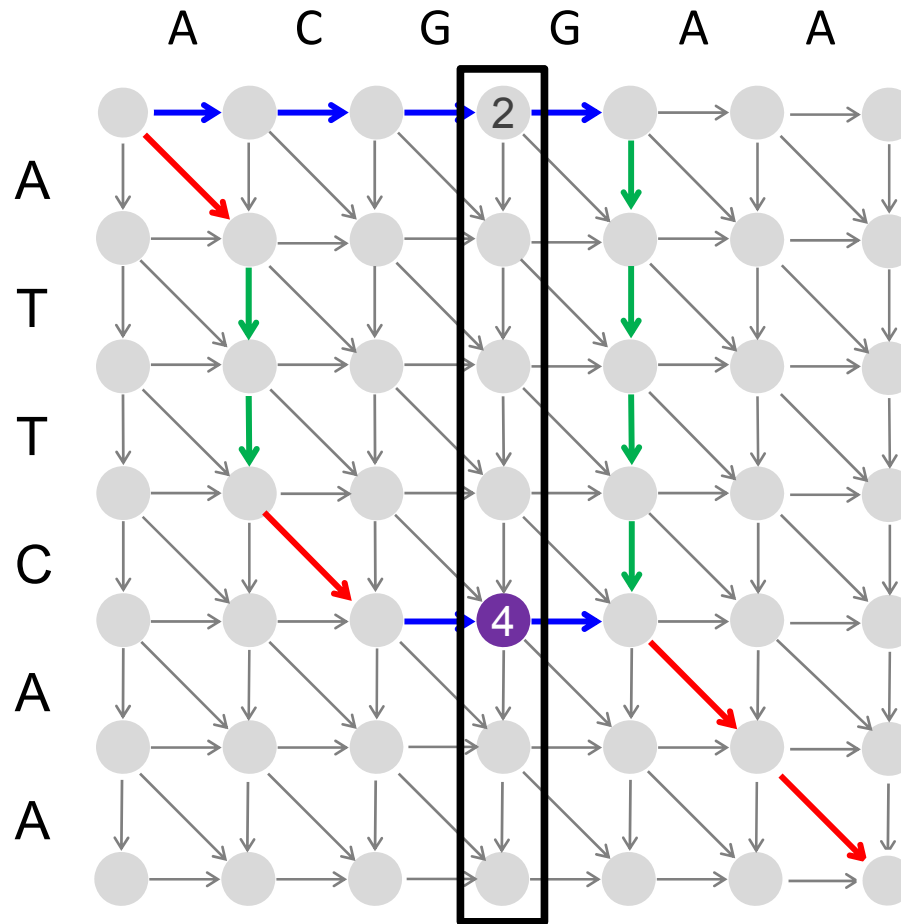
Can We Find the Middle Node without Constructing the Longest Path?



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

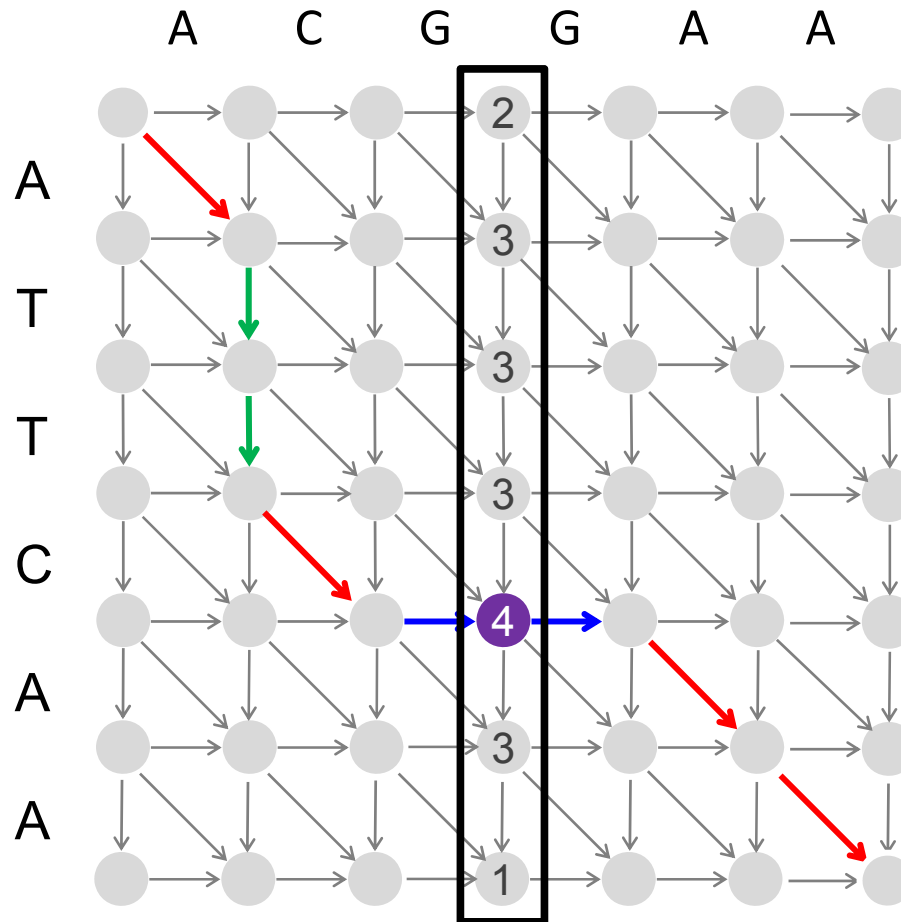
Can We Find The Lengths of All i -paths?



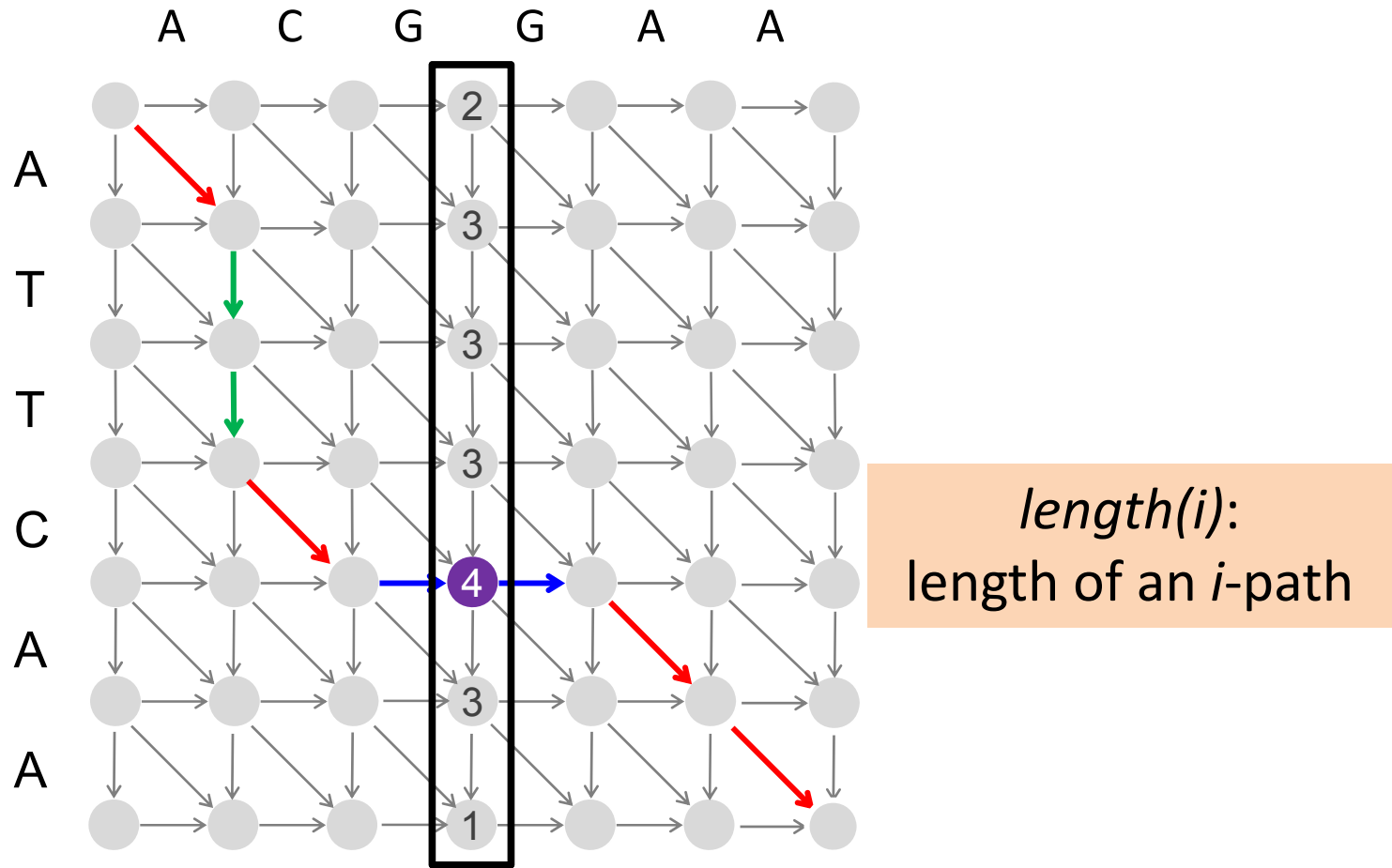
$length(i)$:
length of an i -path:

$length(0)=2$
 $length(4)=4$

Can We Find The Lengths of All i -paths?

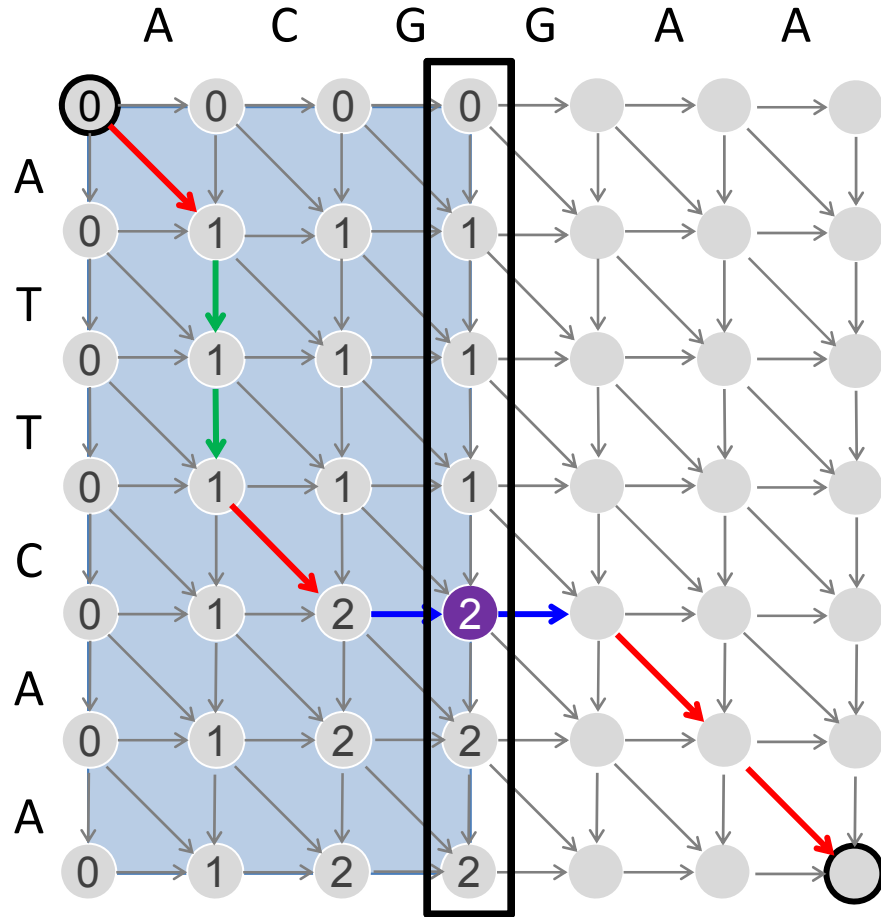


Can We Find The Lengths of i -paths?

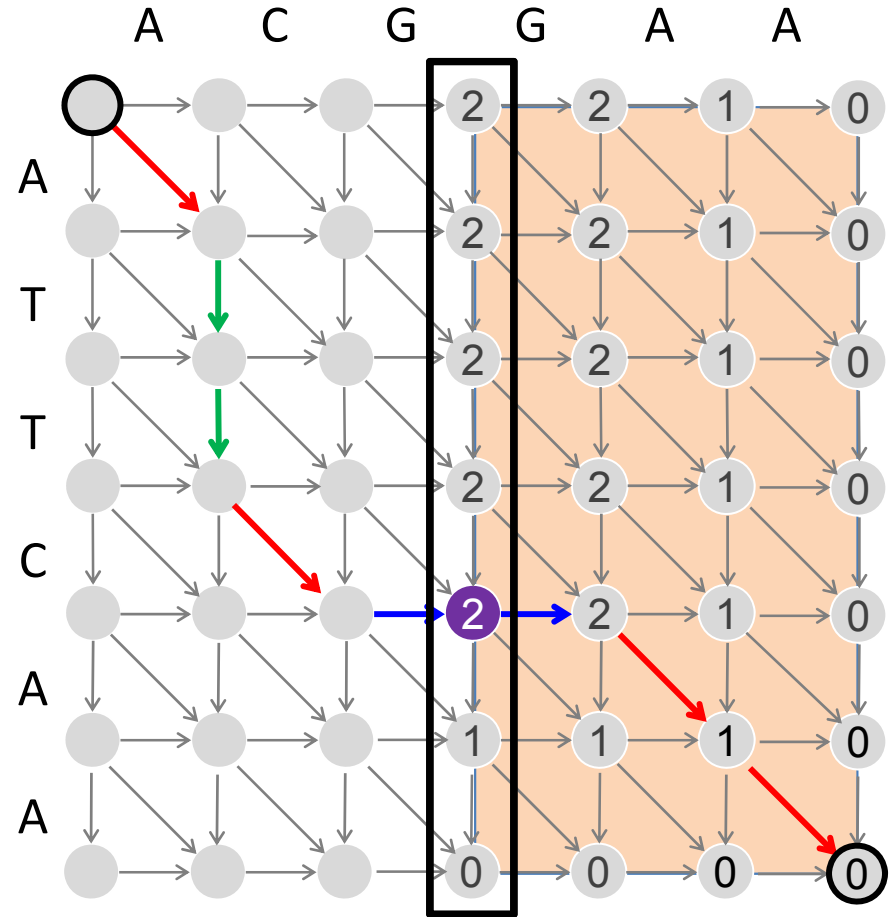


$$\text{length}(i) = \text{fromSource}(i) + \text{toSink}(i)$$

Computing *FromSource* and *toSink*

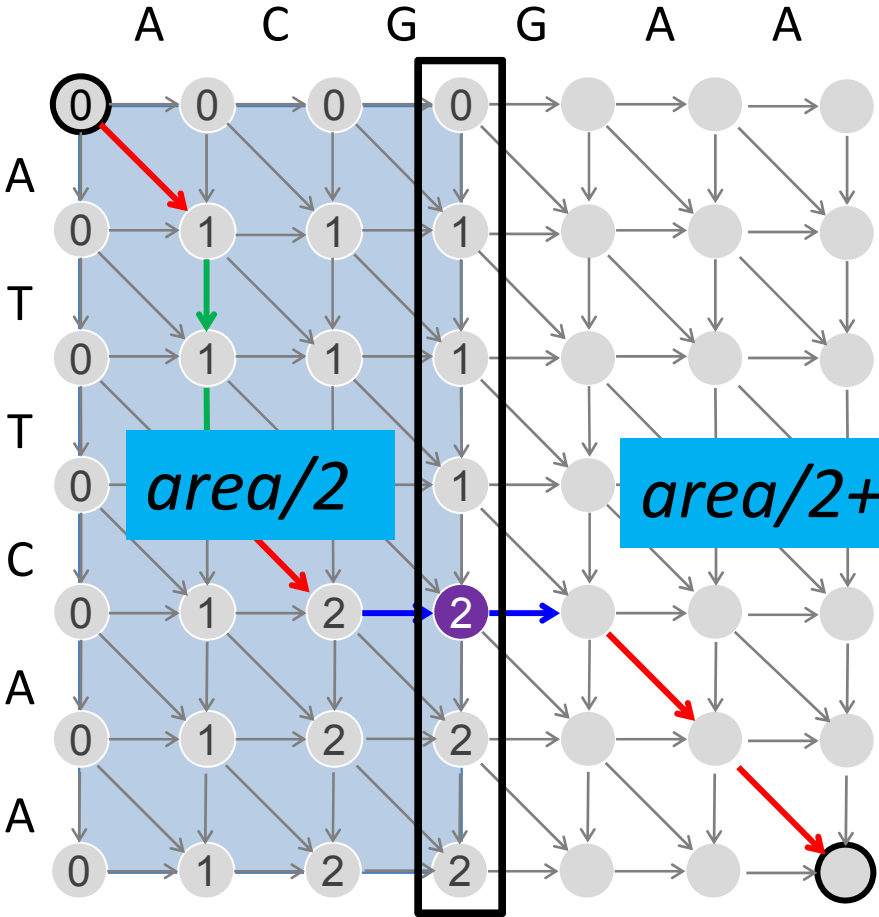


fromSource(i)

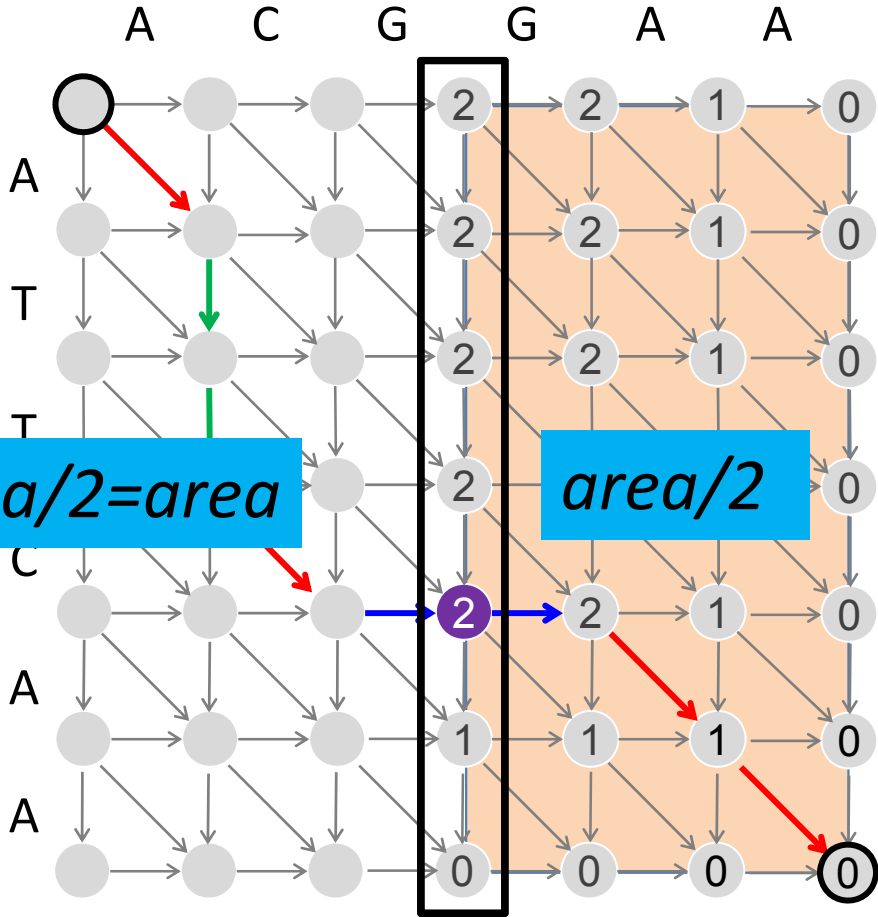


toSink(i)

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

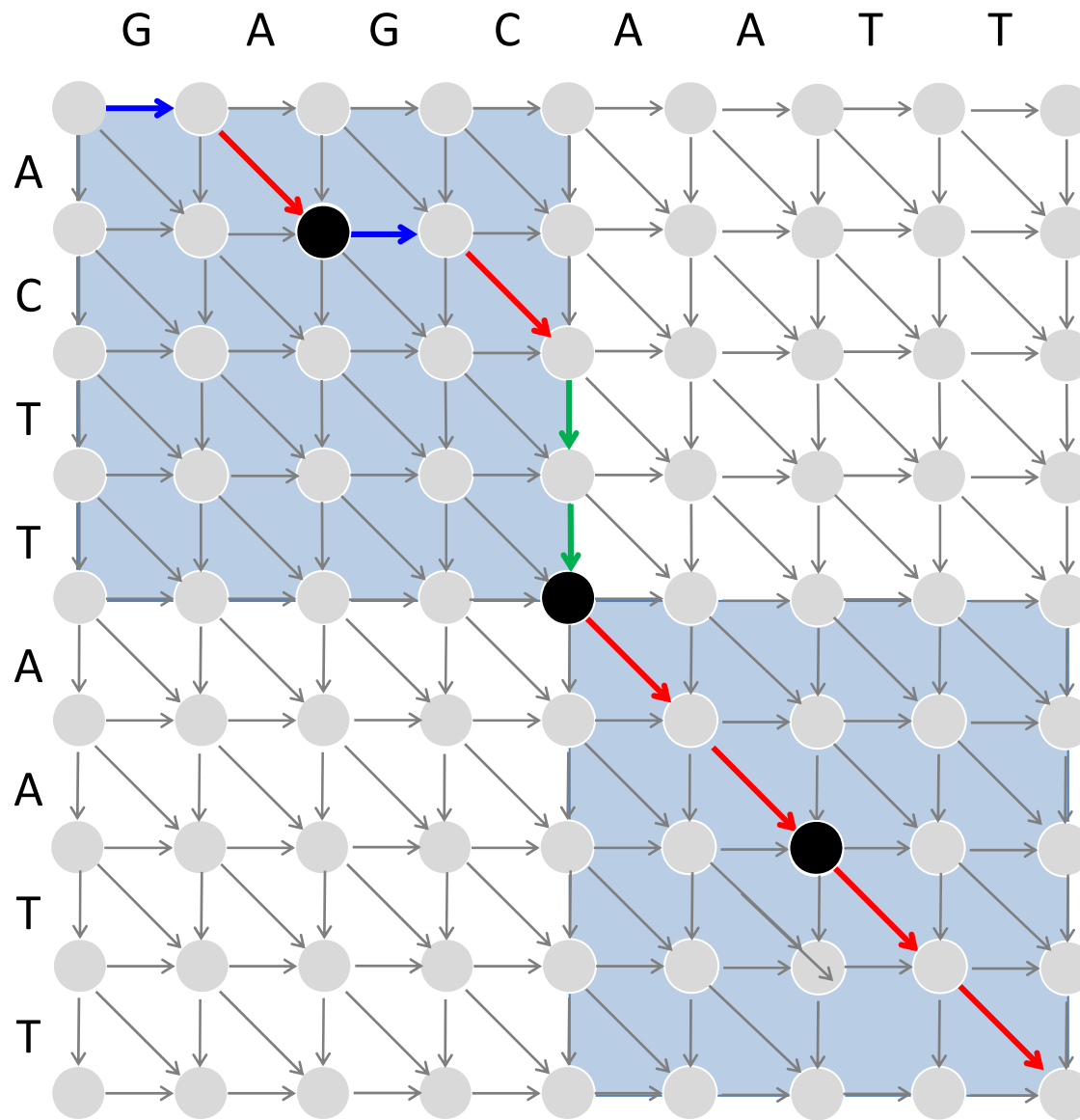


fromSource(i)



toSink(i)

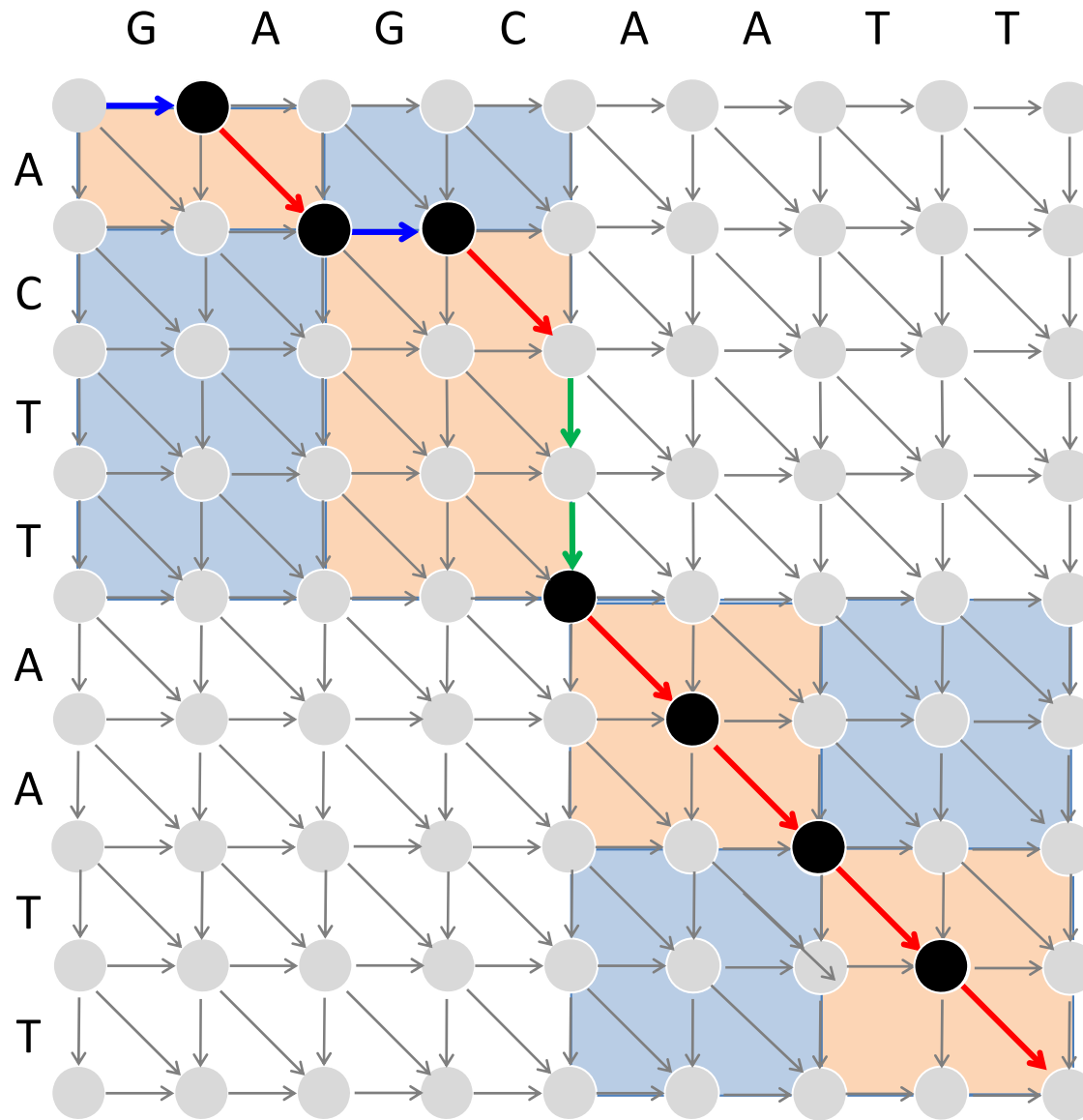
Laughable Progress: $O(nm)$ Time to Find **ONE** Node!



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!

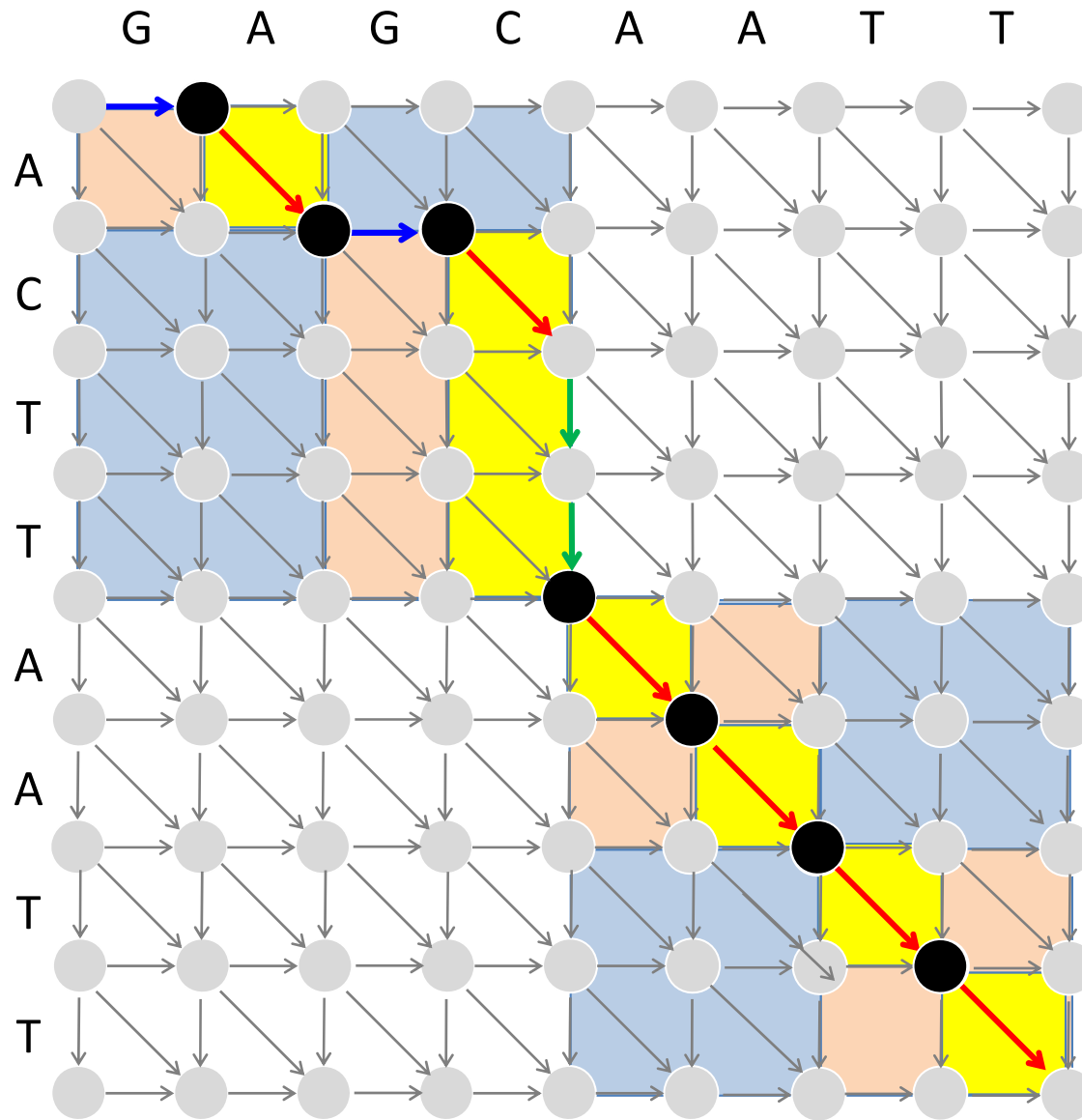


Each subproblem can be conquered in time proportional to its area:

$$\text{area}/8 + \text{area}/8 + \text{area}/8 + \text{area}/8 = \text{area}/4$$

How much time would it take to conquer 4 subproblems?

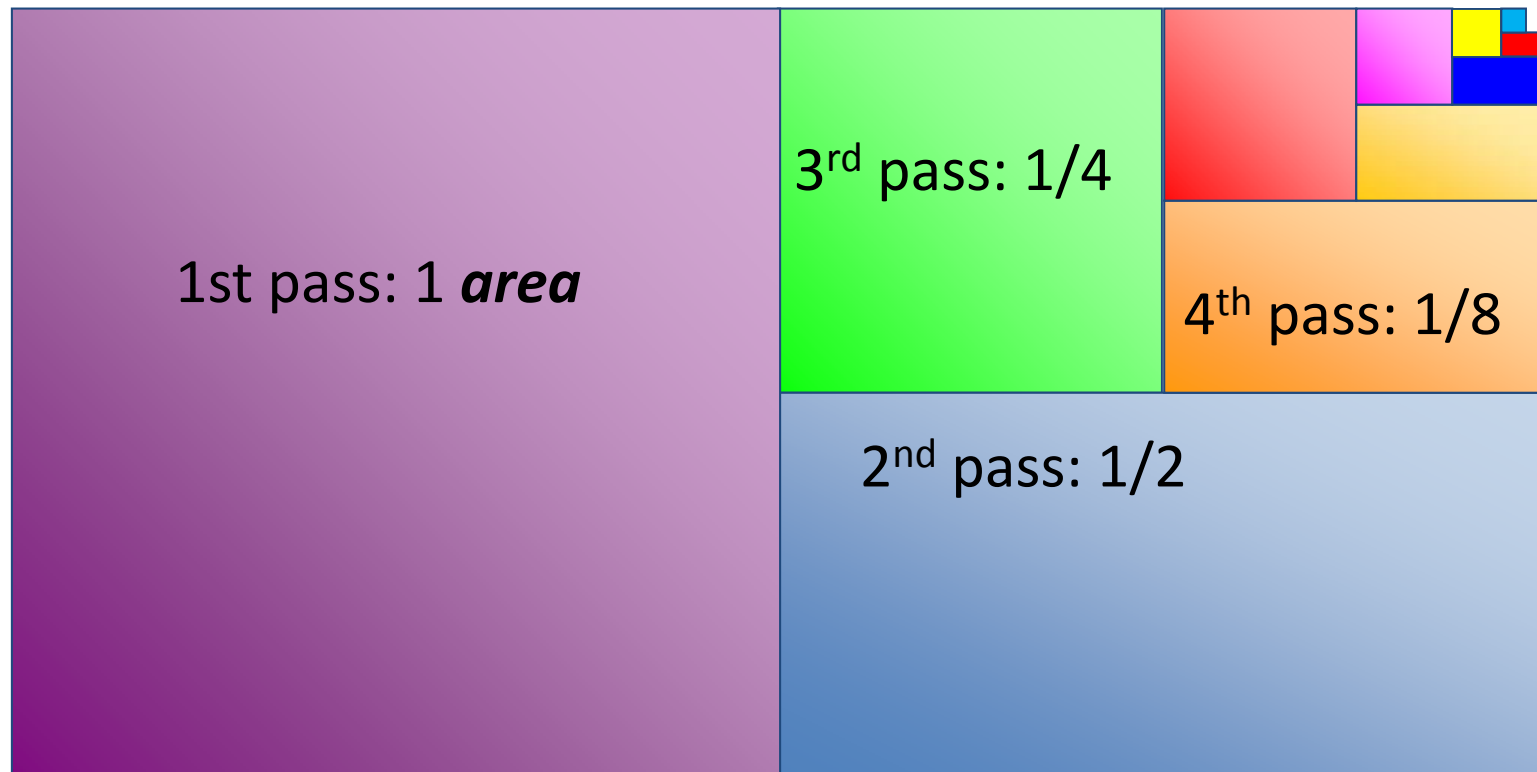
$O(nm+nm/2+nm/4)$ Time to Find **NEARLY ALL** Nodes!



$$\begin{aligned}
 & \text{area} + \\
 & \text{area}/2 + \\
 & \text{area}/4 + \\
 & \text{area}/8 + \\
 & \text{area}/16 + \\
 & \dots + \\
 & < \\
 & 2 \cdot \text{area}
 \end{aligned}$$

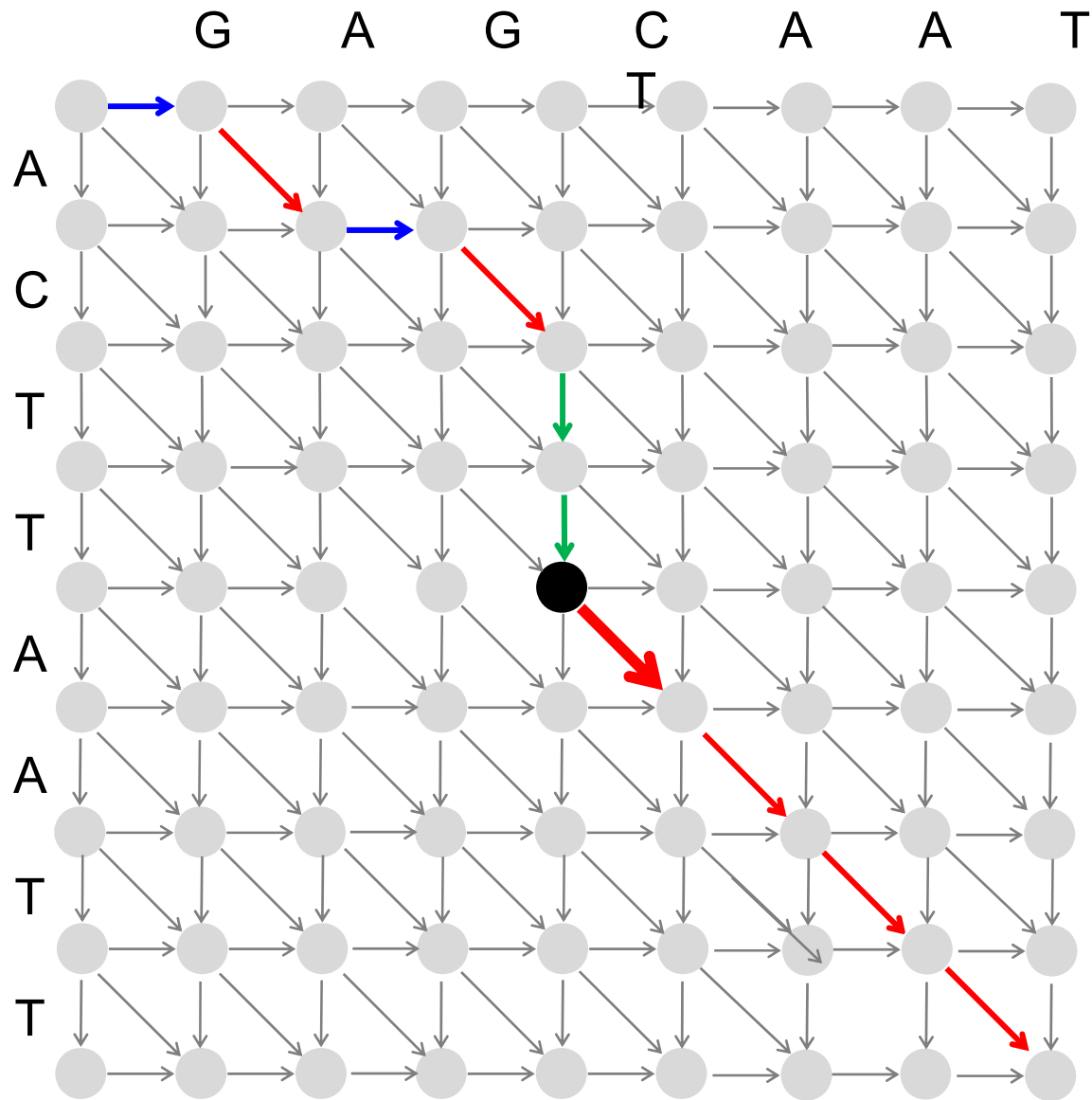
How much time would it take to conquer ALL subproblems?

Total Time: $area + area/2 + area/4 + area/8 + area/16 + \dots$



$$1 + \frac{1}{2} + \frac{1}{4} + \dots < 2$$

The Middle Edge

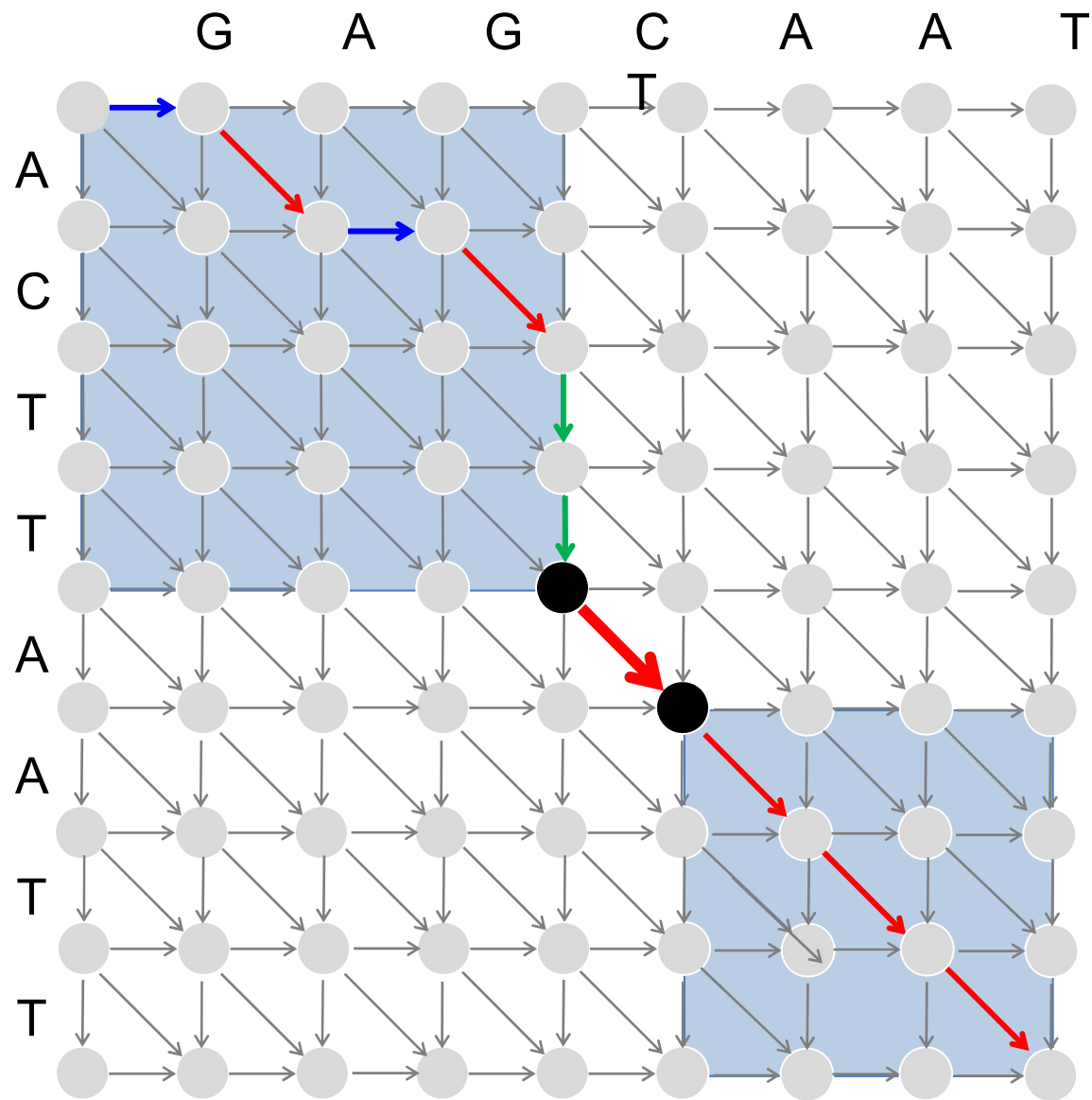


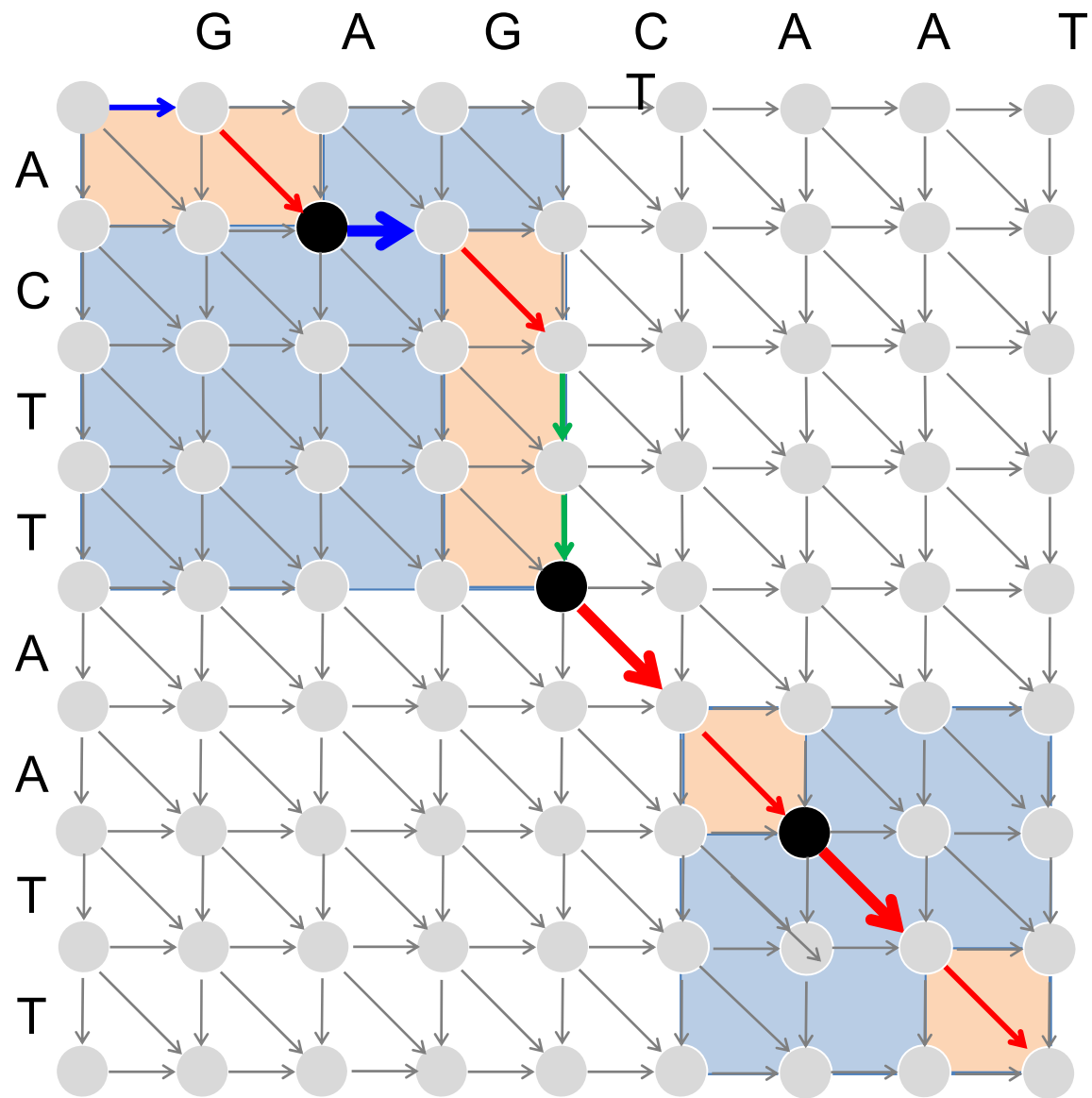
Middle Edge:
an edge in an
optimal
alignment path
starting at the
middle node

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*).





Recursive LinearSpaceAlignment

```
LinearSpaceAlignment(top,bottom,left,right)
  if left = right
    return alignment formed by bottom-top edges “↓”
  middle ←  $\lfloor (\textit{left} + \textit{right}) / 2 \rfloor$ 
  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)
```

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

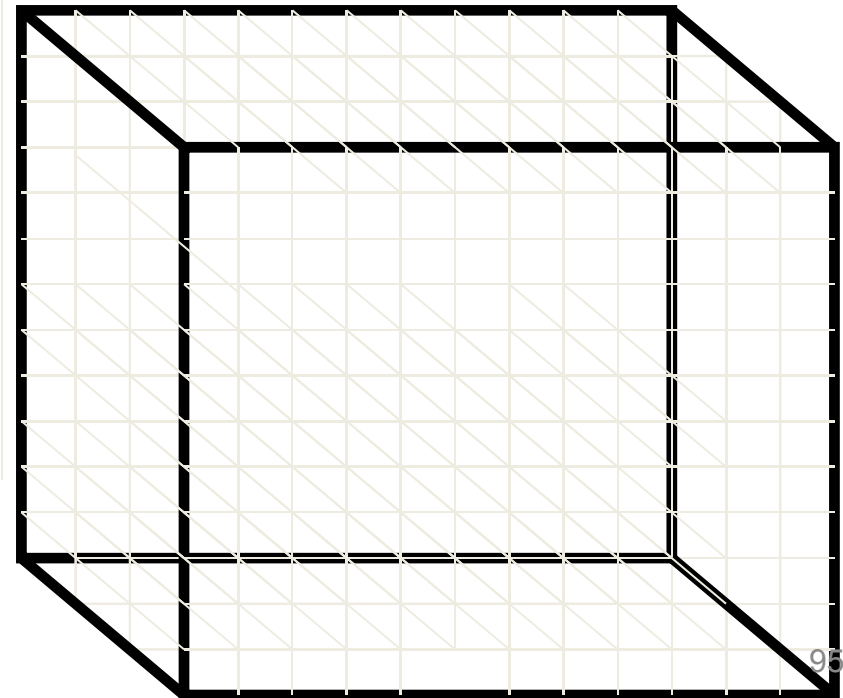
#symbols up to a given position

Alignments = Paths in 3-D

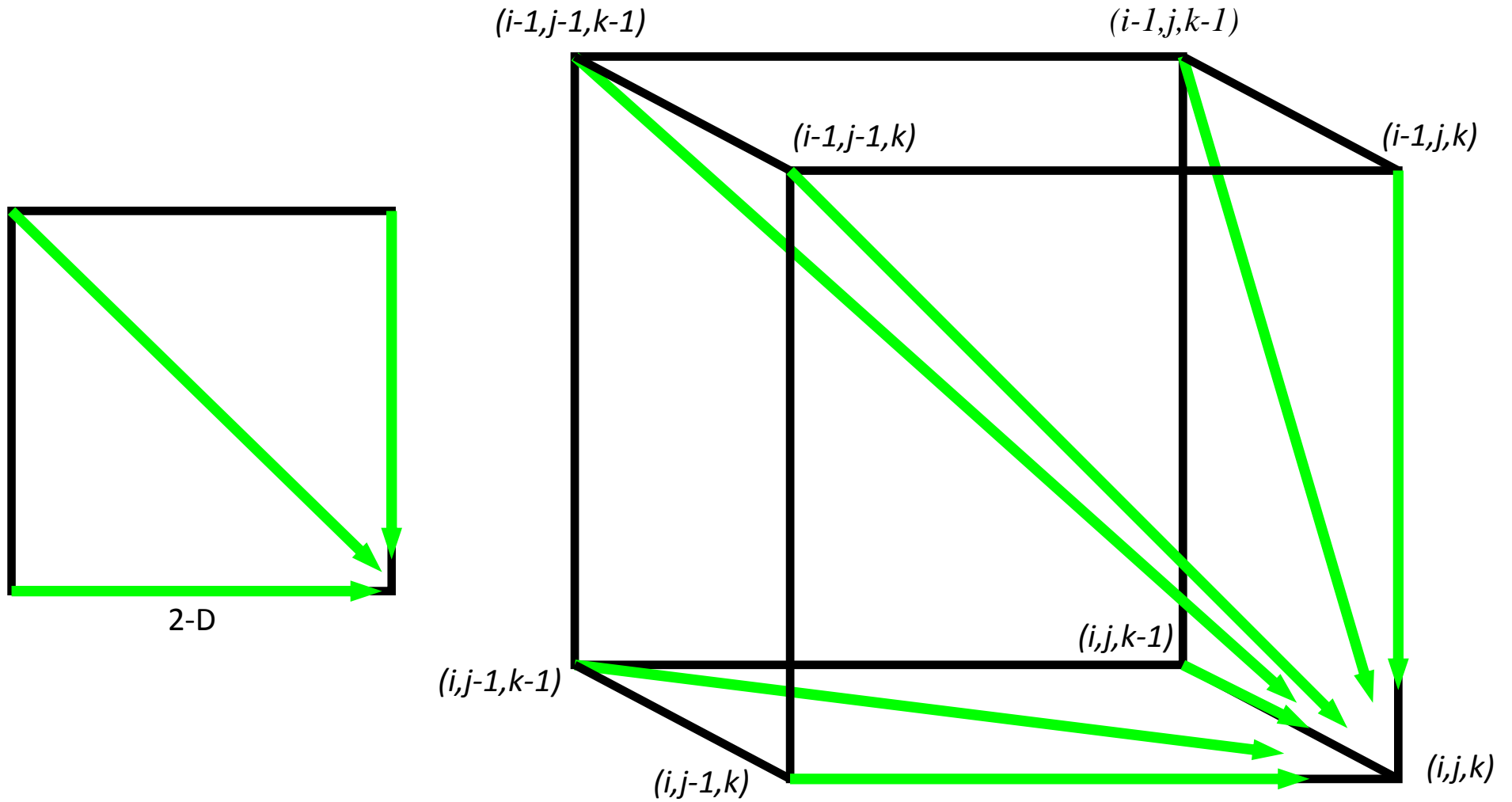
- Alignment of ATGC, AATC, and ATGC

$(0,0,0) \rightarrow (1,1,0) \rightarrow (1,2,1) \rightarrow (2,3,2) \rightarrow (3,3,3) \rightarrow (4,4,4)$

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

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

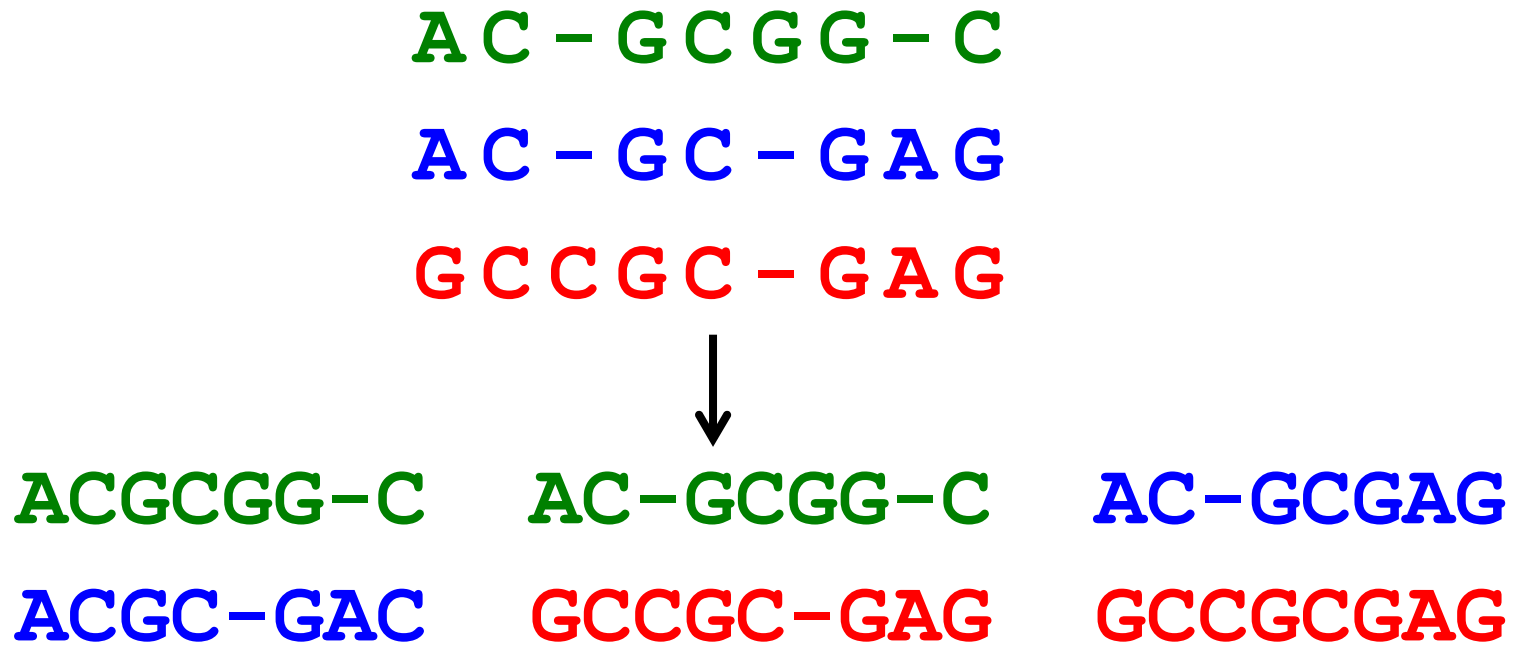
- $\delta(x, y, z)$ is an entry in the 3-D scoring matrix.

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:



Idea: Construct Multiple from Pairwise Alignments

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

AAAATTTT-----
-----TTTTGGGG

-----AAAATTTT
GGGGAAAA-----

TTTTGGGG-----
-----GGGGAAAA

Aligning Profile Against Profile

- In the past we were aligning a **sequence against a sequence**.
 - Can we align a **sequence against a profile**?
 - Can we align a **profile against a profile**?

	-	A	G	G	C	T	A	T	C	A	C	C	T	G
T	A	G	-	C	T	A	C	C	A	-	-	-	-	G
C	A	G	-	C	T	A	C	C	A	-	-	-	-	G
C	A	G	-	C	T	A	T	C	A	C	-	G	G	G
C	A	G	-	C	T	A	T	C	G	C	-	G	G	G
A	0	1	0	0	0	0	1	0	0	.8	0	0	0	0
C	.6	0	0	0	1	0	0	.4	1	0	.6	.2	0	0
G	0	0	1	.2	0	0	0	0	0	.2	0	0	.4	1
T	.2	0	0	0	0	1	0	.6	0	0	0	0	.2	0
-	.2	0	0	.8	0	0	0	0	0	0	.4	.8	.4	0

Multiple Alignment: Greedy Approach

- Choose the most similar sequences and combine them into a profile, thereby reducing alignment of k sequences to an alignment of $k - 2$ sequences and 1 profile.
- Iterate

Greedy Approach: Example

- Sequences: GATTCA, GTCTGA, GATATT, GTCAGC.
- 6 pairwise alignments (premium for **match** +1, penalties for **indels** and **mismatches** -1)

s2 GTCTGA
s4 GTCAGC (score = 2)

s1 GATTCA--
s4 G-T-CAGC (score = 0)

s1 GAT-TCA
s2 G-TCTGA (score = 1)

s2 G-TCTGA
s3 GATAT-T (score = -1)

s1 GAT-TCA
s3 GATAT-T (score = 1)

s3 GAT-ATT
s4 G-TCAGC (score = -1)

Greedy Approach: Example

- Since s_2 and s_4 are closest, we consolidate them into a profile:

$$\left. \begin{array}{l} s_2 \quad \text{GTC} \mathbf{TGA} \\ s_4 \quad \text{GTC} \mathbf{AGC} \end{array} \right\} s_{2,4} = \text{GTC} \mathbf{t/aGa/cA}$$

- New set of 3 sequences to align:

$$\begin{array}{l} s_1 \quad \text{GATTCA} \\ s_3 \quad \text{GATATT} \\ s_{2,4} \quad \text{GTC} \mathbf{t/aGa/c} \end{array}$$

Other alphabets

- Amino acid space (n=20)
- Codon space (n=64)

Summary

- Pairwise alignment can be performed with dynamic programming
- Multiple alignment often uses heuristics
- How best to choose the scoring matrix?
- Current work in this area focuses on aligning coding sequences at the codon (3 bases) level
 - Complicated due to:
 - Insertions and deletions
 - Different reading frames

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

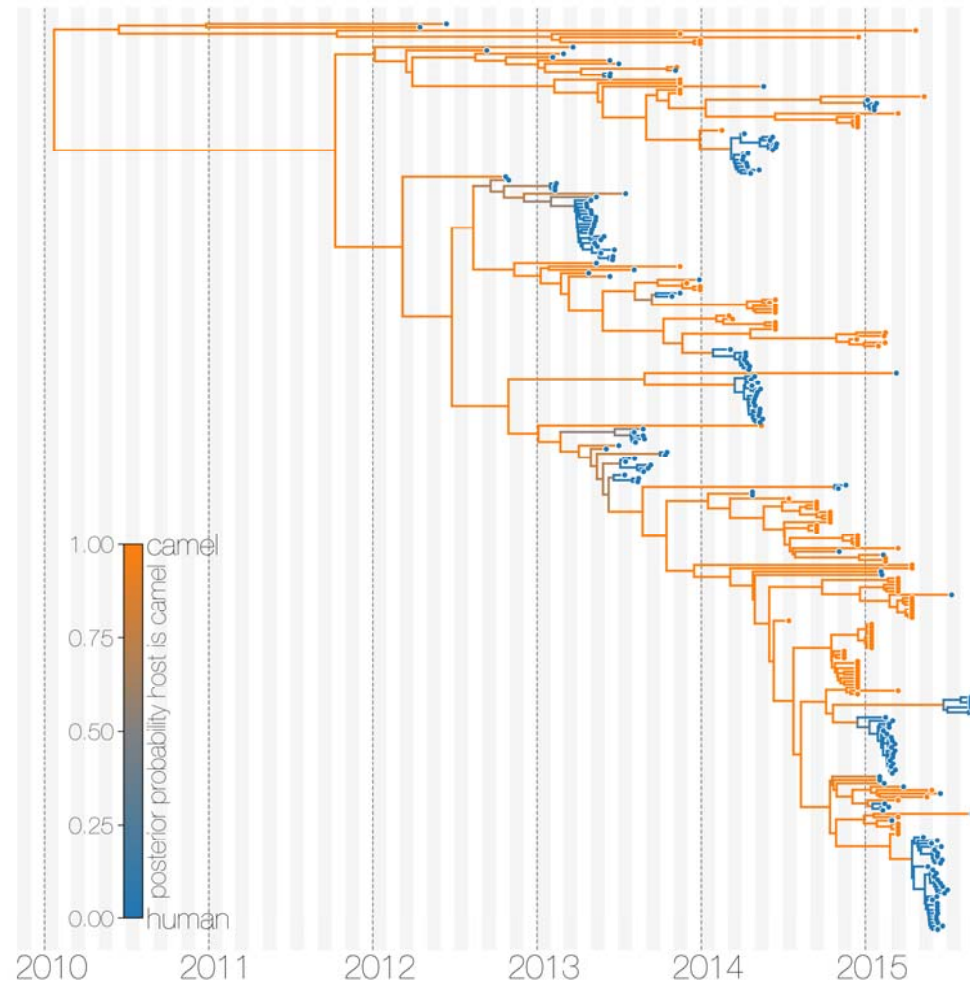
Phylogeny

- A phylogenetic tree represents inferred evolutionary relationships among various biological species or other entities based upon similarities and differences in their physical or genetic characteristics
- The taxa joined together in the tree are implied to have descended from a common ancestor

What do we use phylogenies for?

- Inferring relationships between taxa
- Identifying common ancestors
- With additional information, we can infer when different taxa diverged

Phylogenetic tree of MERS-Coronavirus in humans and camels



Methods for reconstructing phylogenies

- Distance
- Parsimony
- Likelihood

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	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.

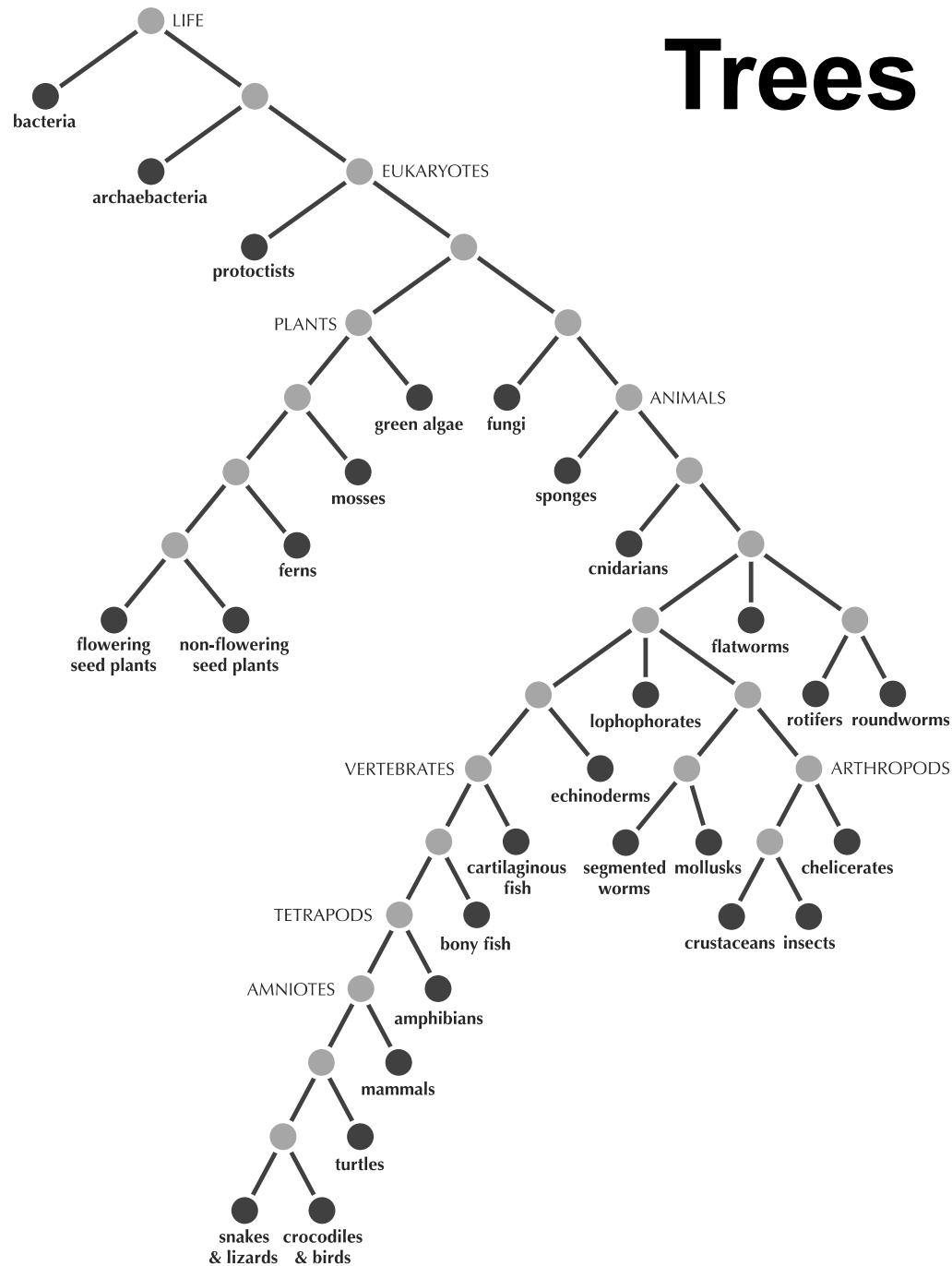
SPECIES	ALIGNMENT	DISTANCE MATRIX			
		Chimp	Human	Seal	Whale
Chimp	ACGTAGGCCT	0	3	6	4
Human	ATGTAAGACT	3	0	7	5
Seal	TCGAGAGCAC	6	7	0	2
Whale	TCGAAAGCAT	4	5	2	0

How else could we form a distance matrix?

Aside: more realistic models of nucleotide distance

- Counting the number of differences may not be a good measure of divergence time
 - Base frequencies are skewed
 - The rate at which one base substitutes for another differ
 - ‘Models_of_DNA_evolution’ in Wikipedia
- What about:
 - Insertions and deletions?
 - Ambiguous nucleotides

Trees

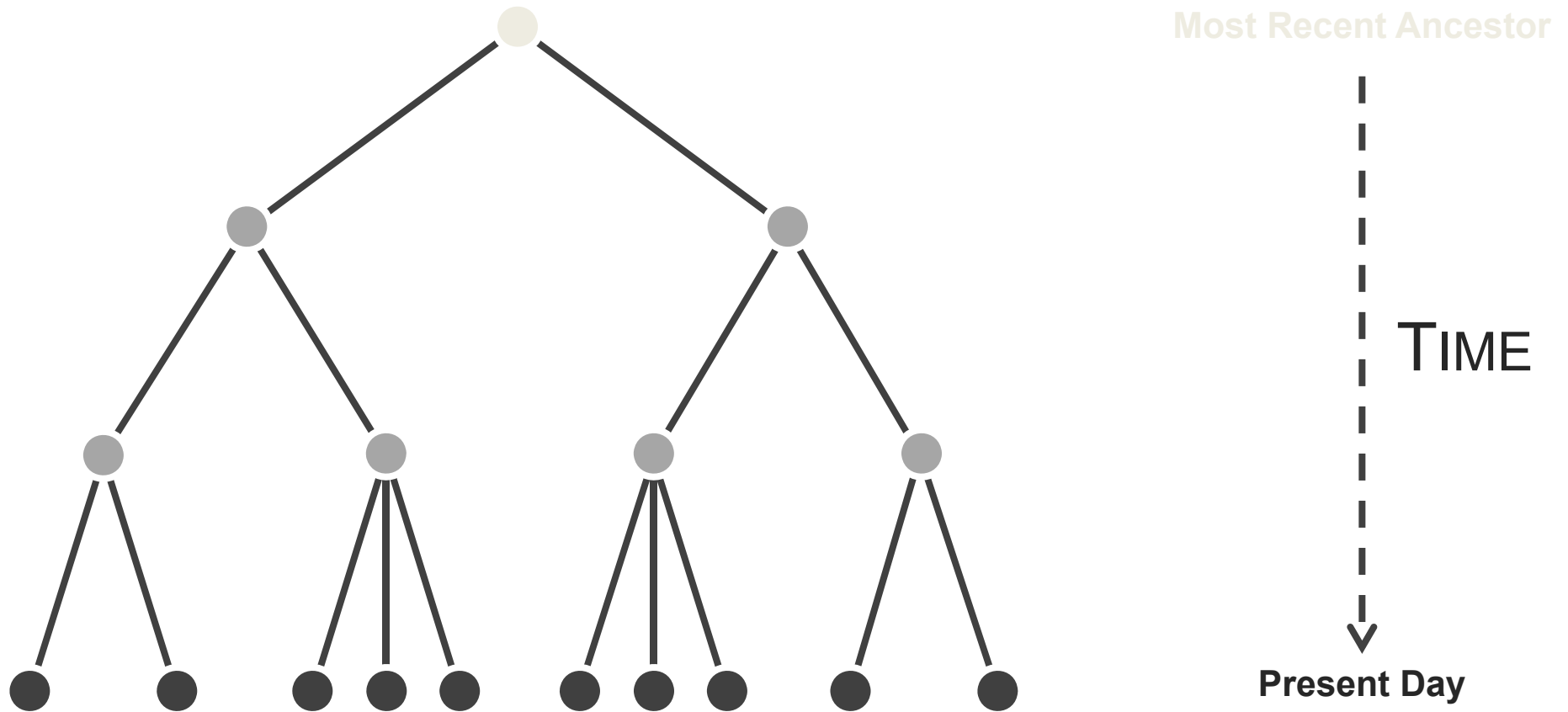


Tree: Connected graph containing no cycles.

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)

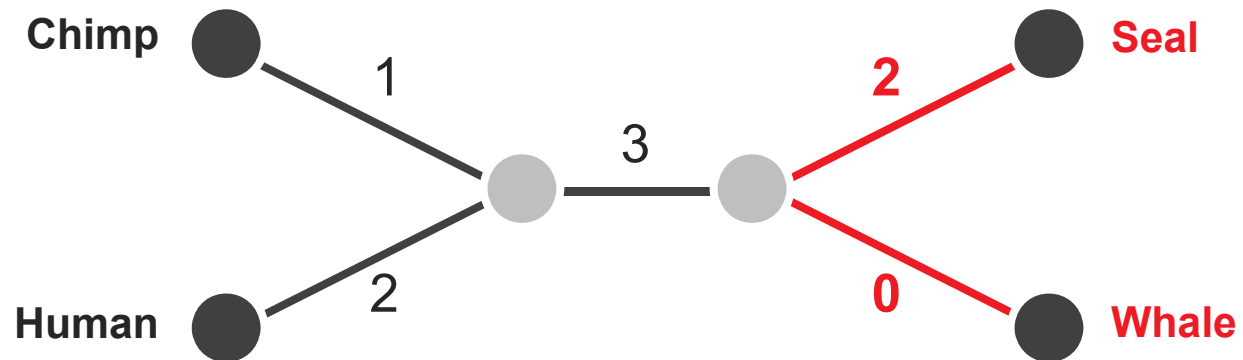
Distance-Based Phylogeny

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

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

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>i</i>	<i>j</i>	<i>k</i>	<i>l</i>
<i>i</i>	0	3	4	3
<i>j</i>	3	0	4	5
<i>k</i>	4	4	0	2
<i>l</i>	3	5	2	0

No Tree Fits a Matrix

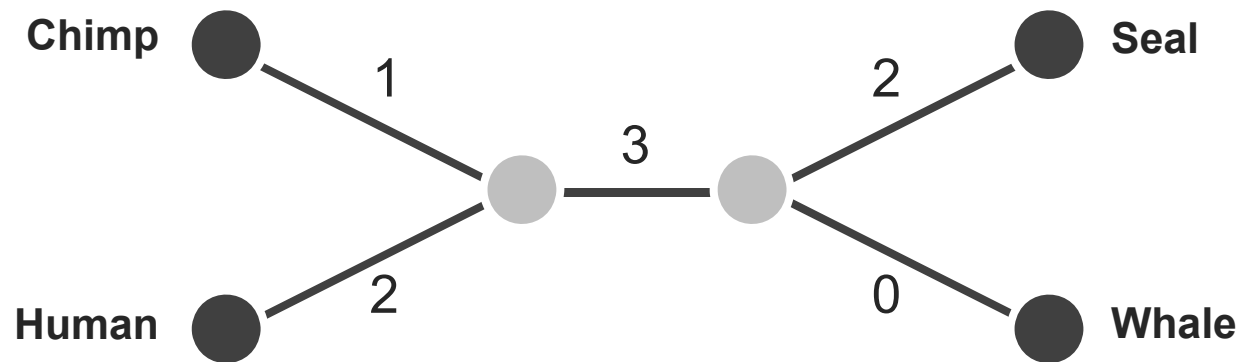
Exercise Break: Try fitting a tree to the following matrix.

	<i>i</i>	<i>j</i>	<i>k</i>	<i>l</i>
<i>i</i>	0	3	4	3
<i>j</i>	3	0	4	5
<i>k</i>	4	4	0	2
<i>l</i>	3	5	2	0

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

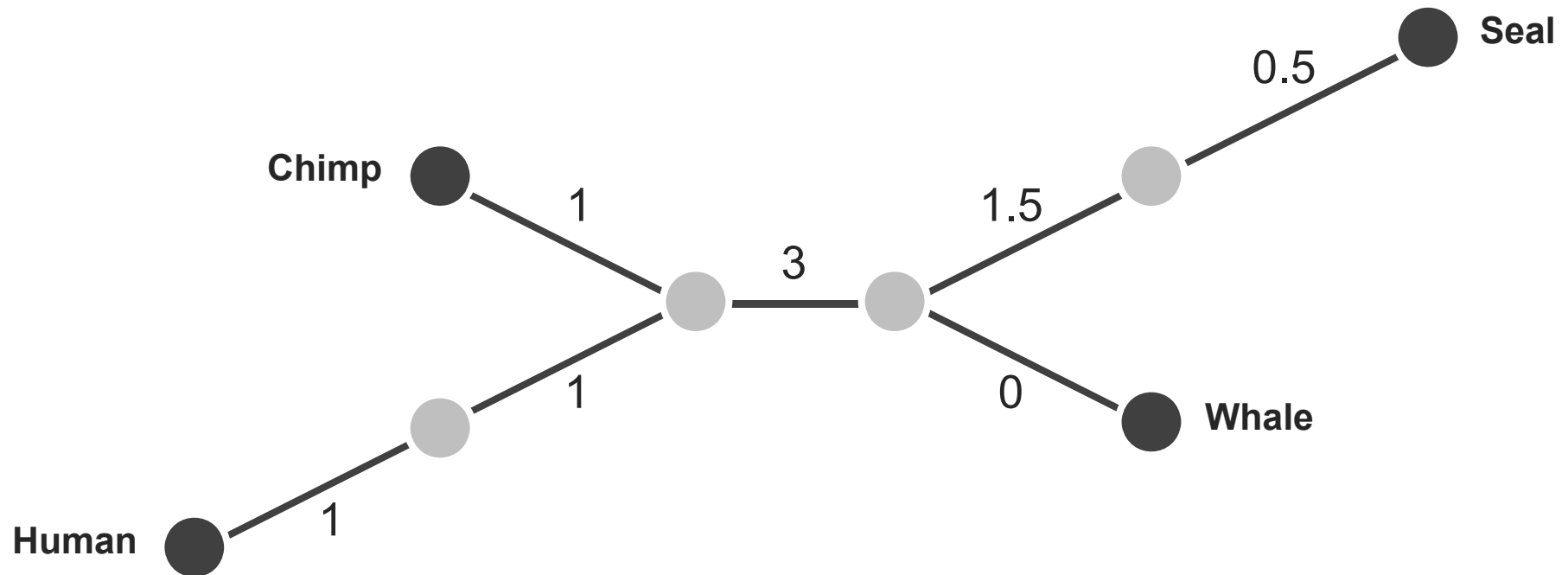
More Than One Tree Fits a Matrix

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

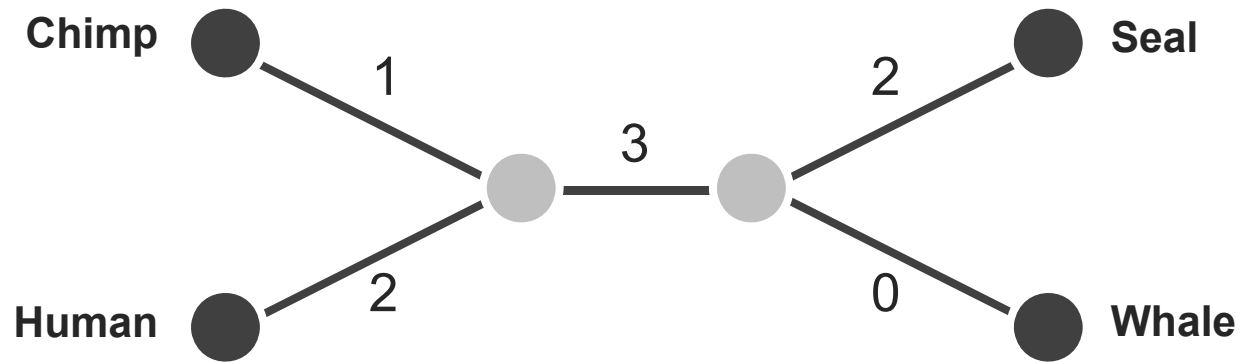


More Than One Tree Fits a Matrix

	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.

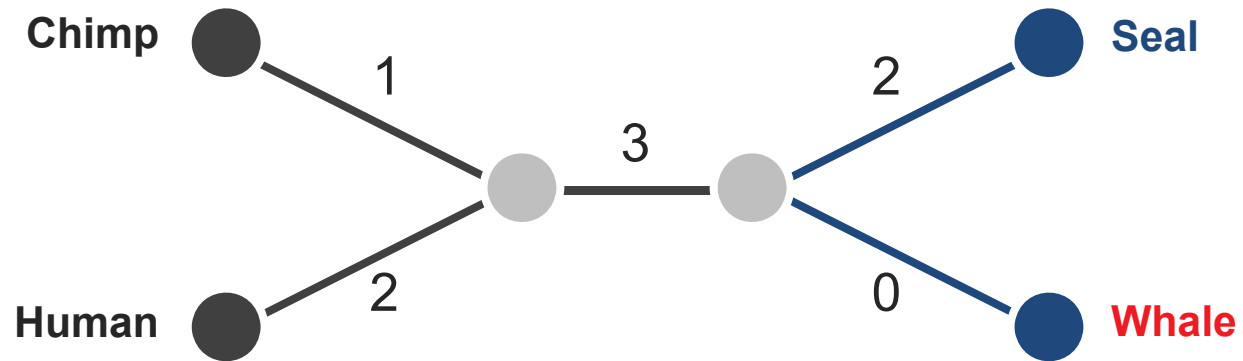
Reformulating Distance-Based Phylogeny

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

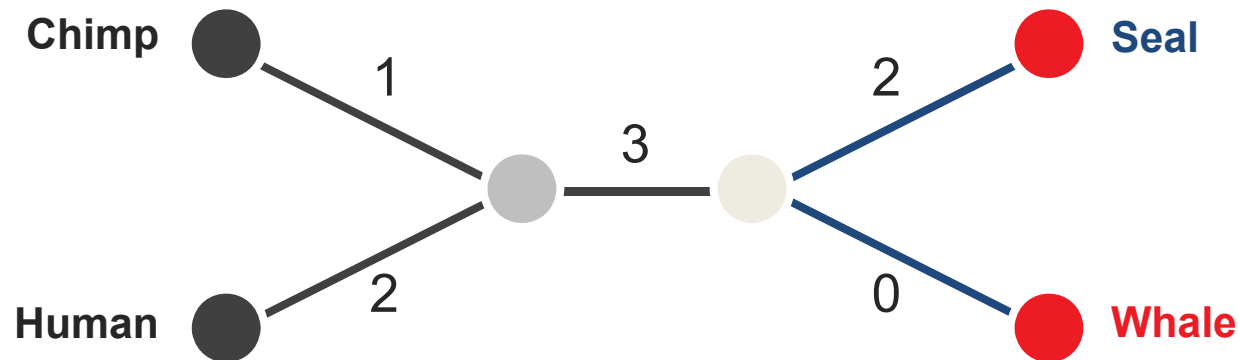
	Chimp	Human	Seal	Whale
Chimp	0	3	6	4
Human	3	0	7	5
Seal	6	7	0	2
Whale	4	5	2	0



An Idea for Distance-Based Phylogeny

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

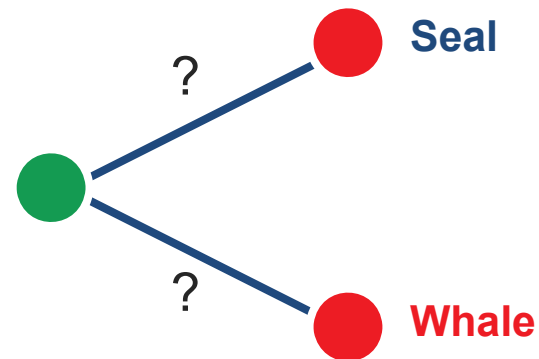
Theorem: Every simple tree with at least two nodes has at least one pair of neighbouring leaves.



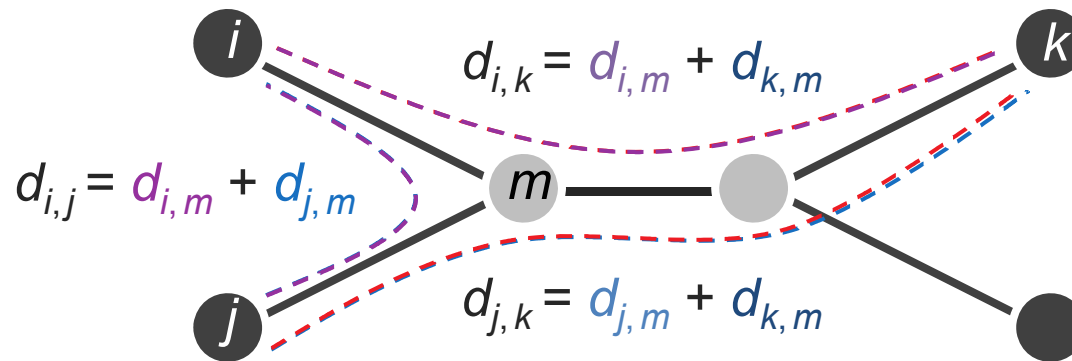
An Idea for Distance-Based Phylogeny

	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?

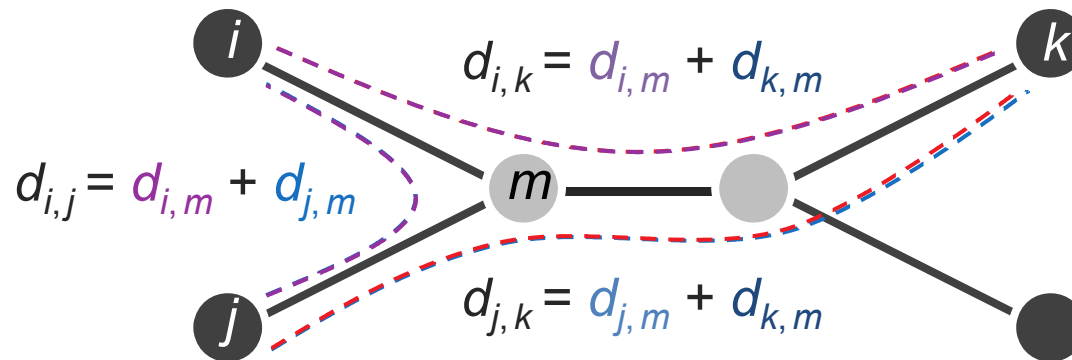


Toward a Recursive Algorithm



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

Toward a Recursive Algorithm



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

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

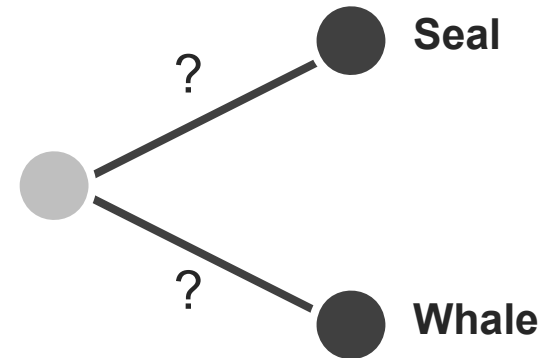
$$d_{k,m} = (D_{i,k} + D_{j,k} - D_{i,j}) / 2$$

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

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

An Idea for Distance-Based Phylogeny

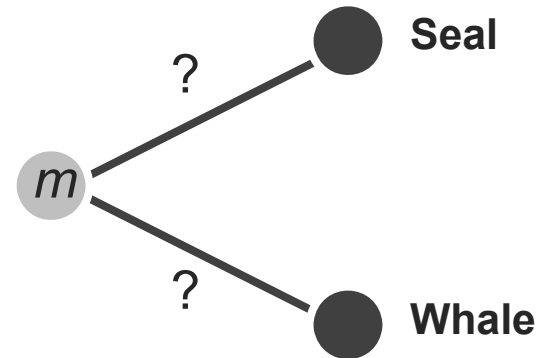
	Chimp	Human	Seal	Whale
Chimp	0	3	6	4
Human	3	0	7	5
Seal	6	7	0	2
Whale	4	5	2	0



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

An Idea for Distance-Based Phylogeny

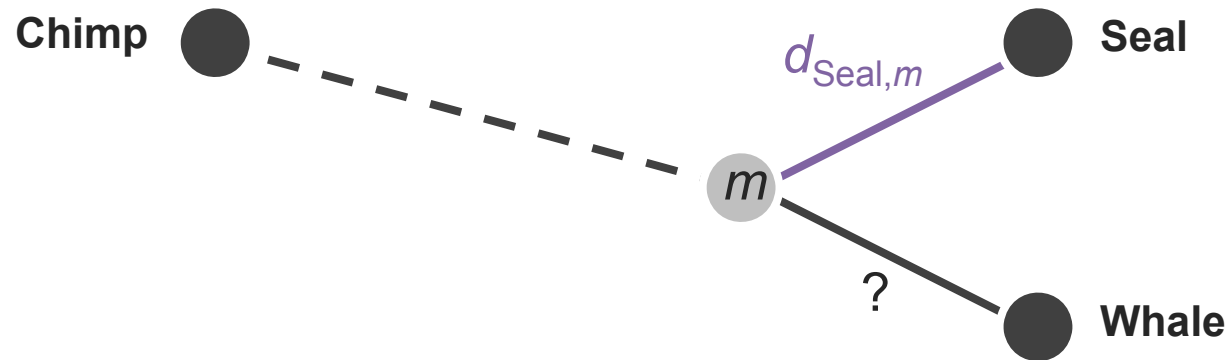
	Chimp	Human	Seal	Whale
Chimp	0	3	6	4
Human	3	0	7	5
Seal	6	7	0	2
Whale	4	5	2	0



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

An Idea for Distance-Based Phylogeny

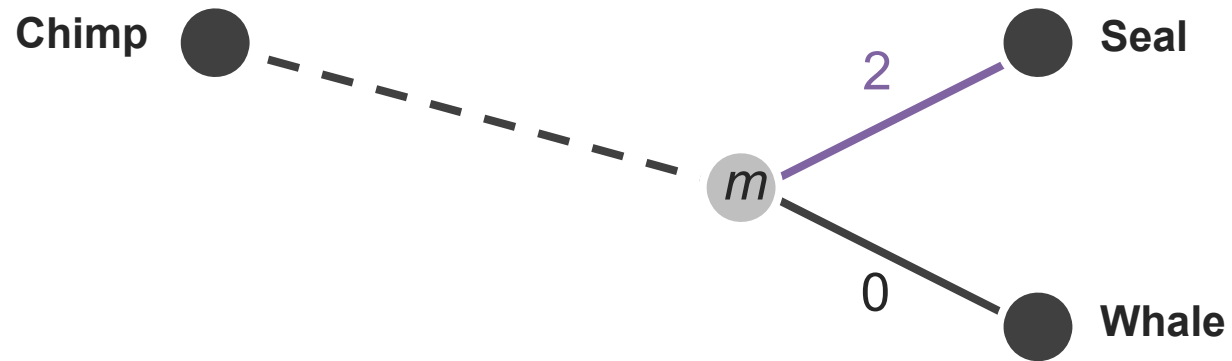
	Chimp	Human	Seal	Whale
Chimp	0	3	6	4
Human	3	0	7	5
Seal	6	7	0	2
Whale	4	5	2	0



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

An Idea for Distance-Based Phylogeny

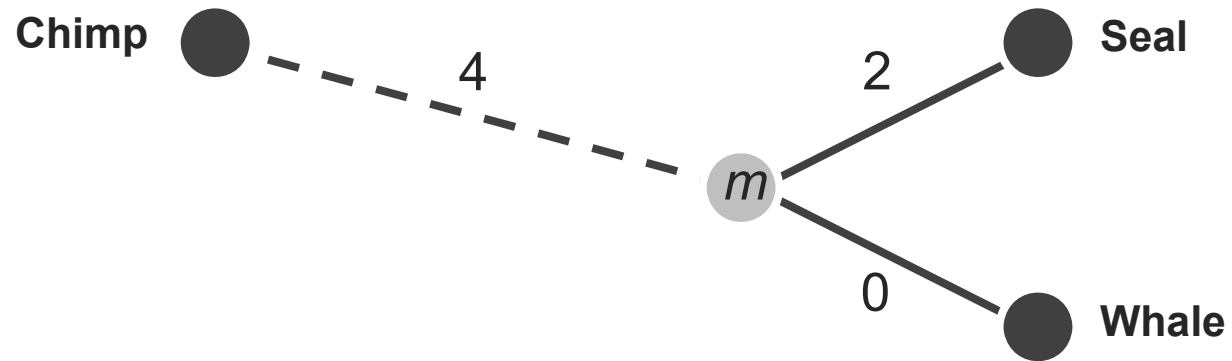
	Chimp	Human	Seal	Whale
Chimp	0	3	6	4
Human	3	0	7	5
Seal	6	7	0	2
Whale	4	5	2	0



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

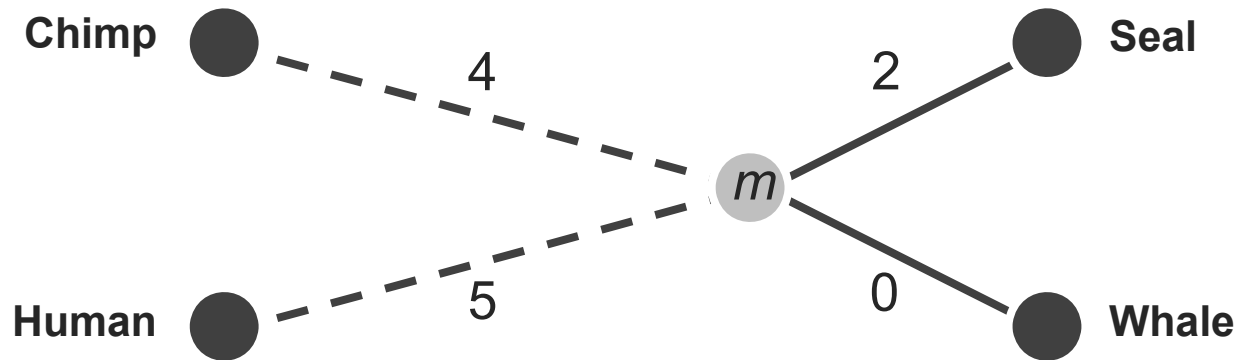
An Idea for Distance-Based Phylogeny

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



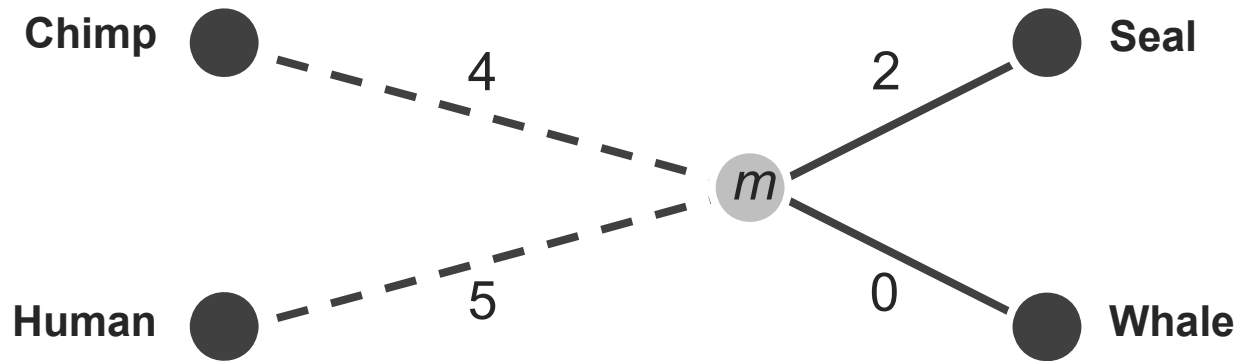
An Idea for Distance-Based Phylogeny

	Chimp	Human	Seal	Whale	<i>m</i>
Chimp	0	3	6	4	4
Human	3	0	7	5	5
Seal	6	7	0	2	2
Whale	4	5	2	0	0
<i>m</i>	4	5	2	0	0



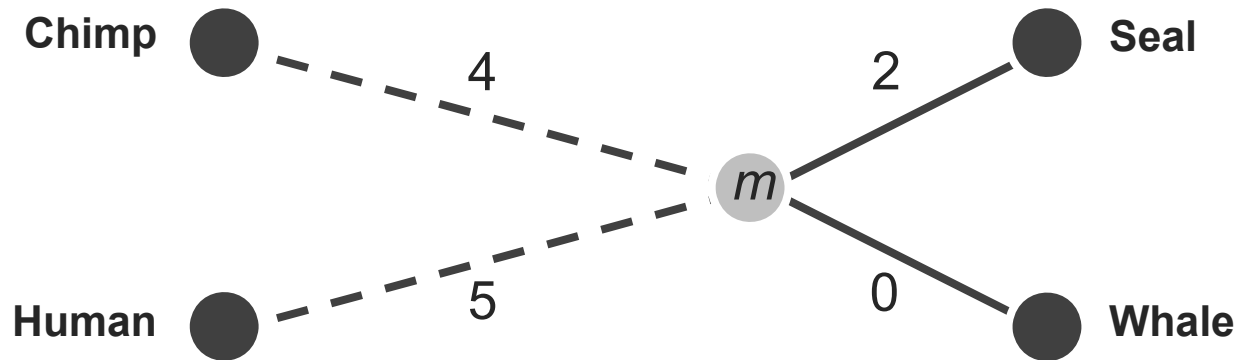
An Idea for Distance-Based Phylogeny

	Chimp	Human	Seal	Whale	<i>m</i>
Chimp	0	3	6	4	4
Human	3	0	7	5	5
Seal	6	7	0	2	2
Whale	4	5	2	0	0
<i>m</i>	4	5	2	0	0



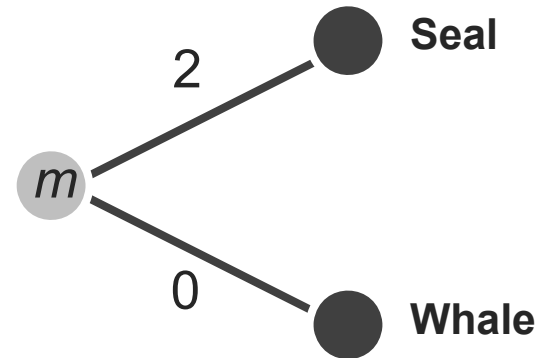
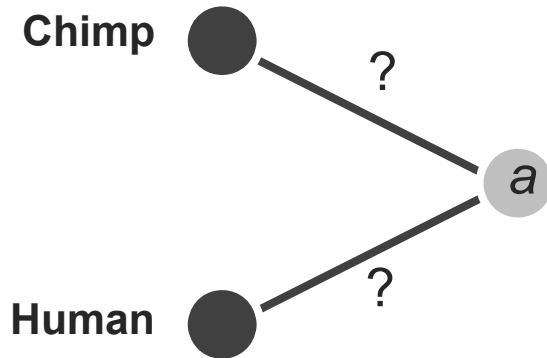
An Idea for Distance-Based Phylogeny

	Chimp	Human	<i>m</i>
Chimp	0	3	4
Human	3	0	5
<i>m</i>	4	5	0



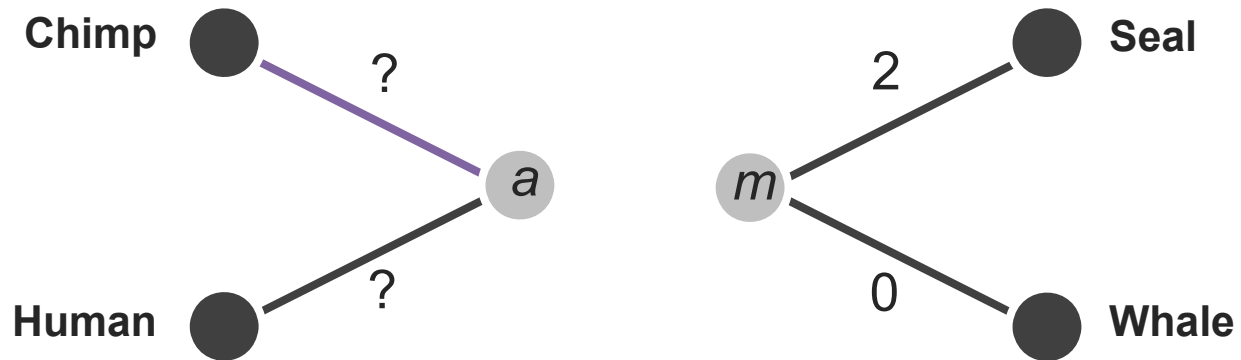
An Idea for Distance-Based Phylogeny

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



An Idea for Distance-Based Phylogeny

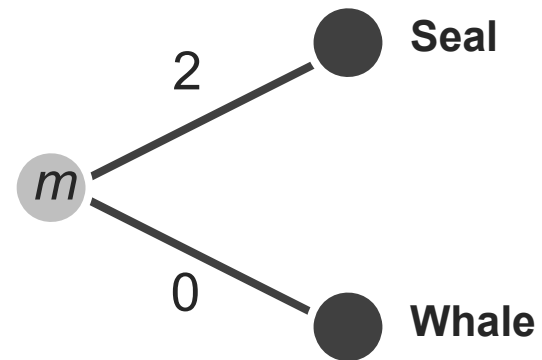
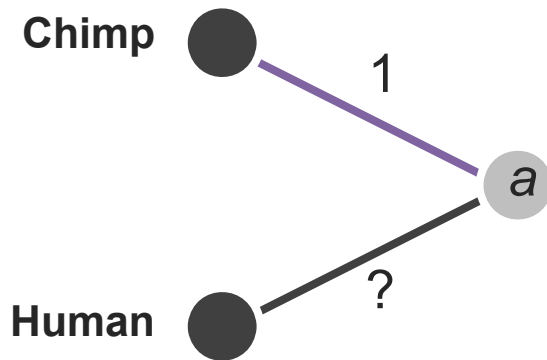
	Chimp	Human	<i>m</i>
Chimp	0	3	4
Human	3	0	5
<i>m</i>	4	5	0



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

An Idea for Distance-Based Phylogeny

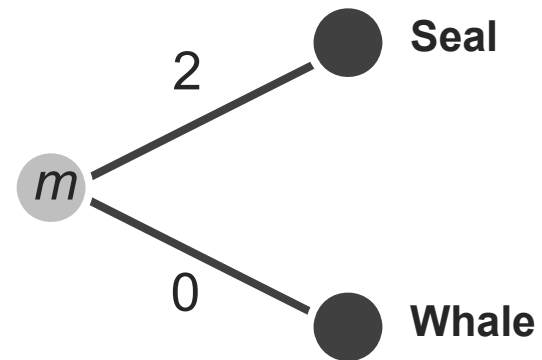
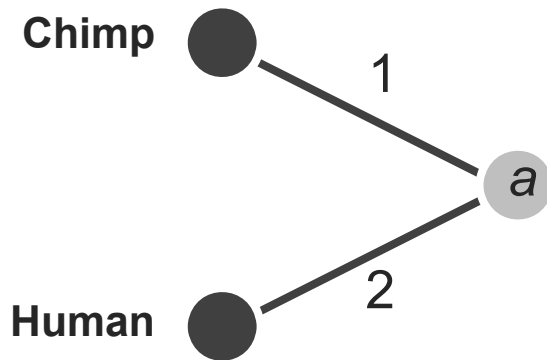
	Chimp	Human	<i>m</i>
Chimp	0	3	4
Human	3	0	5
<i>m</i>	4	5	0



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

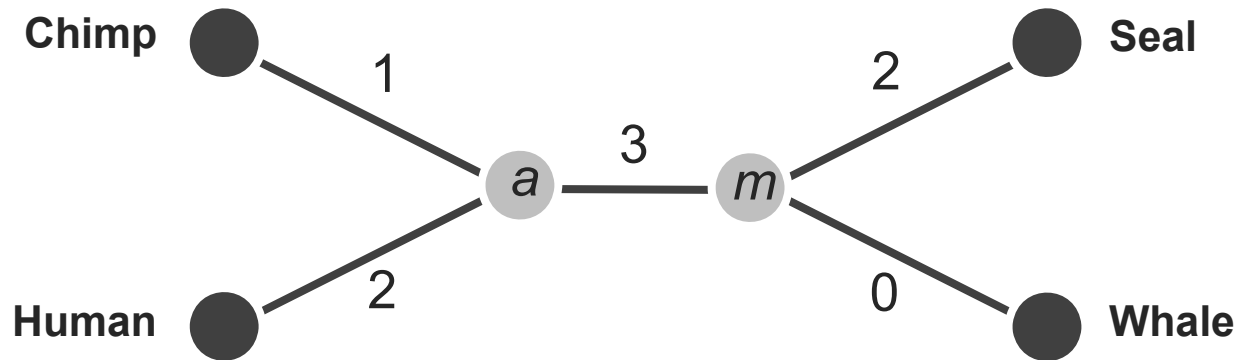
An Idea for Distance-Based Phylogeny

	Chimp	Human	<i>m</i>
Chimp	0	3	4
Human	3	0	5
<i>m</i>	4	5	0



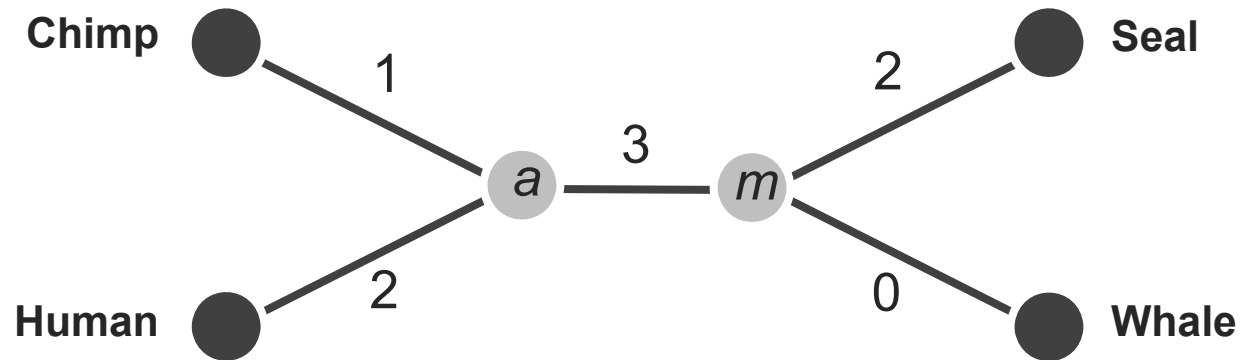
An Idea for Distance-Based Phylogeny

	Chimp	Human	<i>m</i>
Chimp	0	3	4
Human	3	0	5
<i>m</i>	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.

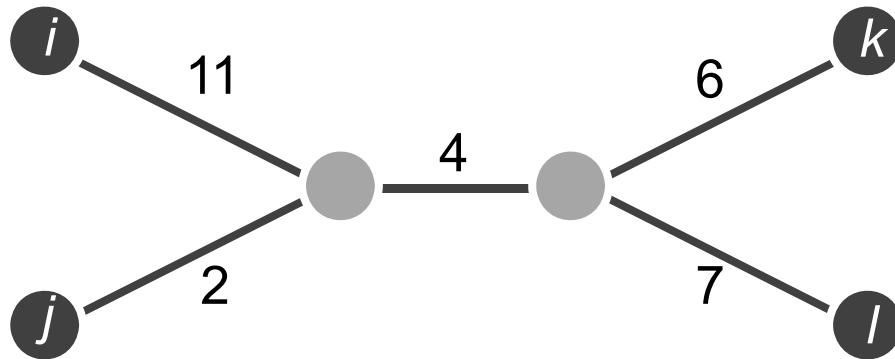
	<i>i</i>	<i>j</i>	<i>k</i>	<i>l</i>
<i>i</i>	0	13	21	22
<i>j</i>	13	0	12	13
<i>k</i>	21	12	0	13
<i>l</i>	22	13	13	0

What Was Wrong With Our Algorithm?

	<i>i</i>	<i>j</i>	<i>k</i>	<i>l</i>
<i>i</i>	0	13	21	22
<i>j</i>	13	0	12	13
<i>k</i>	21	12	0	13
<i>l</i>	22	13	13	0

What Was Wrong With Our Algorithm?

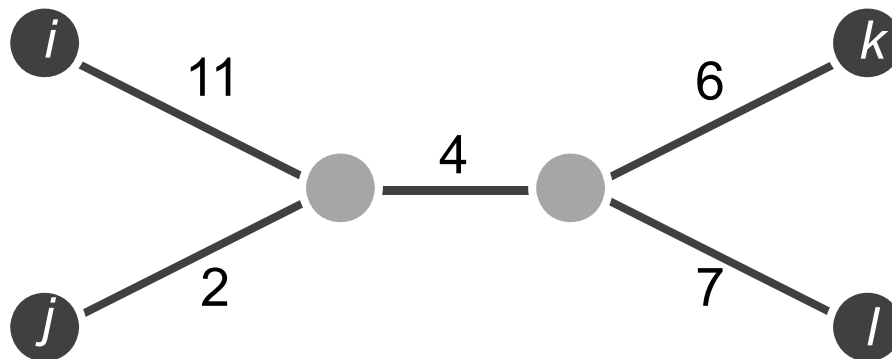
	<i>i</i>	<i>j</i>	<i>k</i>	<i>l</i>
<i>i</i>	0	13	21	22
<i>j</i>	13	0	12	13
<i>k</i>	21	12	0	13
<i>l</i>	22	13	13	0



What Was Wrong With Our Algorithm?

	<i>i</i>	<i>j</i>	<i>k</i>	<i>l</i>
<i>i</i>	0	13	21	22
<i>j</i>	13	0	12	13
<i>k</i>	21	12	0	13
<i>l</i>	22	13	13	0

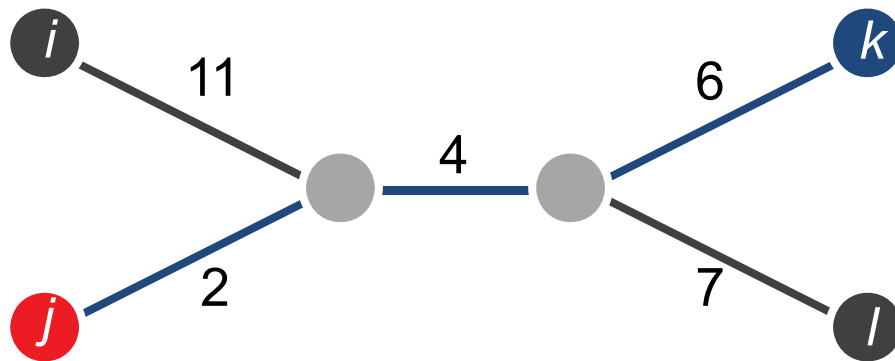
minimum
element is $D_{j,k}$



What Was Wrong With Our Algorithm?

	<i>i</i>	<i>j</i>	<i>k</i>	<i>l</i>
<i>i</i>	0	13	21	22
<i>j</i>	13	0	12	13
<i>k</i>	21	12	0	13
<i>l</i>	22	13	13	0

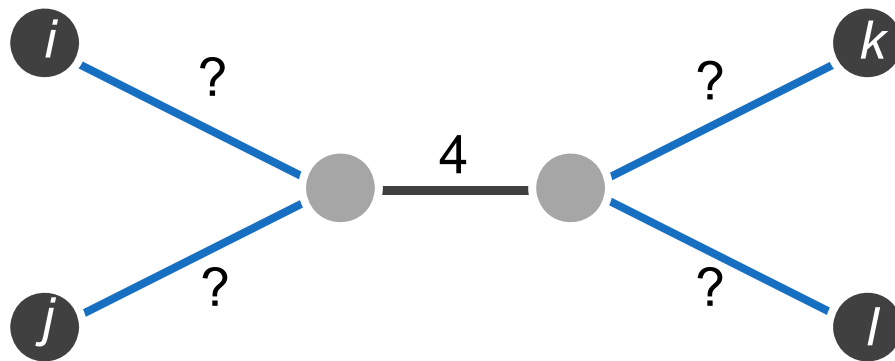
minimum
element is $D_{j,k}$



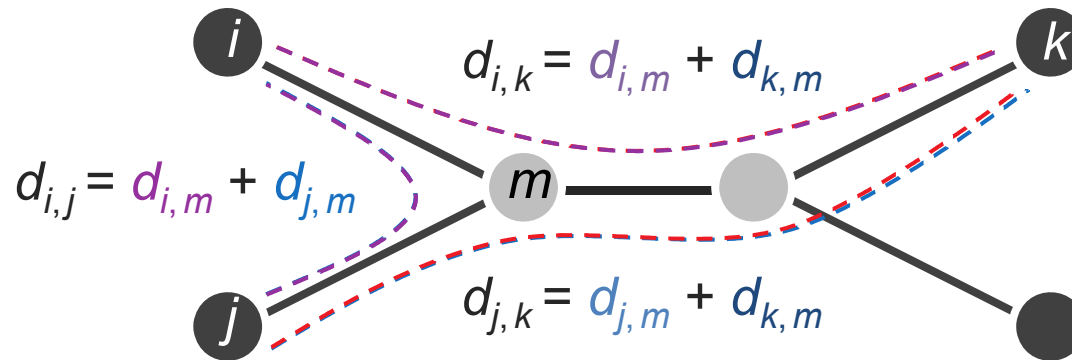
j and *k* are
not neighbors!

From Neighbours to Limbs

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



From Neighbours to Limbs



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

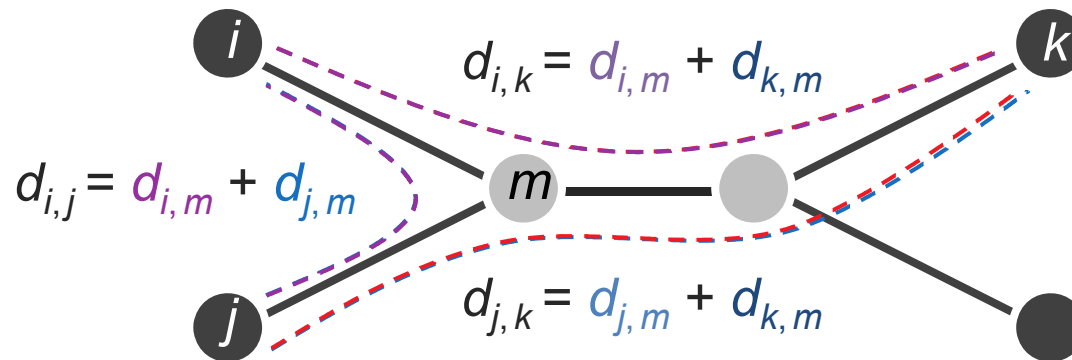
$$d_{k,m} = (d_{i,k} + d_{j,k} - d_{i,j}) / 2$$

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

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

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

From Neighbours 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 $(D_{chimp,k} + D_{chimp,j} - D_{chimp,k})/2$ 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

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

1

Computing Limb Lengths

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

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

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

Computing Limb Lengths

Limb Length Theorem: $LimbLength(chimp)$ is equal to the minimum value of $(D_{chimp,k} + D_{chimp,j} - D_{chimp,k})/2$ 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

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

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

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

Computing Limb Lengths

Limb Length Theorem: $LimbLength(chimp)$ is equal to the minimum value of $(D_{chimp,k} + D_{chimp,j} - D_{chimp,k})/2$ 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

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

1

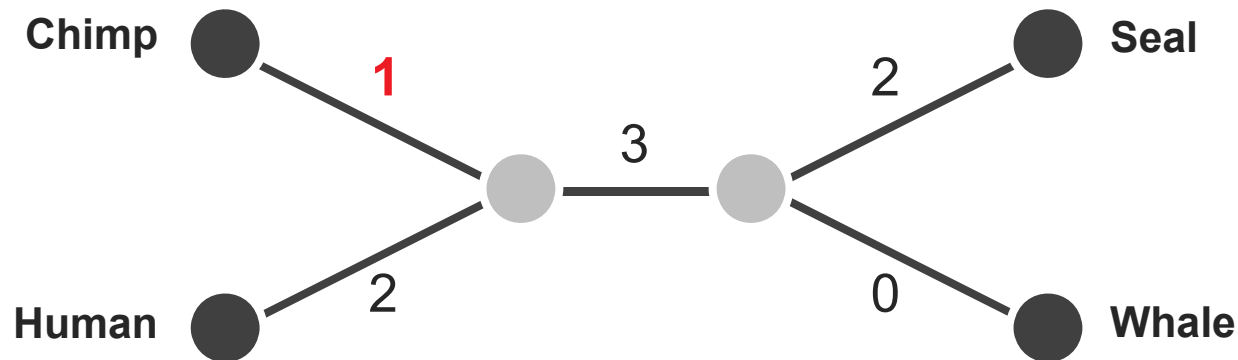
$$(D_{human, chimp} + D_{chimp, whale} - D_{human, whale}) / 2 = (3 + 4 - 5) / 2 = \mathbf{1}$$

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

Computing Limb Lengths

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

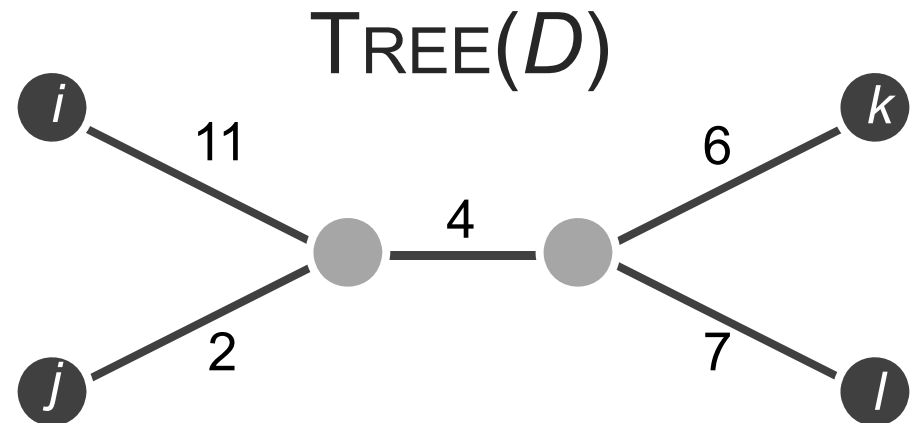
	Chimp	Human	Seal	Whale
Chimp	0	3	6	4
Human	3	0	7	5
Seal	6	7	0	2
Whale	4	5	2	0



Additive Phylogeny In Action

D

	<i>i</i>	<i>j</i>	<i>k</i>	<i>l</i>
<i>i</i>	0	13	21	22
<i>j</i>	13	0	12	13
<i>k</i>	21	12	0	13
<i>l</i>	22	13	13	0



Additive Phylogeny In Action

D

	<i>i</i>	<i>j</i>	<i>k</i>	<i>l</i>
<i>i</i>	0	13	21	22
<i>j</i>	13	0	12	13
<i>k</i>	21	12	0	13
<i>l</i>	22	13	13	0

1. Pick an arbitrary leaf *j*.

Additive Phylogeny In Action

D

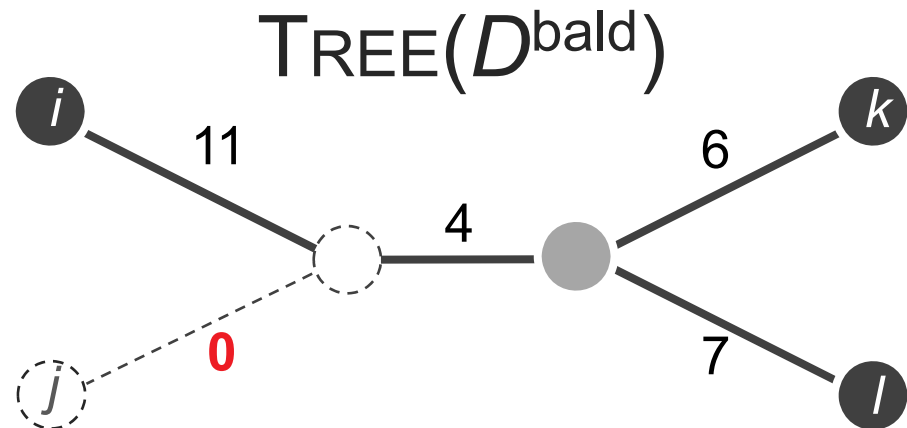
	<i>i</i>	<i>j</i>	<i>k</i>	<i>l</i>
<i>i</i>	0	13	21	22
<i>j</i>	13	0	12	13
<i>k</i>	21	12	0	13
<i>l</i>	22	13	13	0

$$\text{LimbLength}(j) = 2$$

2. Compute its limb length, $\text{LimbLength}(j)$.

Additive Phylogeny In Action

	<i>i</i>	<i>j</i>	<i>k</i>	<i>l</i>
<i>i</i>	0	11	21	22
<i>j</i>	11	0	10	11
<i>k</i>	21	10	0	13
<i>l</i>	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).

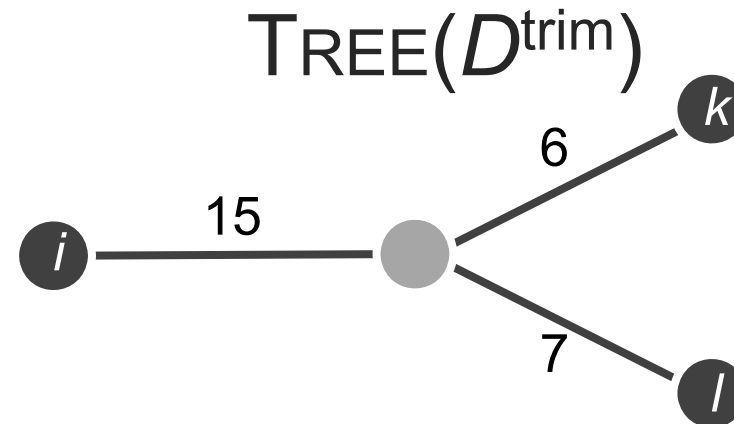
Additive Phylogeny In Action

		<i>i</i>	<i>j</i>	<i>k</i>	<i>l</i>
	<i>i</i>	0	11	21	22
D^{trim}	<i>j</i>	11	0	10	11
	<i>k</i>	21	10	0	13
	<i>l</i>	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} .

Additive Phylogeny In Action

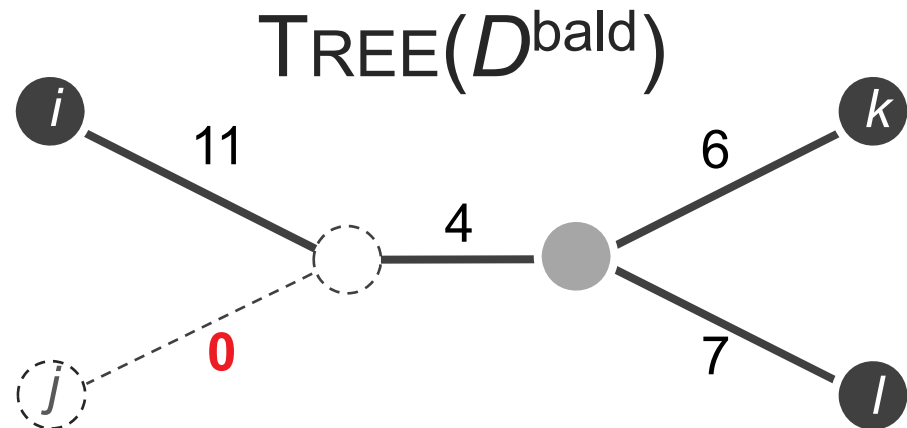
	<i>i</i>	<i>j</i>	<i>k</i>	<i>l</i>
<i>i</i>	0	11	21	22
<i>j</i>	11	0	10	11
<i>k</i>	21	10	0	13
<i>l</i>	22	11	13	0



5. Construct $Tree(D^{\text{trim}})$.

Additive Phylogeny In Action

	<i>i</i>	<i>j</i>	<i>k</i>	<i>l</i>
<i>i</i>	0	11	21	22
<i>j</i>	11	0	10	11
<i>k</i>	21	10	0	13
<i>l</i>	22	11	13	0



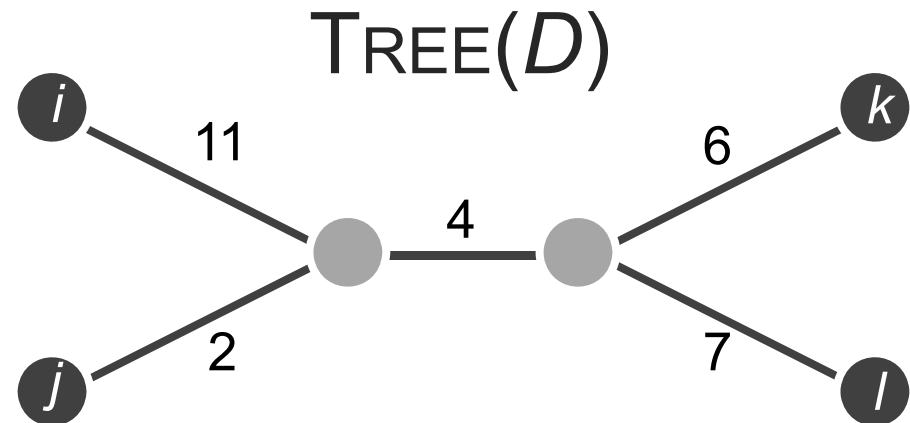
6. Identify the point in $\text{Tree}(D^{\text{trim}})$ where leaf *j* should be attached.

Additive Phylogeny 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

$$\text{LimbLength}(j) = 2$$



7. Attach j by an edge of length $\text{LimbLength}(j)$ in order to form $\text{Tree}(D)$.

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 $Tree(D)$.

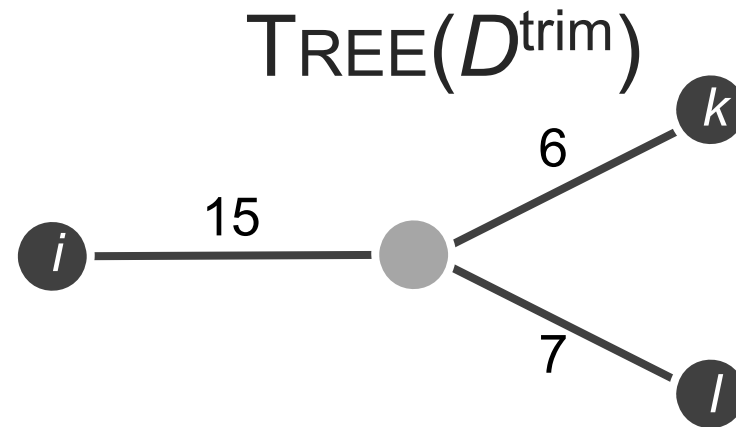
AdditivePhylogeny

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

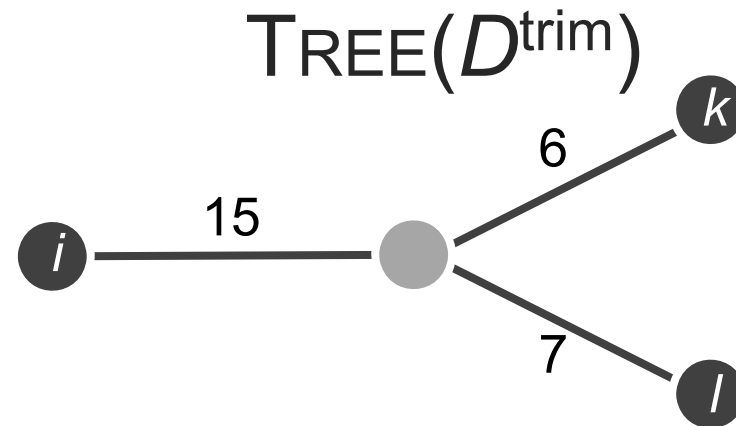
	<i>i</i>	<i>j</i>	<i>k</i>	<i>l</i>
<i>i</i>	0	11	21	22
<i>j</i>	11	0	10	11
<i>k</i>	21	10	0	13
<i>l</i>	22	11	13	0



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

Attaching a Limb

	<i>i</i>	<i>j</i>	<i>k</i>	<i>l</i>
<i>i</i>	0	11	21	22
<i>j</i>	11	0	10	11
<i>k</i>	21	10	0	13
<i>l</i>	22	11	13	0

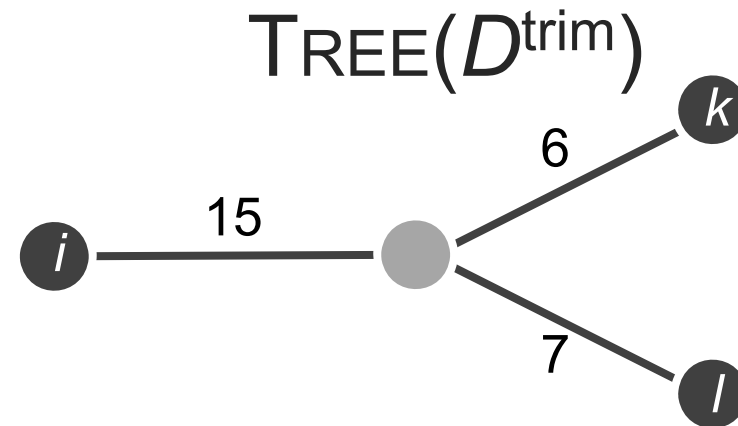


Limb Length Theorem: the length of the limb of *j* is equal to the minimum value of $(D^{\text{bald}}_{i,j} + D^{\text{bald}}_{j,k} - D^{\text{bald}}_{i,k})/2$ over all leaves *i* and *k*.

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

Attaching a Limb

	<i>i</i>	<i>j</i>	<i>k</i>	<i>l</i>
<i>i</i>	0	11	21	22
<i>j</i>	11	0	10	11
<i>k</i>	21	10	0	13
<i>l</i>	22	11	13	0

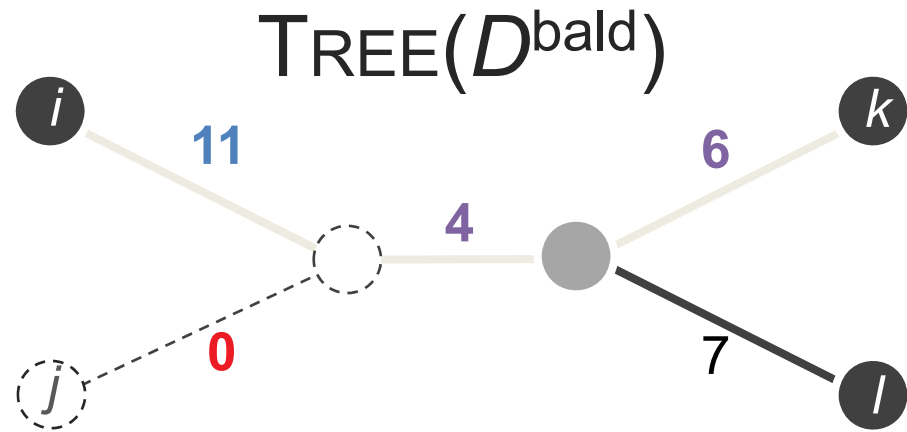


$$(D^{\text{bald}}_{i,j} + D^{\text{bald}}_{j,k} - D^{\text{bald}}_{i,k})/2 = 0$$

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

Attaching a Limb

	i	j	k	l	
D^{bald}	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 $D^{\text{bald}}_{i,j}$ from i .

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

AdditivePhylogeny

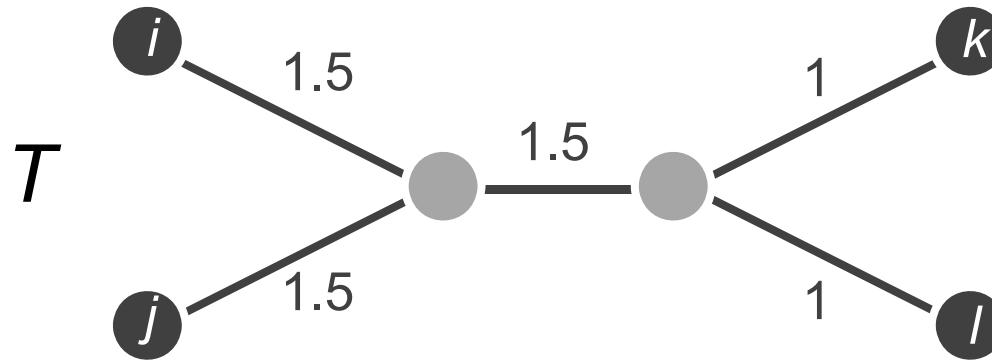
AdditivePhylogeny(D):

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

$$\begin{aligned} \text{Discrepancy}(T, D) &= \sum_{1 \leq i < j \leq n} (d_{i,j}(T) - D_{i,j})^2 \\ &= 1^2 + 1^2 = 2 \end{aligned}$$



D

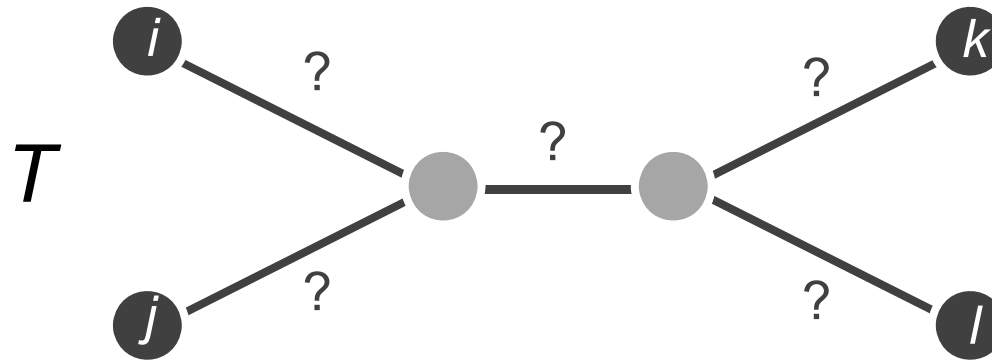
	<i>i</i>	<i>j</i>	<i>k</i>	<i>l</i>
<i>i</i>	0	3	4	3
<i>j</i>	3	0	4	5
<i>k</i>	4	4	0	2
<i>l</i>	3	5	2	0

d

	<i>i</i>	<i>j</i>	<i>k</i>	<i>l</i>
<i>i</i>	0	3	4	4
<i>j</i>	3	0	4	4
<i>k</i>	4	4	0	2
<i>l</i>	4	4	2	0

Sum of Squared Errors

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



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 $n \times n$ distance matrix D .
- **Output:** A weighted tree T with n leaves minimizing $Discrepancy(T, D)$ over all weighted trees with n leaves.

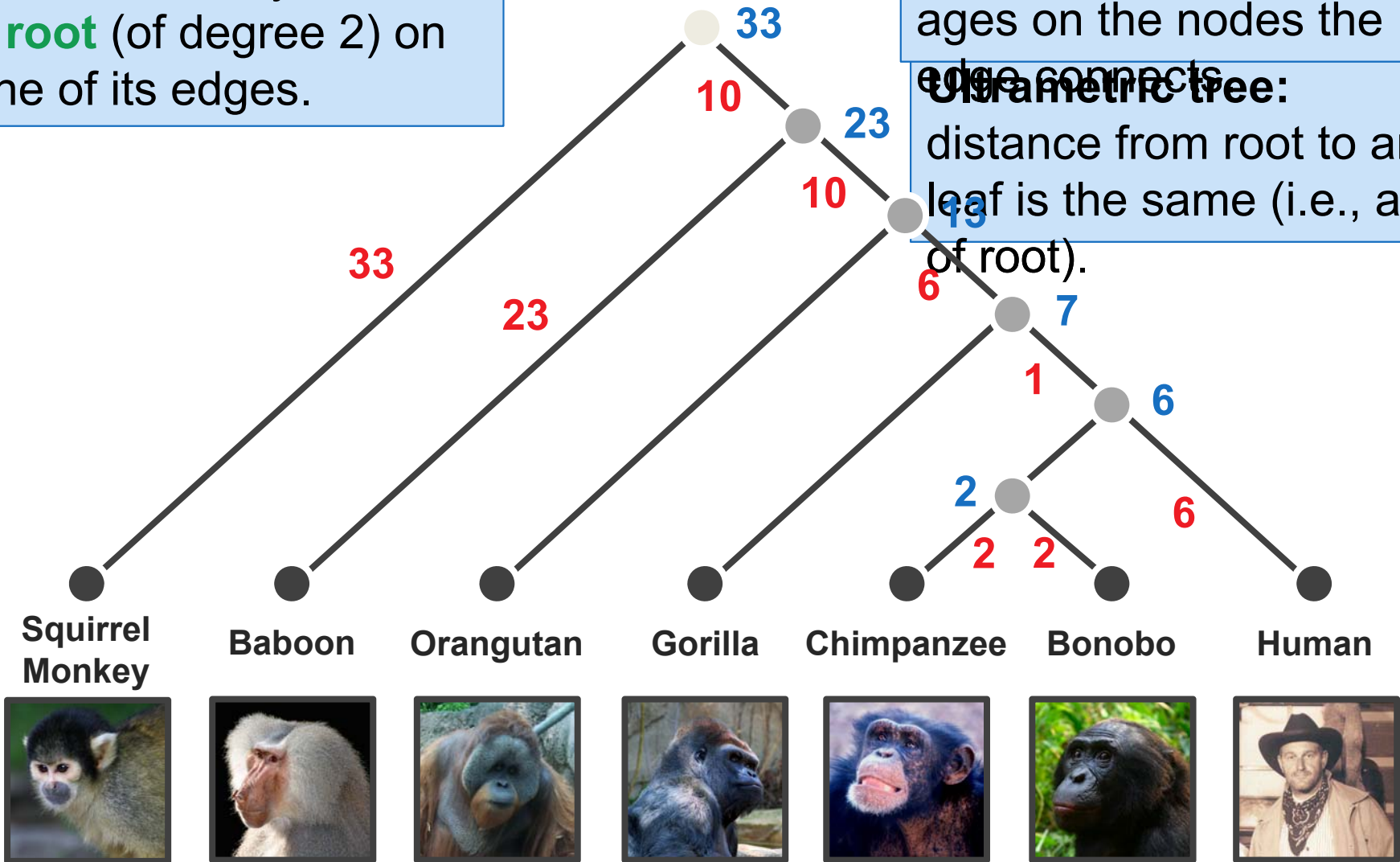
Unfortunately, this problem is *NP-Complete*...

Ultrametric Trees

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

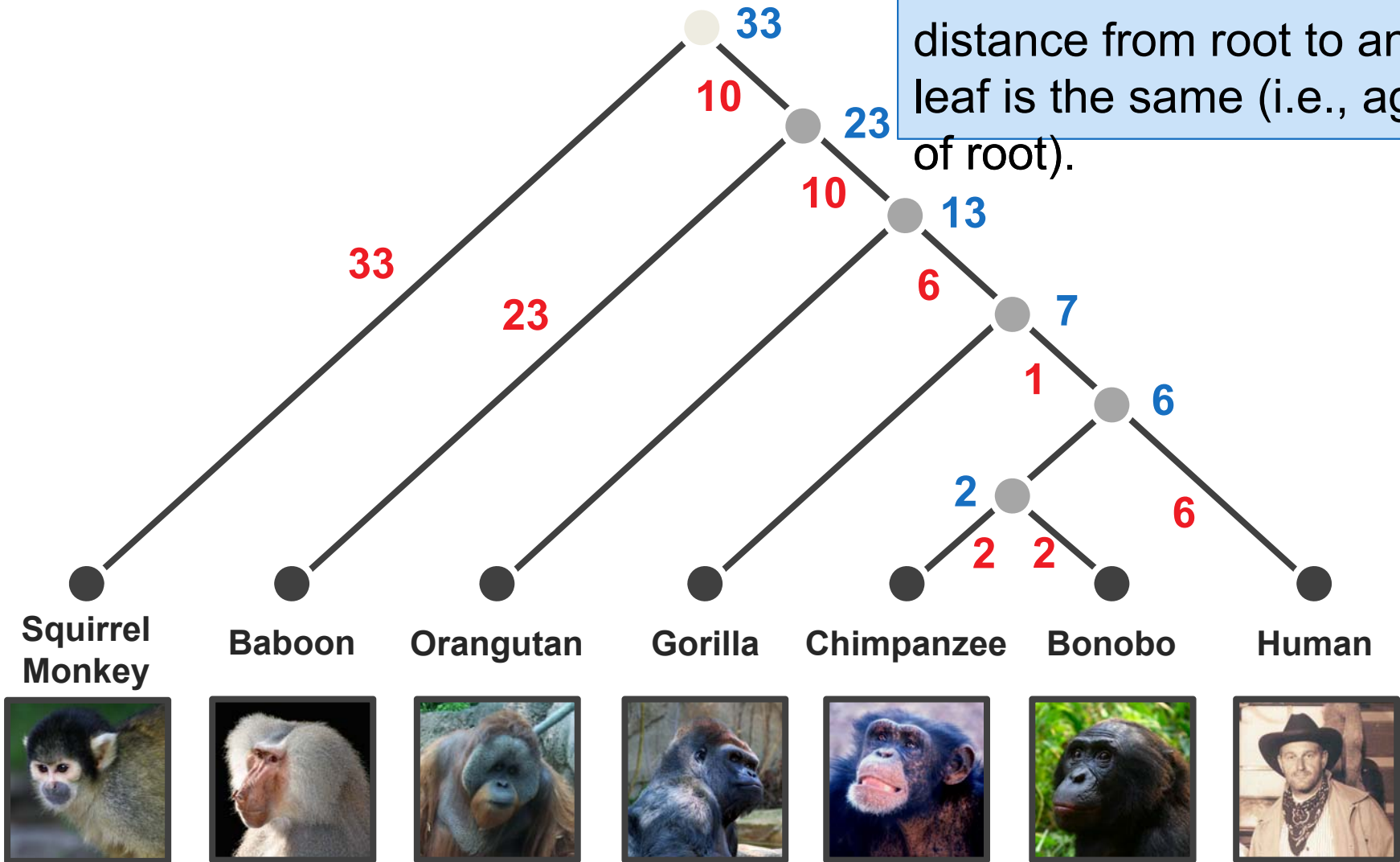
edge weights: correspond to difference in ages on the nodes the edge connects.

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>i</i>	<i>j</i>	<i>k</i>	<i>l</i>
<i>i</i>	0	3	4	3
<i>j</i>	3	0	4	5
<i>k</i>	4	4	0	2
<i>l</i>	3	5	2	0



UPGMA: A Clustering Heuristic

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

$$D_{\text{avg}}(C_1, C_2) = \sum_{i \in C_1, j \in C_2} D_{i,j} / |C_1| \cdot |C_2|$$

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

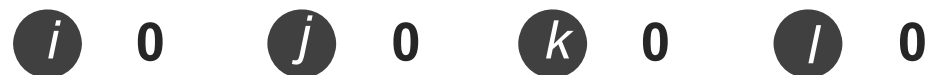
	<i>i</i>	<i>j</i>	<i>k</i>	<i>l</i>
<i>i</i>	0	3	4	3
<i>j</i>	3	0	4	5
<i>k</i>	4	4	0	2
<i>l</i>	3	5	2	0



UPGMA: A Clustering Heuristic

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

	<i>i</i>	<i>j</i>	<i>k</i>	<i>l</i>
<i>i</i>	0	3	4	3
<i>j</i>	3	0	4	5
<i>k</i>	4	4	0	2
<i>l</i>	3	5	2	0

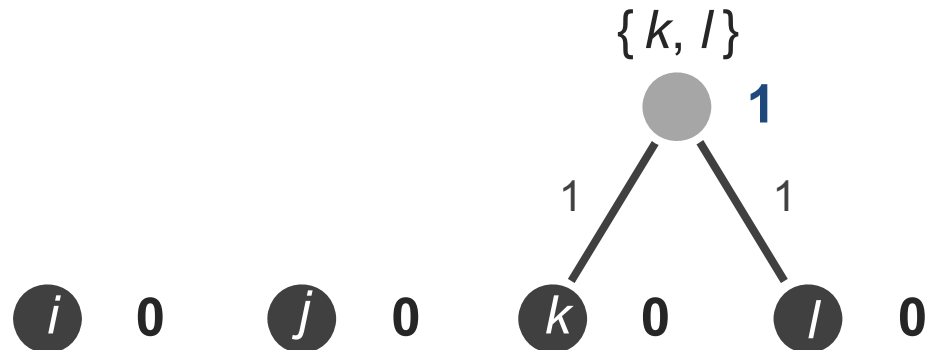


$\{k, l\}$

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 $D_{\text{avg}}(C_1, C_2)/2$.

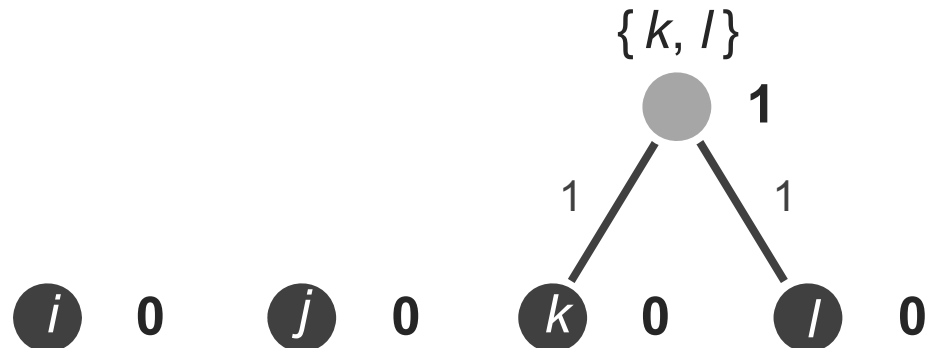
	<i>i</i>	<i>j</i>	<i>k</i>	<i>l</i>
<i>i</i>	0	3	4	3
<i>j</i>	3	0	4	5
<i>k</i>	4	4	0	2
<i>l</i>	3	5	2	0



UPGMA: A Clustering Heuristic

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

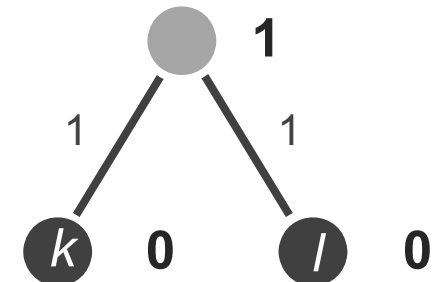
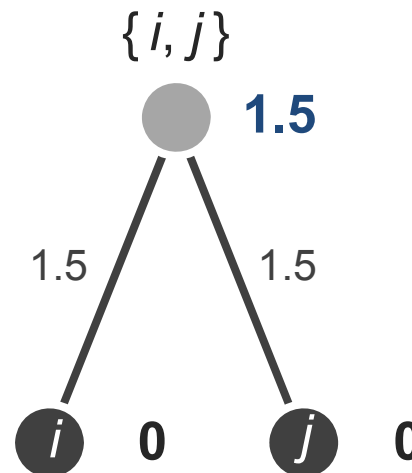
	<i>i</i>	<i>j</i>	{ <i>k</i> , <i>l</i> }
<i>i</i>	0	3	3.5
<i>j</i>	3	0	4.5
{ <i>k</i> , <i>l</i> }	3.5	4.5	0



UPGMA: A Clustering Heuristic

6. Iterate until a single cluster contains all species.

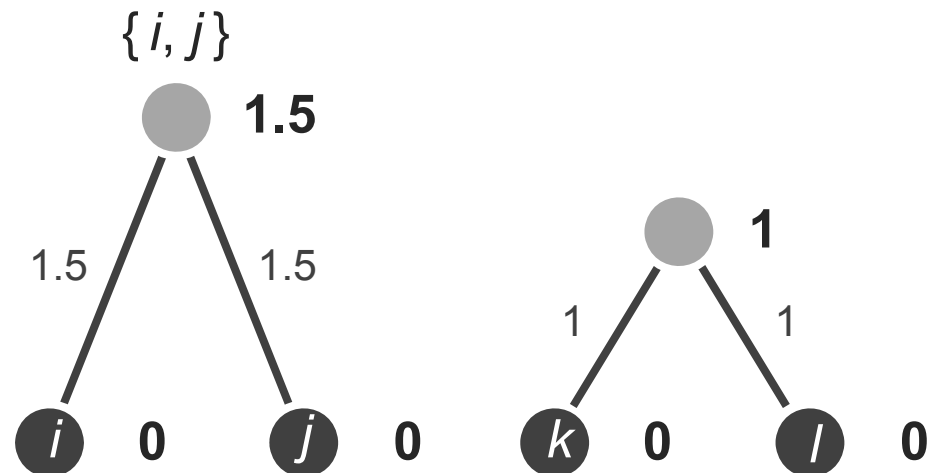
	<i>i</i>	<i>j</i>	{ <i>k</i> , <i>l</i> }
<i>i</i>	0	3	3.5
<i>j</i>	3	0	4.5
{ <i>k</i> , <i>l</i> }	3.5	4.5	0



UPGMA: A Clustering Heuristic

6. Iterate until a single cluster contains all species.

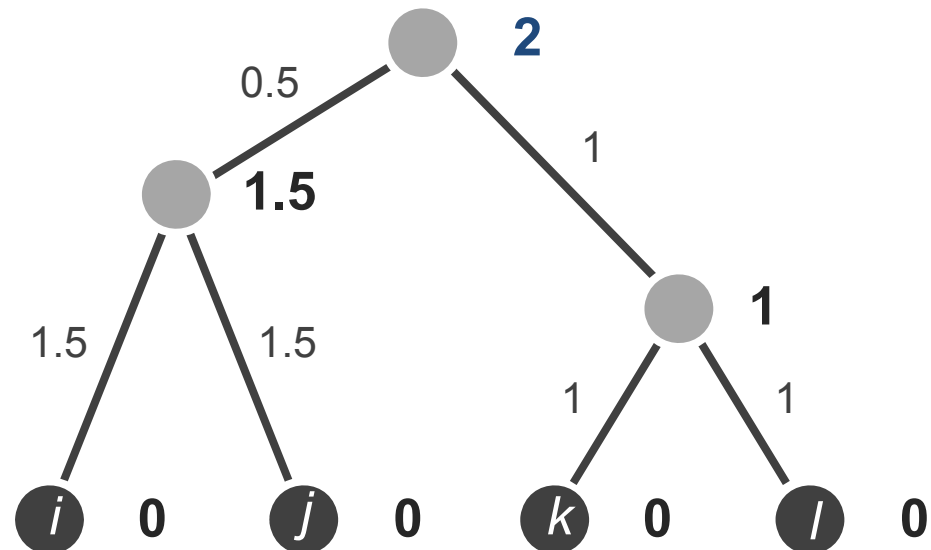
	$\{i, j\}$	$\{k, l\}$
$\{i, j\}$	0	4
$\{k, l\}$	4	0



UPGMA: A Clustering Heuristic

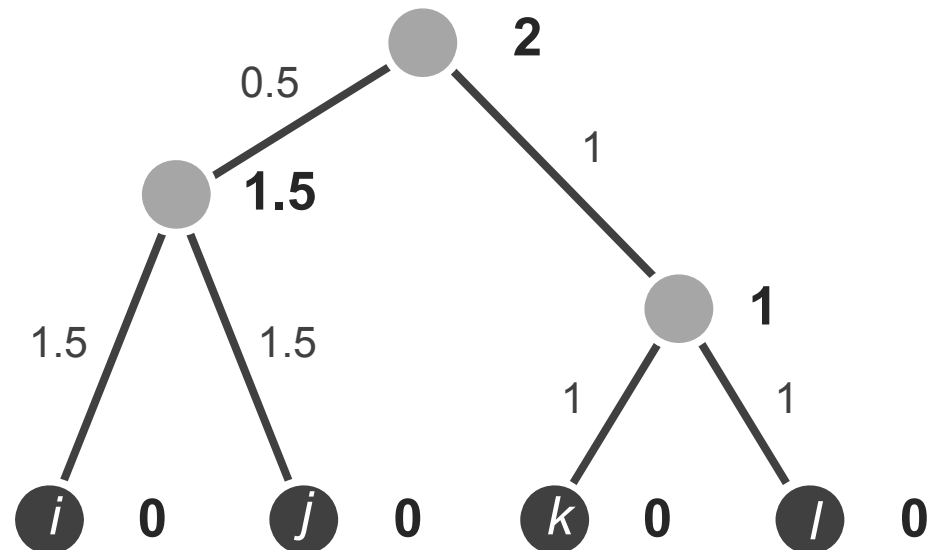
6. Iterate until a single cluster contains all species.

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

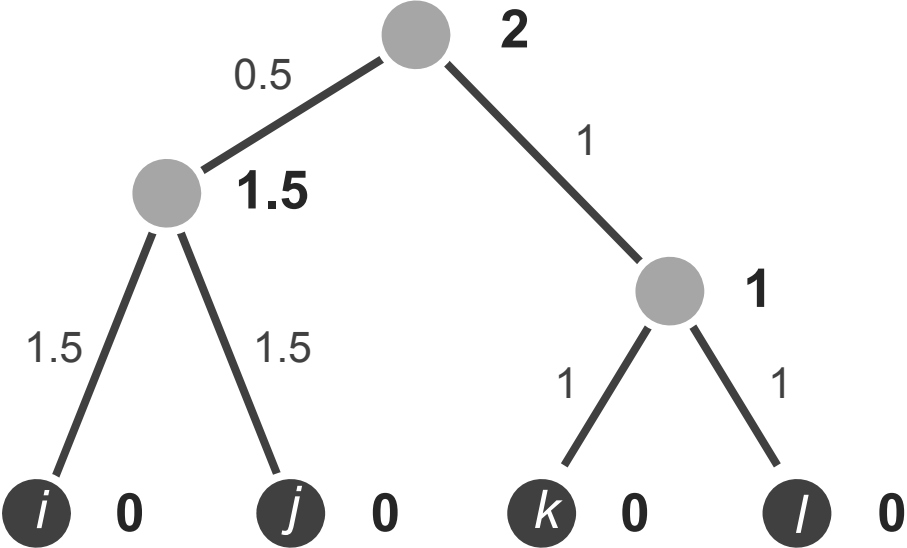
$$D_{\text{avg}}(C_1, C_2) = \sum_{i \in C_1, j \in C_2} D_{i,j} / |C_1| \cdot |C_2|$$

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 $D_{\text{avg}}(C_1, C_2)/2$.
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.

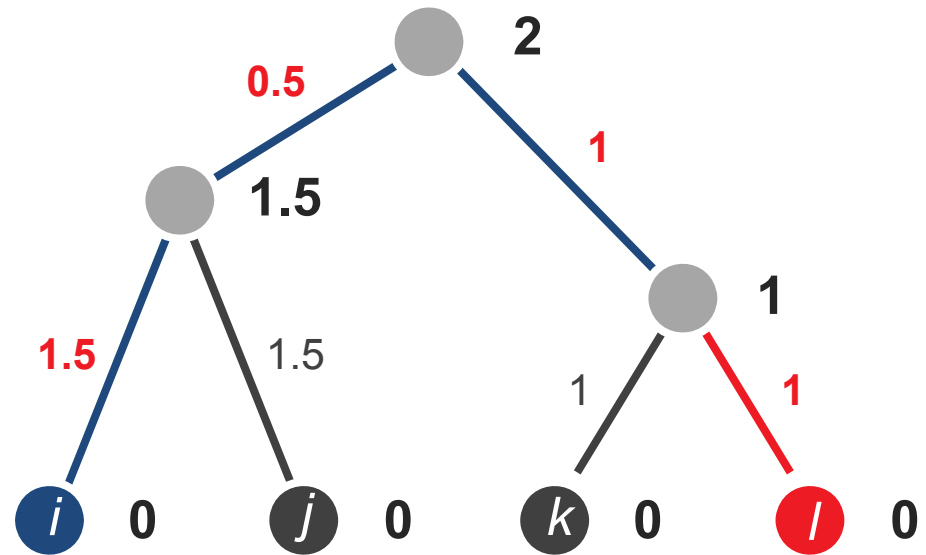
UPGMA Doesn't "Fit" a Tree to a Matrix

	<i>i</i>	<i>j</i>	<i>k</i>	<i>l</i>
<i>i</i>	0	3	4	3
<i>j</i>	3	0	4	5
<i>k</i>	4	4	0	2
<i>l</i>	3	5	2	0



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

	<i>i</i>	<i>j</i>	<i>k</i>	<i>l</i>
<i>i</i>	0	3	4	3
<i>j</i>	3	0	4	5
<i>k</i>	4	4	0	2
<i>l</i>	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

Neighbour-Joining Theorem

Given an $n \times n$ distance matrix D , its **neighbour-joining matrix** is the matrix D^* defined as

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

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

	i	j	k	l	$TotalDistance_D$		i	j	k	l		
D	i	0	13	21	22	56	D^*	i	0	-68	-60	-60
	j	13	0	12	13	38		j	-	0	-60	-60
	k	21	12	0	13	46		k	68	-	0	-68
	l	22	13	13	0	48		l	-	-60	-68	0

Neighbour-Joining in Action

	<i>i</i>	<i>j</i>	<i>k</i>	<i>l</i>	<i>TotalDistance_D</i>	
<i>D</i> *	<i>i</i>	0	-68	-60	-60	56
	<i>j</i>	-	0	-60	-60	38
		68				46
	<i>k</i>	-	-60	0	-68	48
		60				
	<i>l</i>	-	-60	-68	0	
		60				

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

Neighbour-Joining in Action

	<i>i</i>	<i>j</i>	<i>k</i>	<i>l</i>	<i>TotalDistance_D</i>
<i>i</i>	0	-68	-60	-60	56
<i>D*</i> <i>j</i>	68	0	-60	-60	38
<i>k</i>	-	-60	0	-68	46
	60				48
<i>l</i>	-	-60	-68	0	
	60				

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

Neighbour-Joining in Action

	<i>i</i>	<i>j</i>	<i>k</i>	<i>l</i>	<i>TotalDistance_D</i>	
<i>i</i>	0	-68	-60	-60	56	
<i>D*</i>	<i>j</i>	-	0	-60	-60	38
		68				46
<i>k</i>	-	-60	0	-68	48	
	60					
<i>l</i>	-	-60	-68	0		
	60					

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

Neighbour-Joining in Action

		<i>i</i>	<i>j</i>	<i>k</i>	<i>l</i>	<i>TotalDistance_D</i>	
<i>D</i> *	<i>i</i>	0	-68	-60	-60	56	$\Delta_{i,j} = (56 - 38) / (4 - 2)$ $= 9$
	<i>j</i>	-	0	-60	-60	38	
	<i>k</i>	68	-60	0	-68	46	
		60				48	
	<i>l</i>	-	-60	-68	0		
		60					

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

Neighbour-Joining in Action

	<i>i</i>	<i>j</i>	<i>k</i>	<i>l</i>	<i>TotalDistance_D</i>		
<i>D</i>	<i>i</i>	0	13	21	22	56	$\Delta_{i,j} = (56 - 38) / (4 - 2)$ $= 9$
	<i>j</i>	13	0	12	13	38	
	<i>k</i>	21	12	0	13	46	
	<i>l</i>	22	13	13	0	48	

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

$$LimbLength(j) = \frac{1}{2}(13 - 9) = 2$$

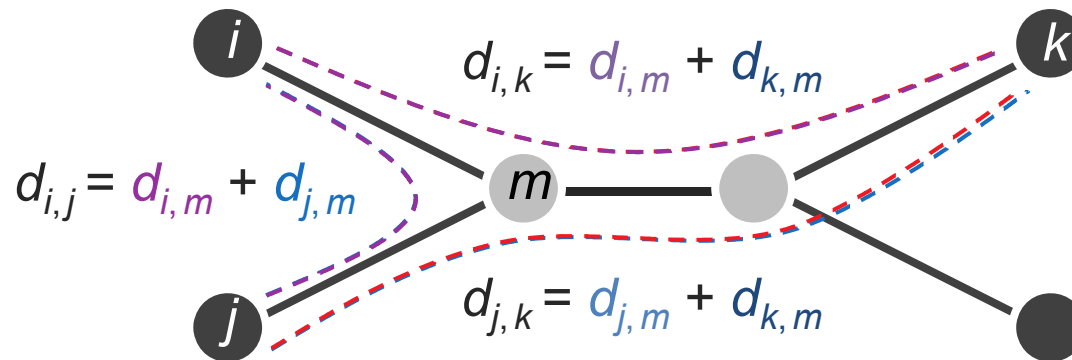
4. Set $LimbLength(i)$ equal to $\frac{1}{2}(D_{i,j} + \Delta_{i,j})$ and $LimbLength(j)$ equal to $\frac{1}{2}(D_{i,j} - \Delta_{i,j})$.

Neighbour-Joining in Action

	<i>m</i>	<i>k</i>	<i>l</i>	<i>TotalDistance_D</i>	
<i>D'</i>	<i>m</i>	0	10	11	21
	<i>k</i>	10	0	13	23
	<i>l</i>	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 $d_{k,m}$

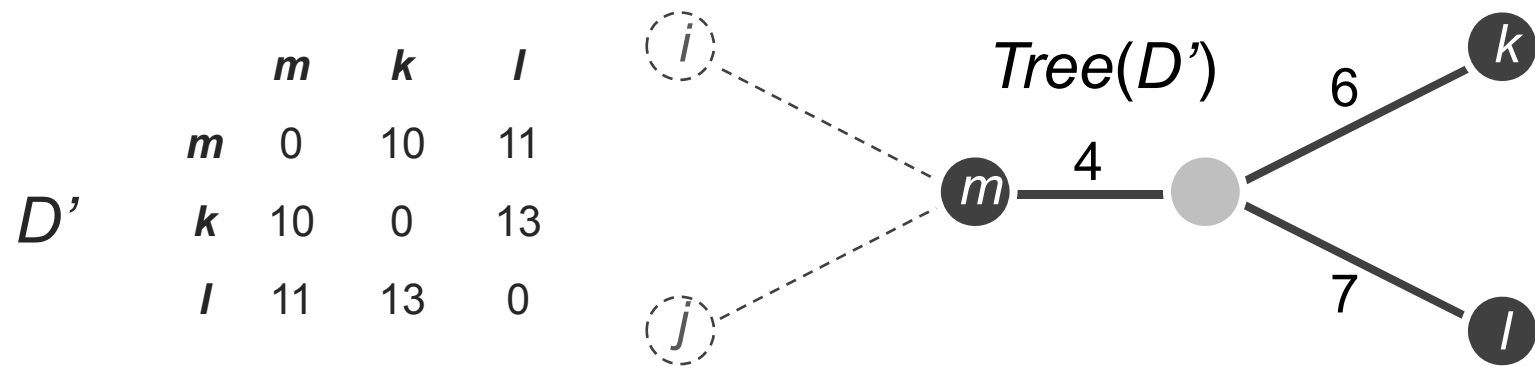


$$d_{k,m} = [(d_{i,m} + d_{k,m}) + (d_{j,m} + d_{k,m}) - (d_{i,m} + d_{j,m})] / 2$$

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

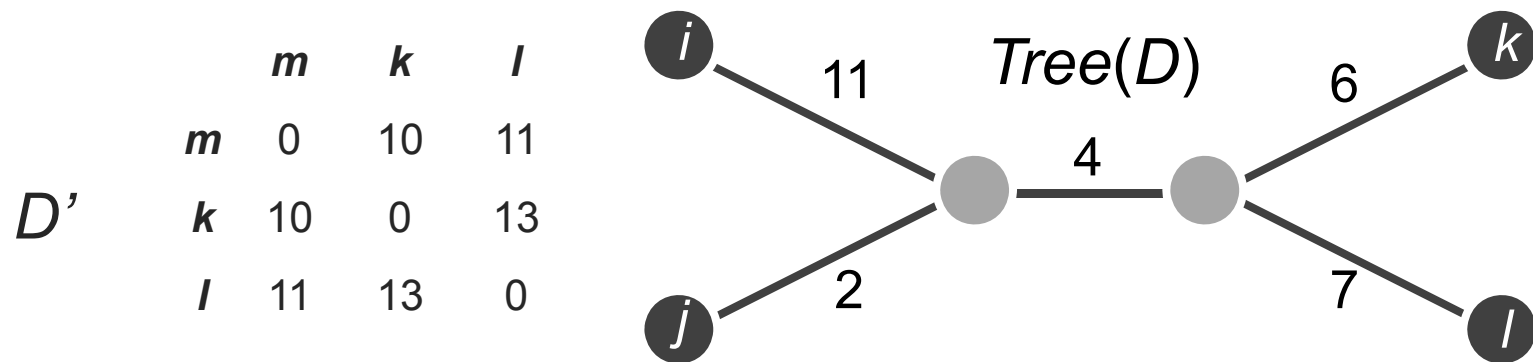
$$d_{k,m} = (D_{i,k} + D_{j,k} - D_{i,j}) / 2$$

Neighbour-Joining in Action



6. Apply **NeighbourJoining** to D' to obtain $Tree(D')$.

Neighbour-Joining in Action

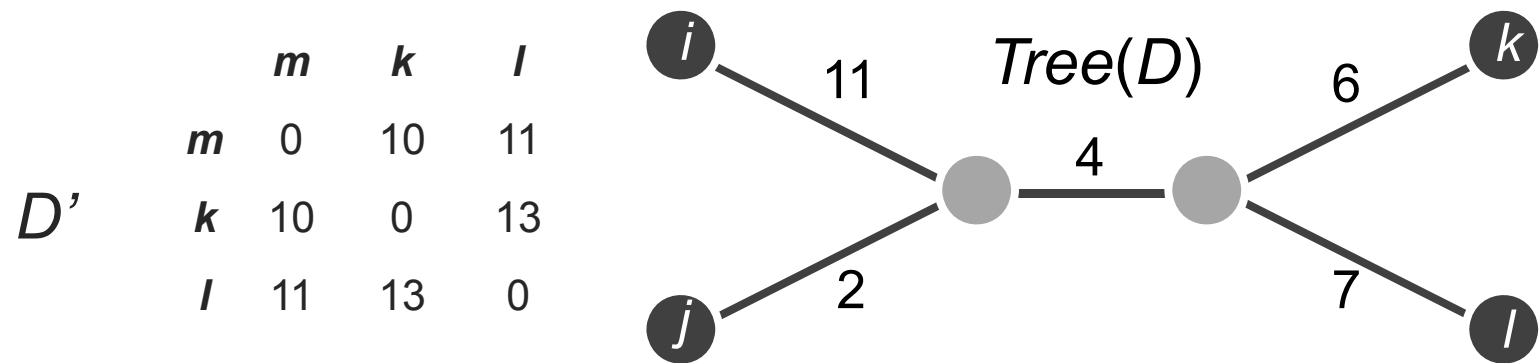


$$\text{LimbLength}(i) = \frac{1}{2}(13 + 9) = 11$$

$$\text{LimbLength}(j) = \frac{1}{2}(13 - 9) = 2$$

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

Neighbour-Joining in Action



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

Neighbour-Joining

NeighborJoining(D):

1. Construct neighbour-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 $\frac{1}{2}(D_{i,j} - \Delta_{i,j})$.
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 **NeighbourJoining** to D' to obtain $Tree(D')$.
7. Reattach limbs of i and j to obtain $Tree(D)$.

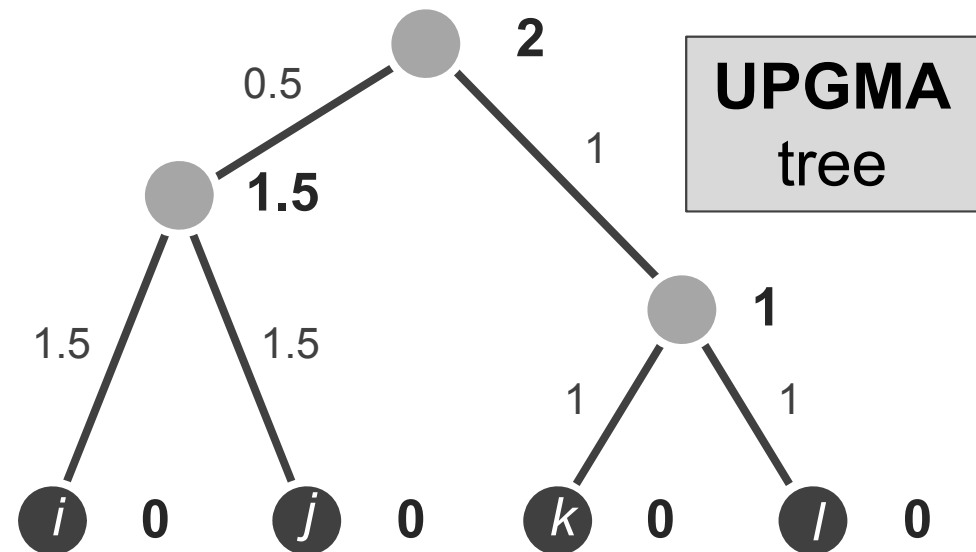
Code Challenge: Implement **NeighbourJoining**.

Neighbour-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



Current work

- What if some taxa are ancestors of others?
 - ‘Family joining’
<https://doi.org/10.1093/molbev/msw123>
- What if sequences are sampled at several points in time?
 - Serial UPGMA
<https://doi.org/10.1093/oxfordjournals.molbev.a026281>
 - Least squares dating (adjusts branch lengths of existing tree) <https://doi.org/10.1093/sysbio/syv068>

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

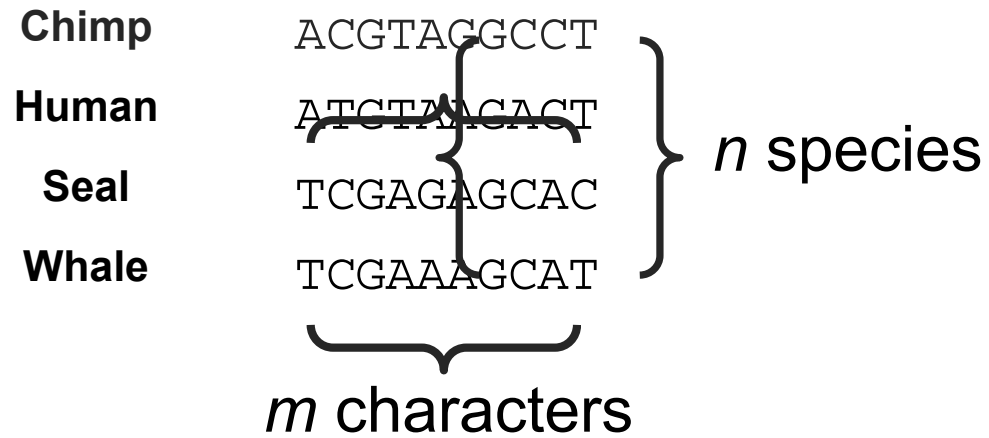
An Alignment As a Character Table

SPECIES	ALIGNMENT
Chimp	ACGTAGGCCT
Human	ATGTAAGACT
Seal	TCGAGAGCAC
Whale	TCGAAAGCAT

n species

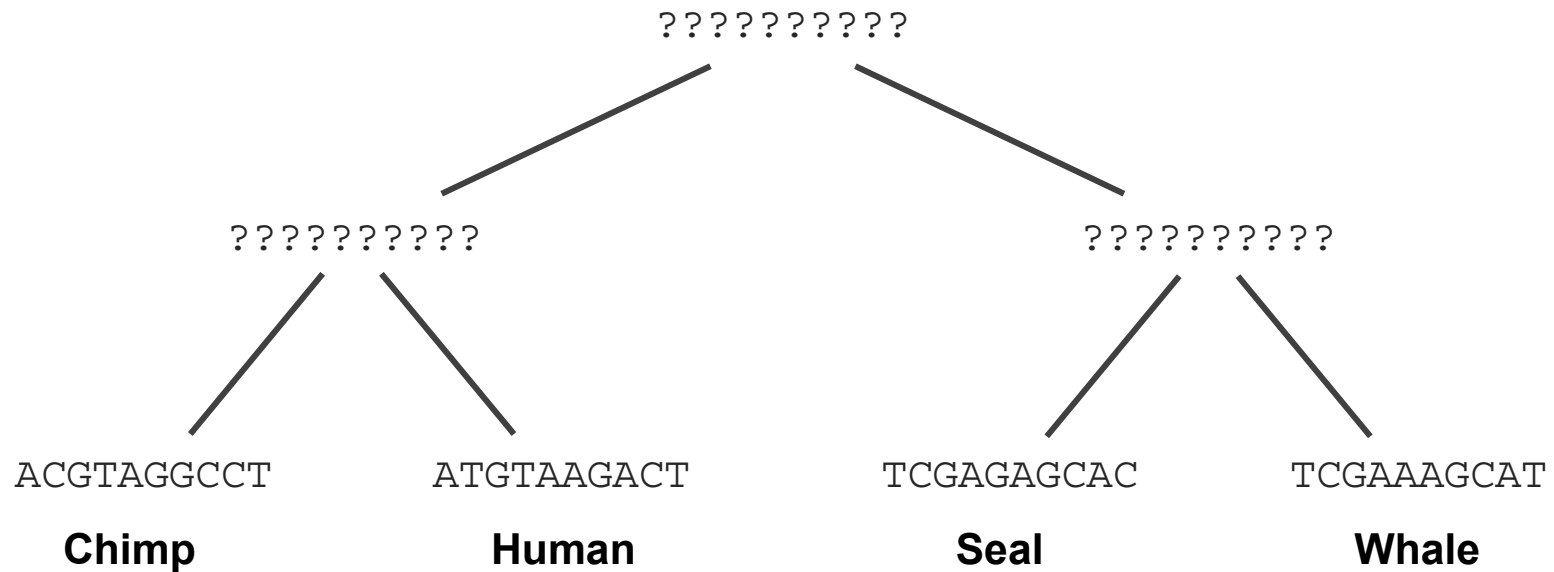
m characters

Toward a Computational Problem

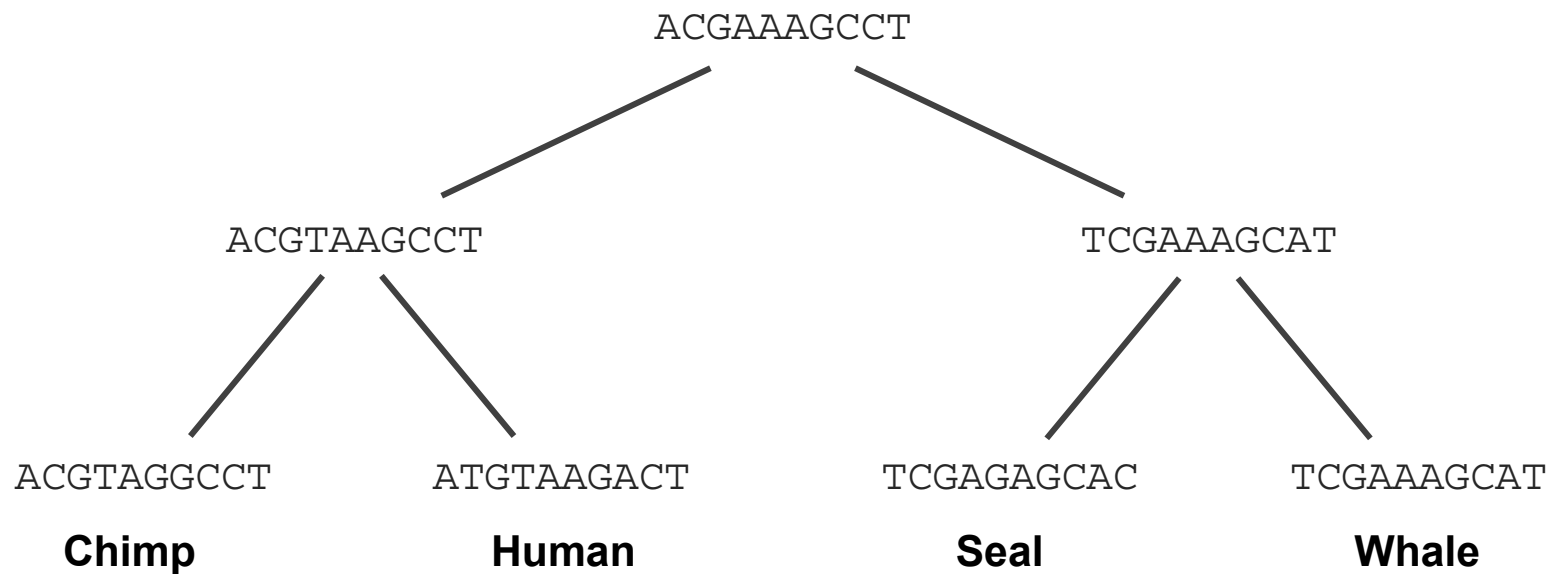


Toward a Computational Problem

Chimp	ACGTAGGCCT
Human	ATGTAAGACT
Seal	TCGAGAGCAC
Whale	TCGAAAGCAT

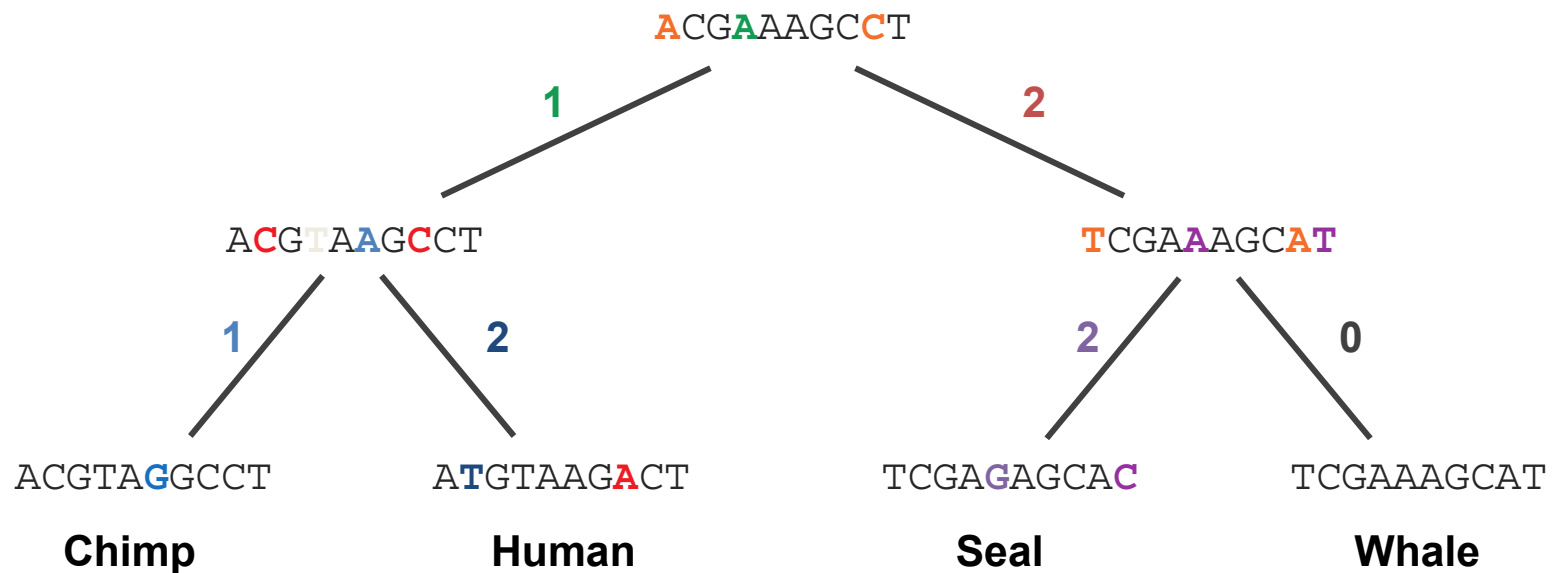


Toward a Computational Problem



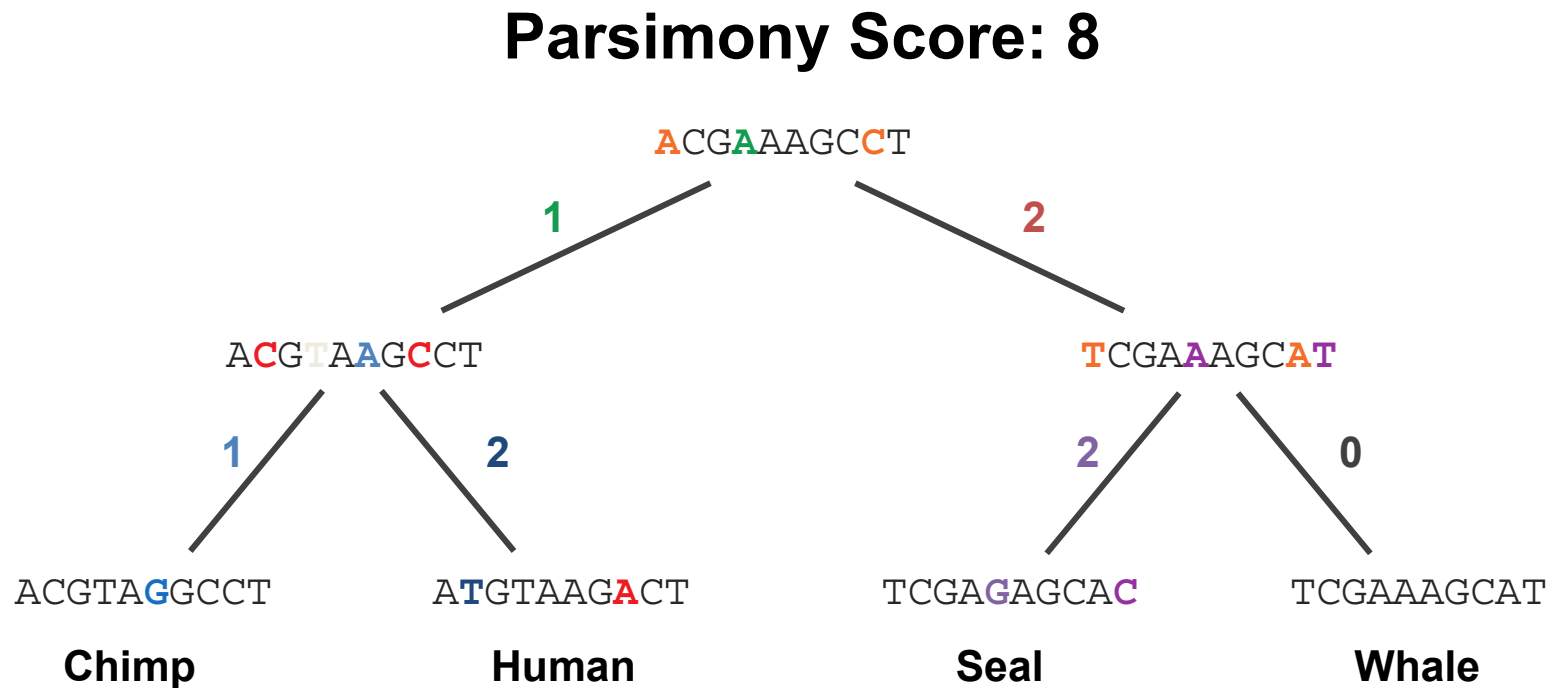
Toward a Computational Problem

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.

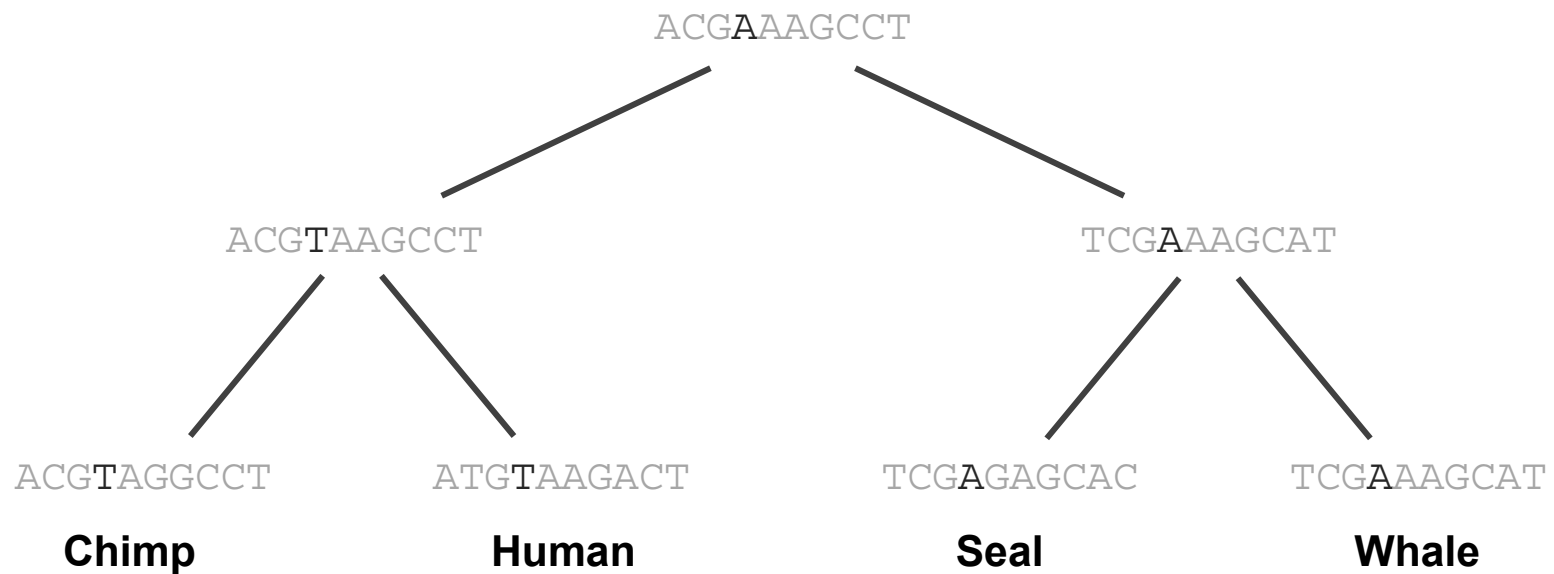
Is there any way we can simplify this problem statement?

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.

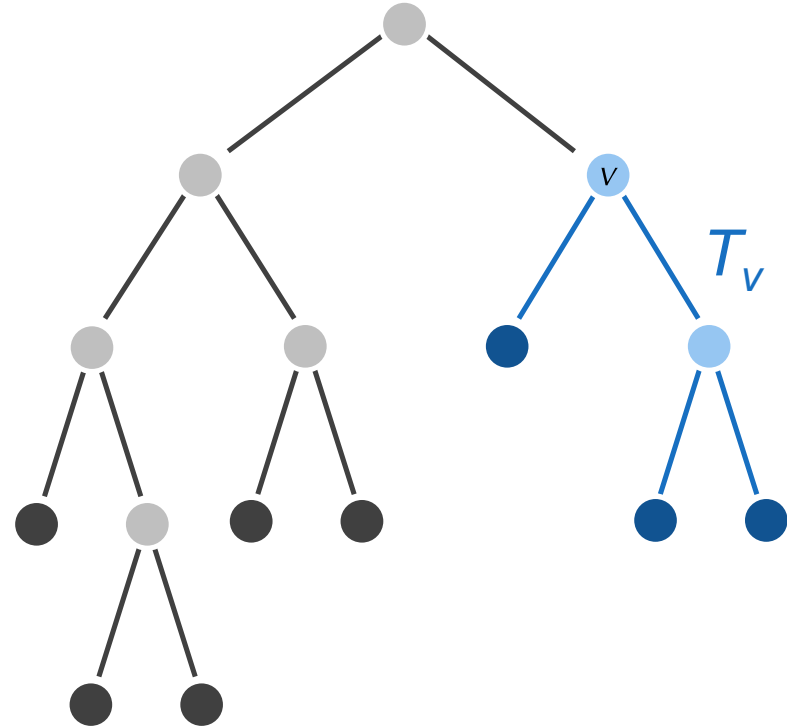
Toward a Computational Problem



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 .

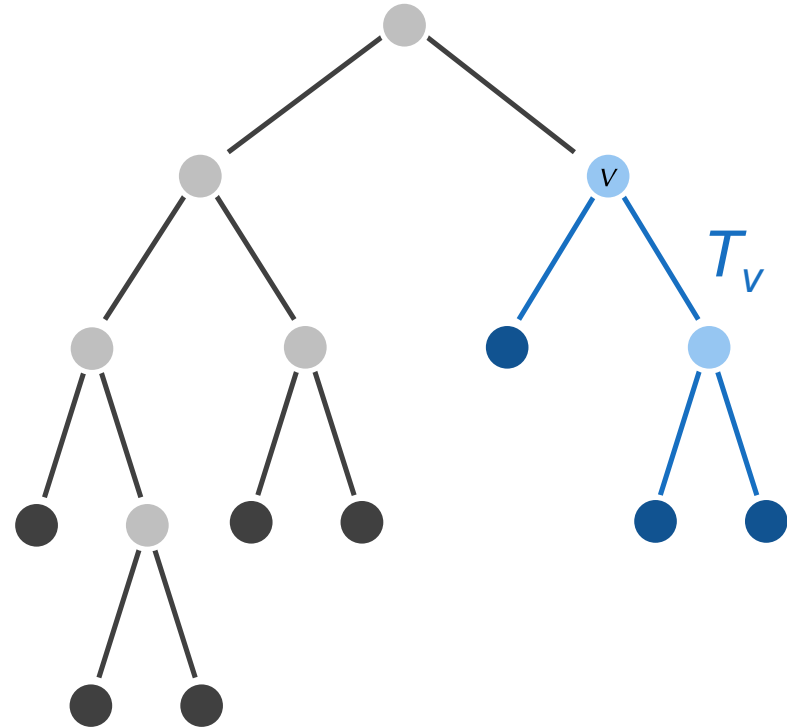


The minimum parsimony score for the tree is equal to the minimum value of $s_k(\text{root})$ over all symbols k .

A Dynamic Programming Algorithm

For symbols i and j , define

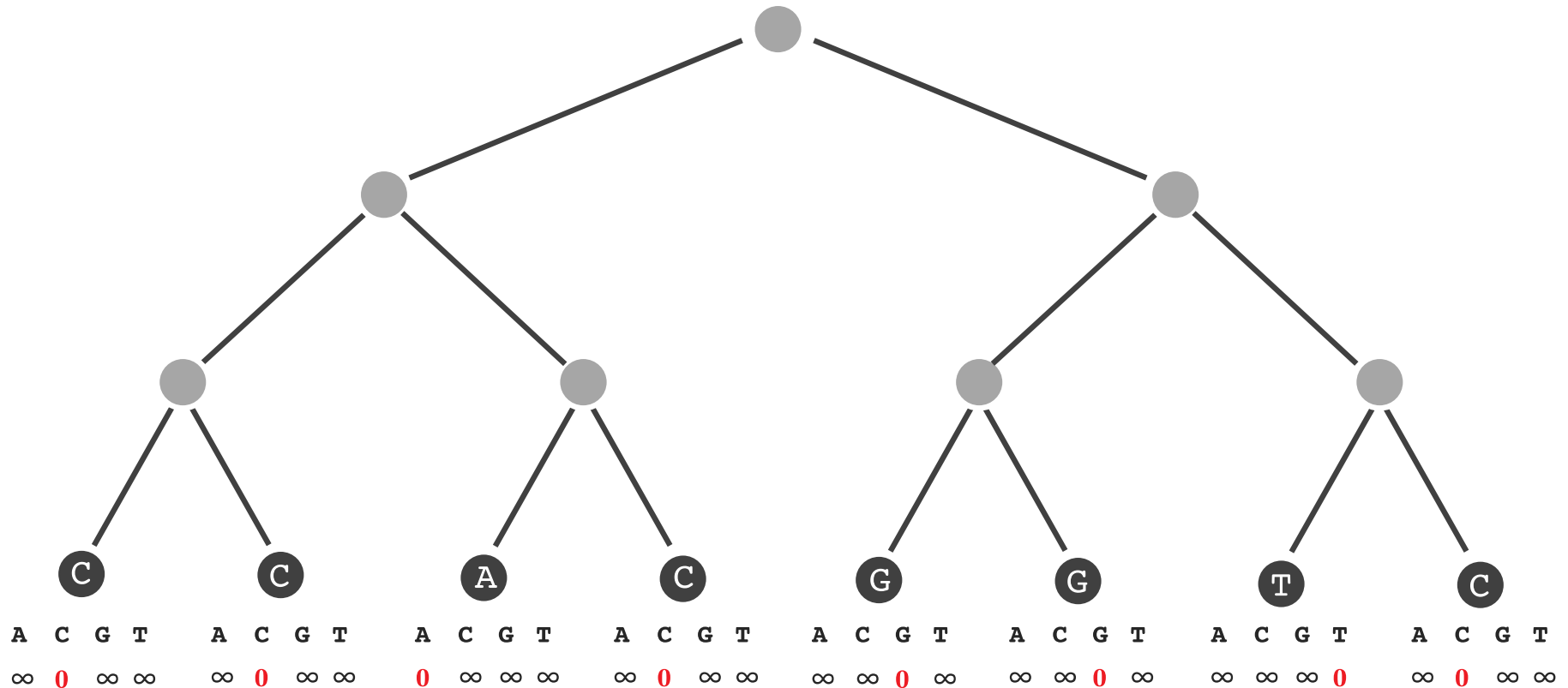
- $\delta_{i,j} = 0$ if $i = j$
- $\delta_{i,j} = 1$ otherwise.



Exercise Break: Prove the following recurrence relation:

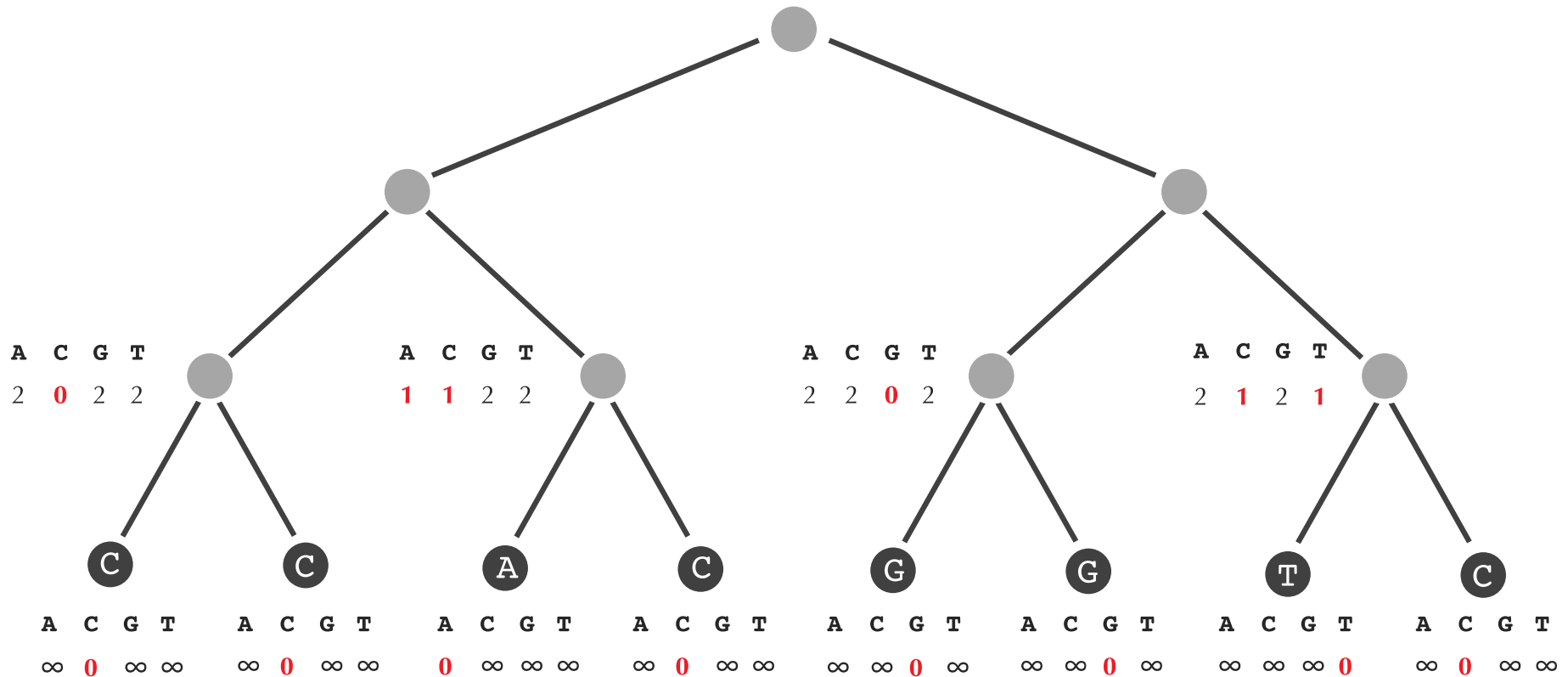
$$s_k(v) = \min_{\text{all symbols } i} \{s_i(\text{Daughter}(v)) + \delta_{i,k}\} + \min_{\text{all symbols } i} \{s_i(\text{Son}(v)) + \delta_{j,k}\}$$

A Dynamic Programming Algorithm



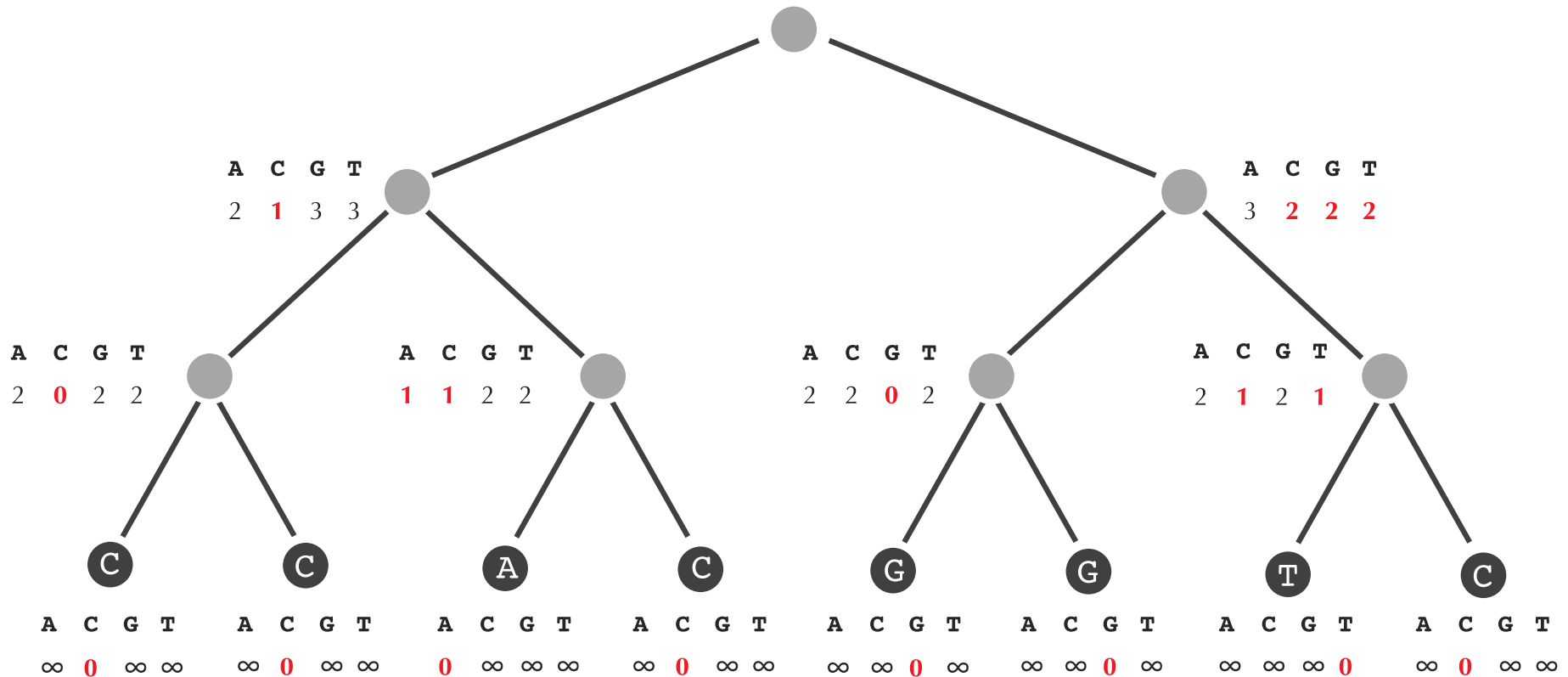
$$s_k(v) = \min_{\text{all symbols } i} \{s_i(\text{Daughter}(v)) + \delta_{i,k}\} + \min_{\text{all symbols } i} \{s_i(\text{Son}(v)) + \delta_{j,k}\}$$

A Dynamic Programming Algorithm



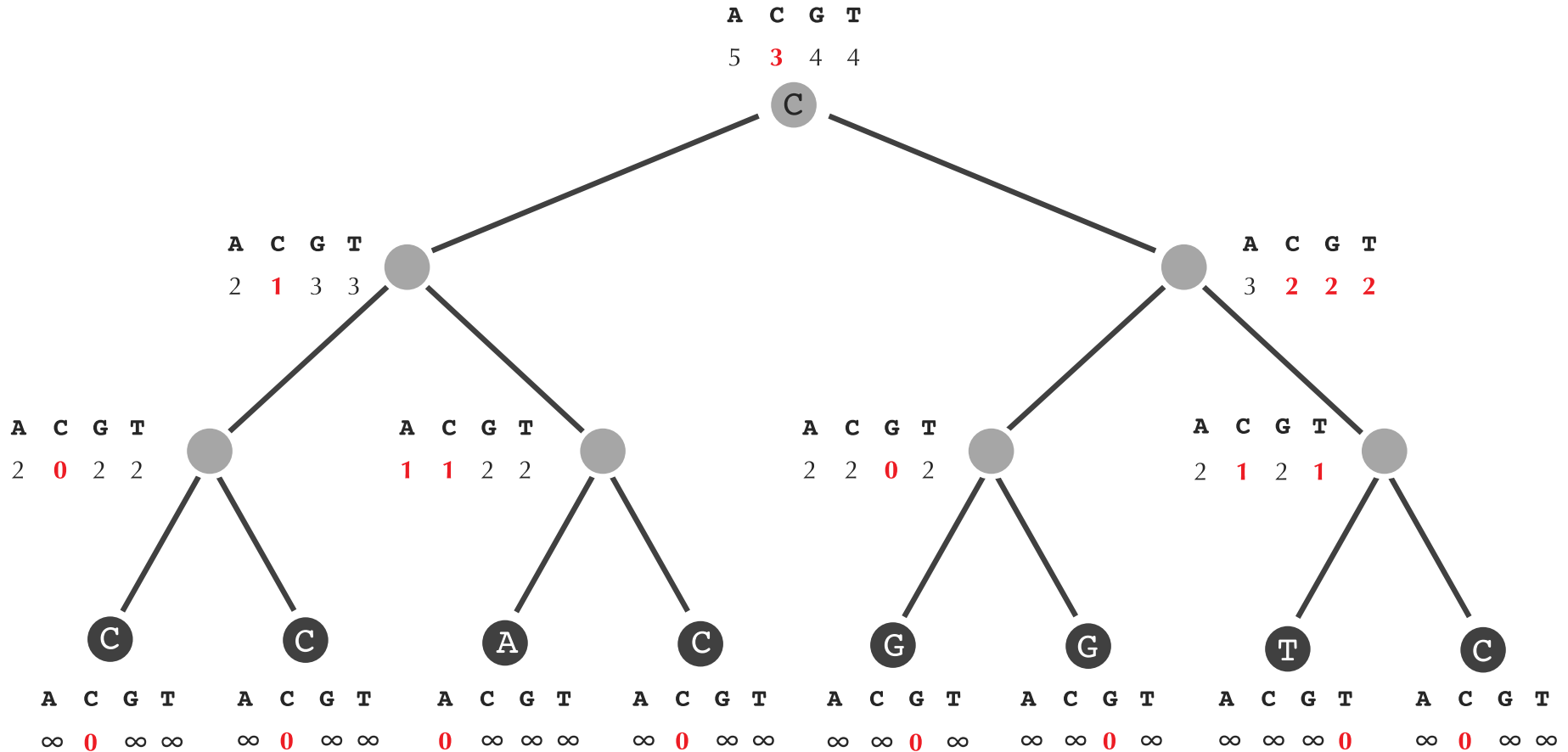
$$s_k(v) = \min_{\text{all symbols } i} \{s_i(\text{Daughter}(v)) + \delta_{i,k}\} + \min_{\text{all symbols } i} \{s_i(\text{Son}(v)) + \delta_{j,k}\}$$

A Dynamic Programming Algorithm



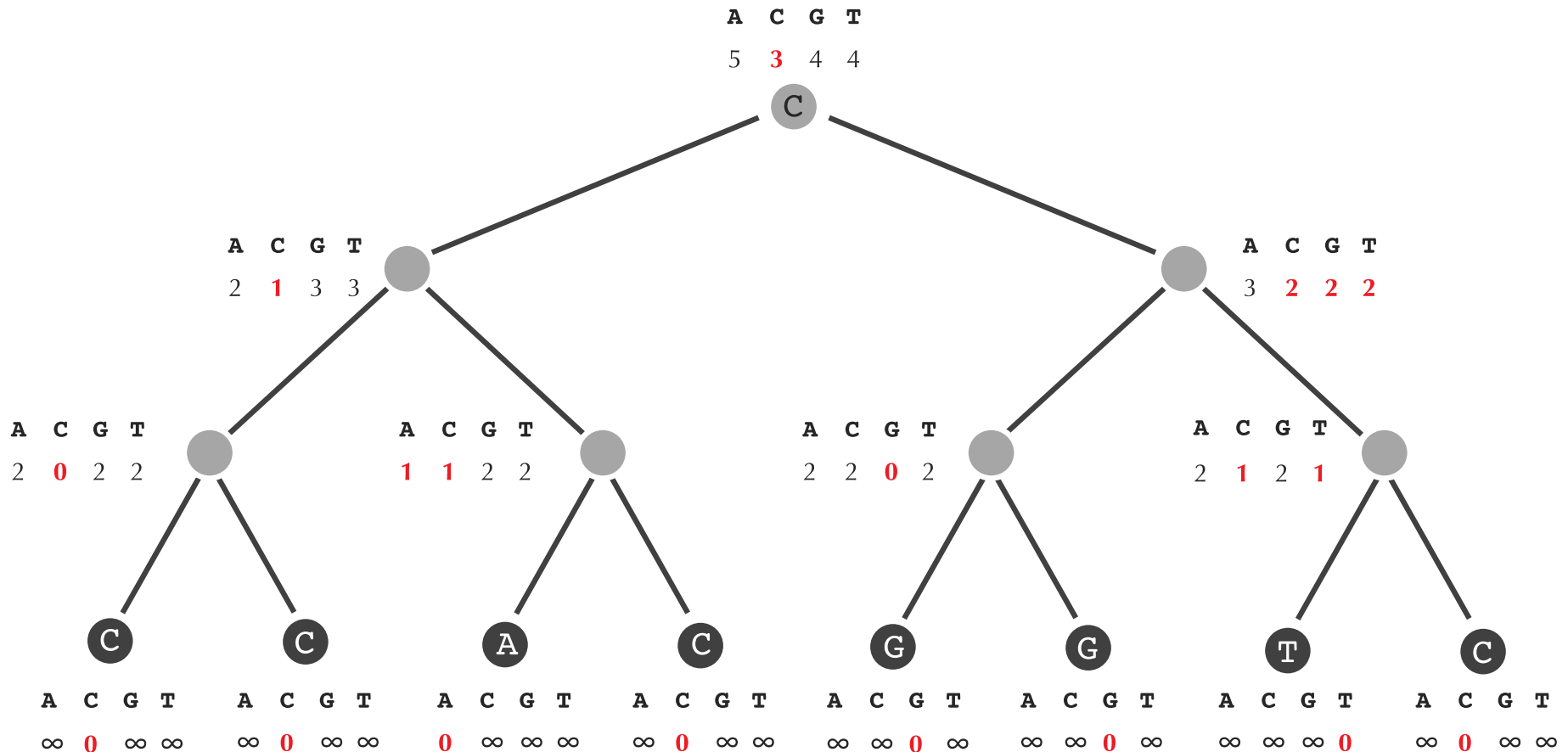
$$s_k(v) = \min_{\text{all symbols } i} \{s_i(\text{Daughter}(v)) + \delta_{i,k}\} + \min_{\text{all symbols } i} \{s_i(\text{Son}(v)) + \delta_{j,k}\}$$

A Dynamic Programming Algorithm



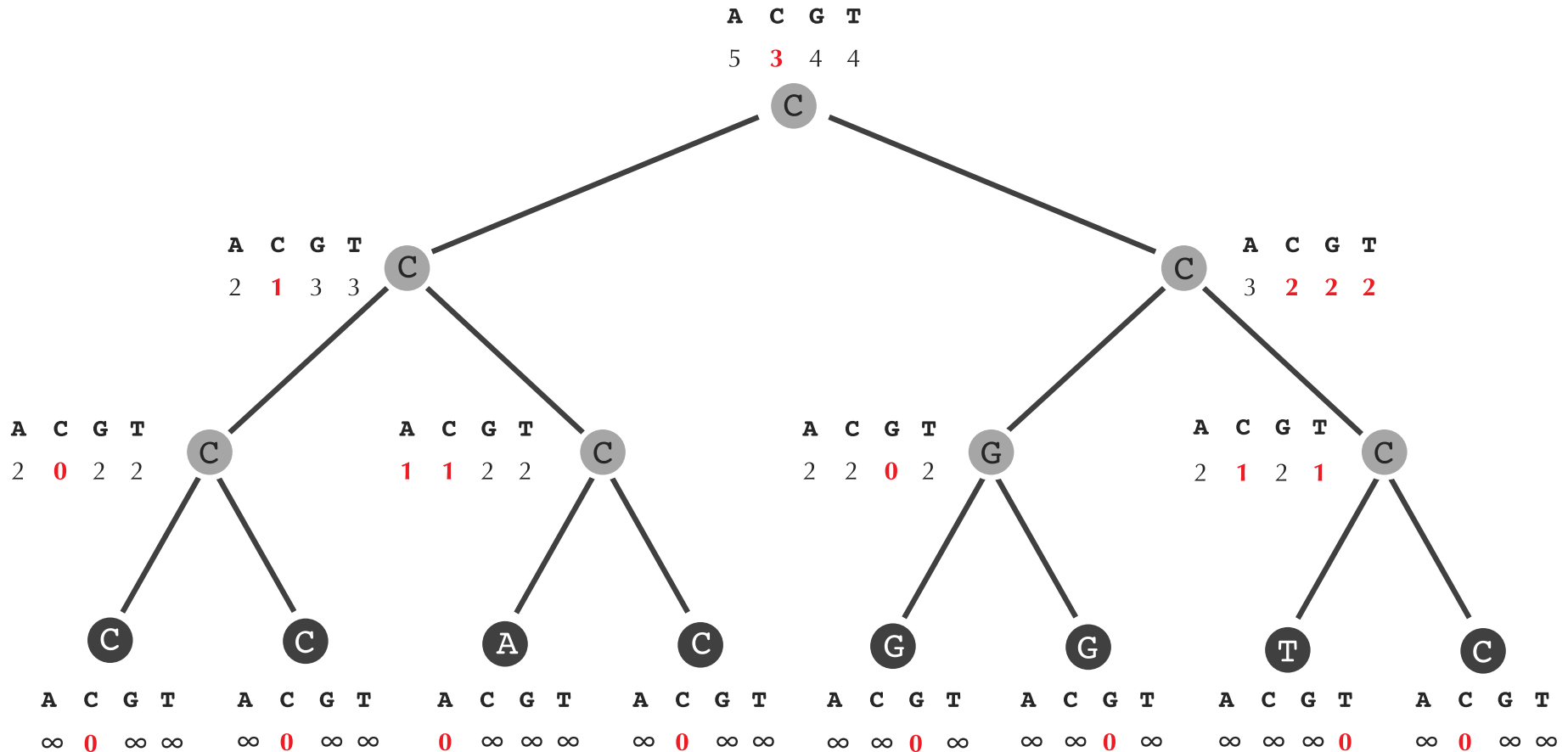
$$s_k(v) = \min_{\text{all symbols } i} \{s_i(\text{Daughter}(v)) + \delta_{i,k}\} + \min_{\text{all symbols } i} \{s_i(\text{Son}(v)) + \delta_{j,k}\}$$

A Dynamic Programming Algorithm

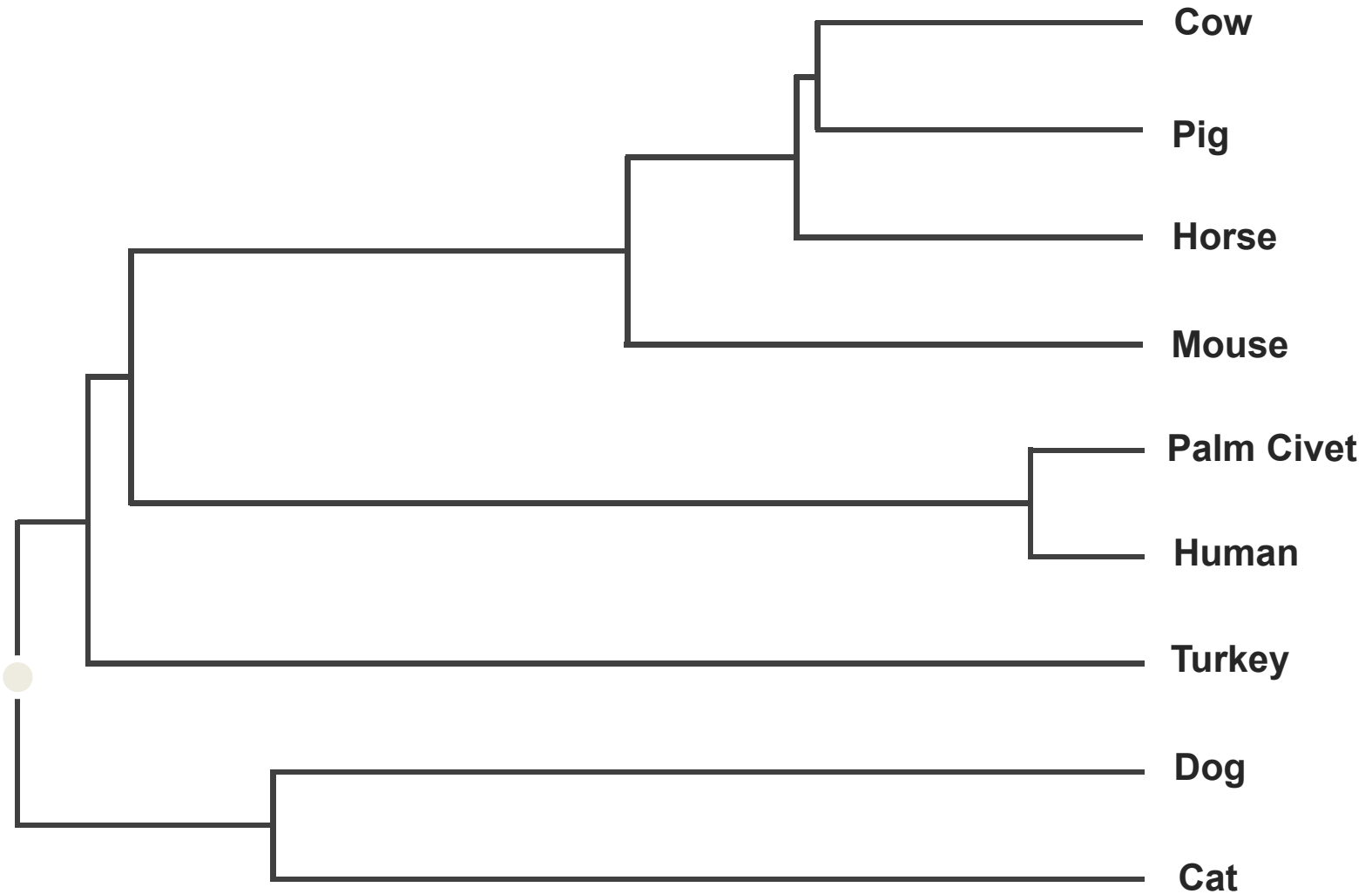


Exercise Break: “Backtrack” to fill in the remaining nodes of the tree.

A Dynamic Programming Algorithm



Code Challenge: Solve the Small Parsimony Problem.



Exercise Break: Apply **SmallParsimony** to this tree to reconstruct ancestral coronavirus sequences.

Small Parsimony for Unrooted Trees

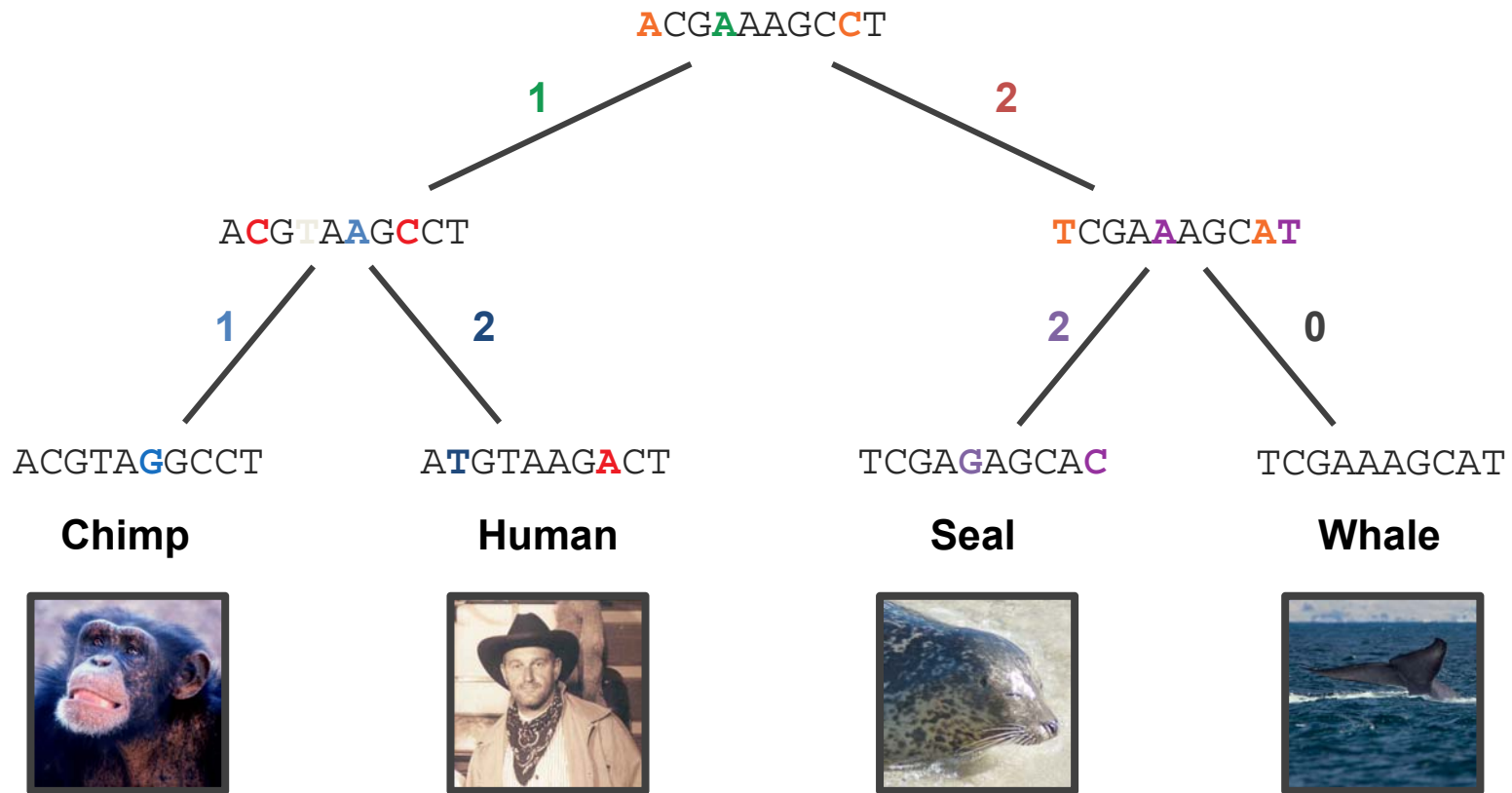
Small Parsimony in an Unrooted Tree Problem:

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.

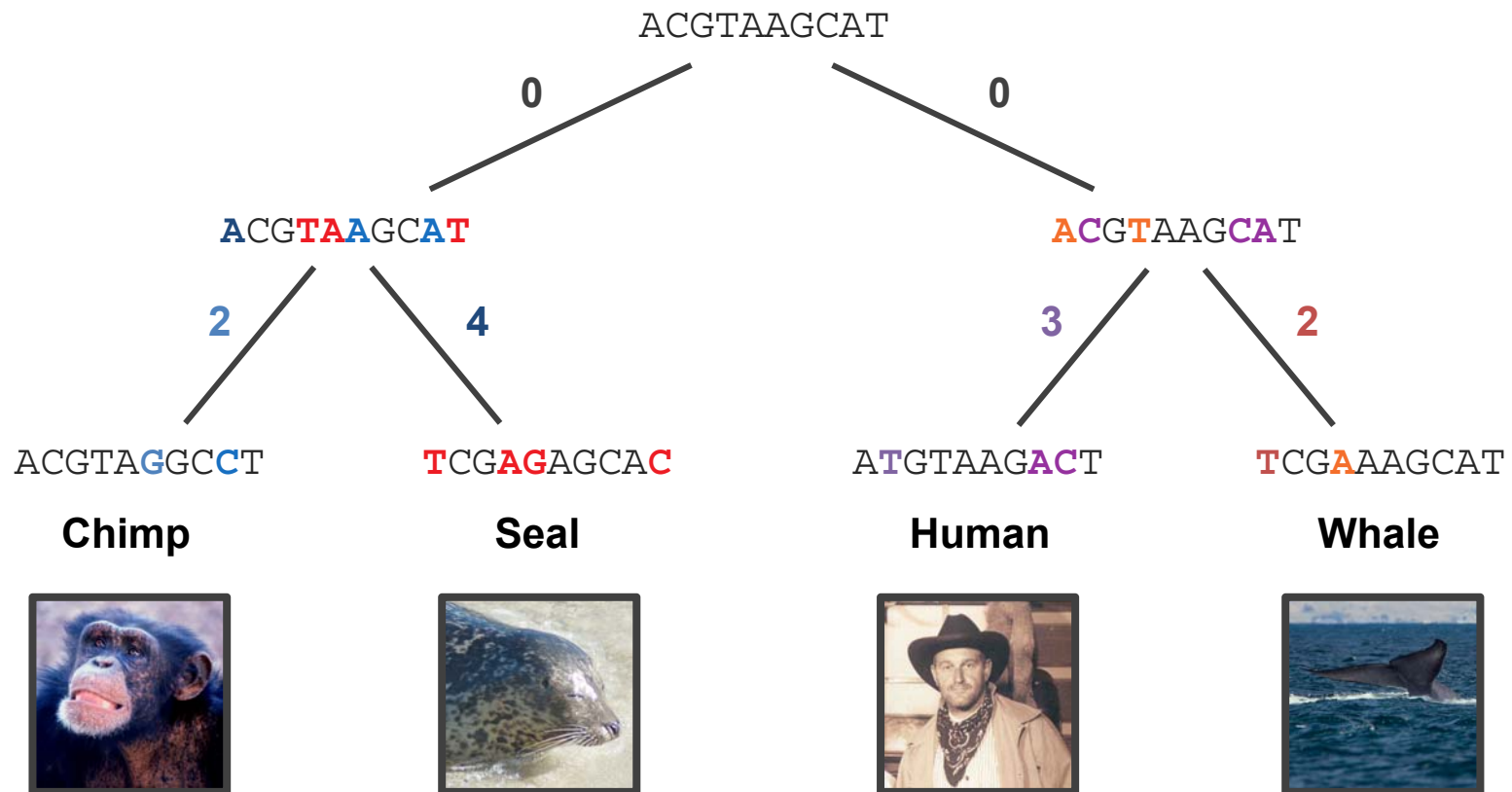
Code Challenge: Solve this problem.

Finding the Most Parsimonious Tree



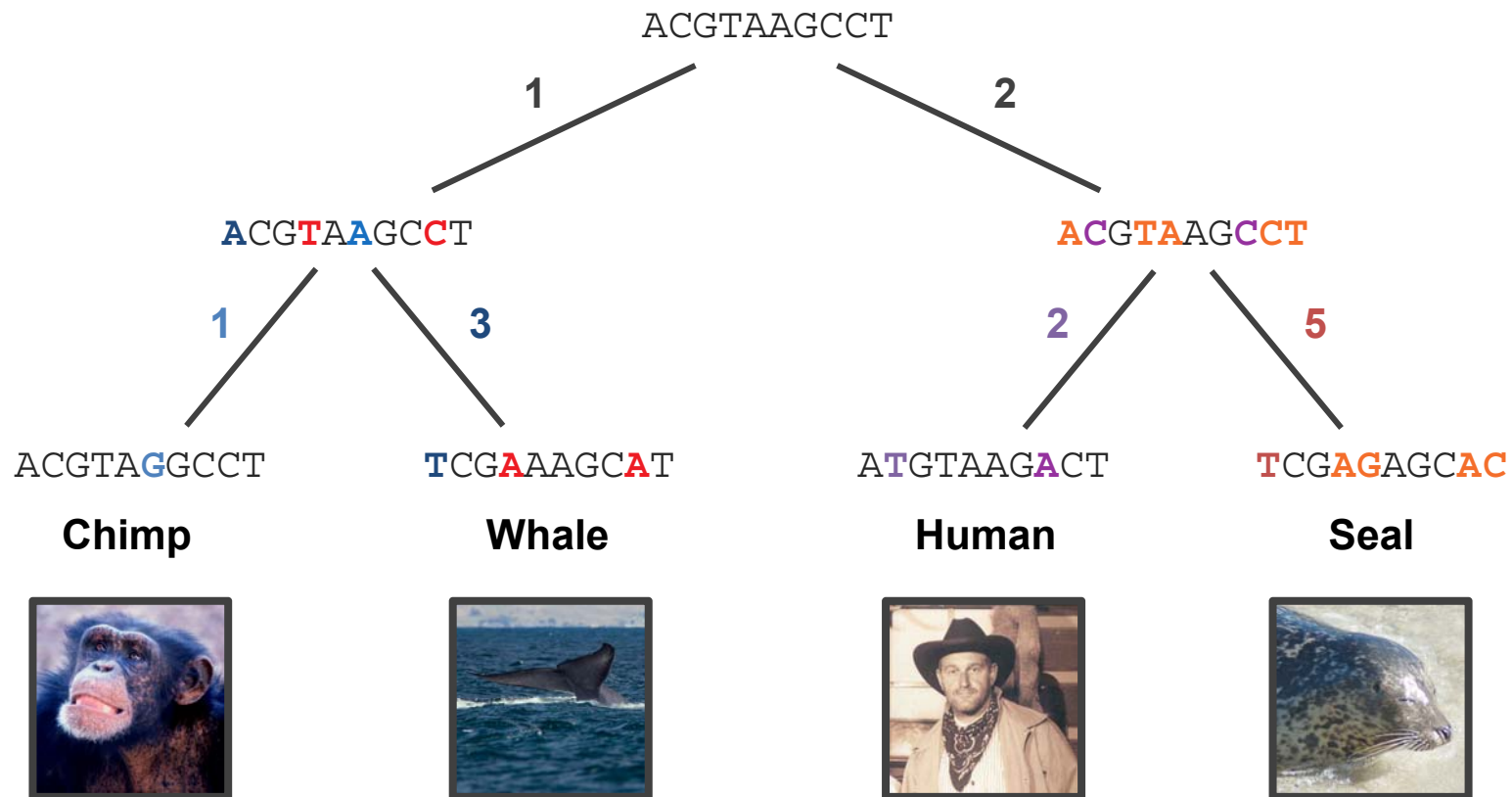
Parsimony Score: 8

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

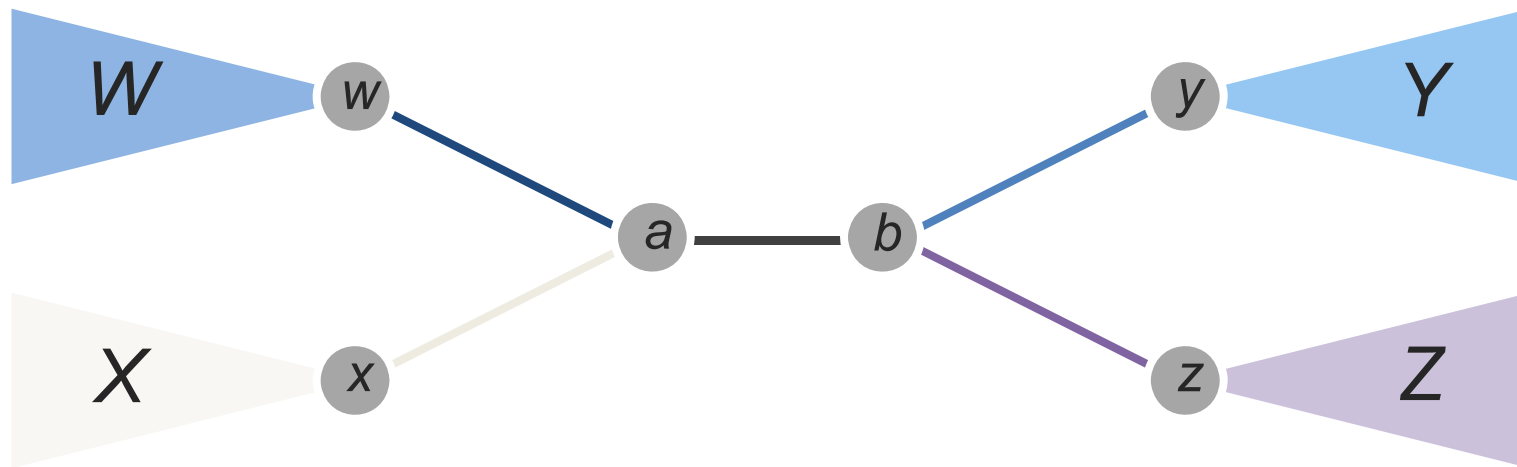
Large Parsimony Problem: *Given a set of strings, find a tree (with leaves labeled by all these strings) having minimum parsimony score.*

- **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...

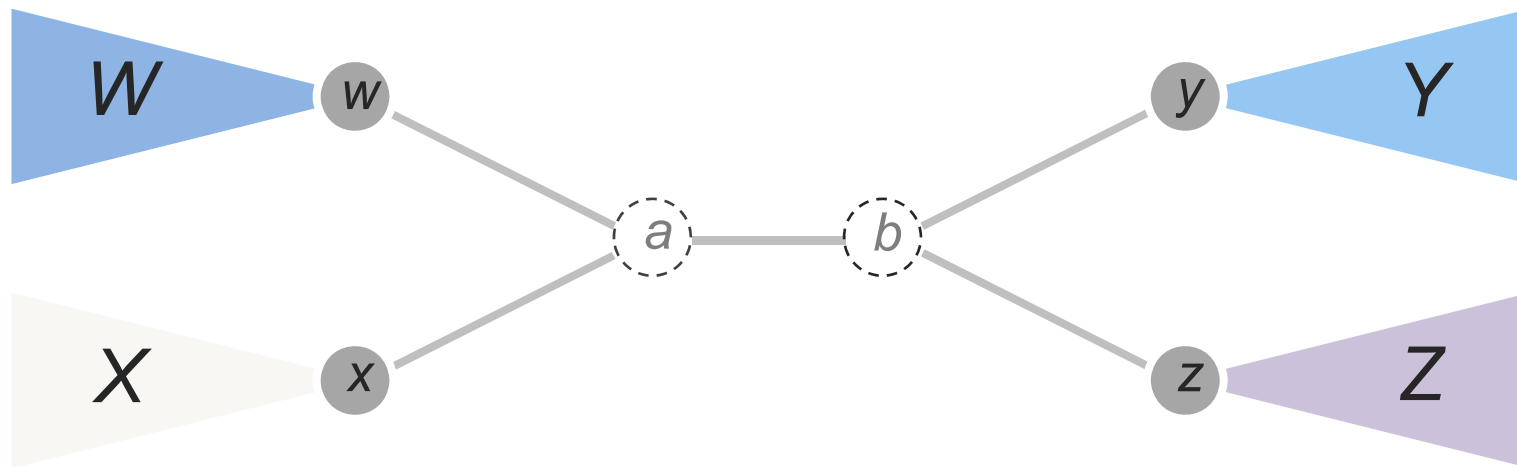
A Greedy Heuristic for Large Parsimony

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



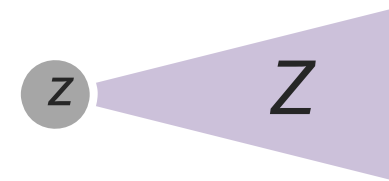
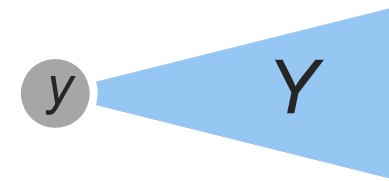
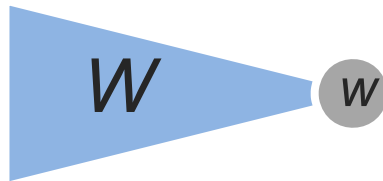
A Greedy Heuristic for Large Parsimony

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



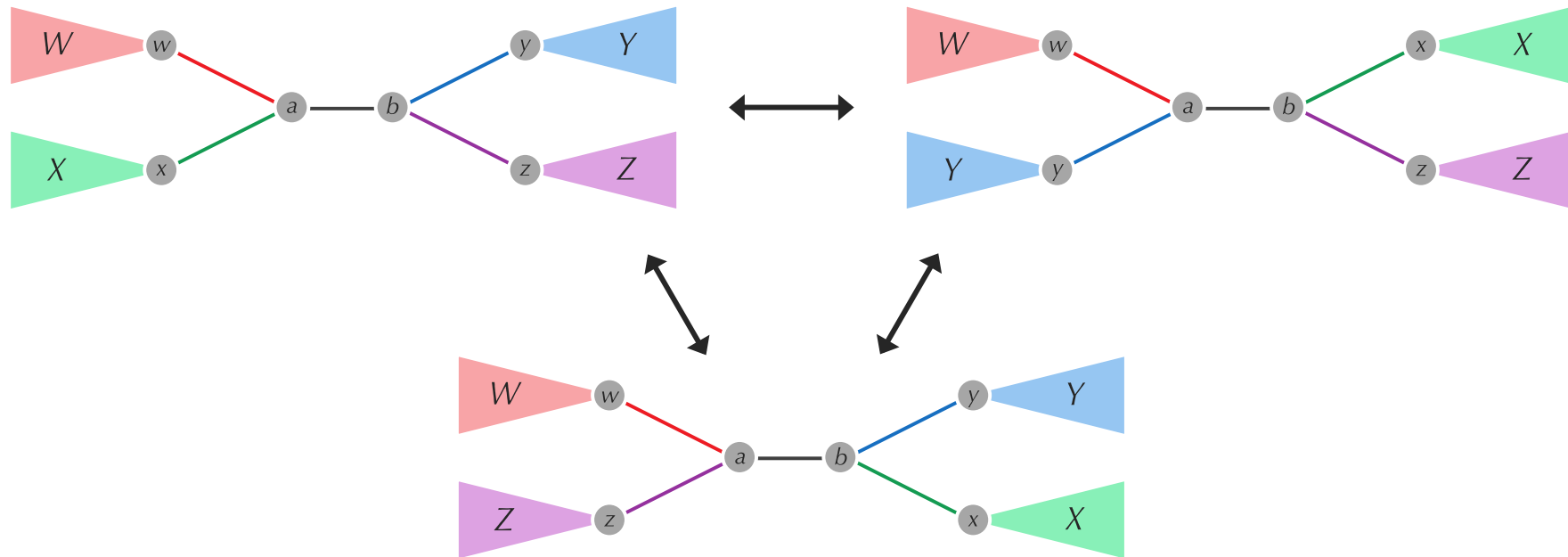
A Greedy Heuristic for Large Parsimony

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.

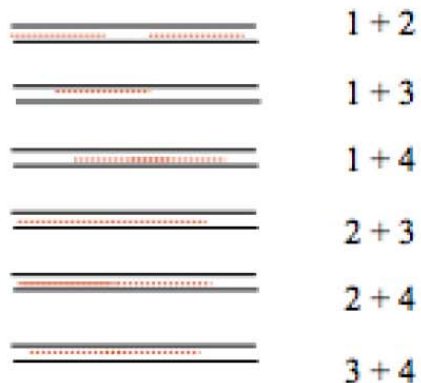
Current uses of parsimony

- Parsimony has generally fallen out of favour in preference for model-based approaches
- Uses:
 - Inference of 'difficult' states e.g. indels
 - Heuristic for guiding iterative tree search

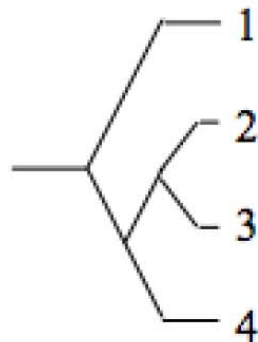
Back to alignment: progressive alignment

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 CLUSTALW

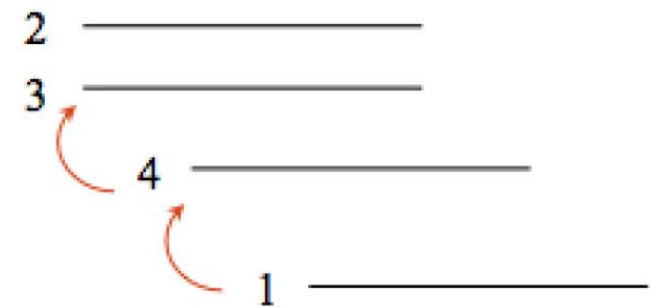
Pairwise Alignment



Guide Tree



Iterative Multiple Alignment



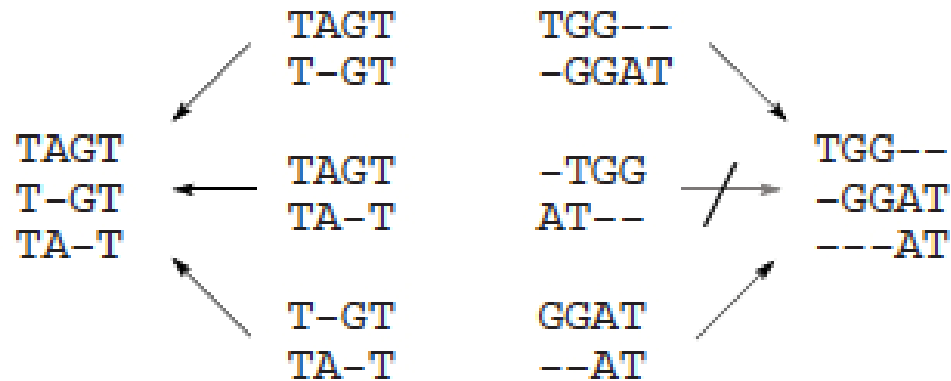
Progressive Alignment

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.

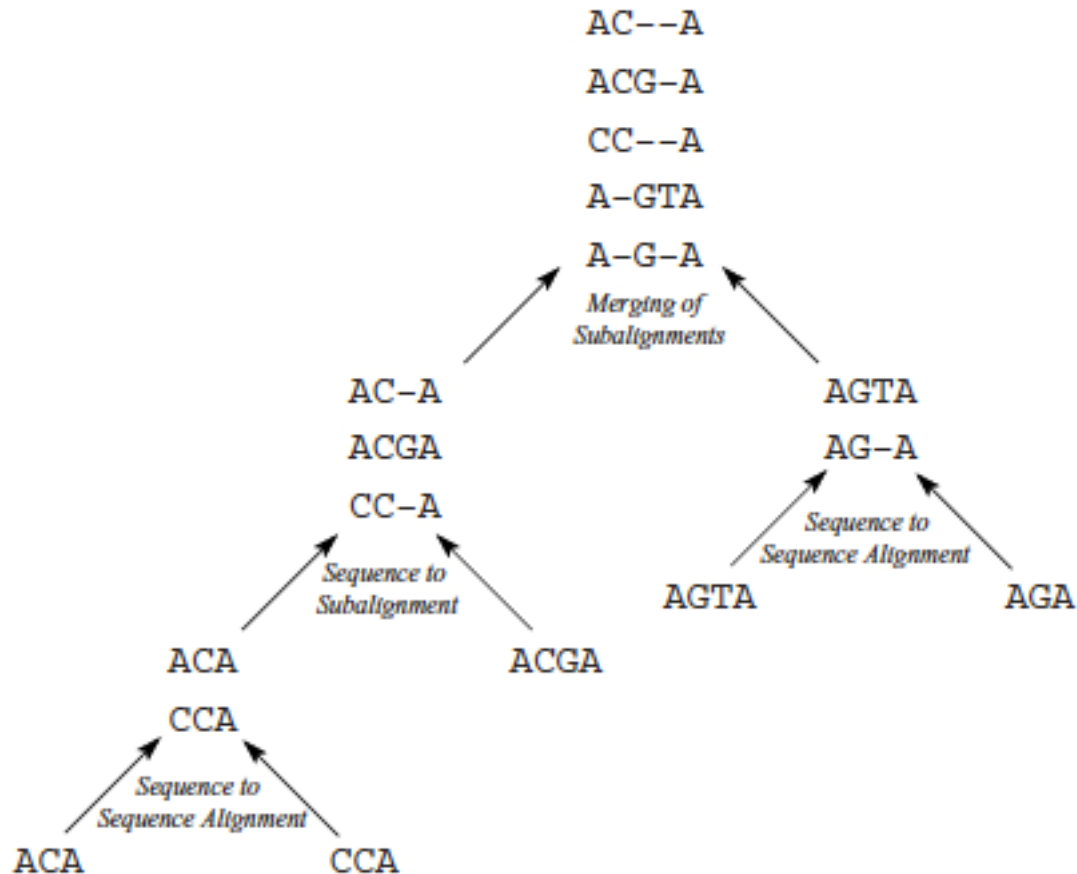
Progressive Alignment

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

AAA
AAA
AAT
ATC

Small section (3 columns) of the alignment of 4 sequences

Let's start from an alignment of four sequences (above the first three columns);
Compute the frequencies for the occurrence of each letter in each column of multiple alignment $p_A = 1, p_T = p_G = p_C = 0$ (1st column);
 $p_A = 0.75, p_T = 0.25, p_G = p_C = 0$ (2nd column);
 $p_A = 0.50, p_T = 0.25, p_C = 0.25, p_G = 0$ (3rd column);
Compute entropy of each column: $E = - \sum_{X=A,C,G,T} p_X \log(p_X)$
The entropy for a multiple alignment is the sum of entropies of each column of the alignment.

Implementation: <http://www.ebi.ac.uk/Tools/msa/>

Genome Sequencing Outline

- What Is Genome Sequencing?
- Exploding Newspapers
- The String Reconstruction Problem
- String Reconstruction as a Hamiltonian Path Problem
- String Reconstruction as an Eulerian Path Problem
- Similar Problems with Different Fates
- De Bruijn Graphs
- Euler's Theorem
- Assembling Read-Pairs

Next Generation Sequencing Technologies

- **Late 2000s:** The market for new sequencing machines takes off.
 - Illumina reduces the cost of sequencing a human genome from \$3 billion to \$10,000.
 - Complete Genomics builds a genomic factory in Silicon Valley that sequences hundreds of genomes per month.
 - Beijing Genome Institute orders hundreds of sequencing machines, becoming the world's largest sequencing center.



illumina

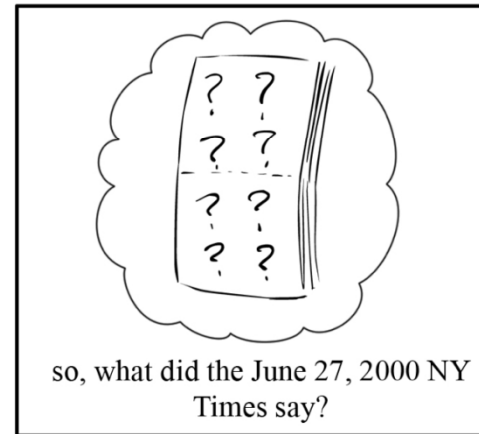
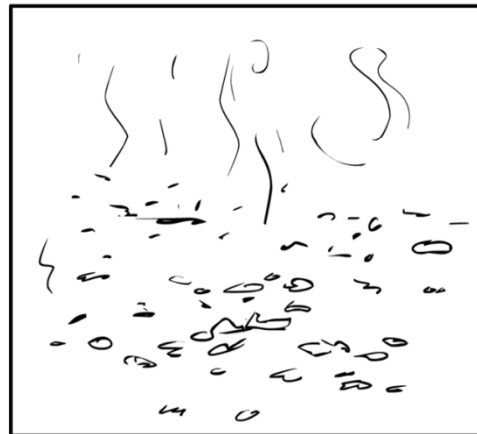
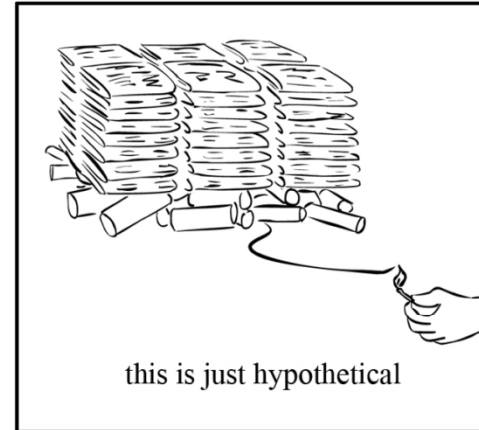
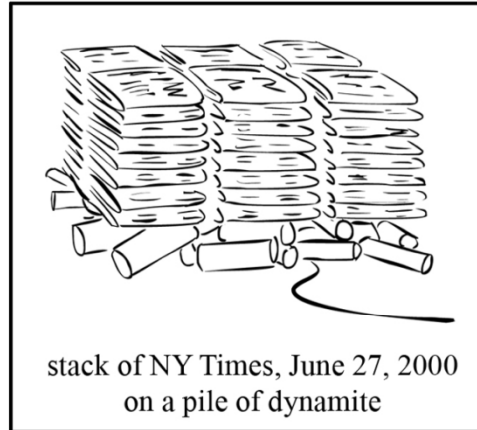
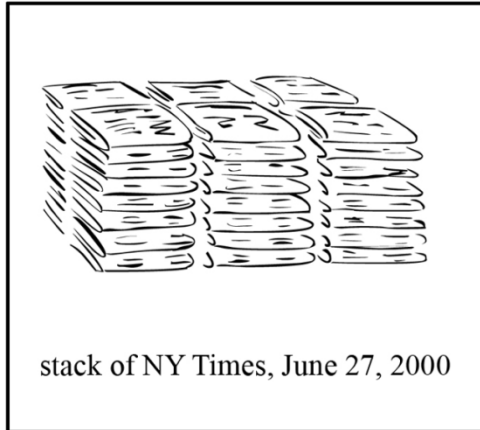
Complete
genomics

华大基因
BGI

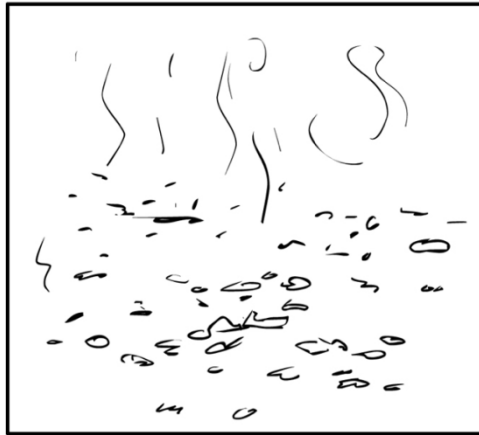
Why Do We Sequence Personal Genomes?

- **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



moodie, appr
e have not yet named
mation is welc

lie, appr
yet named any suspects, alt
is welc

o'2
e ca

The Newspaper Problem as an Overlapping Puzzle



...noodie, app...
...e have not yet name...
...mation is welc...

...2...
...pects, alt...
...e ca...

Multiple Copies of a Genome (Millions of them)



CTGATGATGGACTACGCTACTACTGCTAGCTGTATTACGATCAGCTACCACATCGTAGCTACGATGCATTAGCAAGCTATCGGATCAGCTACCACATCGTAGC
CTGATGATGGACTACGCTACTACTGCTAGCTGTATTACGATCAGCTACCACATCGTAGCTACGATGCATTAGCAAGCTATCGGATCAGCTACCACATCGTAGC
CTGATGATGGACTACGCTACTACTGCTAGCTGTATTACGATCAGCTACCACATCGTAGCTACGATGCATTAGCAAGCTATCGGATCAGCTACCACATCGTAGC
CTGATGATGGACTACGCTACTACTGCTAGCTGTATTACGATCAGCTACCACATCGTAGCTACGATGCATTAGCAAGCTATCGGATCAGCTACCACATCGTAGC

Breaking the Genomes at Random Positions

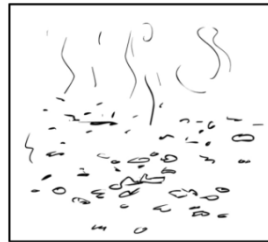


CTGATG★TGGACTACGC★ACTACTGCT★GCTGTATT★ATCAGCTACC★ATCGTAGCT★ATGCATTAGC★AGCTATCG★ATCAGCTAC★ATCGTAGC
CTGA★GATGGACT★CGCTACTACT★CTAGCTGTAT★ACGATCAGC★ACCACATCGT★CTACGATGC★TAGCAAGC★ATCGGATCA★TACCACAT★GTAGC
CTGATG★TGGACTACG★ACTACTGCT★CTGTATTAC★ATCAGCTA★CACATCGTAGC★ACGATGCATT★CAAGCTAT★GGATCAGCT★CCACATCGTAGC
CTGATGATG★CTACGCTAC★ACTGCTAGCT★TATTACGAT★AGCTACCACA★CGTAGCTACG★GCATTAGCA★CTATCGG★CAGCTACC★ATCGTAGC

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

ATCAGCTACCA CTGATGATGGACT AGCTATCGG AGCTACGATGCA ATGCATTAGCA
TACTGCTAG TGGACTAGCTAC TTAGCAAGCT TCGTAGCTACG CTGATGATGG CATCGTAGC
CTGATGA ATCAGCTACC GCTGTATTACG GCTACCACATC GCTATCGGA ATGCATTAGCAA
ATGCATTAGCA TACTGCTAGCT GCAAGCTATC CTGTATTACG CATCGTAGC
CTGATGATG GACTACGCT ATCAGCTACCACA ACGATGCATTA CACATCGTAGC
ACGCTACTACT ATCGGATCA ACTACTGCTA TACGATCAGC
ACATCGTAGCT GGATCAGCTAC ATCAGCTACCACA ACGATGCATTA CACATCGTAGC
TACTGCTAGCT ATCGTAGCTACG GCAAGCTATC ACTAGCTAC TACGATCAGC
GCTAGCTGTAT GTATTACGATC TACGATCAGC ACGATGCATTA CACATCGTAGC
TACCACATCGT
CTGATGATGG
ATCGTAGCTACG

From Experimental to Computational Challenges

Multiple (unsequenced) genome copies



Read generation

Reads



Genome assembly

Assembled genome



...GGCATGCGTCAGAACTATCATAGCTAGATCGTACGTAGCC...

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)
 - Progress in this area via Nanopore and PacBio
- 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(**TA****ATG****CC****ATG****GG****ATG****TT**) =

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

Composition₃(TAATGCCATGGGATGTT) =

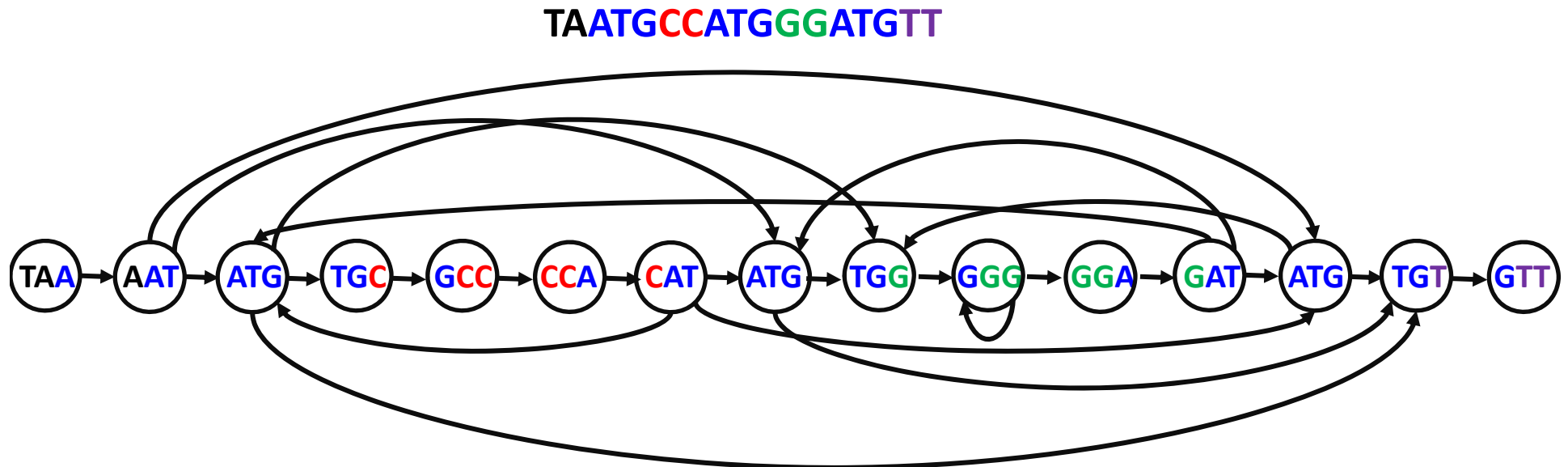


Can we construct this **genome path** without knowing the genome TAATGCCATGGGATGTT, only from its composition?

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

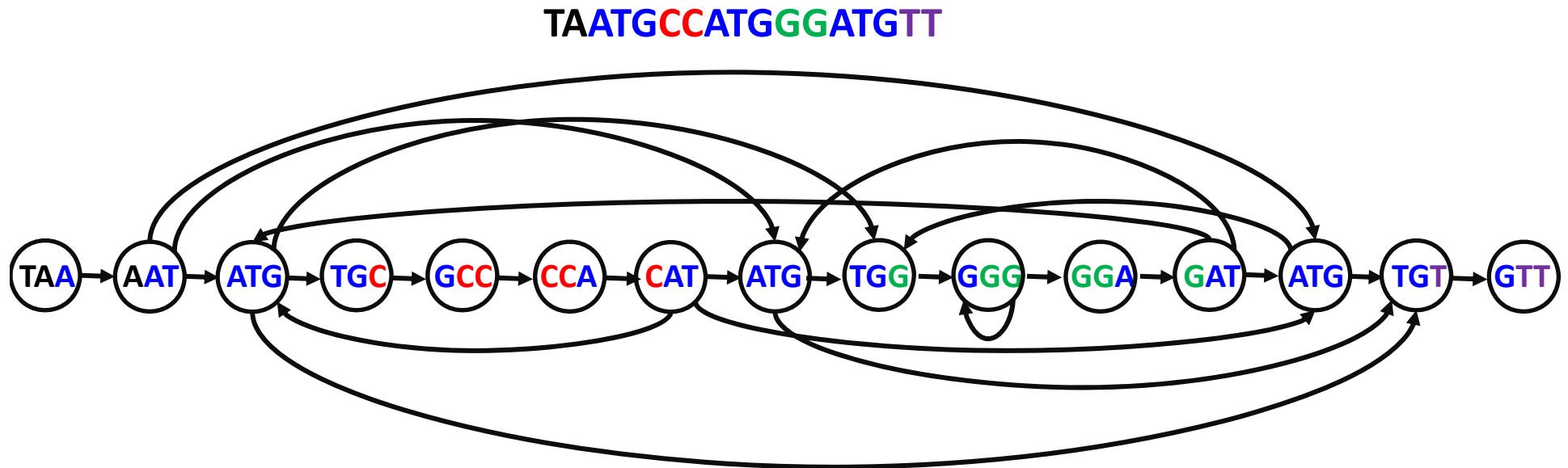


A Path Turns into a Graph



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

A Path Turns into a Graph

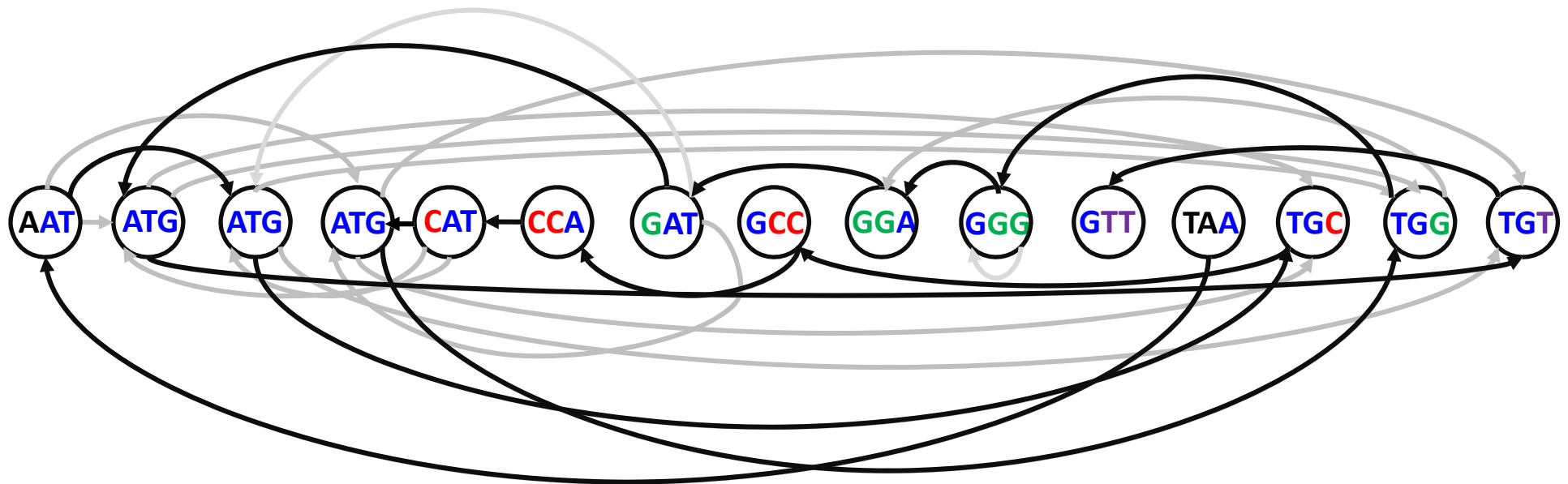


Can we still find the **genome path** in this graph?

Where Is the Genomic Path?

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

TAATGCCATGGGATGTT



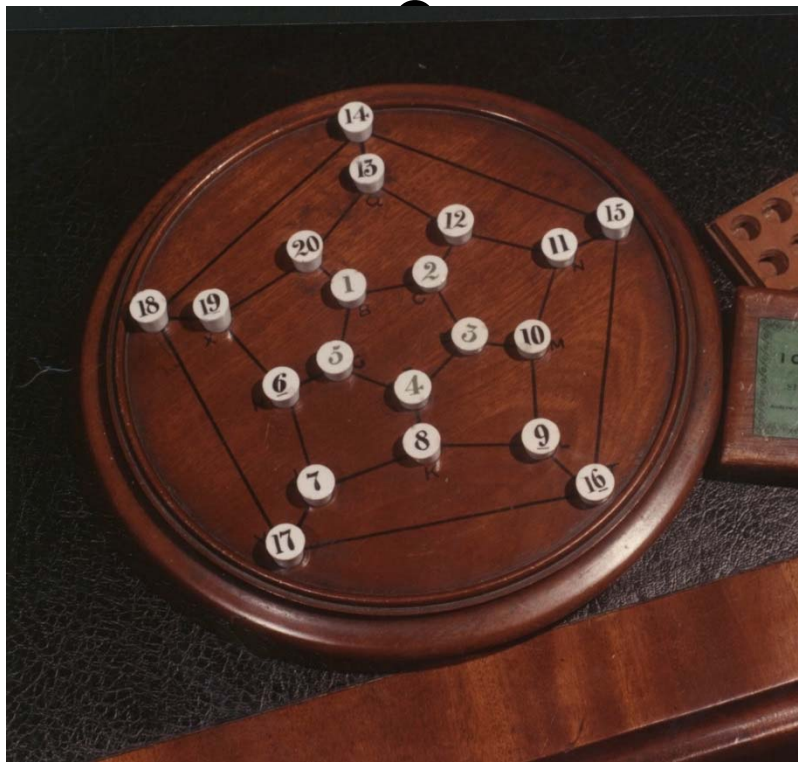
What are we trying to find in this graph?

Does This Graph Have a Hamiltonian Path?

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

Input. A graph.

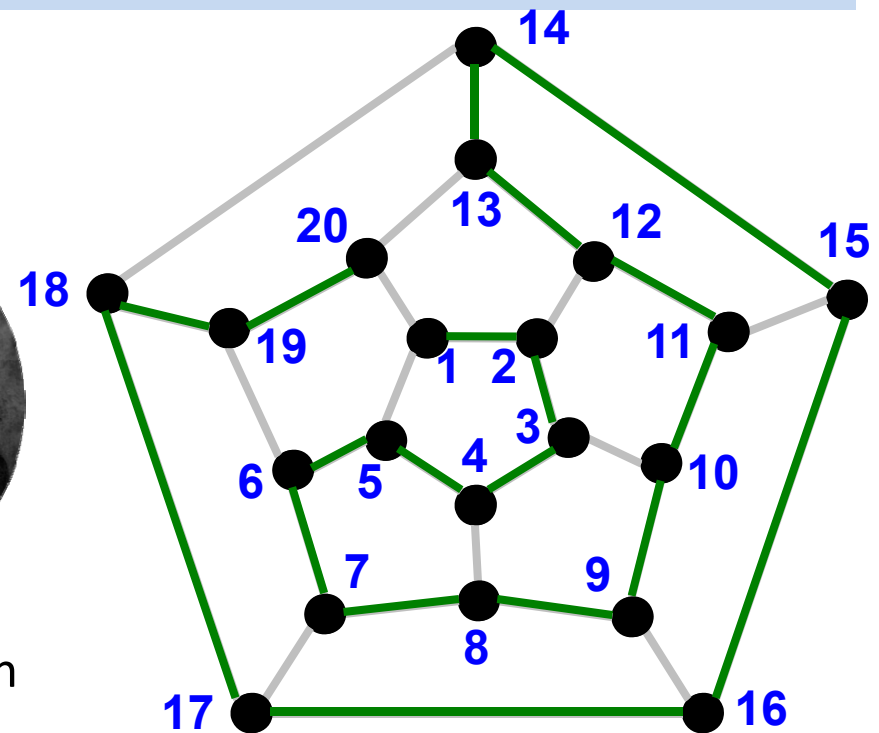
Output. A path visiting every **node** in the graph exactly once.



Icosian game (1857)

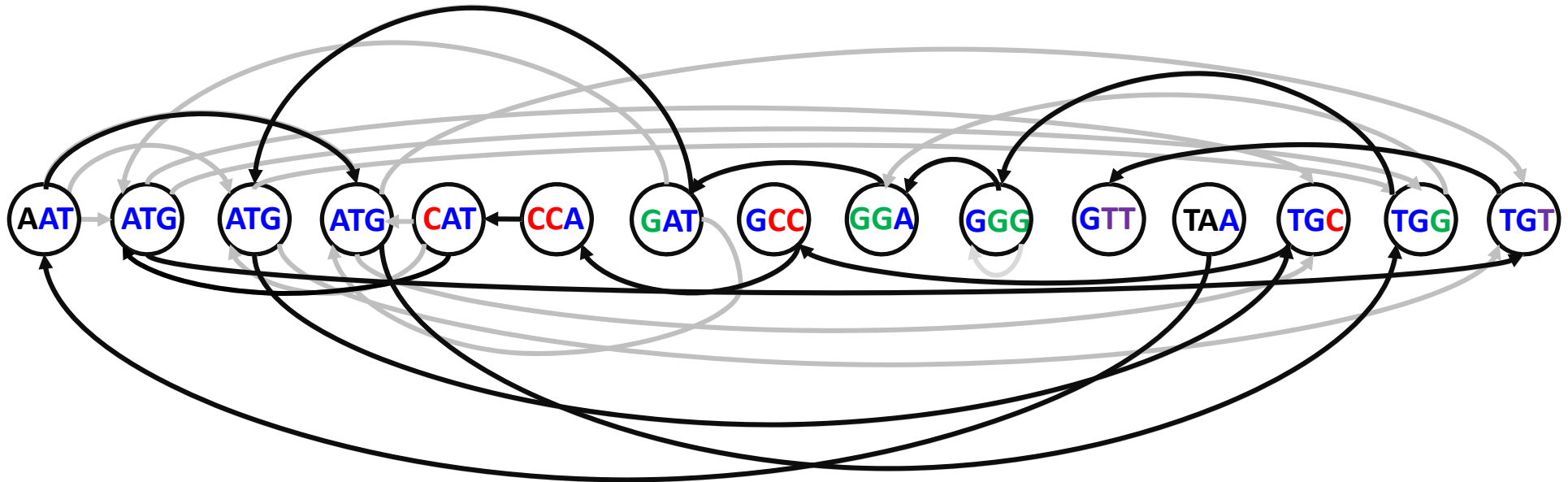


William Hamilton

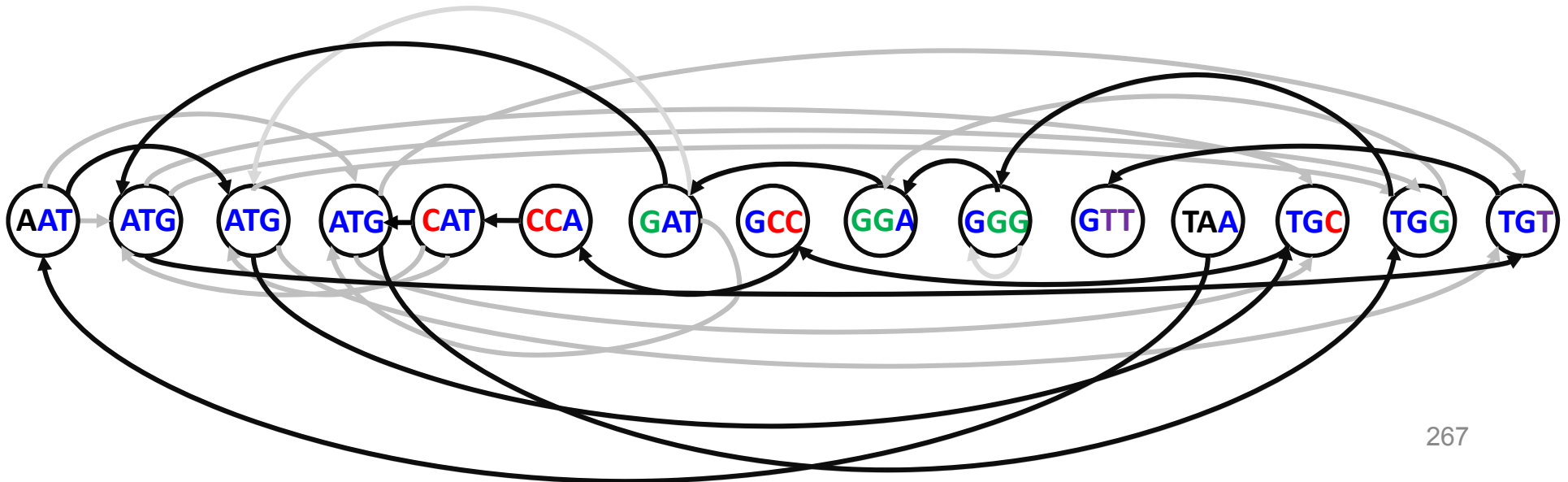


Undirected graph

TAATGGGATGCCATGTT



TAATGCCATGGGATGTT

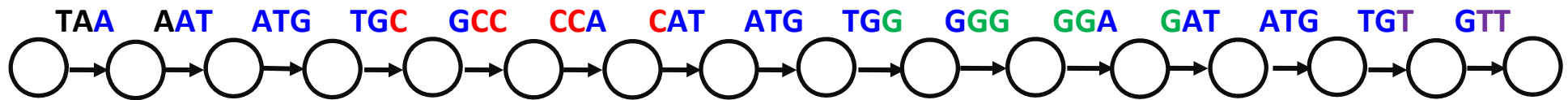


A Slightly Different Path

TAATGCCATGGGATGTT

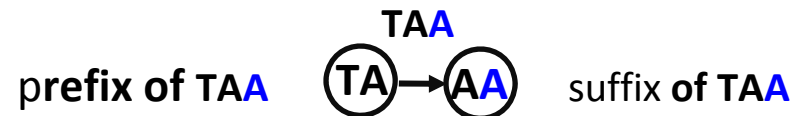


3-mers as nodes



3-mers as edges

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

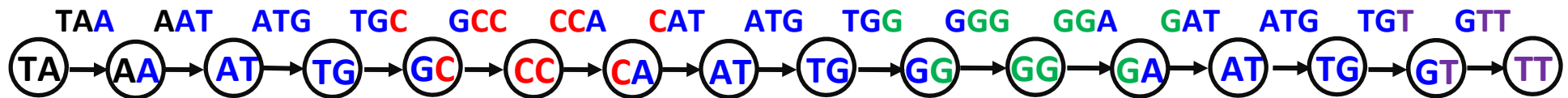


Labeling Nodes in the New Path

TAATGCCATGGGATGTT

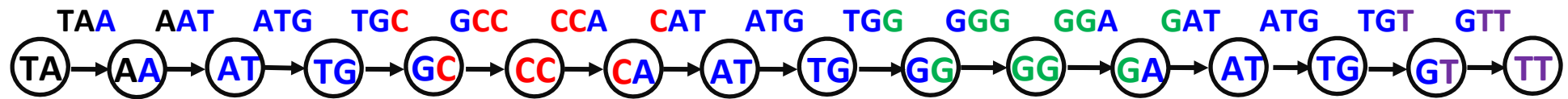


3-mers as nodes



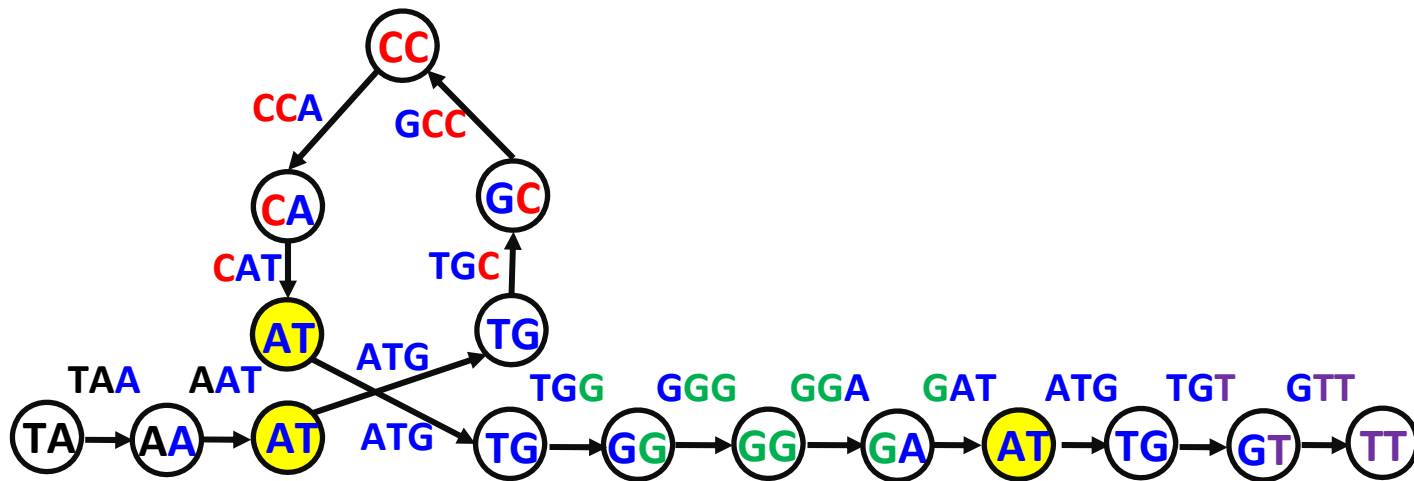
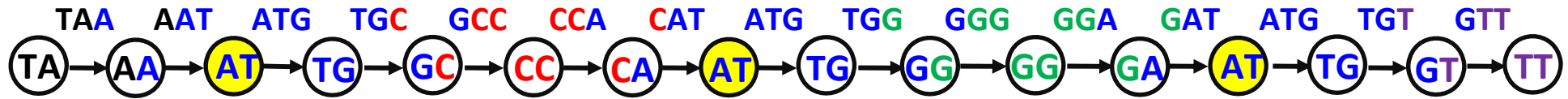
3-mers as edges and 2-mers as nodes

Labeling Nodes in the New Path

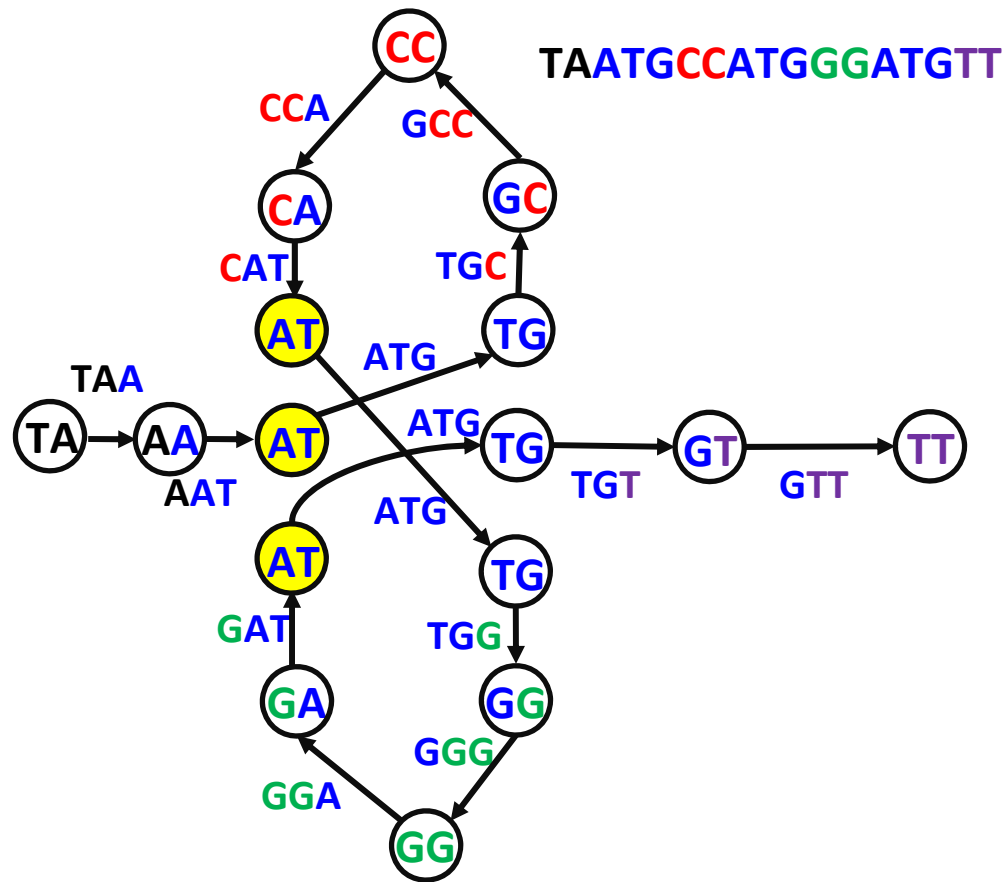


3-mers as edges and 2-mers as nodes

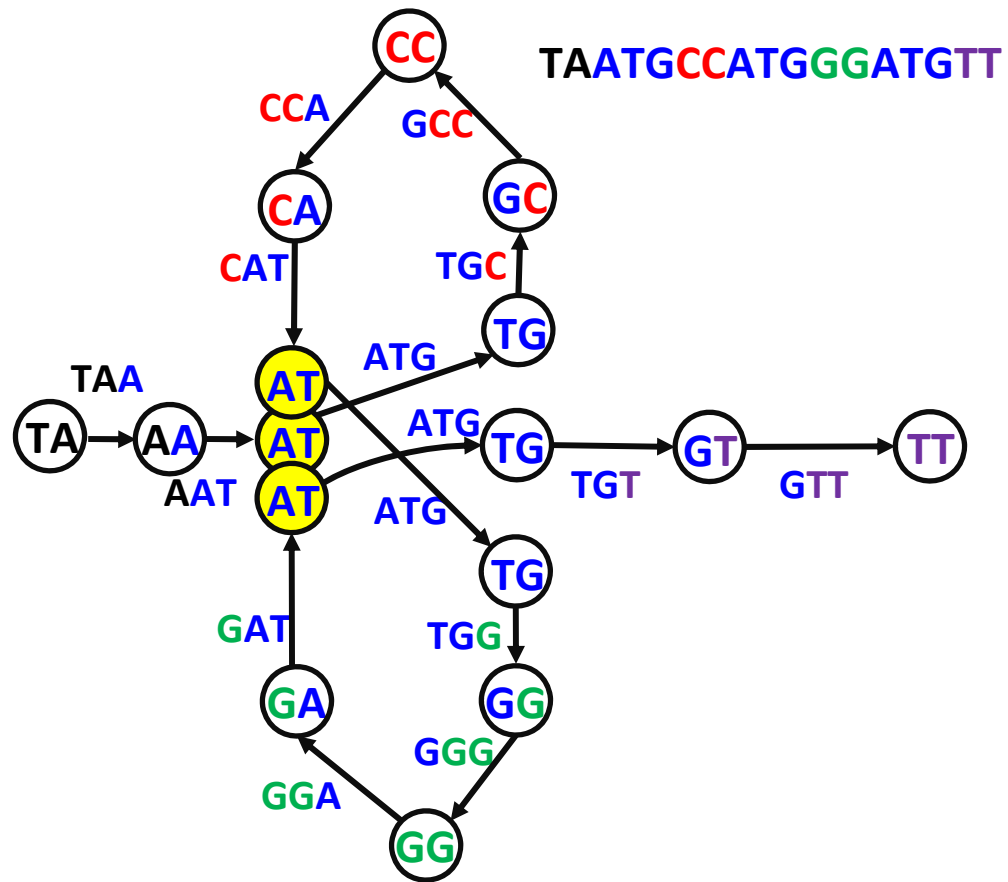
Gluing Identically Labeled Nodes



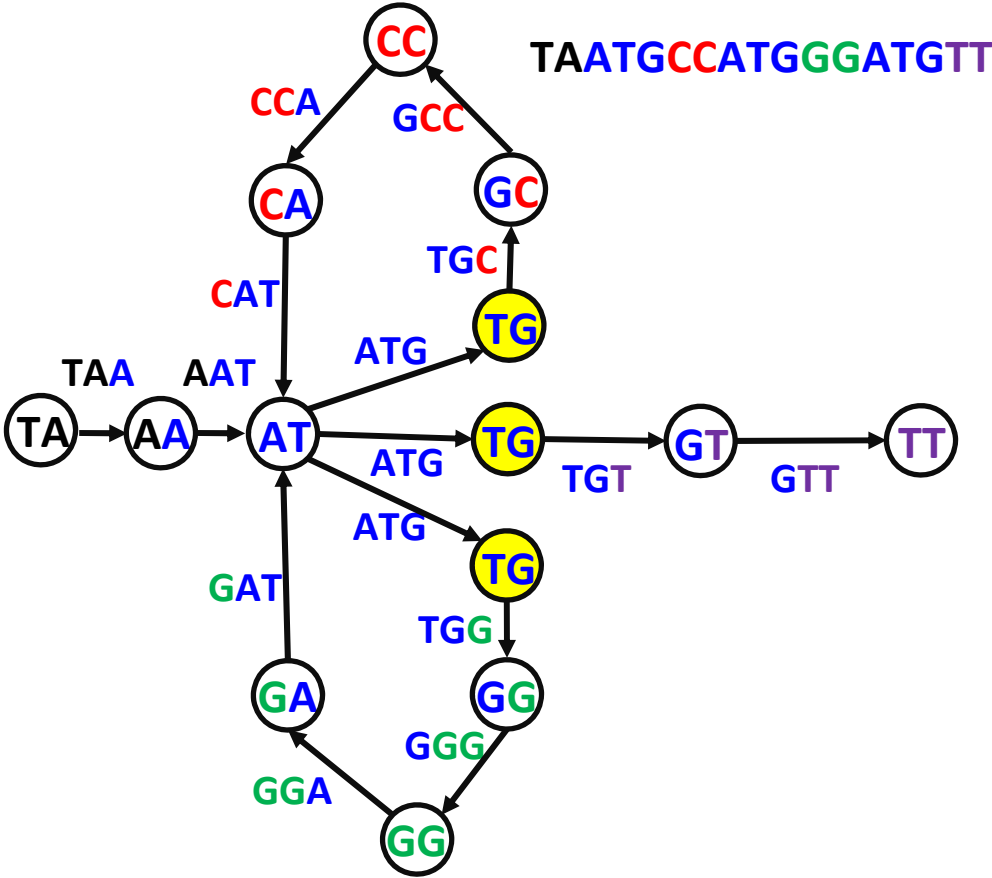
Gluing Identically Labeled Nodes



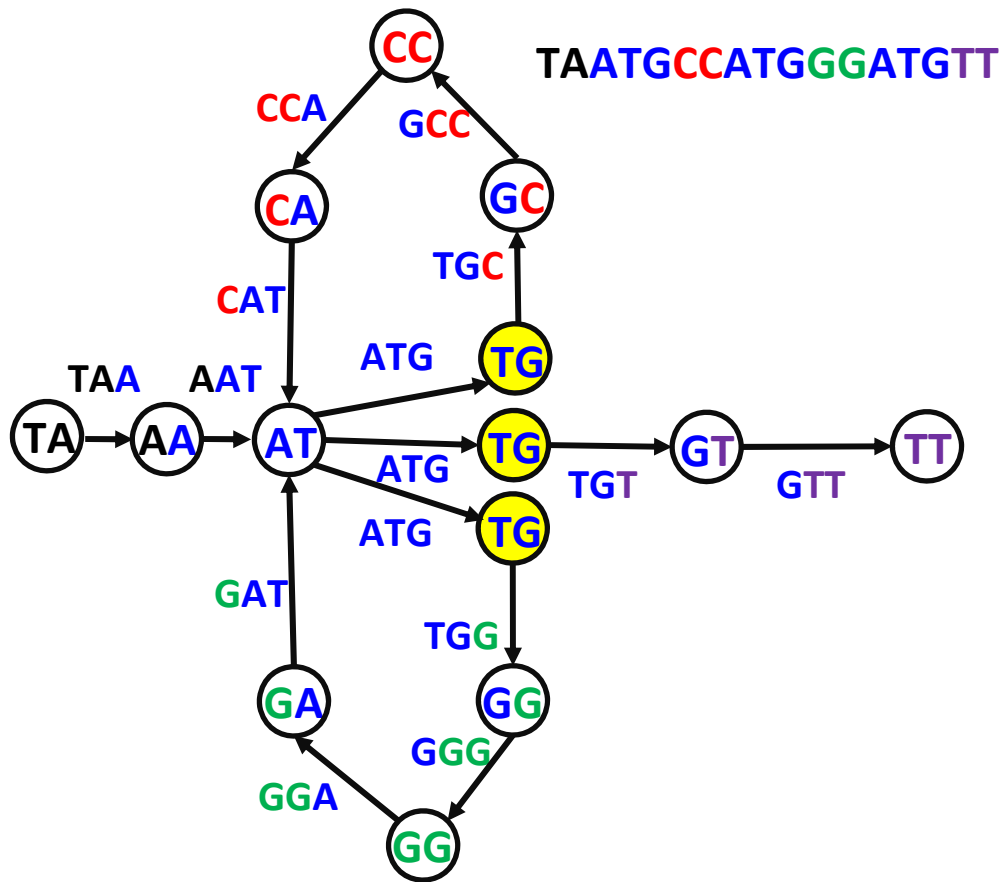
Gluing Identically Labeled Nodes



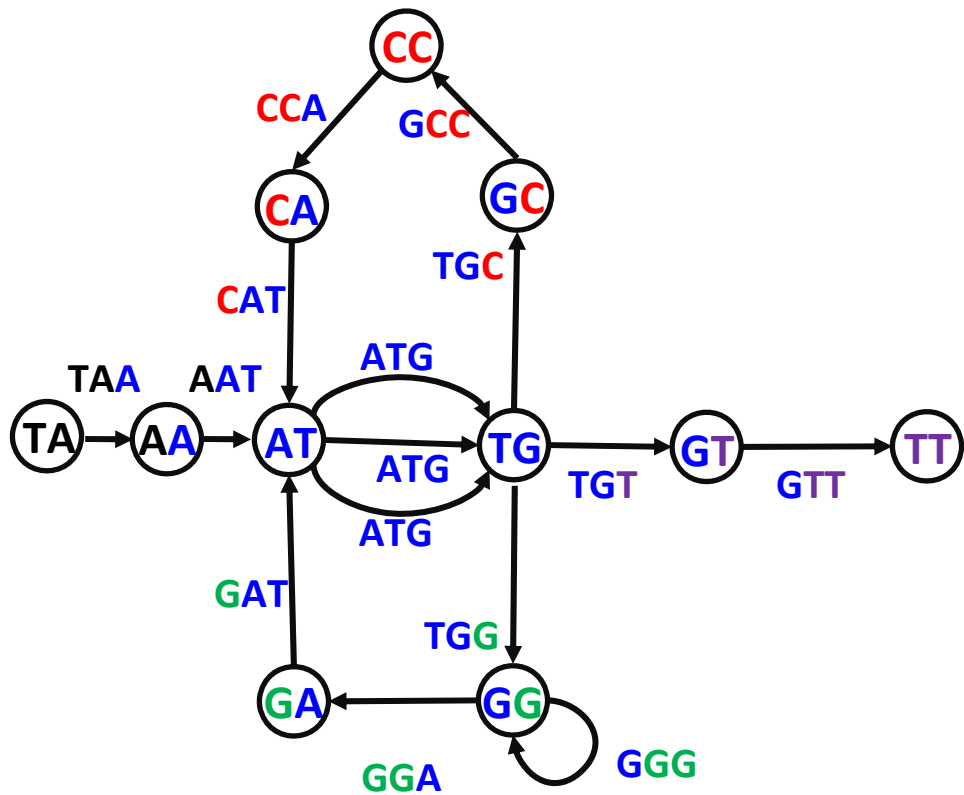
Gluing Identically Labeled Nodes



Gluing Identically Labeled Nodes



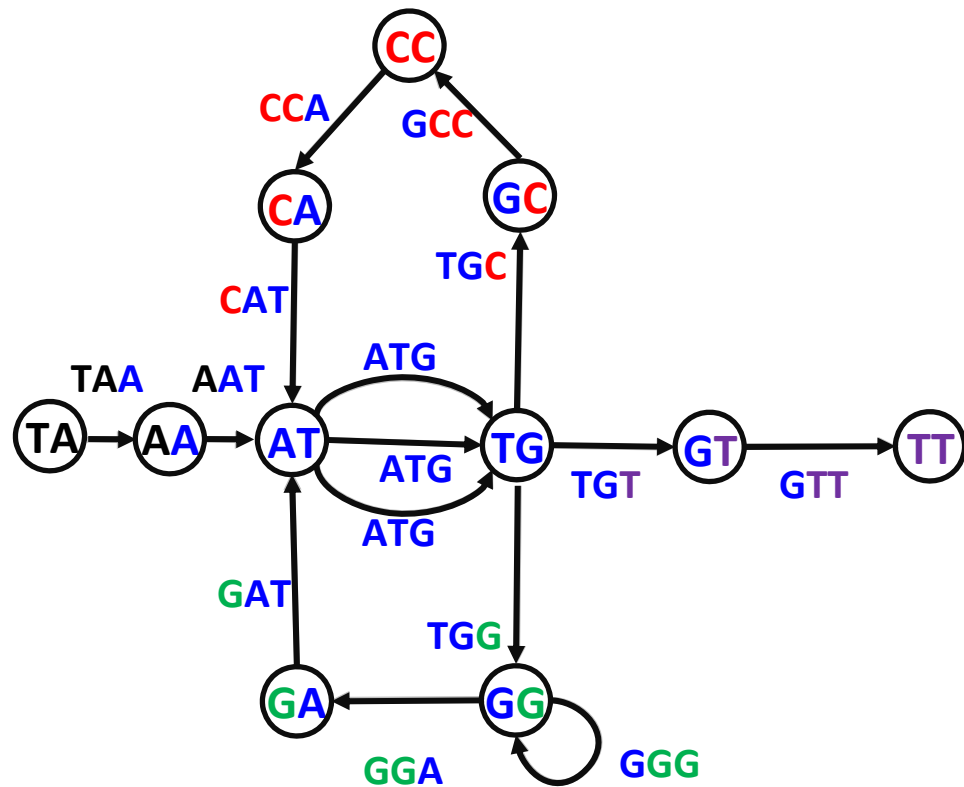
De Bruijn Graph of TAATGCCATGGGATGTT



Where is the Genome hiding in this graph?

It Was Always There!

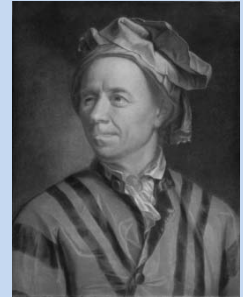
TAATGCCATGGGATGTT



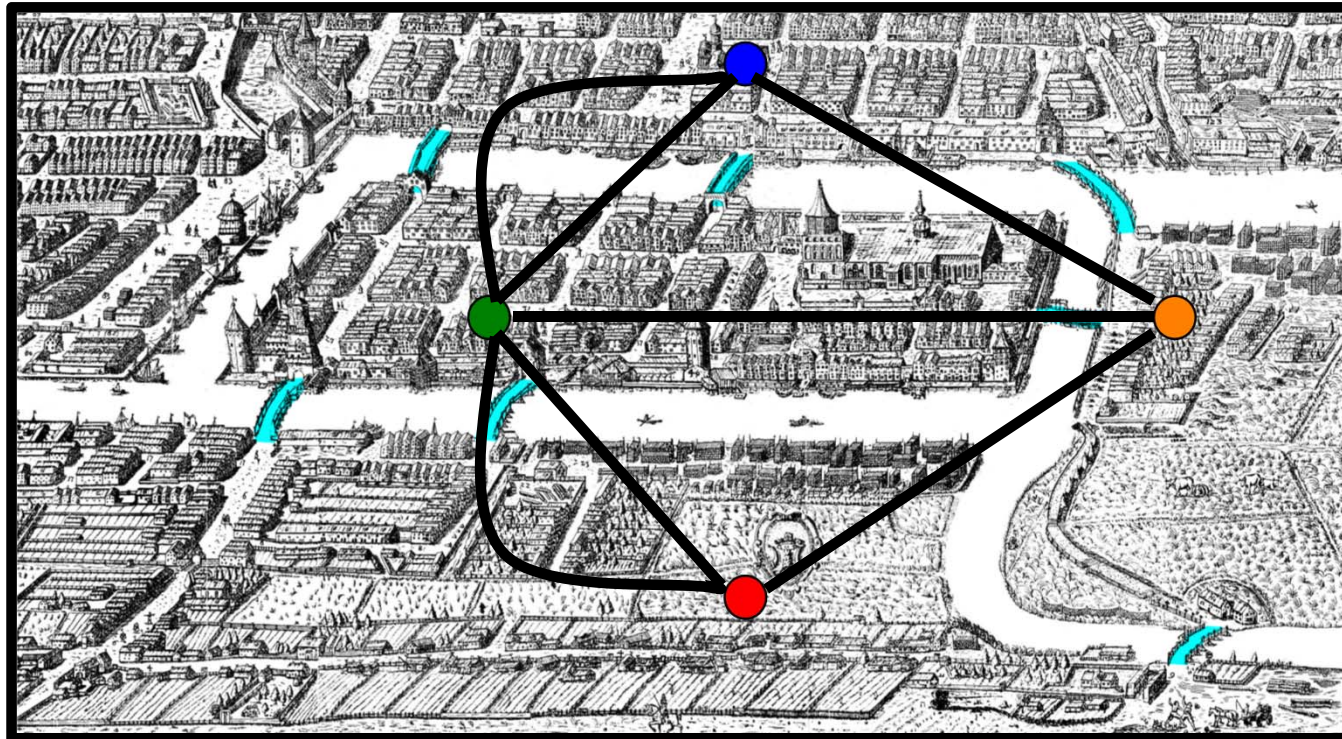
An Eulerian **path** in a graph is a path that visits each **edge** exactly once.

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

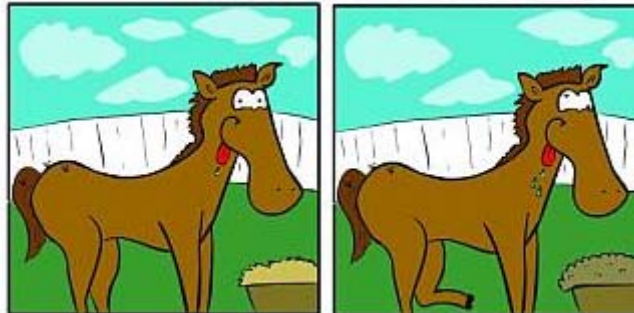
Eulerian Path Problem. Find an Eulerian path in a graph.

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

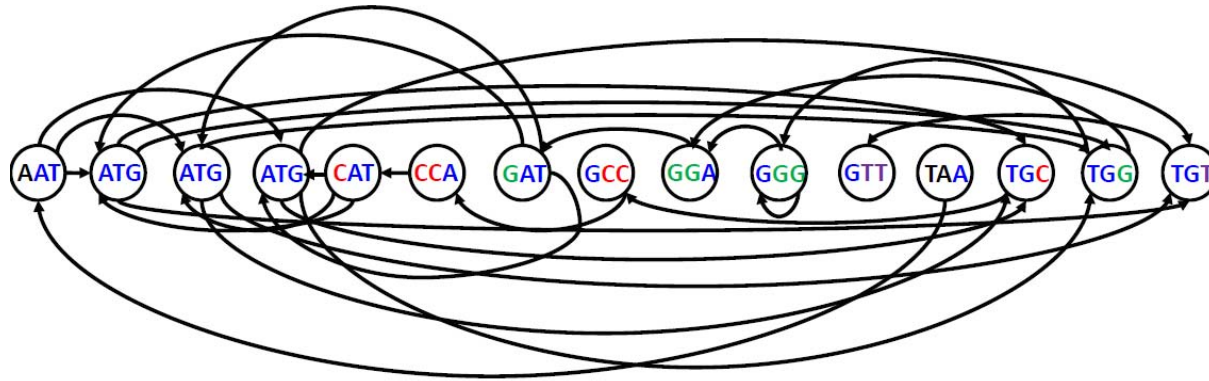
Hamiltonian Path Problem. Find a Hamiltonian path in a graph.

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

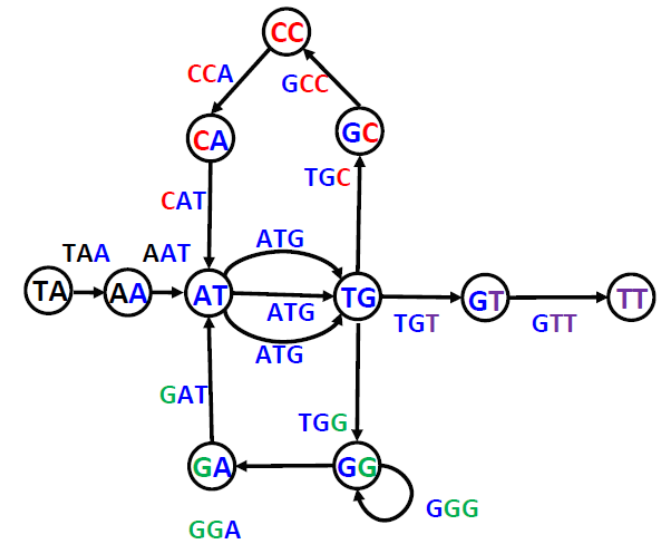
Find a difference!



What Problem Would You Prefer to Solve?

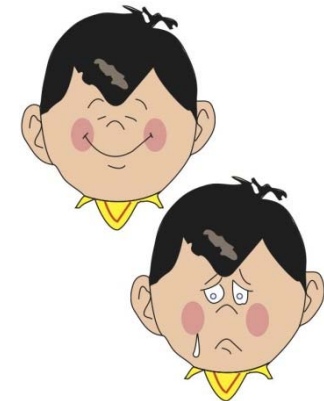


Hamiltonian Path Problem



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.

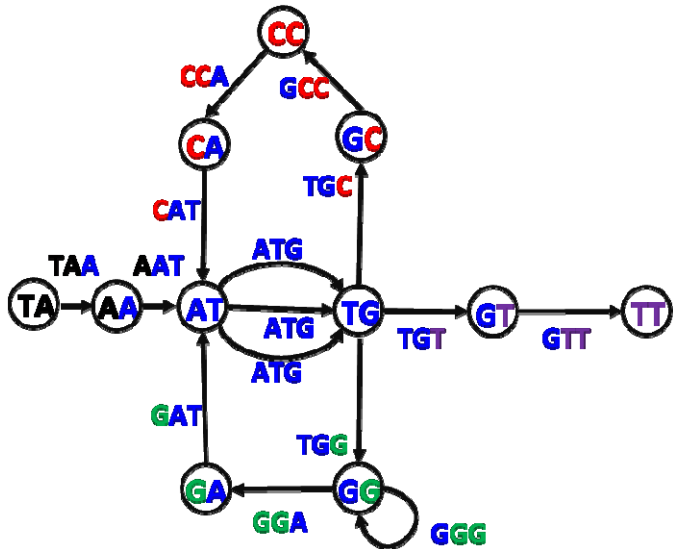
NP-Complete problems are all equivalent: find an efficient solution to one, and you have an efficient solution to them all.

Eulerian Path Problem

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

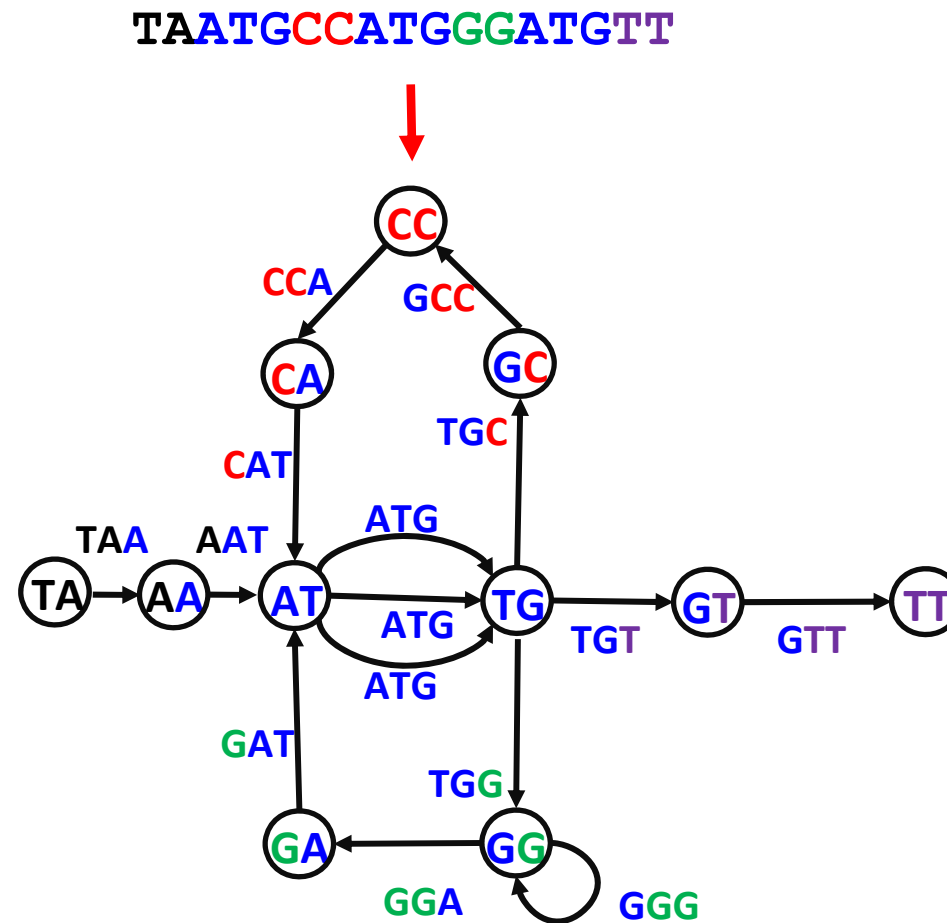


- Input. A graph.
- Output. A path visiting every **edge** in the graph exactly once.



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

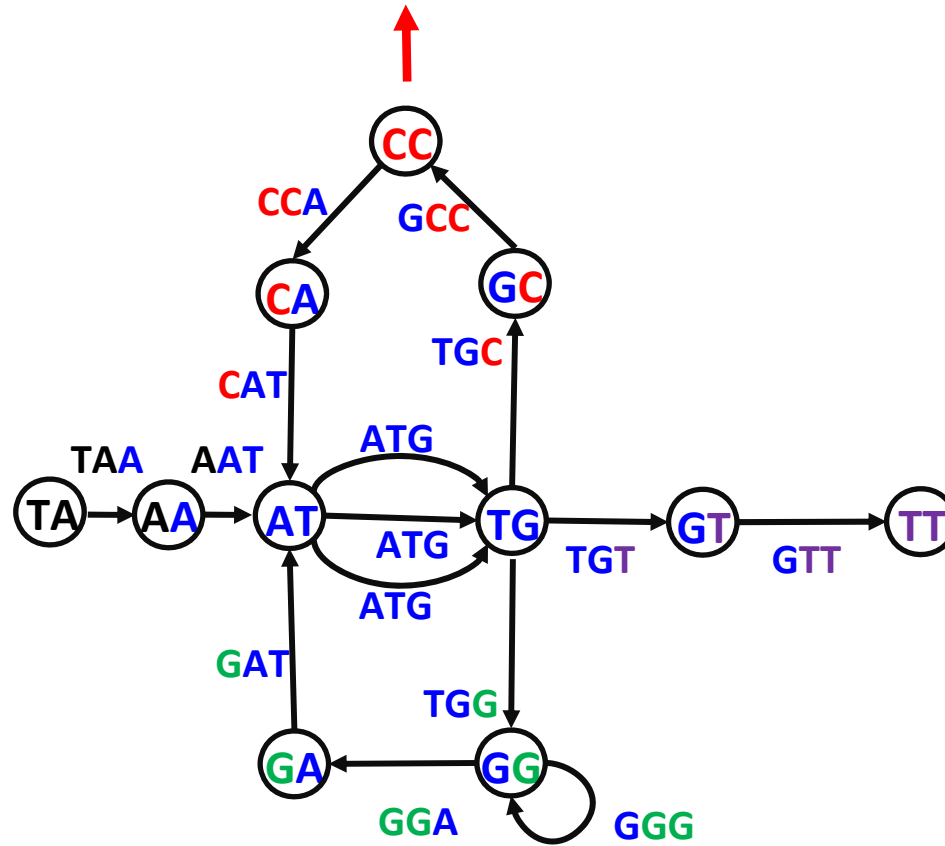
TAATGCCATGGATGTT



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

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

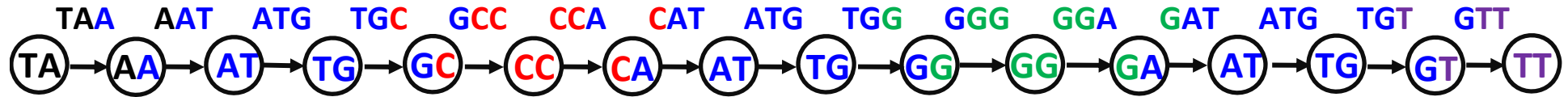
TAATG**CC**ATGGATGTT



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

Constructing de Bruijn Graph when Genome Is Known

TAATGCCATGGGATGTT

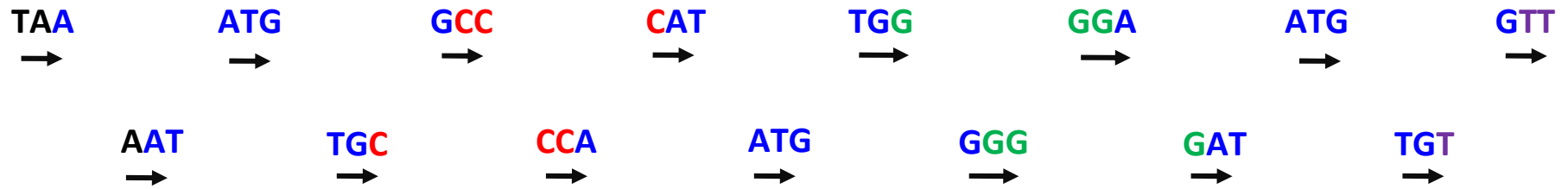


Constructing de Bruijn when Genome Is Unknown

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

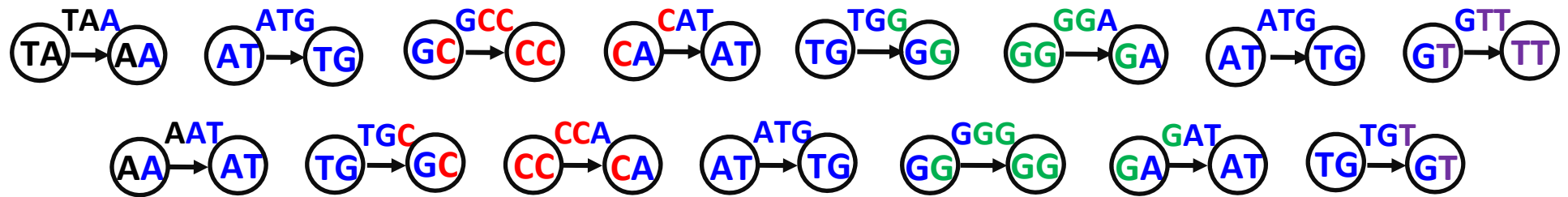
Composition₃(TAATGCCATGGGATGTT)

Representing Composition as a Graph Consisting of Isolated Edges



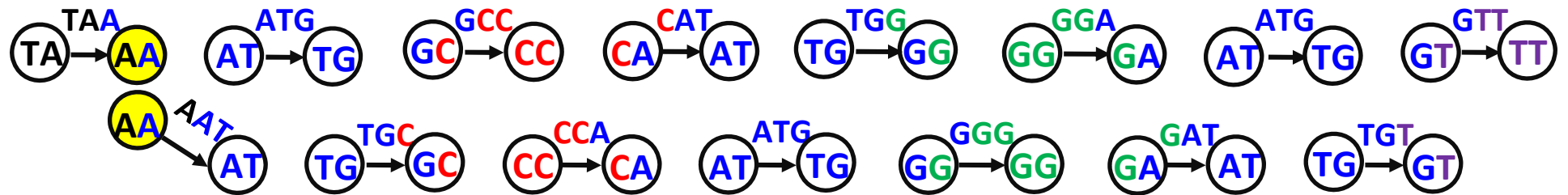
Composition₃(TAATGCCATGGGATGTT)

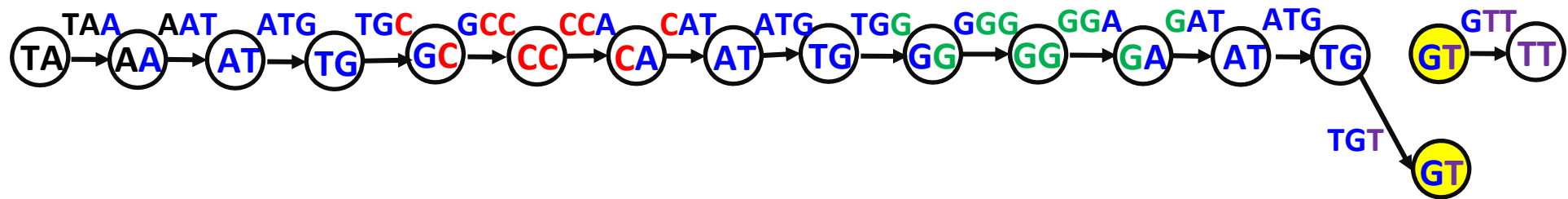
Constructing de Bruijn Graph from k-mer Composition



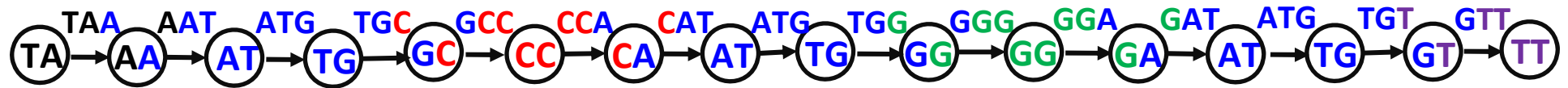
Composition₃(TAATGCCATGGGATGTT)

Glueing Identically Labeled Nodes

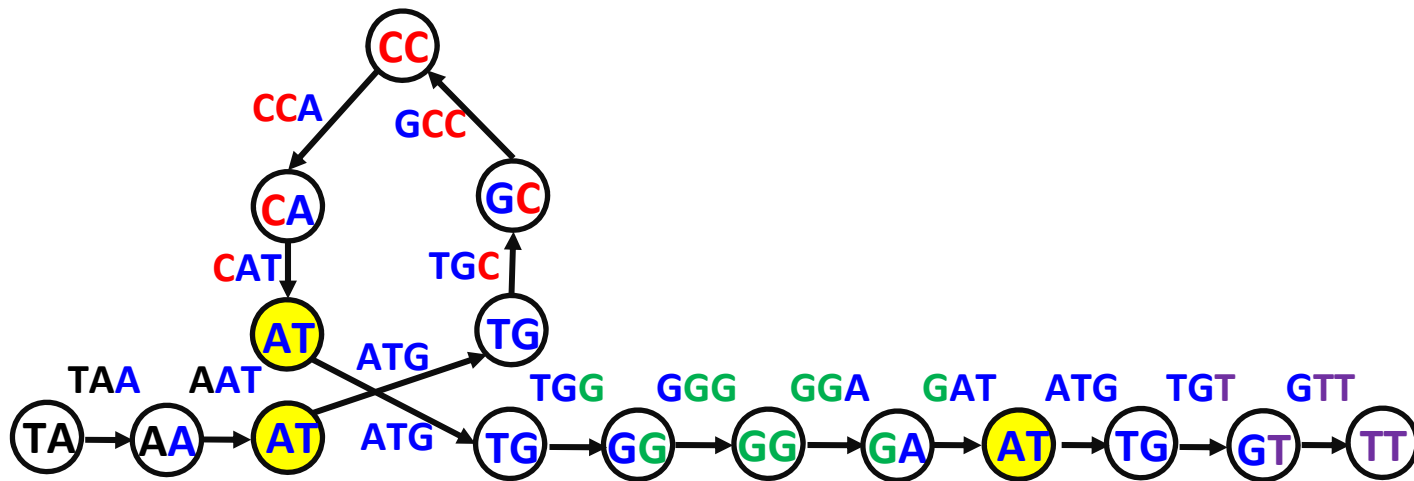
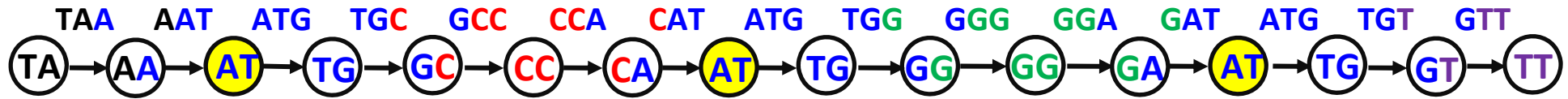




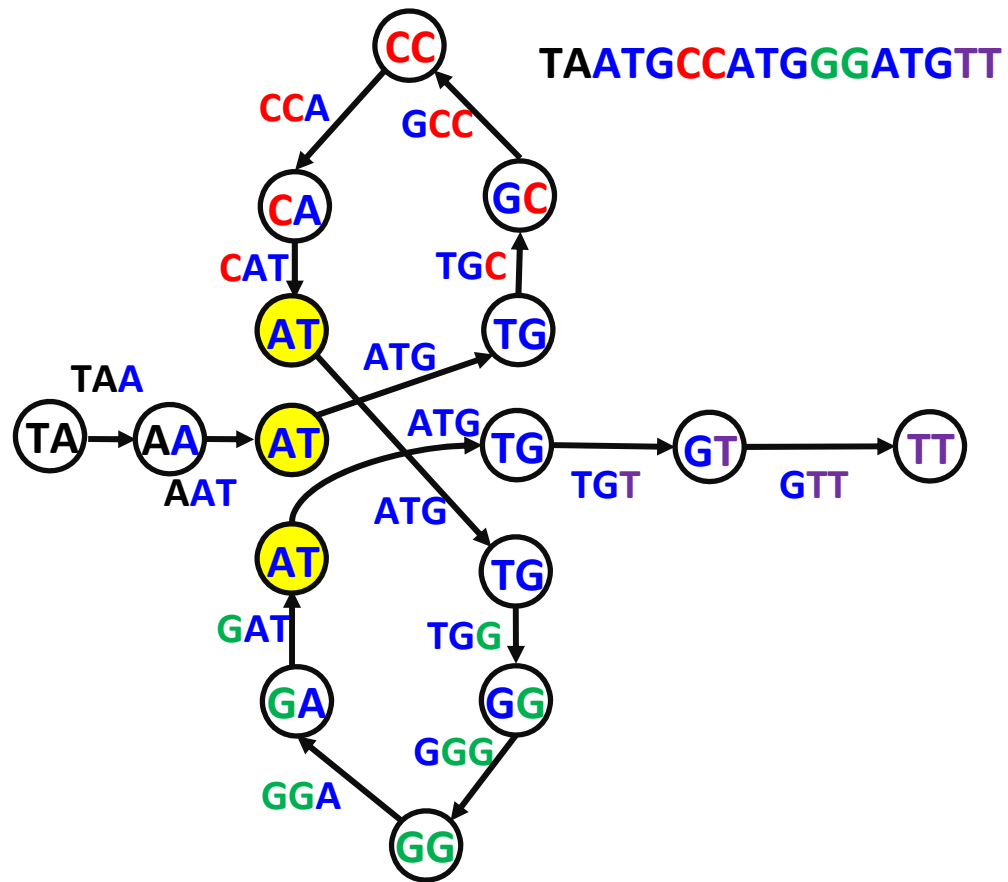
We Are Not Done with Glueing Yet

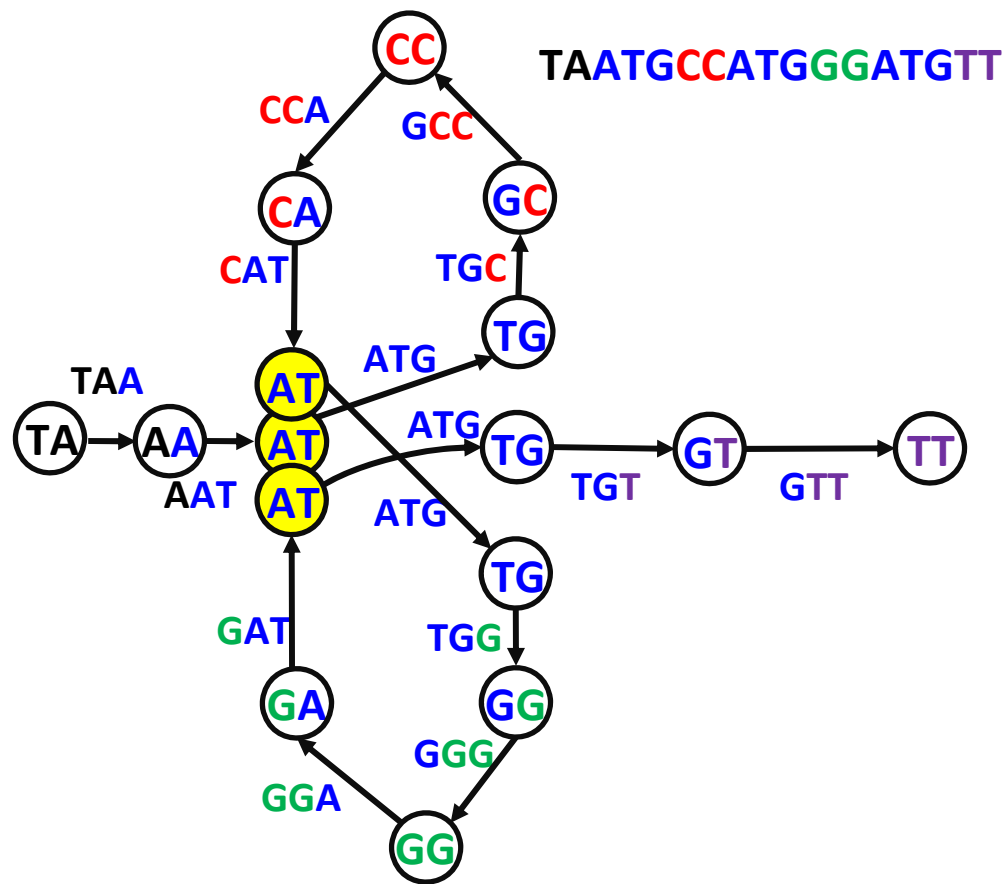


Glueing Identically Labeled Nodes

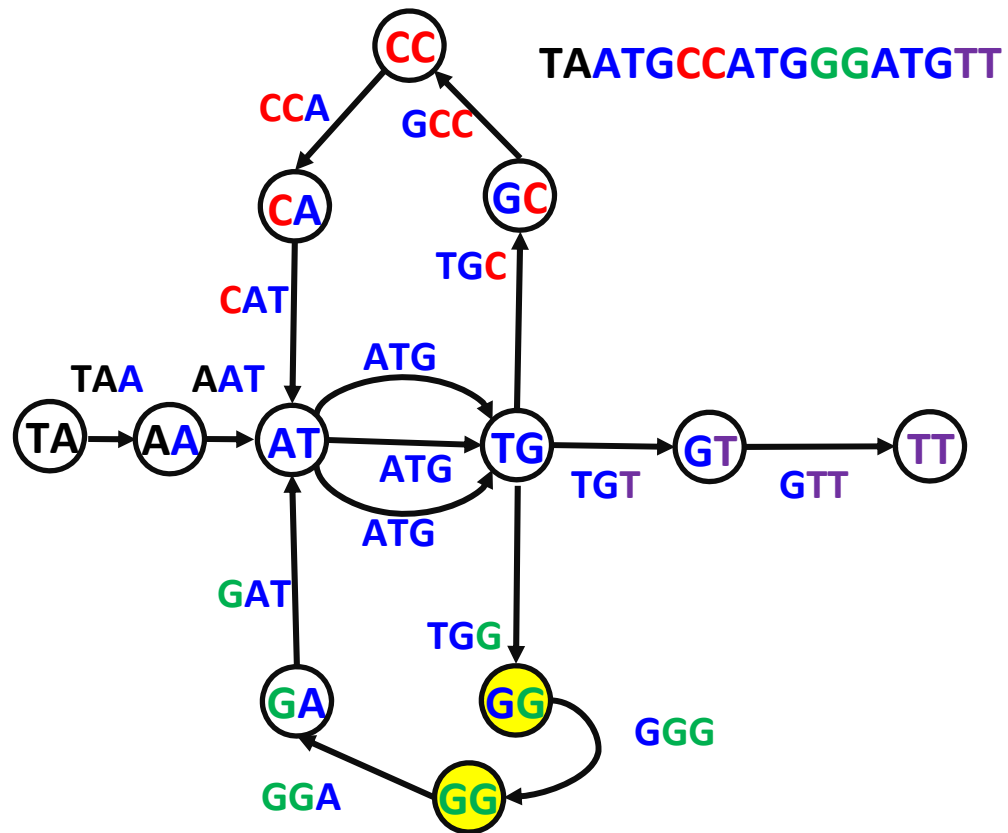


Glueing Identically Labeled Nodes

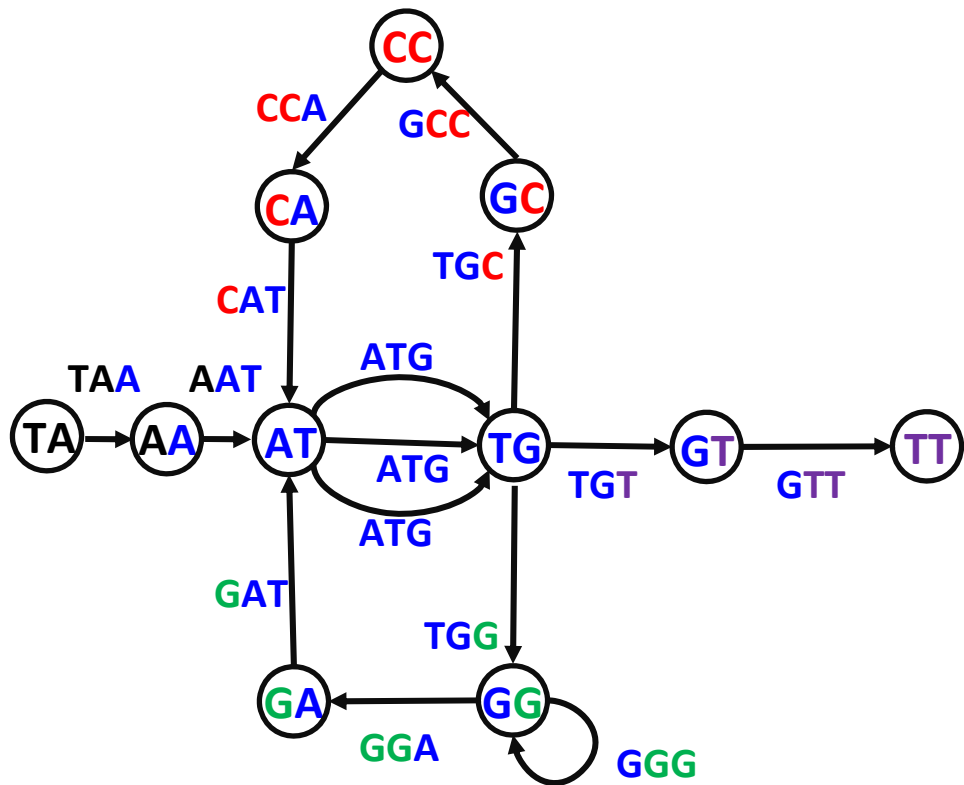




Glueing Identically Labeled Nodes



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.

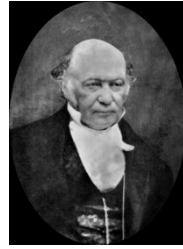
DeBruijn(k -mers)

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

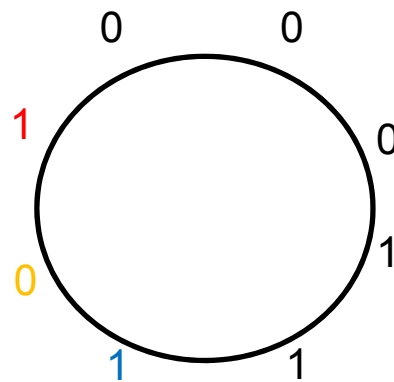


to de Bruijn



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

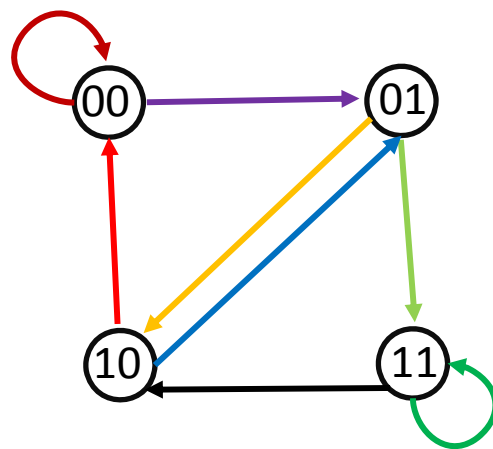


to de Bruijn

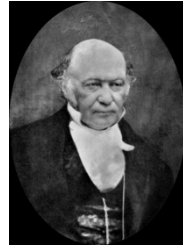


Universal String Problem (Nicolaas de Bruijn, 1946). Find a circular string containing each binary k-mer exactly once.

000 001 010 011 100 101 110 111



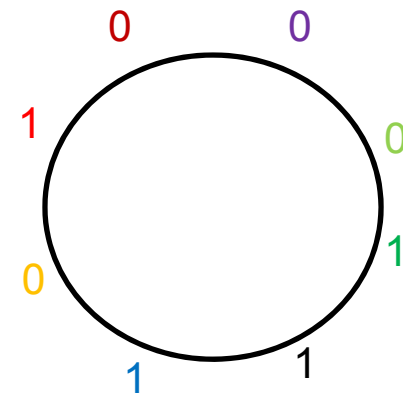
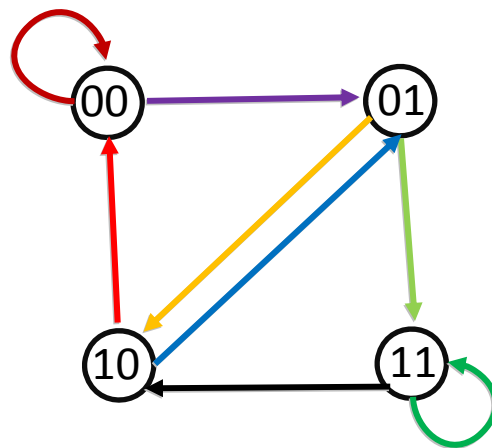
From Hamilton



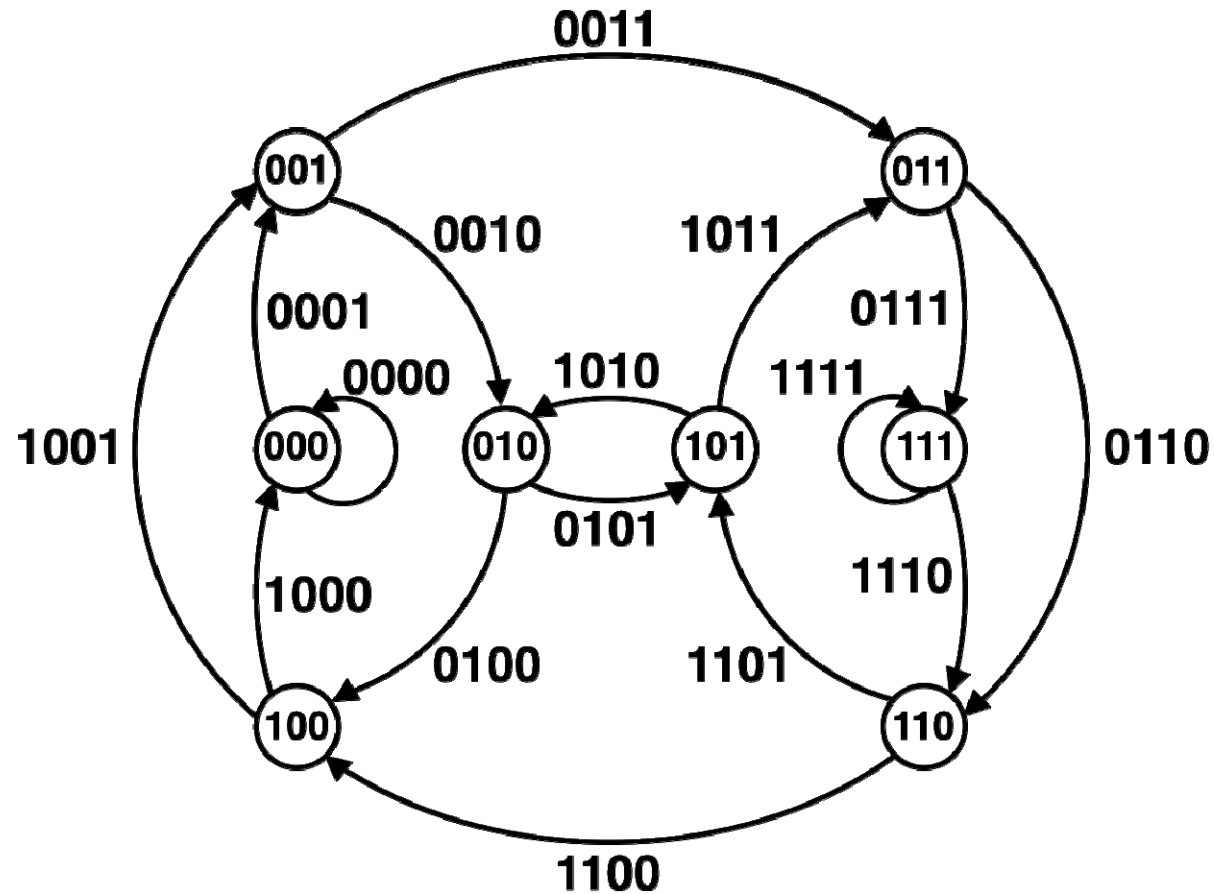
to Euler



to de Bruijn



De Bruijn Graph for 4-Universal String

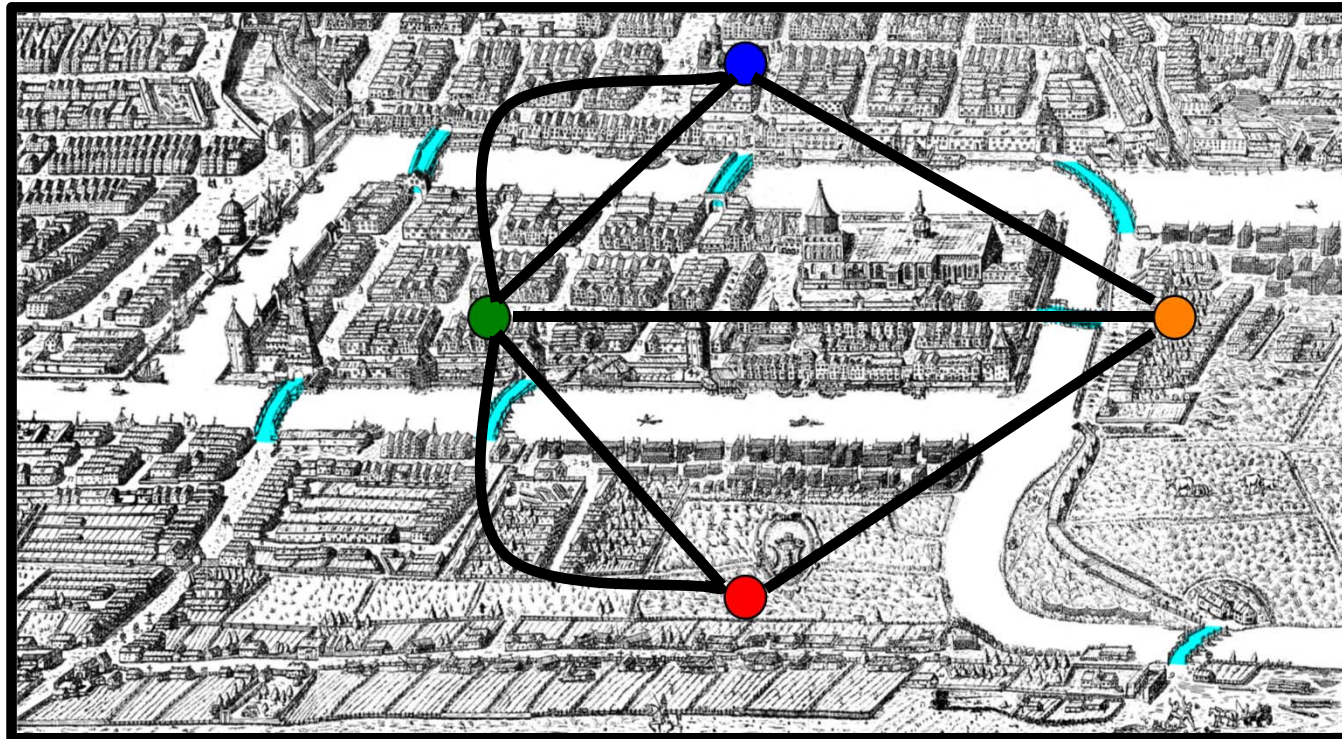


Does it have an Eulerian cycle? If yes, how can we find it?

Eulerian CYCLE Problem

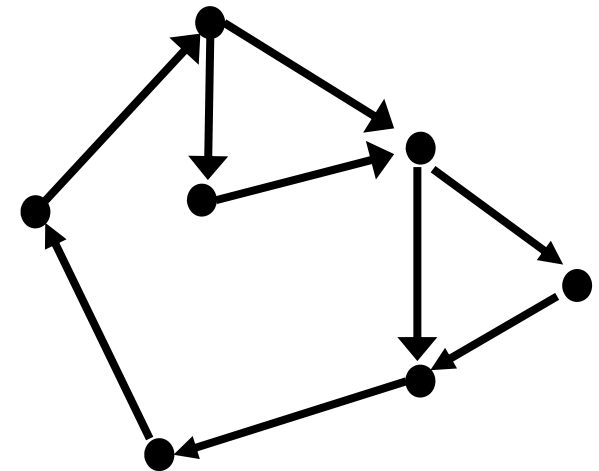
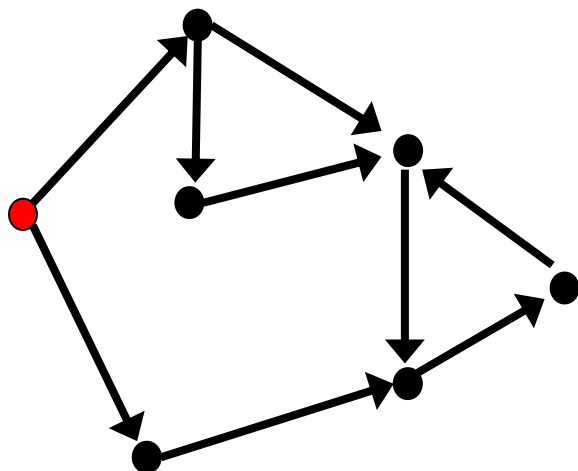
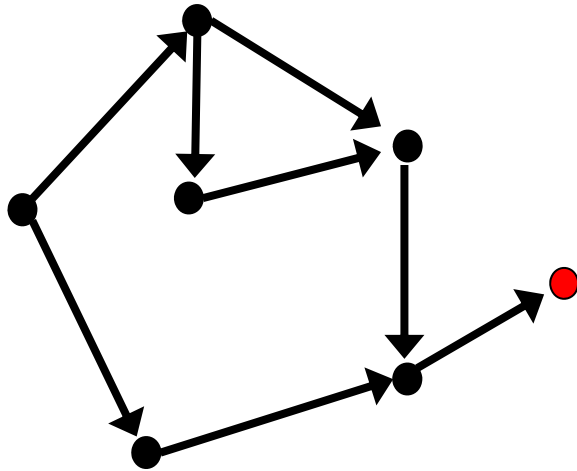
Eulerian CYCLE Problem. Find an Eulerian cycle in a graph.

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



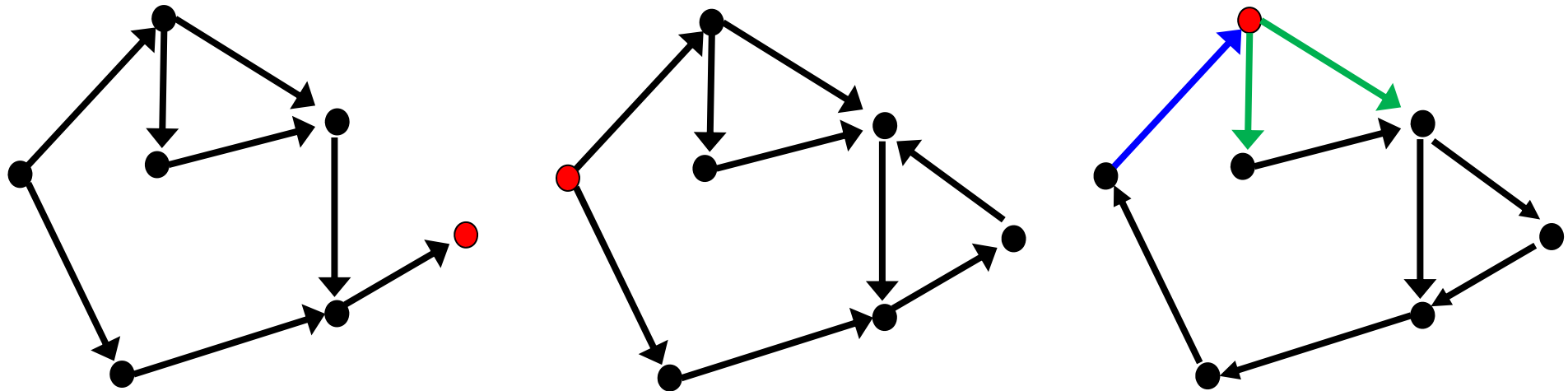
A Graph is **Eulerian** if It Contains an Eulerian Cycle.

Is this graph Eulerian?



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

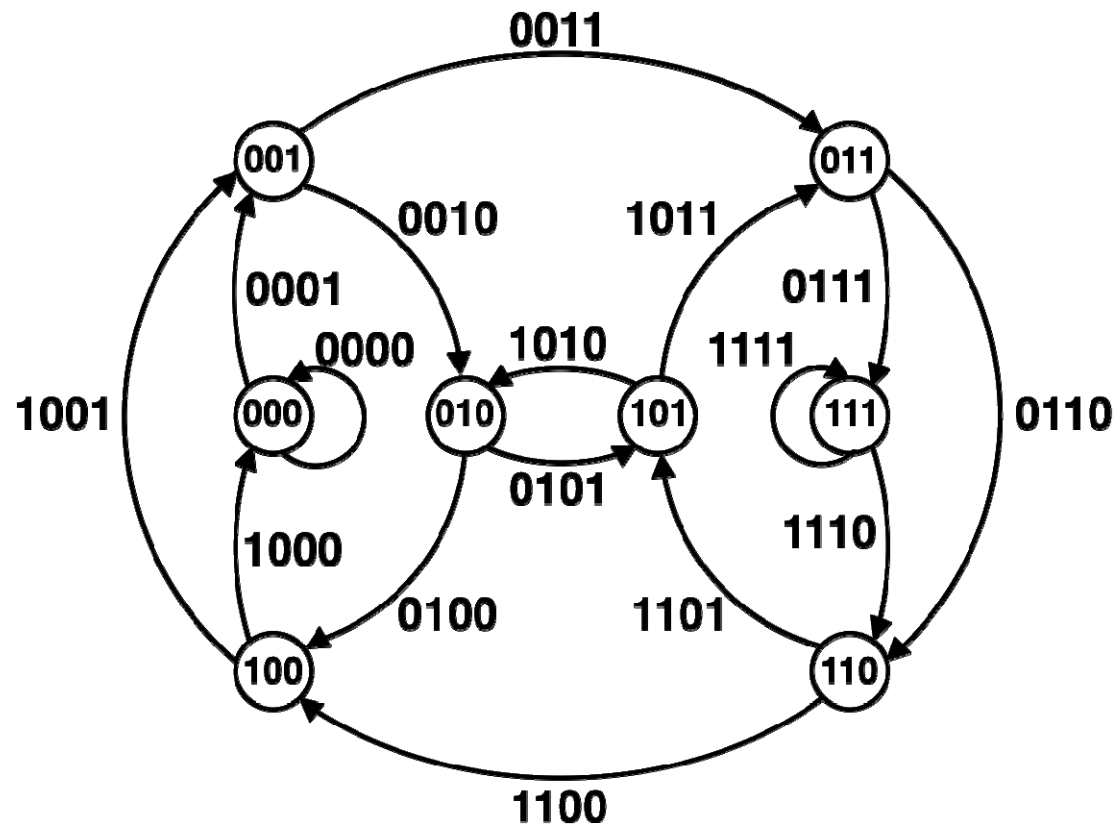
Is this graph Eulerian?



A graph is balanced if **indegree** = **outdegree** for each node

Euler's Theorem

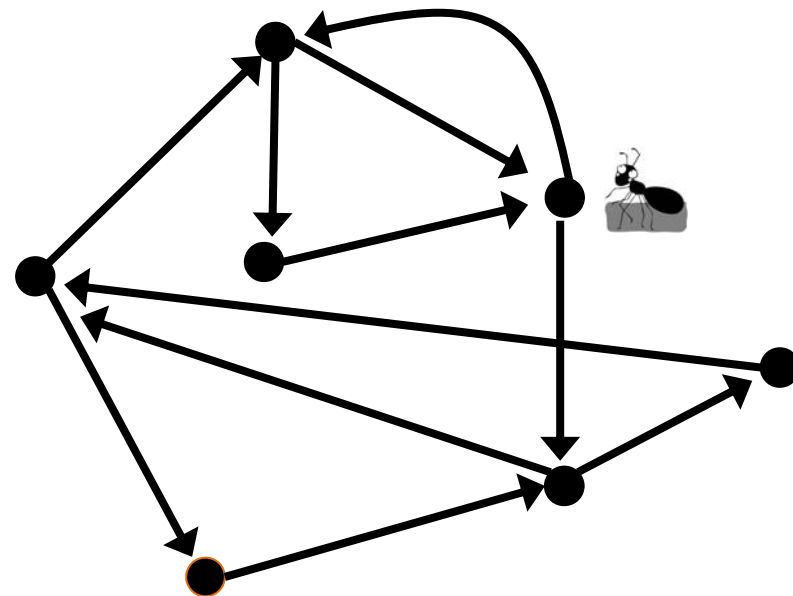
- Every Eulerian graph is balanced
- **Every balanced* graph is Eulerian**



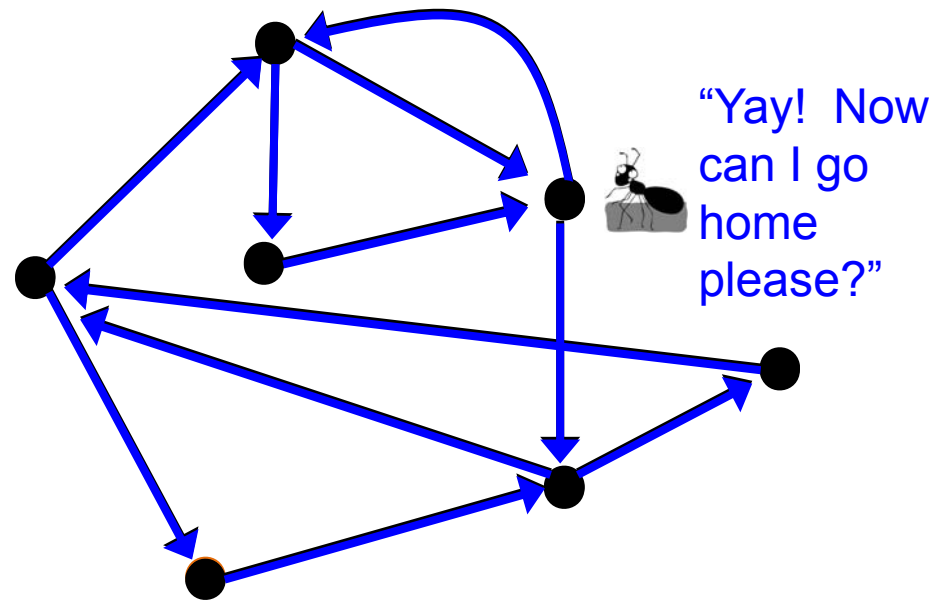
(*) and strongly connected, of course!

Recruiting an Ant to Prove Euler's Theorem

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

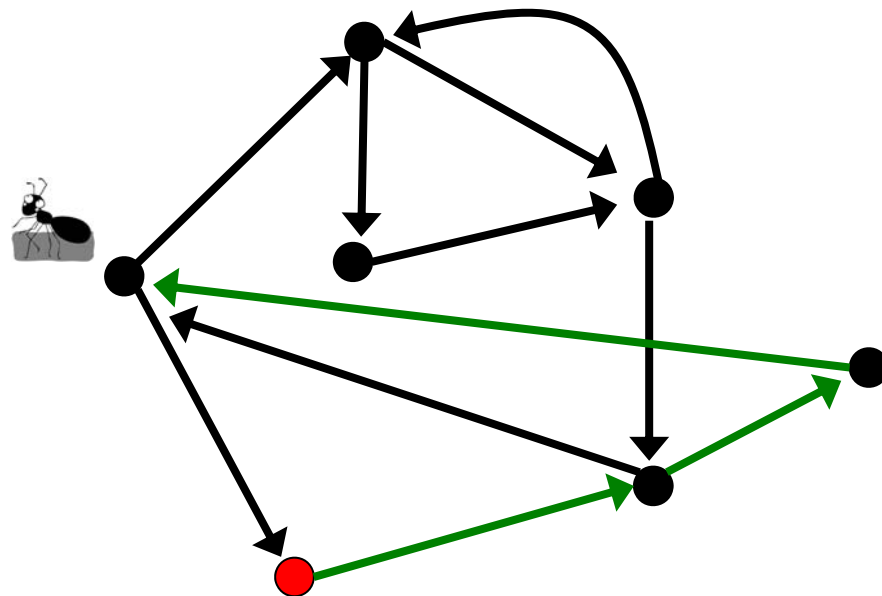


If Ant Was a Genius...



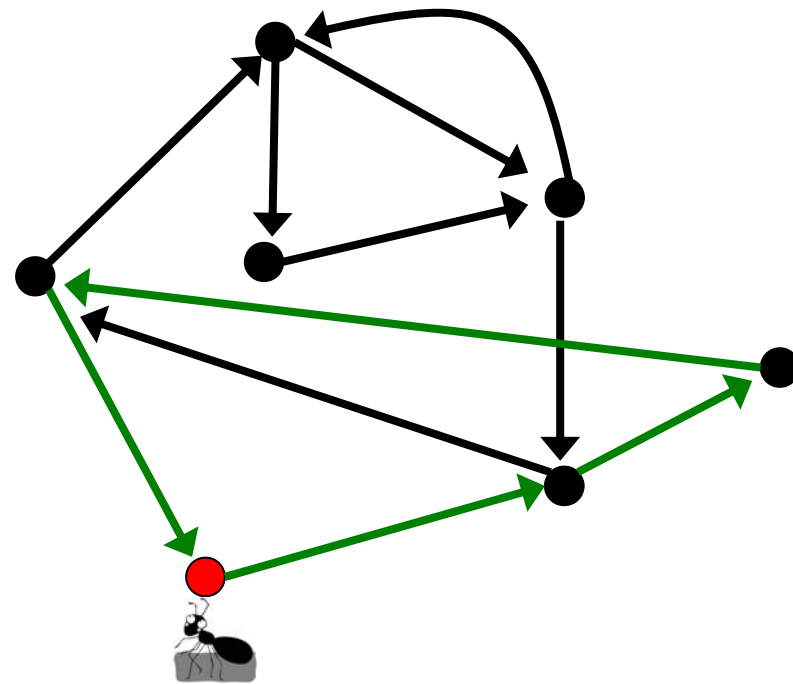
A Less Intelligent Ant Would Randomly Choose a Node and Start Walking...

Can it get stuck? In what node?



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

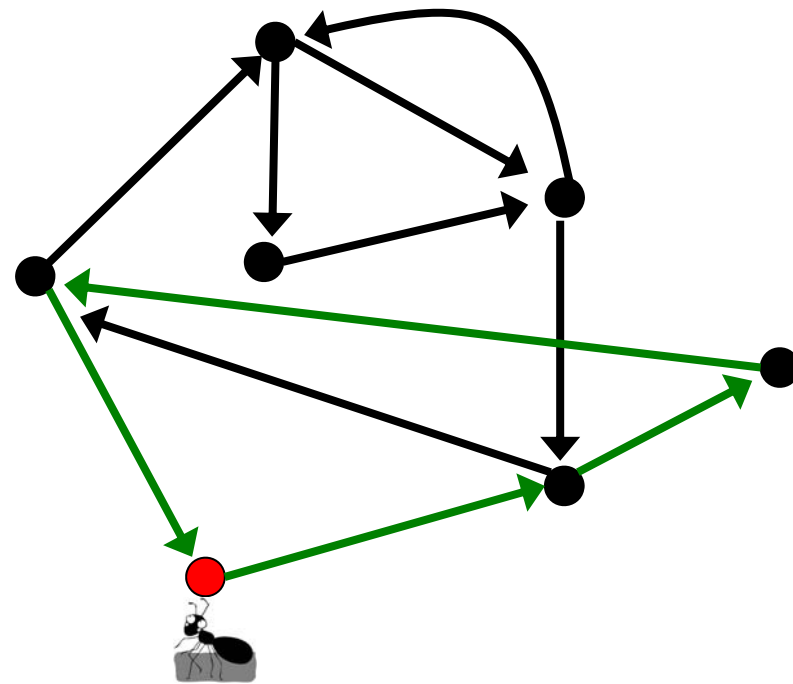
The constructed cycle is not Eulerian. **Can we enlarge it?**



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

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

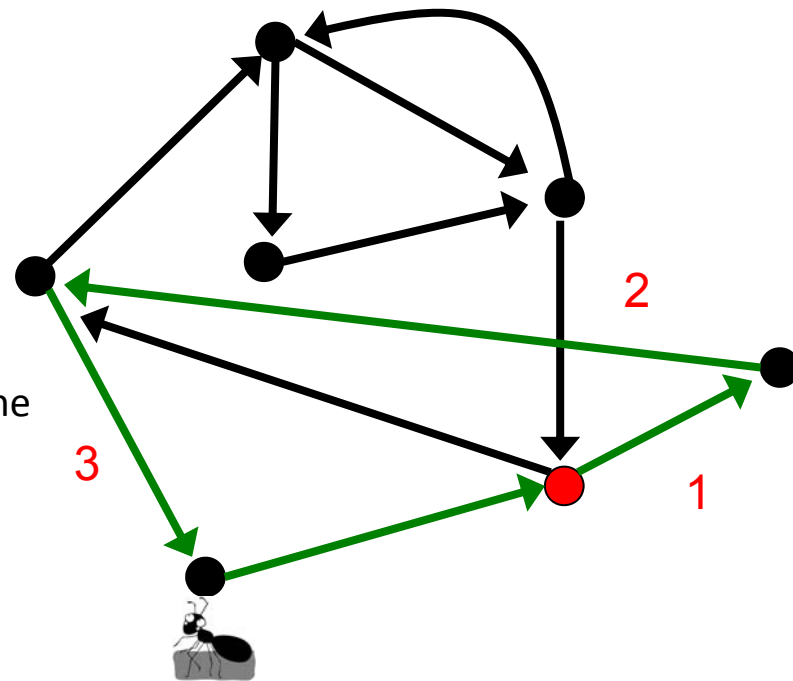
“Why should I start at a different node?
Backtracking? I'm not evolved to walk
backwards! **And what difference does it
make???**”



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.

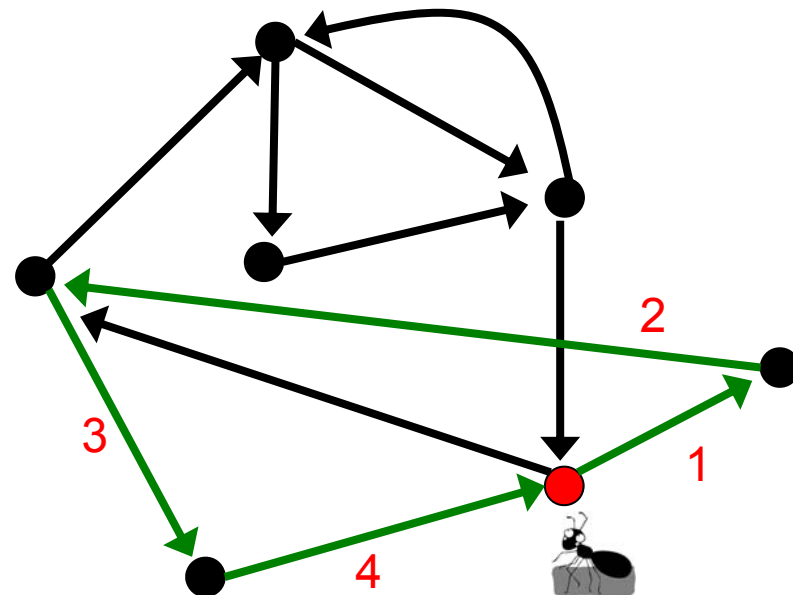
“Why do I have to walk along the same cycle again??? Can I see something new?”



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.

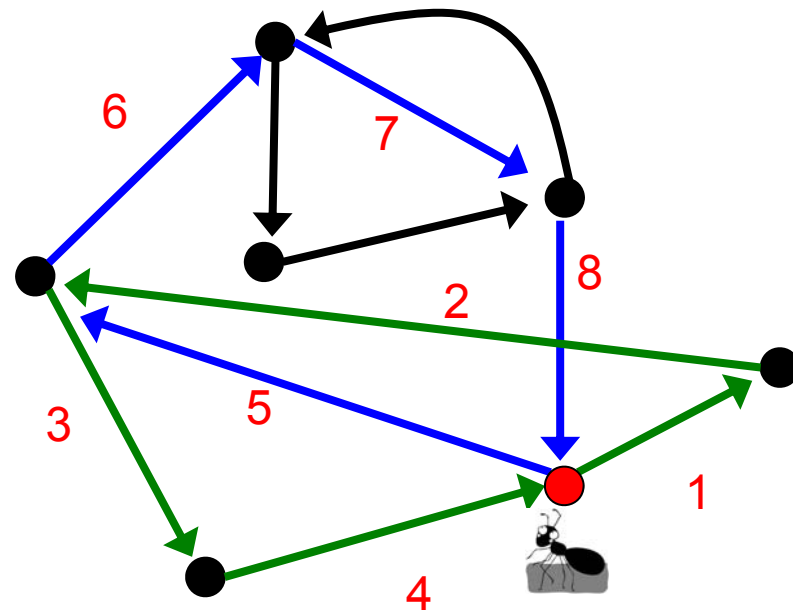
After completing the cycle, start random exploration of still untraversed edges in the graph.



Stuck Again!

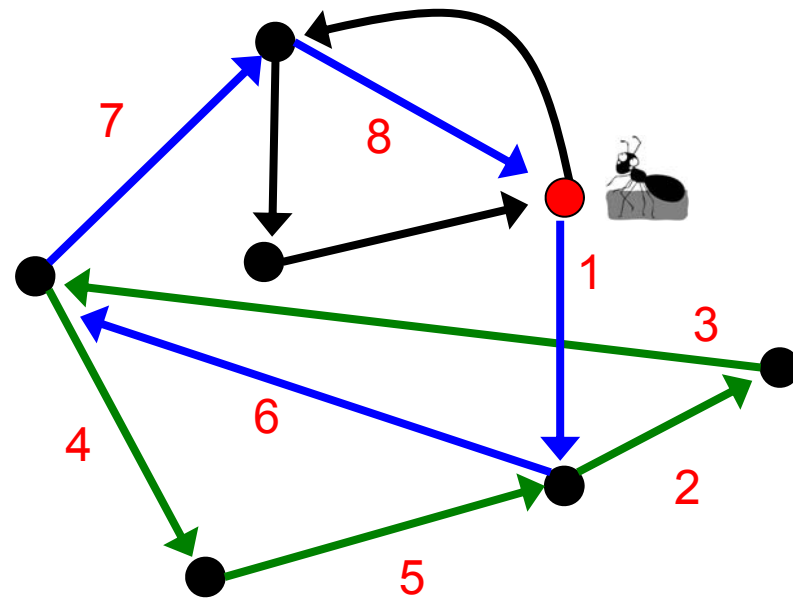
No Eulerian cycle yet... can we enlarge the green-blue cycle?

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(BalancedGraph)

form a *Cycle* by randomly walking in *BalancedGraph* (avoiding already visited edges)

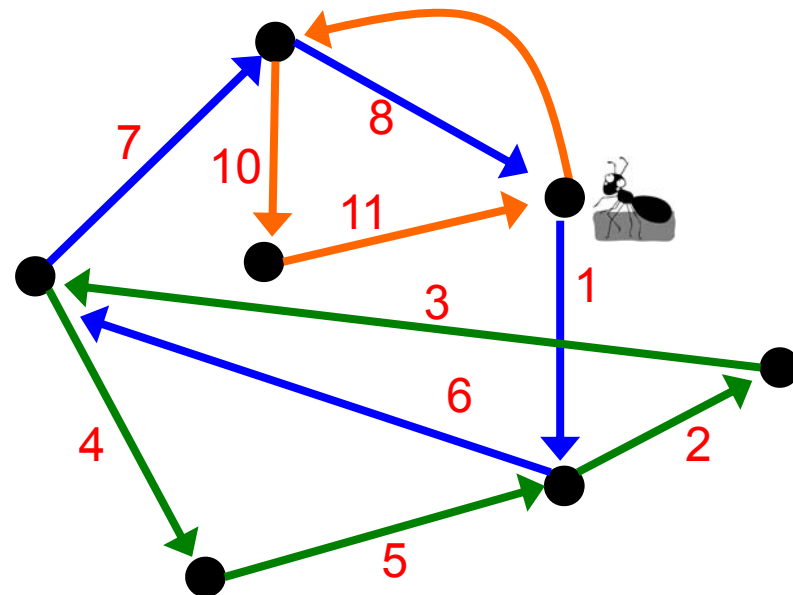
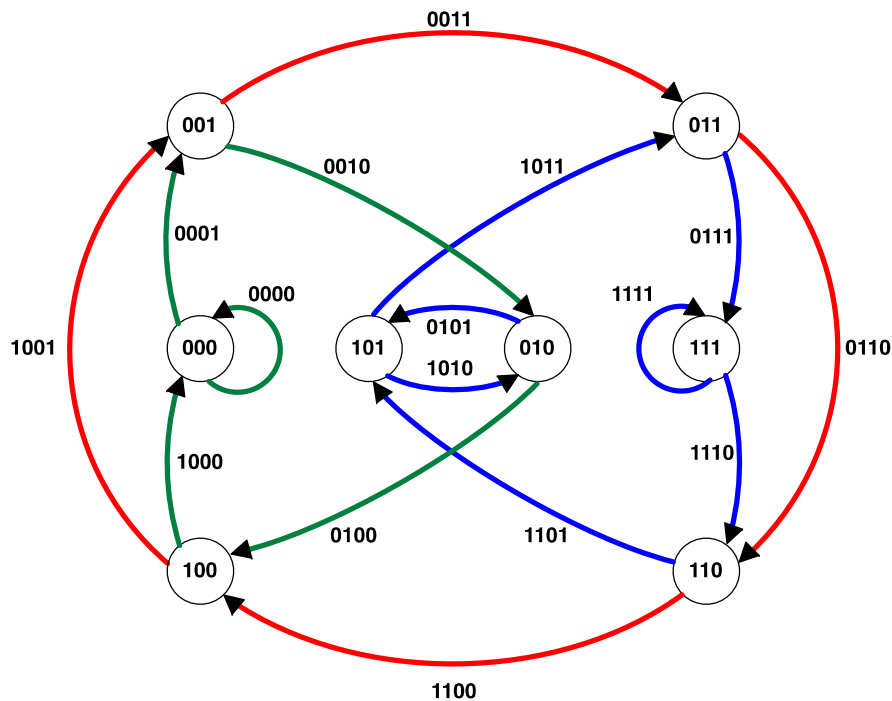
while *Cycle* is not Eulerian

 select a node *newStart* in *Cycle* with still unexplored outgoing edges

 form a *Cycle'* by traversing *Cycle* from *newStart* and randomly walking afterwards

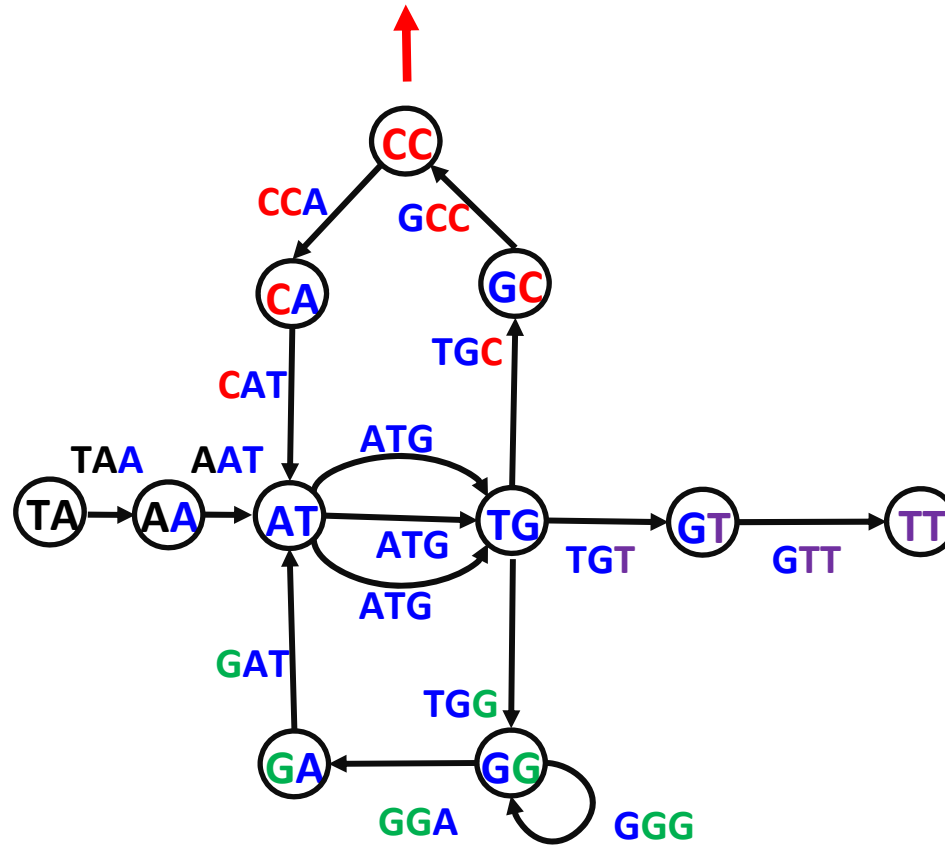
Cycle \leftarrow *Cycle'*

return *Cycle*



From Reads to de Bruijn Graph to Genome

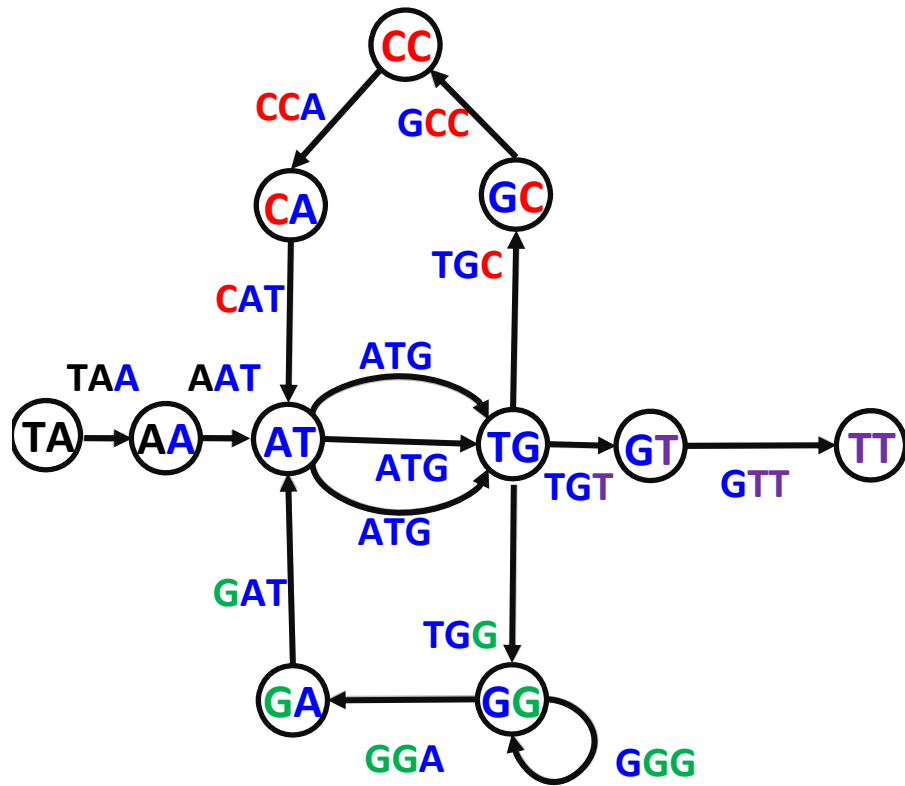
TAATG**CC**ATGGATGTT



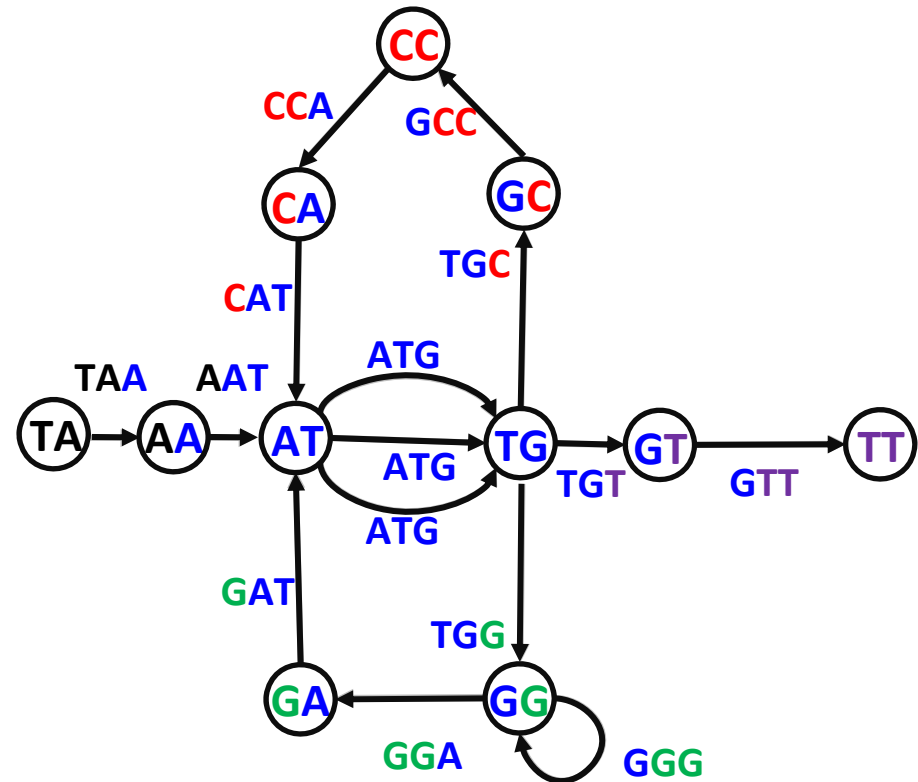
AAT ATG ATG ATG **CAT** CCA GAT **GCC** GGA GGG GTT TAA TGC TGG TGT

Multiple Eulerian Paths

TAATGCCATGGGATGTT

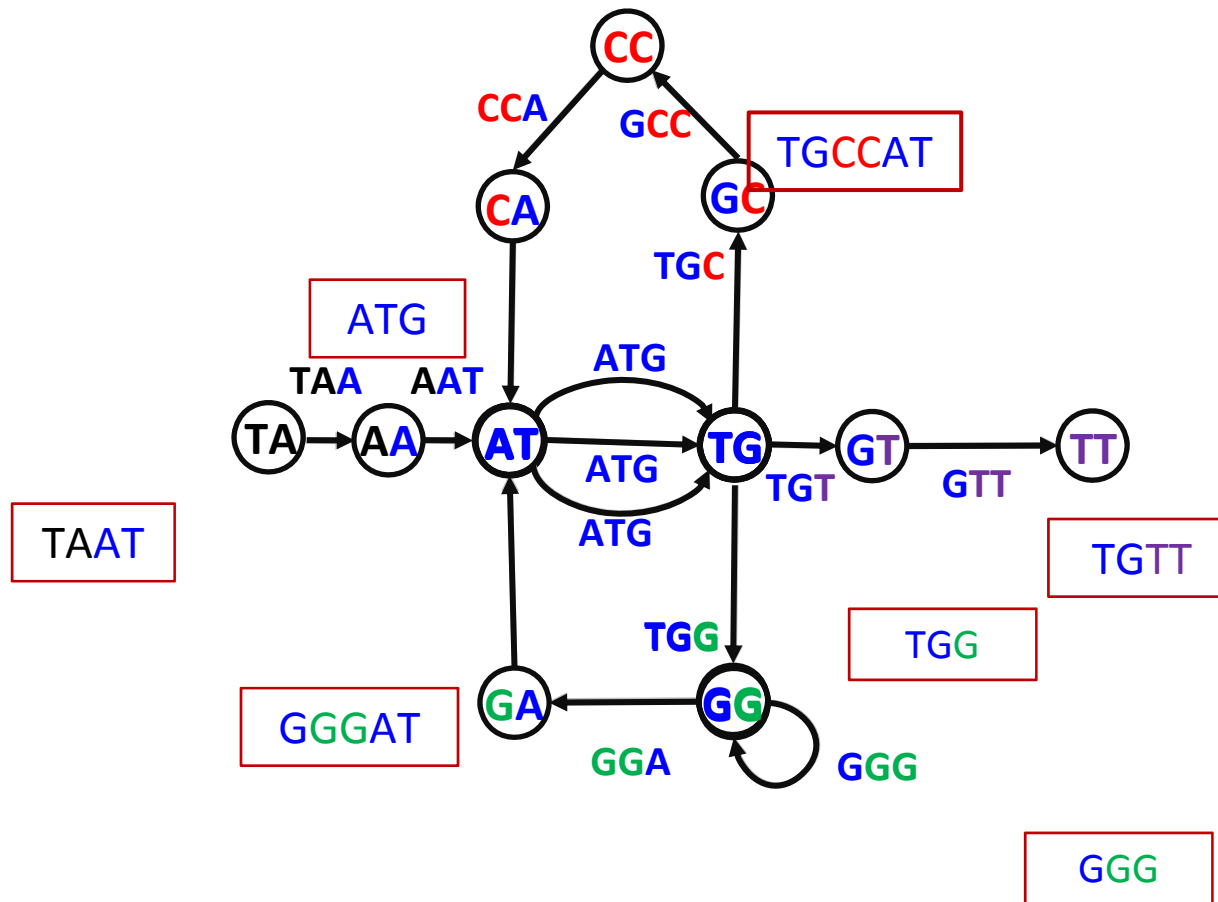


TAATGGGATGCCATGTT



Breaking Genome into Contigs

TAATGCCATGGGATGTT

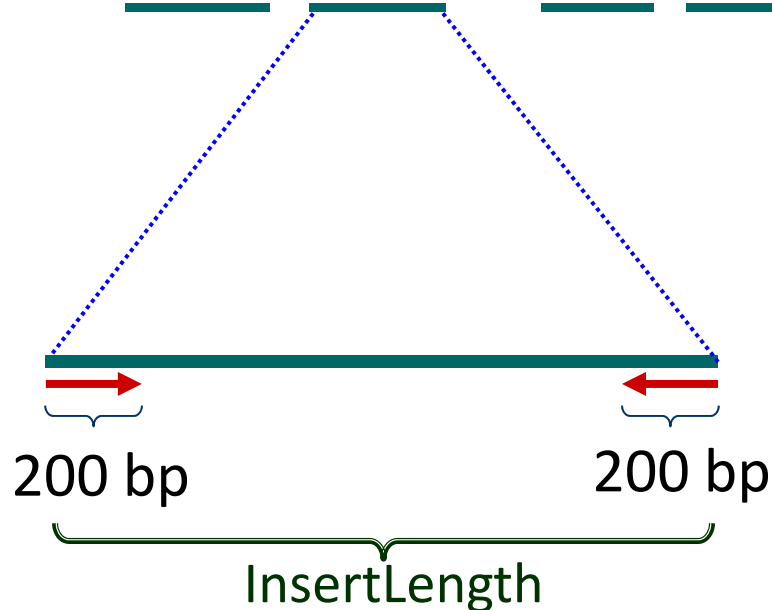


DNA Sequencing with Read-pairs

Multiple identical copies of genome

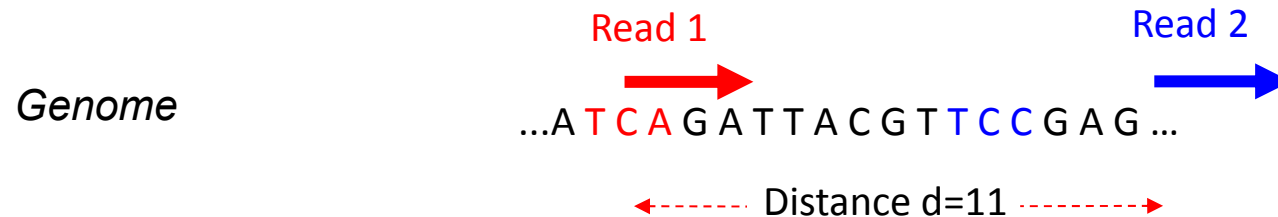


↓ Randomly cut genomes into large equally sized fragments of size InsertLength



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:
(\rightarrow \leftarrow rather than \rightarrow \rightarrow)
2. In reality, the distance d between reads is measured with errors.

What is PairedComposition(**TAATGCCATGGATGTT**)?

```
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(TAATGCCATGGATGTT)

```
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

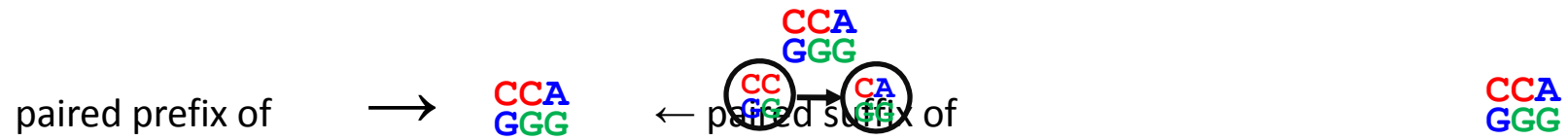
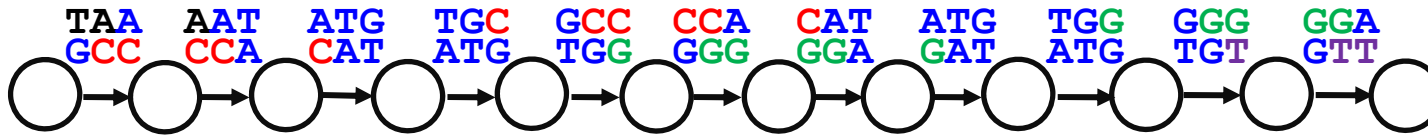
String Reconstruction from Read-Pairs Problem. Reconstruct a string from its paired k -mers.

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

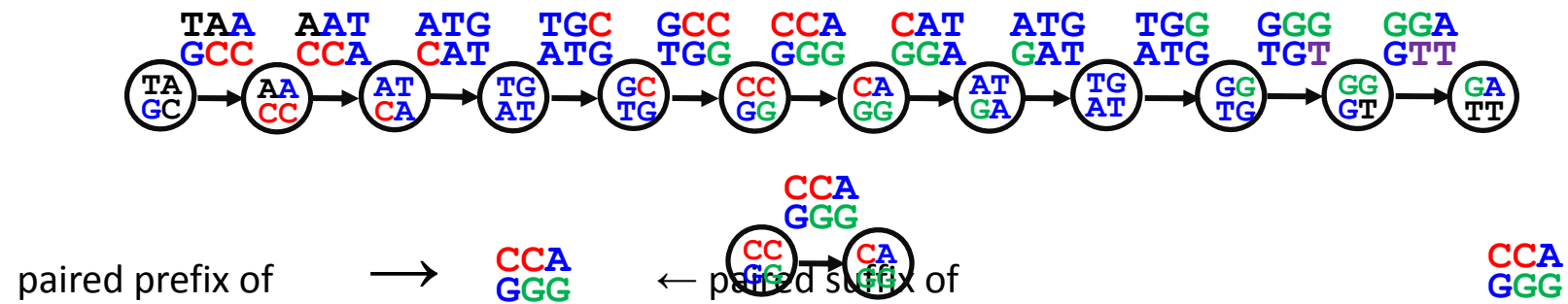
How Would de Bruijn Assemble Paired k -mers?

Representing Genome TAATGCCATGGATGTT as a Path

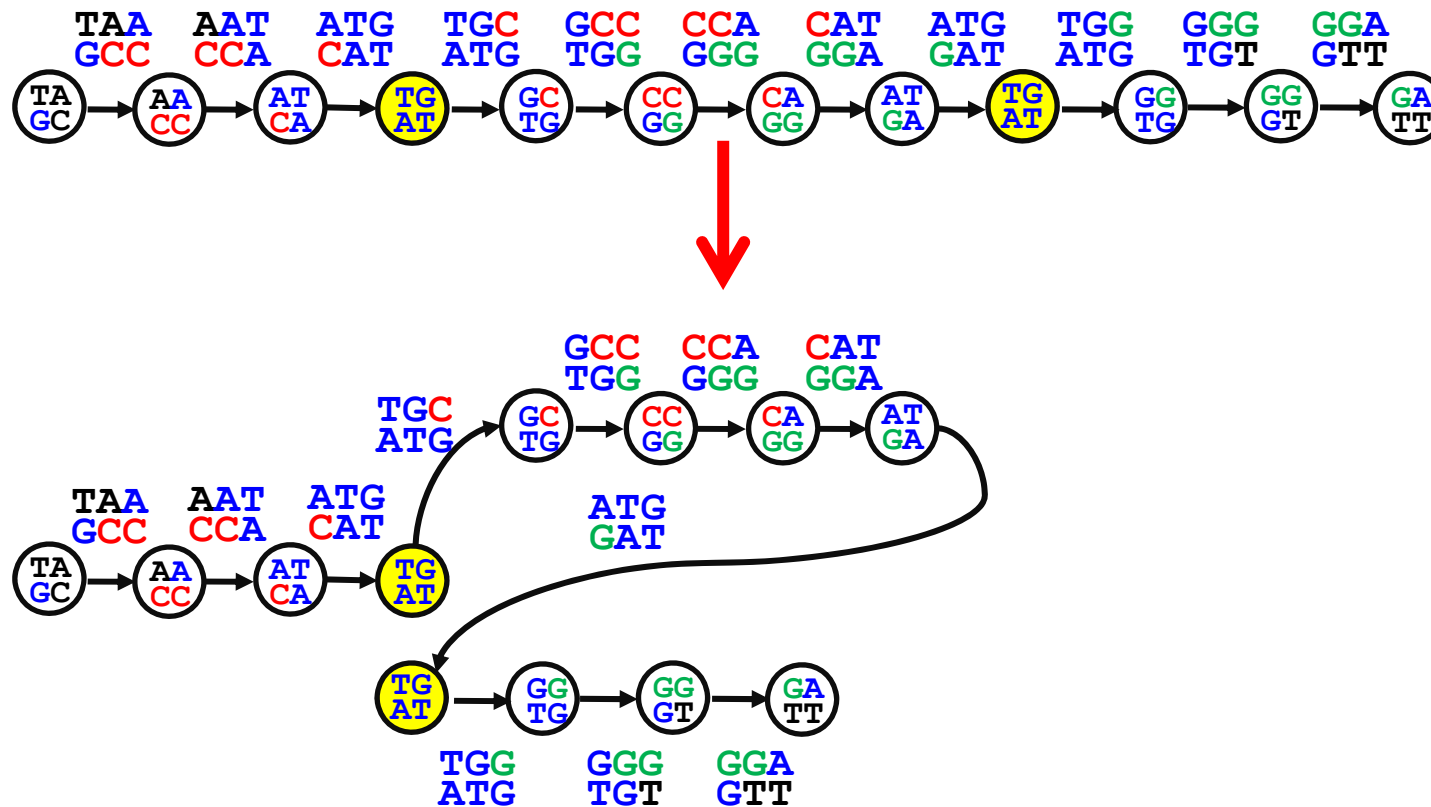
TAA GCC
 AAT CCA
 ATG CAT
 TGC ATG
 GCC TGG
 CCA GGG
 CAT GGA
 ATG GAT
 TGG ATG
 GGG TGT
 GGA GTT



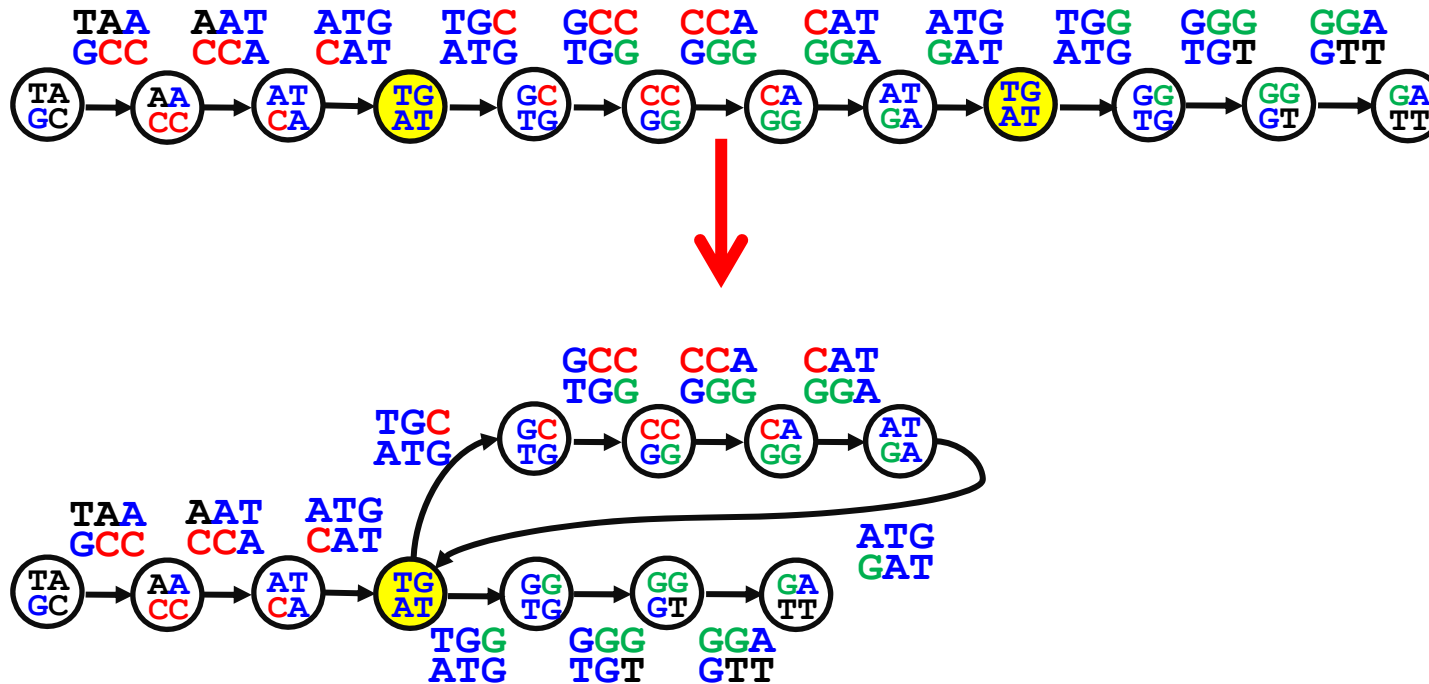
Labeling Nodes by Paired Prefixes and Suffixes



Glue nodes with identical labels

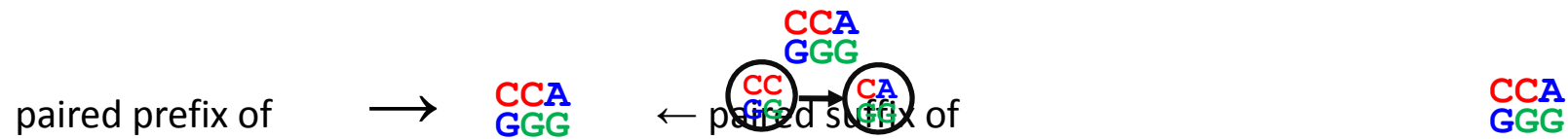
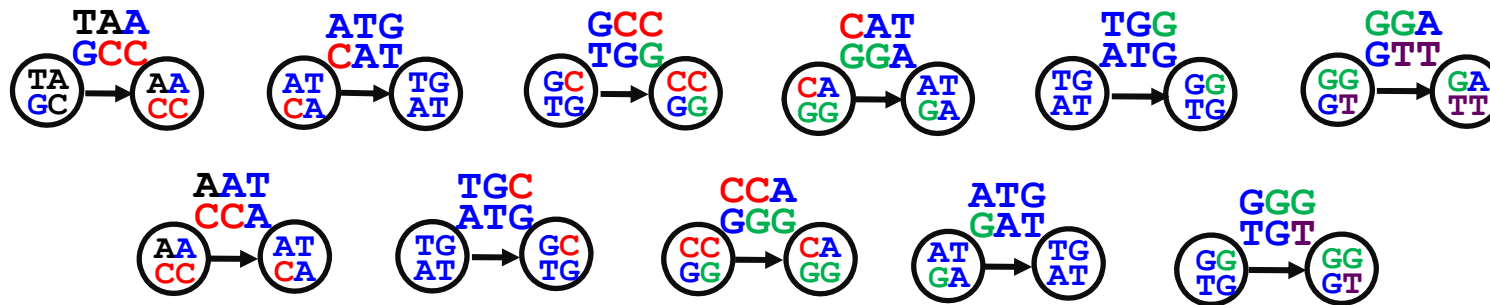


Glue nodes with identical labels

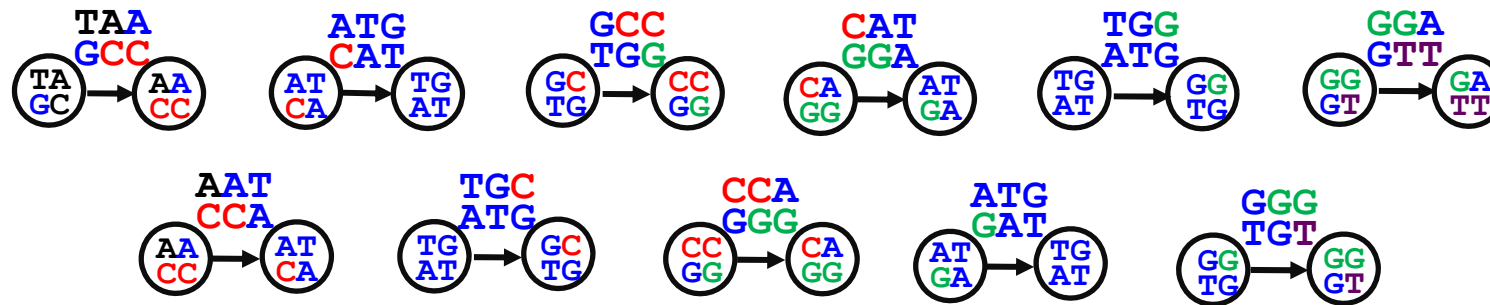


Paired de Bruijn Graph from the Genome

Constructing Paired de Bruijn Graph

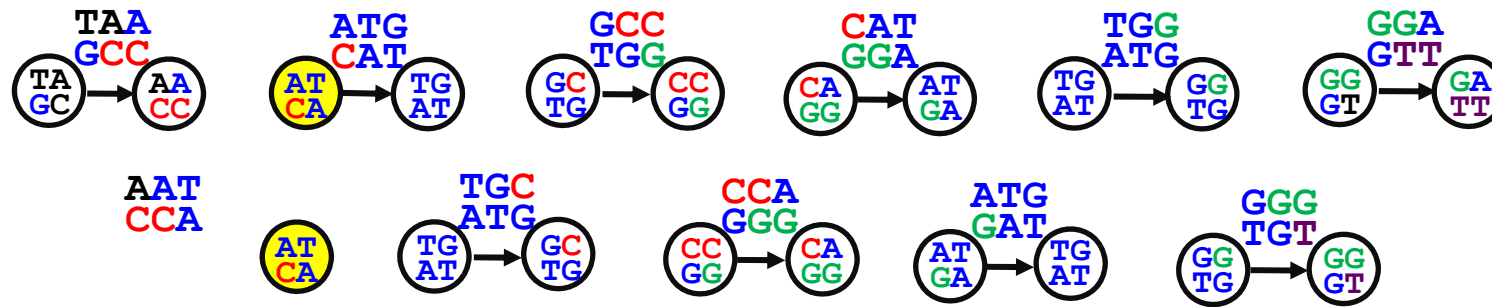


Constructing Paired de Bruijn Graph

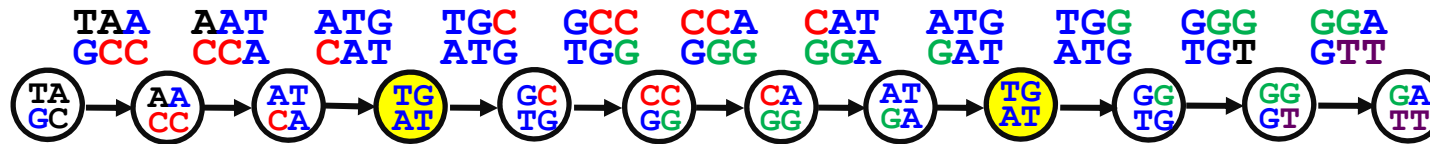


- **Paired de Bruijn graph for a collection of paired k -mers:**
 - Represent every paired k -mer as an edge between its paired prefix and paired suffix.
 - Glue **ALL** nodes with identical labels.

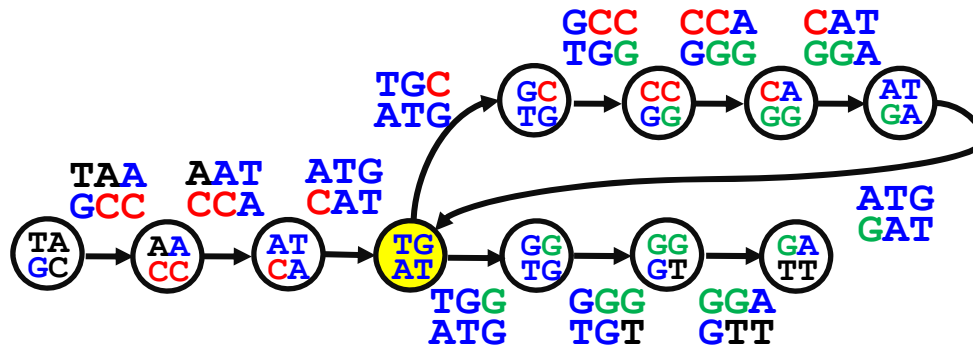
Constructing Paired de Bruijn Graph



We Are Not Done with Glueing 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?

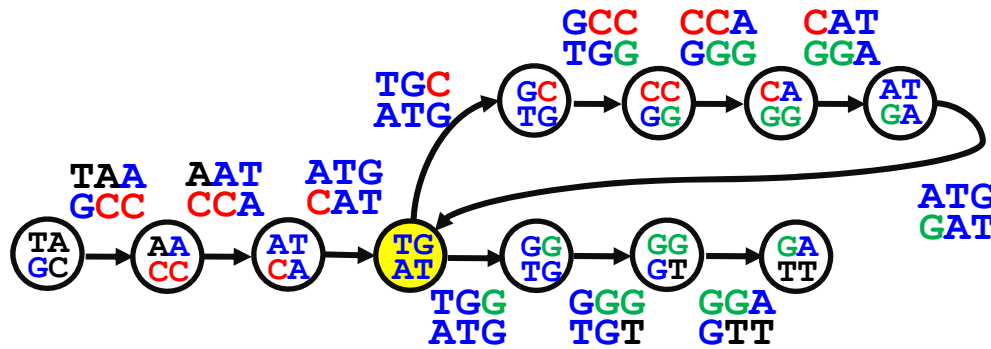
Unique genome reconstruction

Multiple genome reconstructions

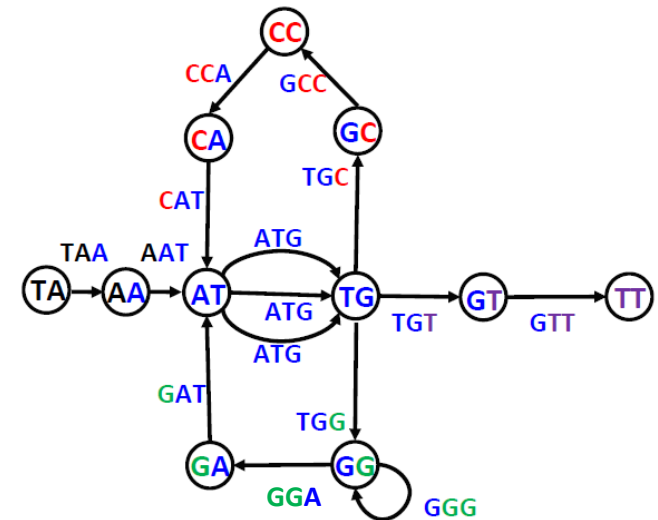
TAATGCCATGGGATGTT

TAATGCCATGGGATGTT

TAATGGGATGCCATGTT



Paired de Bruijn Graph



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
    gtatggaaa
      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

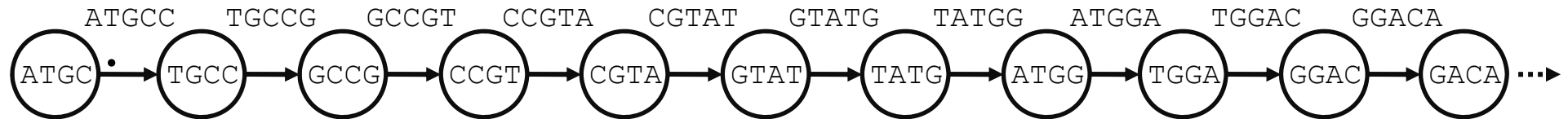
atgccgtatggacaacgact
atgccgtatg
gccgtatgga
gtatggacaa
gacaacgact
cgtaCggaca

Erroneous read
(change of t into C)

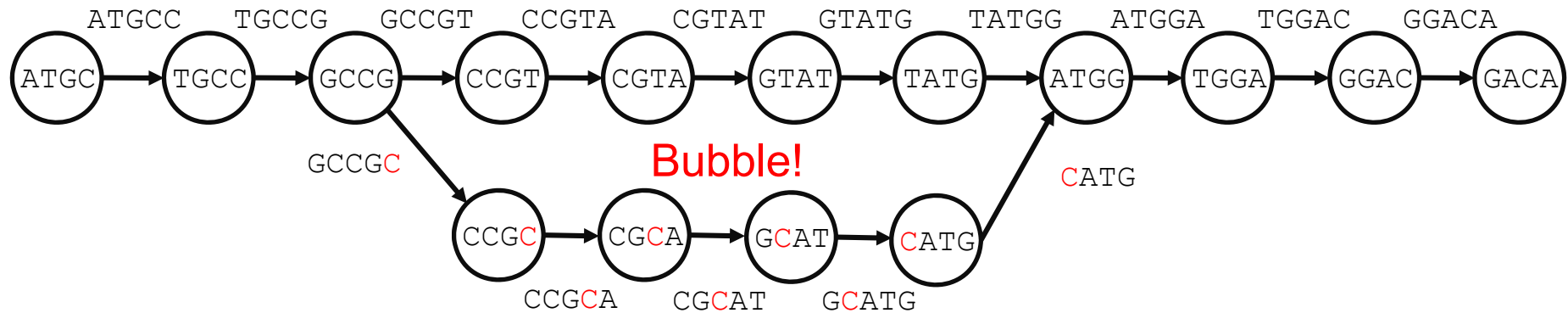
atgccgtatggacaacgact
atgcc
tgccg
gccgt
ccgta
cgtat
gtatg
tatgg
atgga
tggac
ggaca
gacaa
acaac
caacg
aacga
acgac
cgact
cgtaC
gtaCg
taCgg
aCgga
Cggac

De Bruijn Graph of ATGGCGTGCAATG...

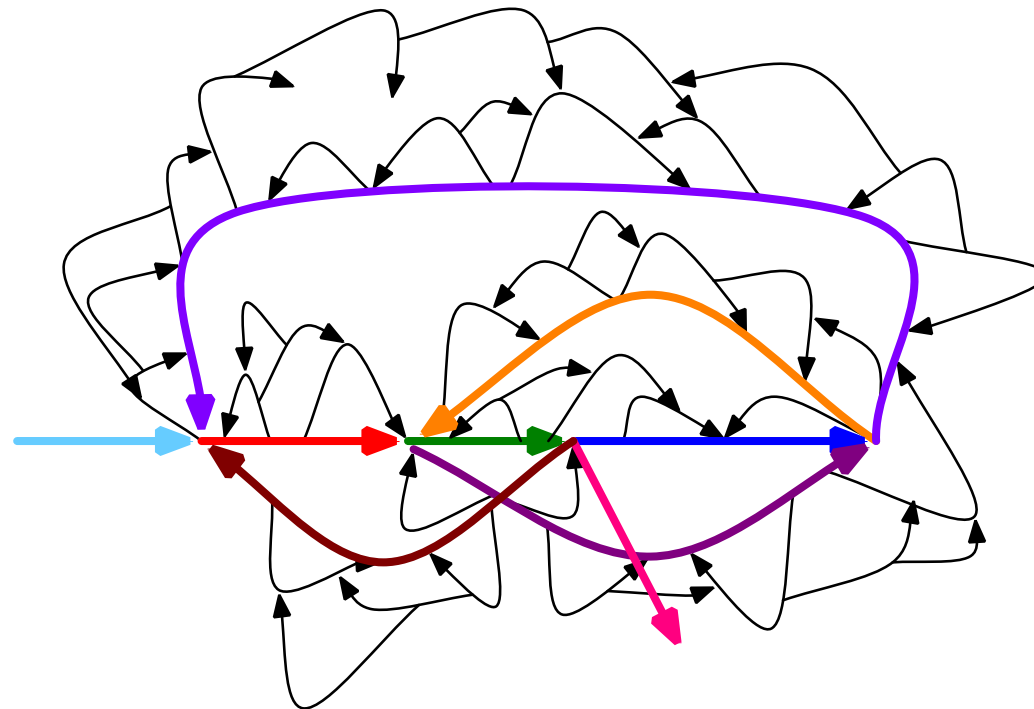
Constructed from Error-Free Reads



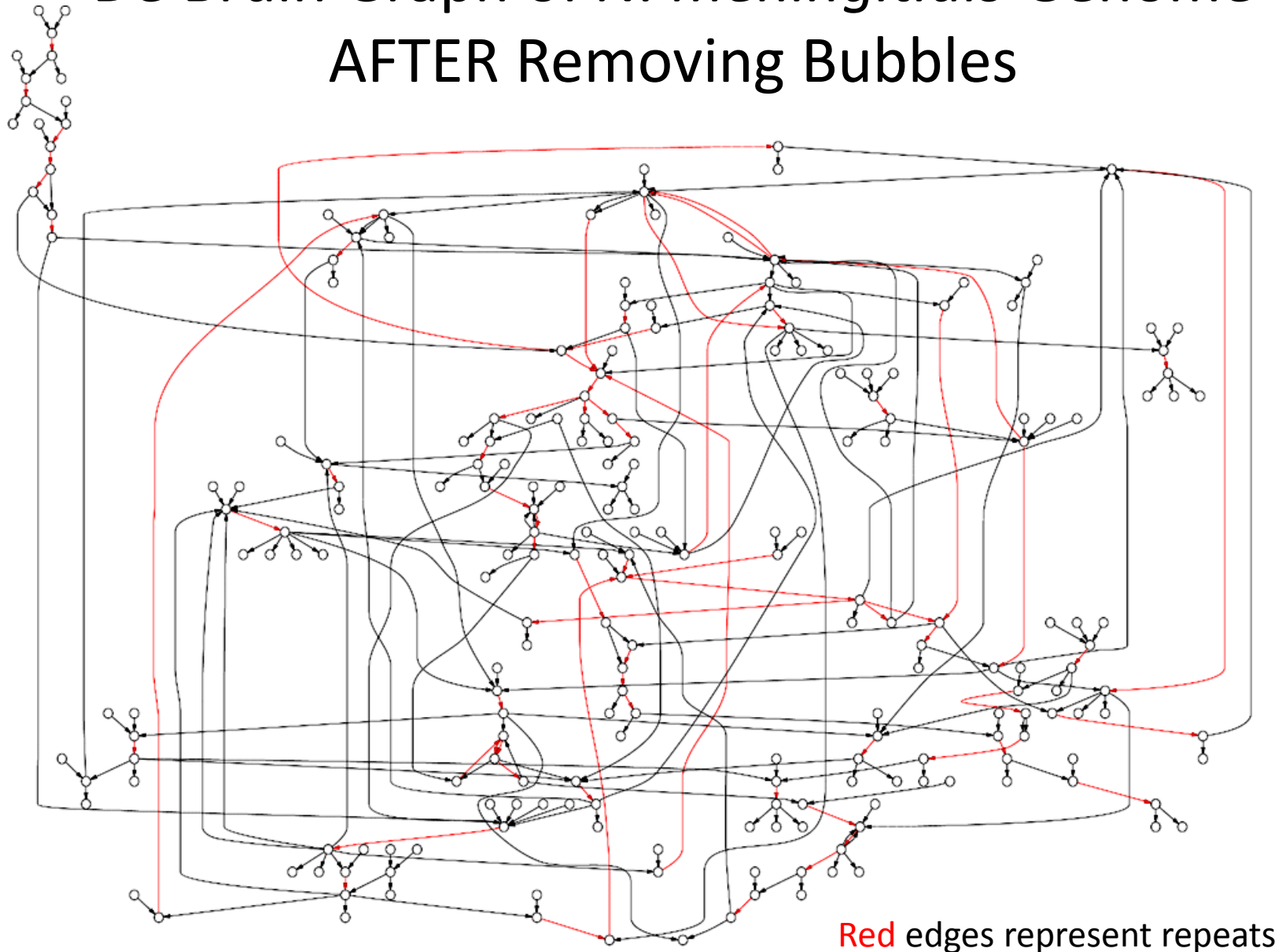
Errors in Reads Lead to **Bubbles** in the De Bruijn Graph



Bubble Explosion...Where Are the Correct Edges of the de Bruijn Graph?



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



Current work

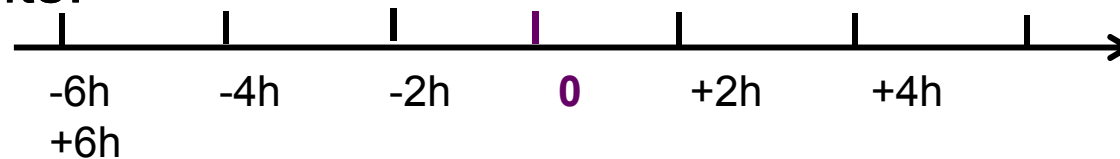
- Extension to multiple samples using coloured de Bruijn graphs
 - <https://doi.org/10.1038/ng.1028>
 - Colour nodes according to sample
 - Bubbles represent biological variation
- Succinct coloured de Bruijn graphs with reduced memory footprint
 - <https://doi.org/10.1093/bioinformatics/btx067>

Clustering Algorithms Outline

- 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

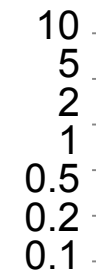
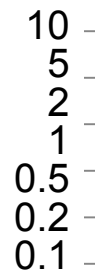
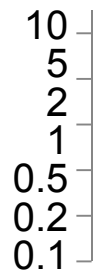
Measuring 3 Genes at 7 Checkpoints

Measure expression of various yeast genes at 7 checkpoints:



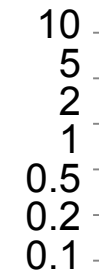
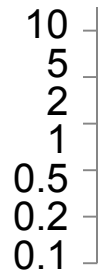
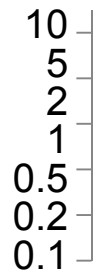
	-6h	-4h	-2h	0	+2h	+4h	+6h
YLR258W	1.1	1.4	1.4	3.7	4.0	10.0	5.9
YPL012W	1.1	0.8	0.9	0.4	0.3	0.1	0.1
YPR055W	1.1	1.1	1.1	1.1	1.1	1.1	1.1

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

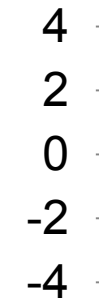
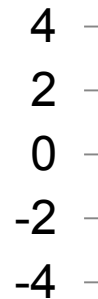
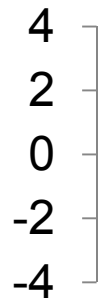


Switching to Logarithms of Expression Levels

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



↓ taking logarithms (base-2)

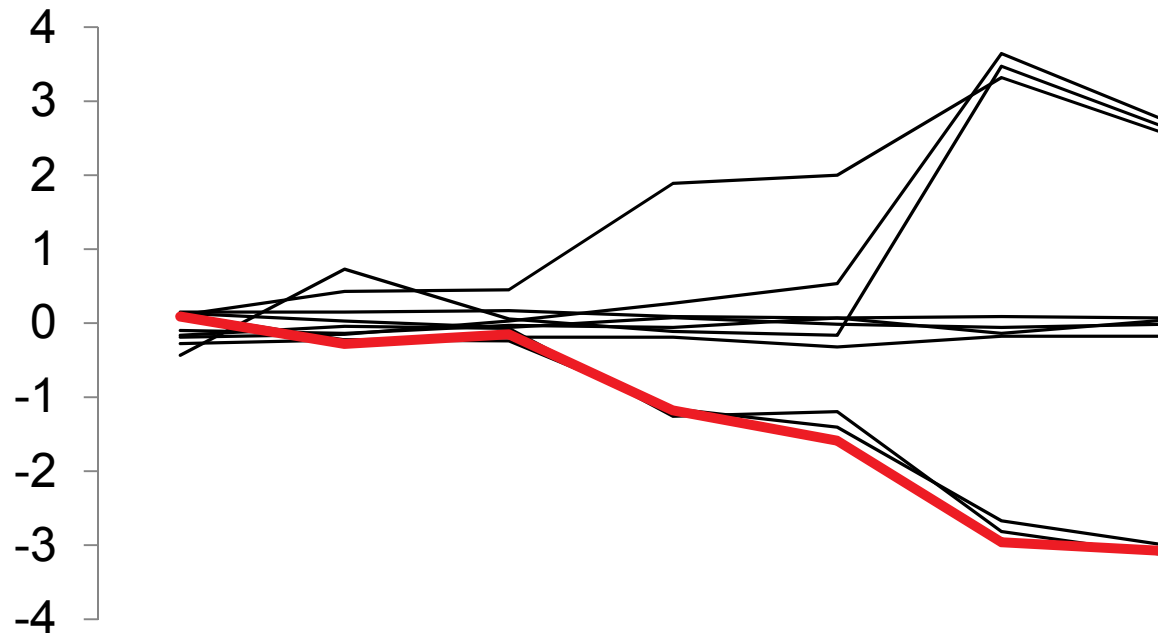


YLR258W	0.1	0.4	0.5	1.9	2.0	3.3	2.6
YPL012W	0.1	-0.3	-0.2	-1.2	-1.6	-3.0	-3.1
YPR055W	0.2	0.2	0.2	0.1	0.1	0.1	0.1

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

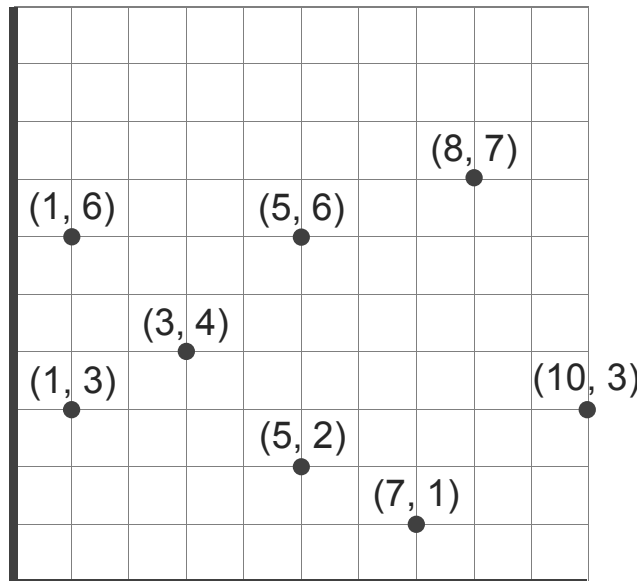
Genes as Points in Multidimensional Space

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

$n \times m$
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.



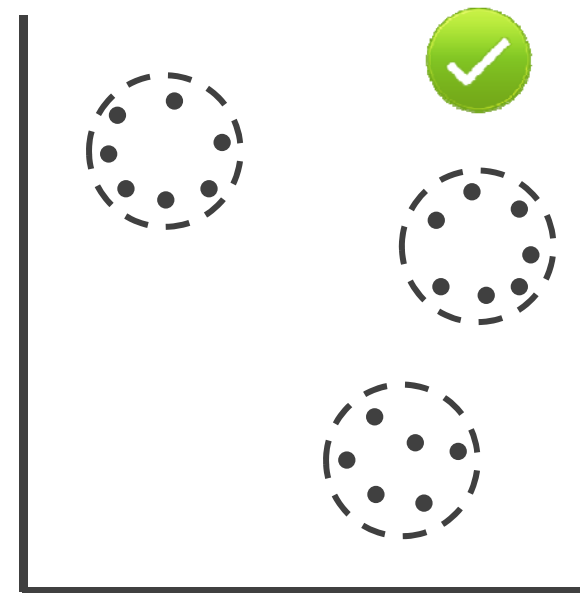
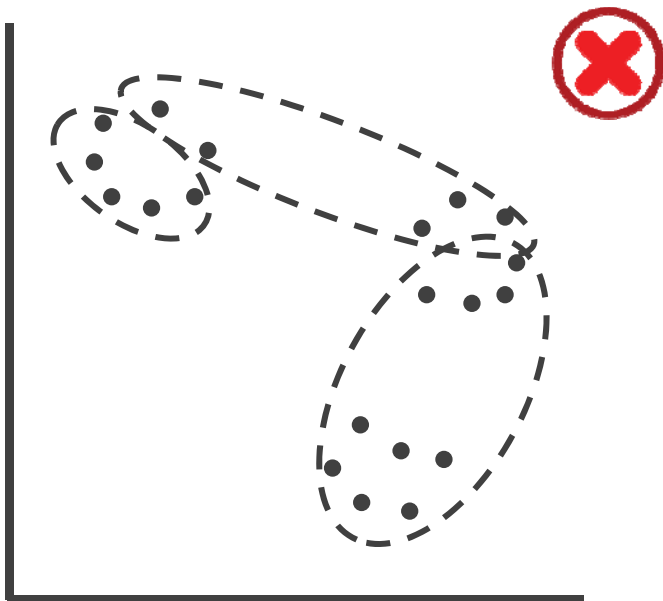
But how did scientists discover these 70 human genes?

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 $> \Delta$

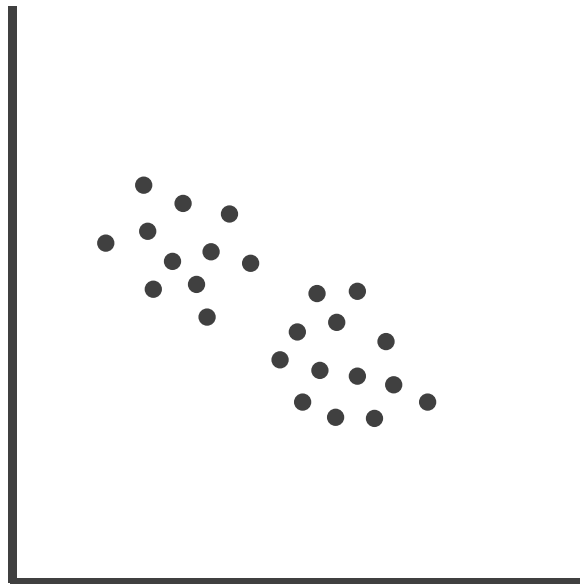


Clustering Problem

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

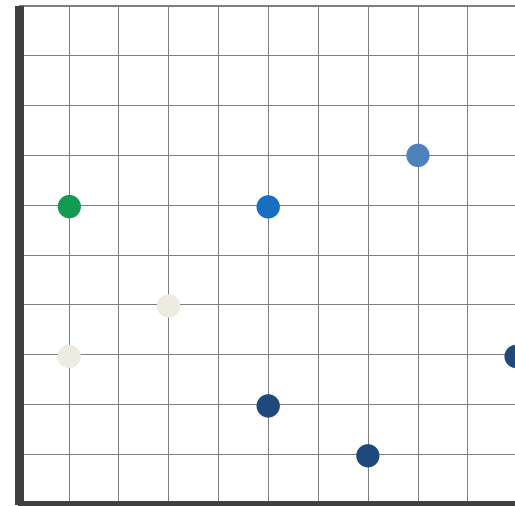
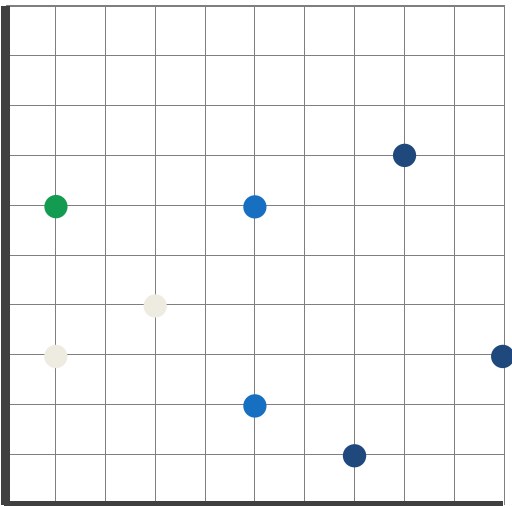
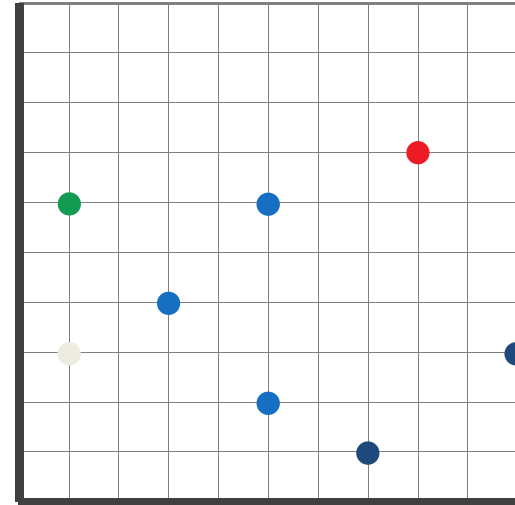
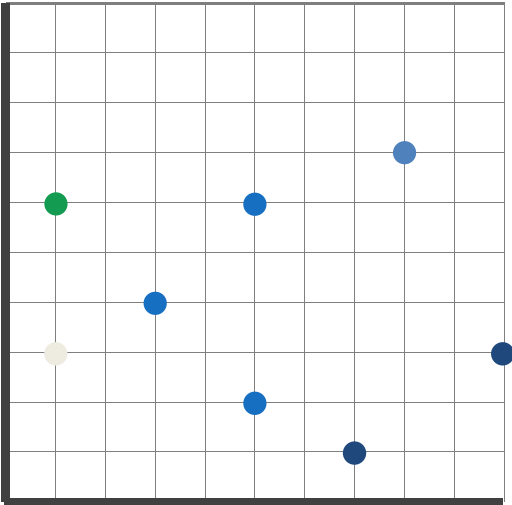
- **Input:** A collection of n vectors and an integer k .
- **Output:** Partition of n vectors into k disjoint clusters satisfying the Good Clustering

Principle.



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

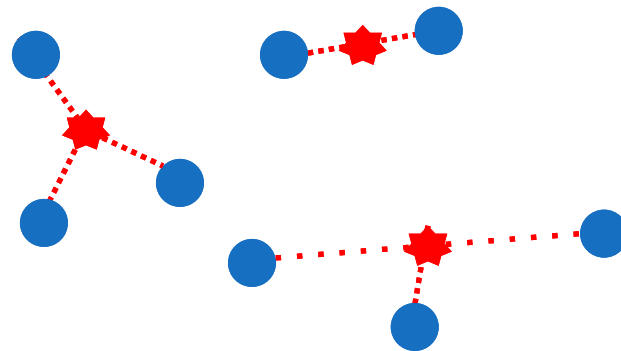
What is the “best” partition into three clusters?



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*.

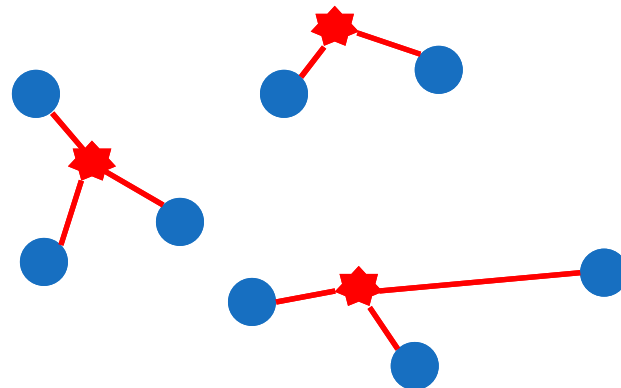


Clustering as Finding Centers

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* .

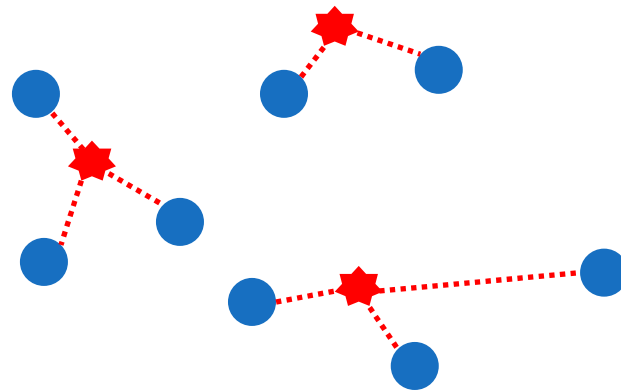
What is the “distance” from *Centers* to *Data*?



Distance from a *Single DataPoint* to *Centers*

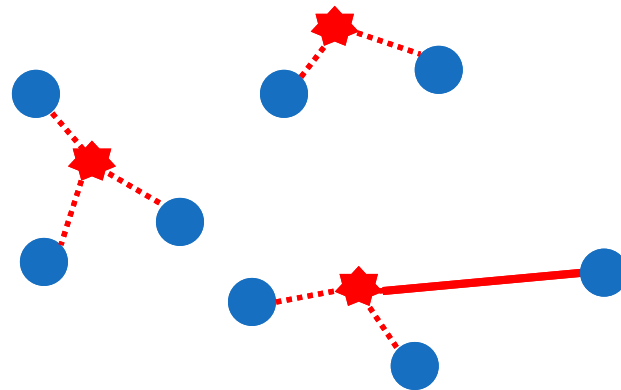
The distance from *DataPoint* in *Data* to *Centers* is the distance from *DataPoint* to the closest center:

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



Distance from *Data* to *Centers*

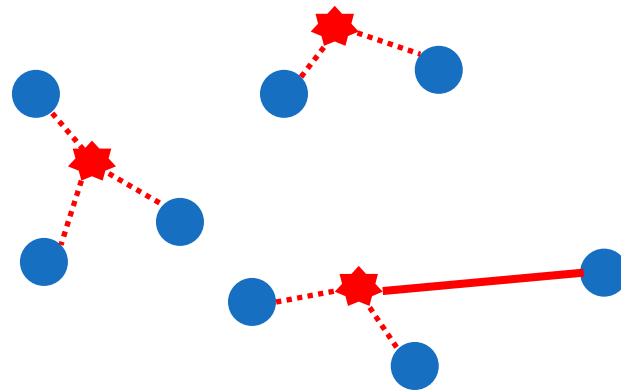
$$\text{MaxDistance}(\text{Data}, \text{Centers}) = \max_{\text{all points } \text{DataPoint} \text{ from } \text{Data}} d(\text{DataPoint}, \text{Centers})$$



k -Center Clustering Problem

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

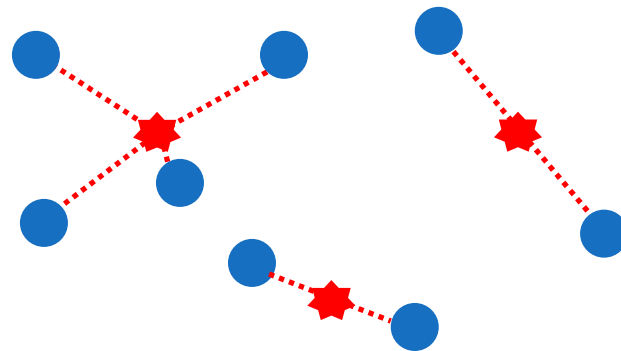
- **Input:** A set of points $Data$ and an integer k .
- **Output:** A set of k points $Centers$ that minimizes $MaxDistance(DataPoints, Centers)$ over all possible choices of $Centers$.



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

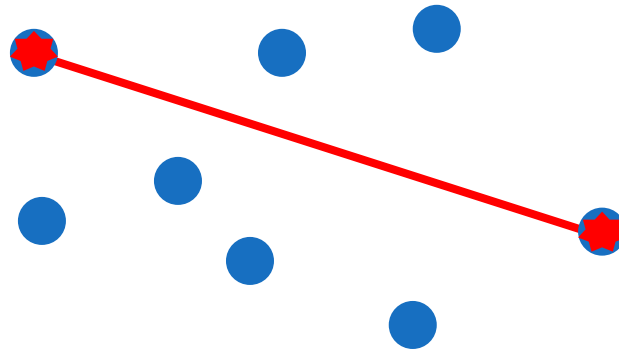
FarthestFirstTraversal($Data, k$)

$Centers$ \leftarrow the set consisting of a single $DataPoint$ from $Data$

while $Centers$ have fewer than k points

$DataPoint$ \leftarrow a point in $Data$ maximizing $d(DataPoint, Centers)$
among all data points

add $DataPoint$ to $Centers$



k -Center Clustering Heuristic

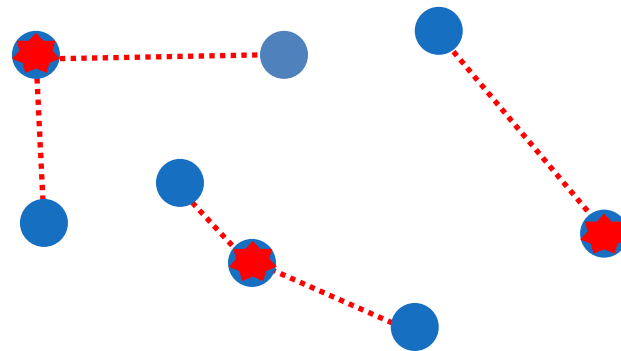
FarthestFirstTraversal($Data, k$)

$Centers$ \leftarrow the set consisting of a single $DataPoint$ from $Data$

while $Centers$ have fewer than k points

$DataPoint$ \leftarrow a point in $Data$ maximizing $d(DataPoint, Centers)$
among all data points

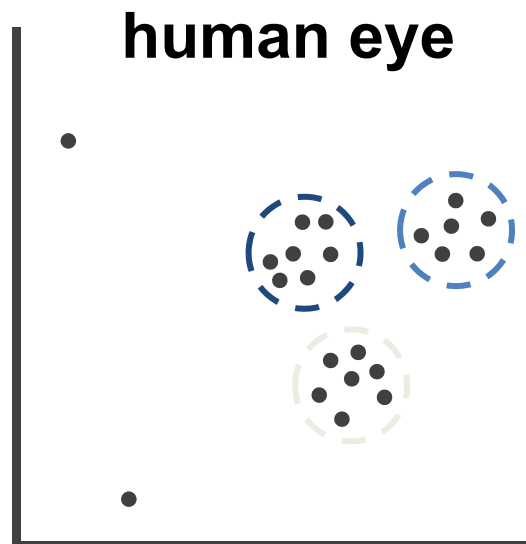
add $DataPoint$ to $Centers$



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

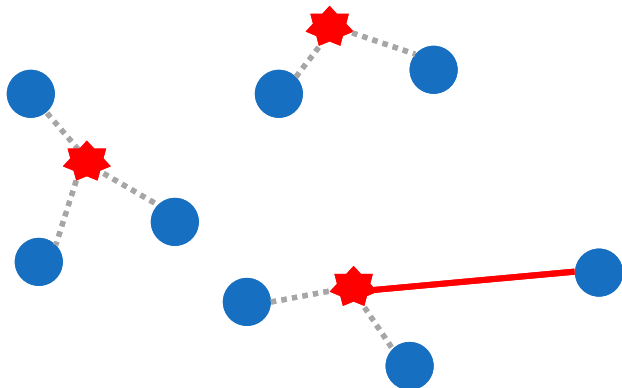
The **maximal distance** between *Data* and *Centers*:

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

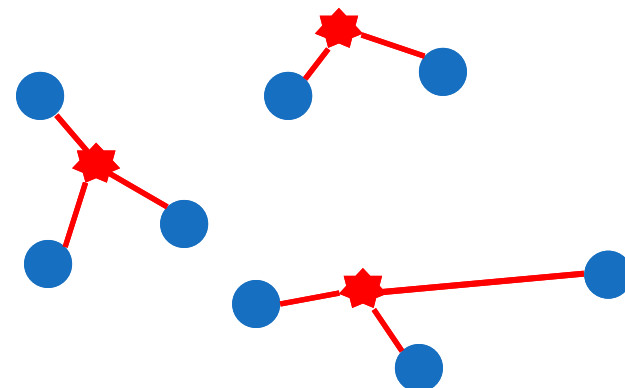
The **squared error distortion** between *Data* and *Centers*:

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

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*.

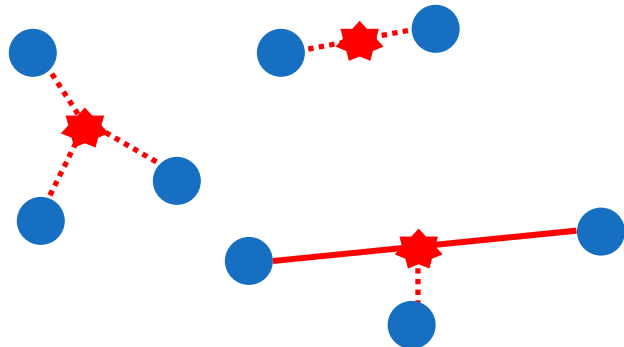
Output: A set of *k* points

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*.

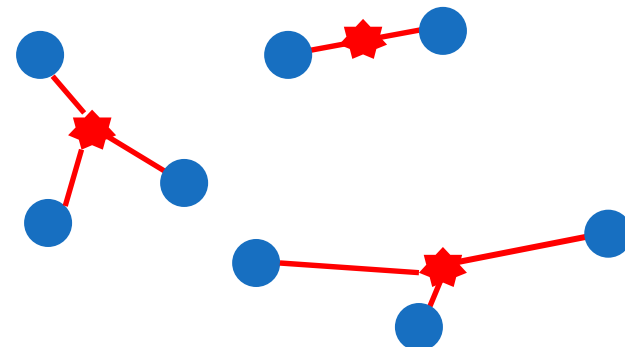
Output: A set of *k* points

Centers

that minimizes

$Distortion(Data, Centers)$

over all choices of *Centers*.
NP Hard for $k \geq 2$.

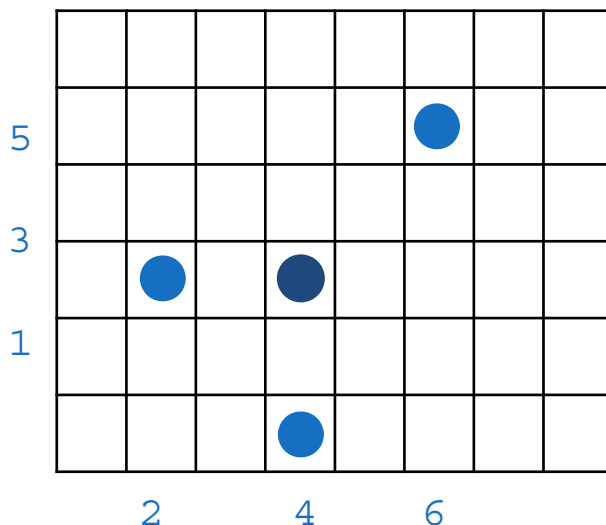


k -Means Clustering for $k = 1$

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

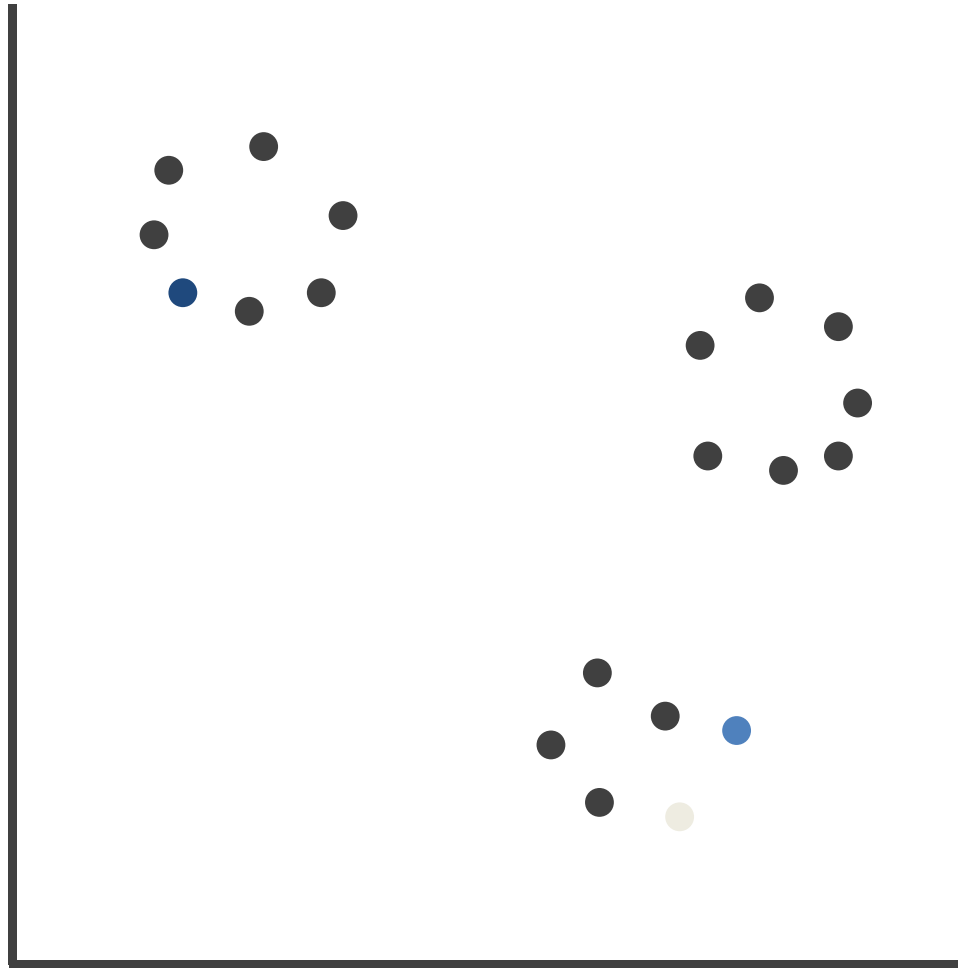
The **center of gravity** of points *Data* is

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



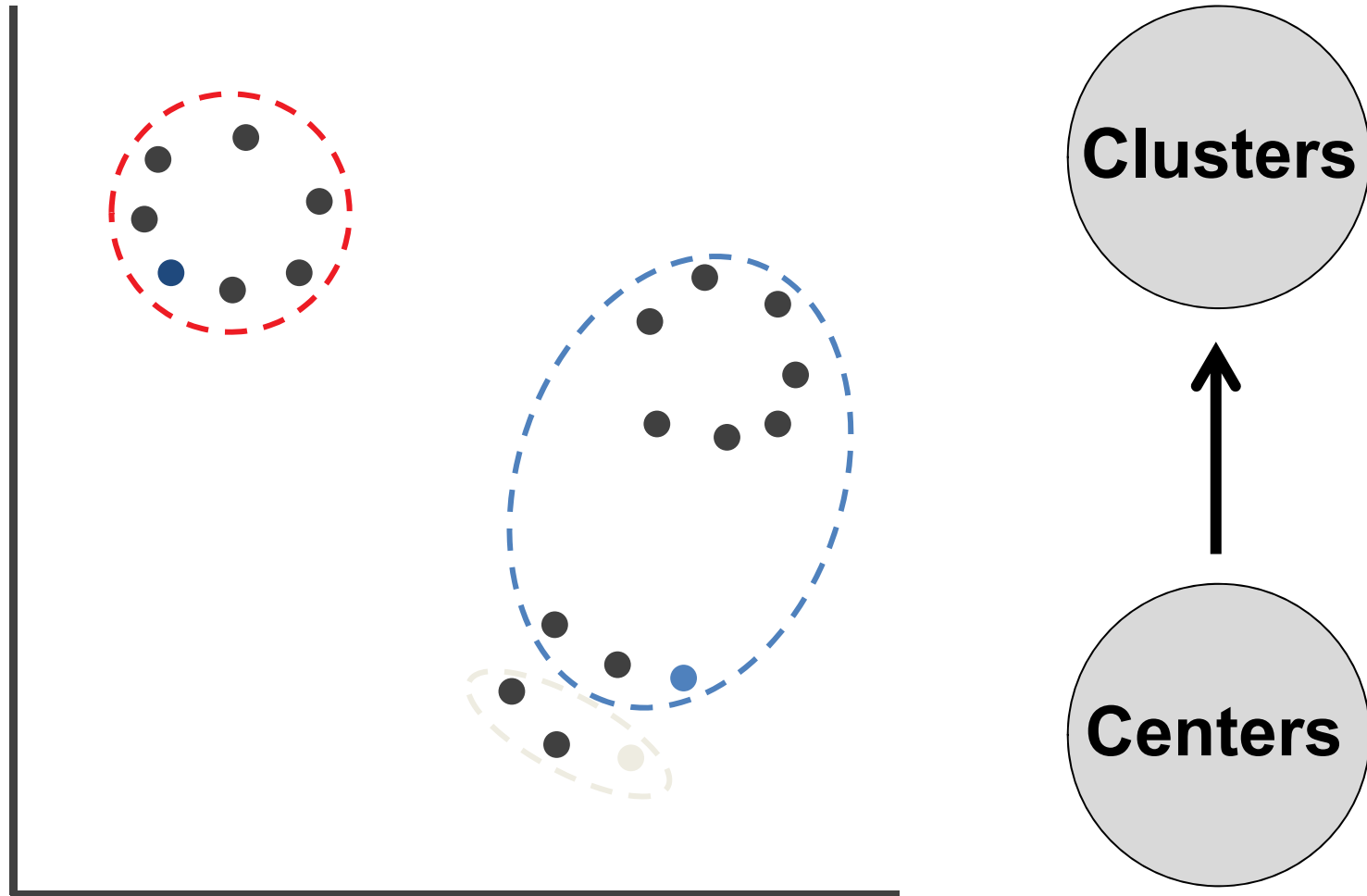
i -th coordinate of the **center of gravity** = the average of the i -th coordinates of datapoints:
 $((2+4+6)/3, (3+1+5)/3) = (4, 3)$

The Lloyd Algorithm in Action



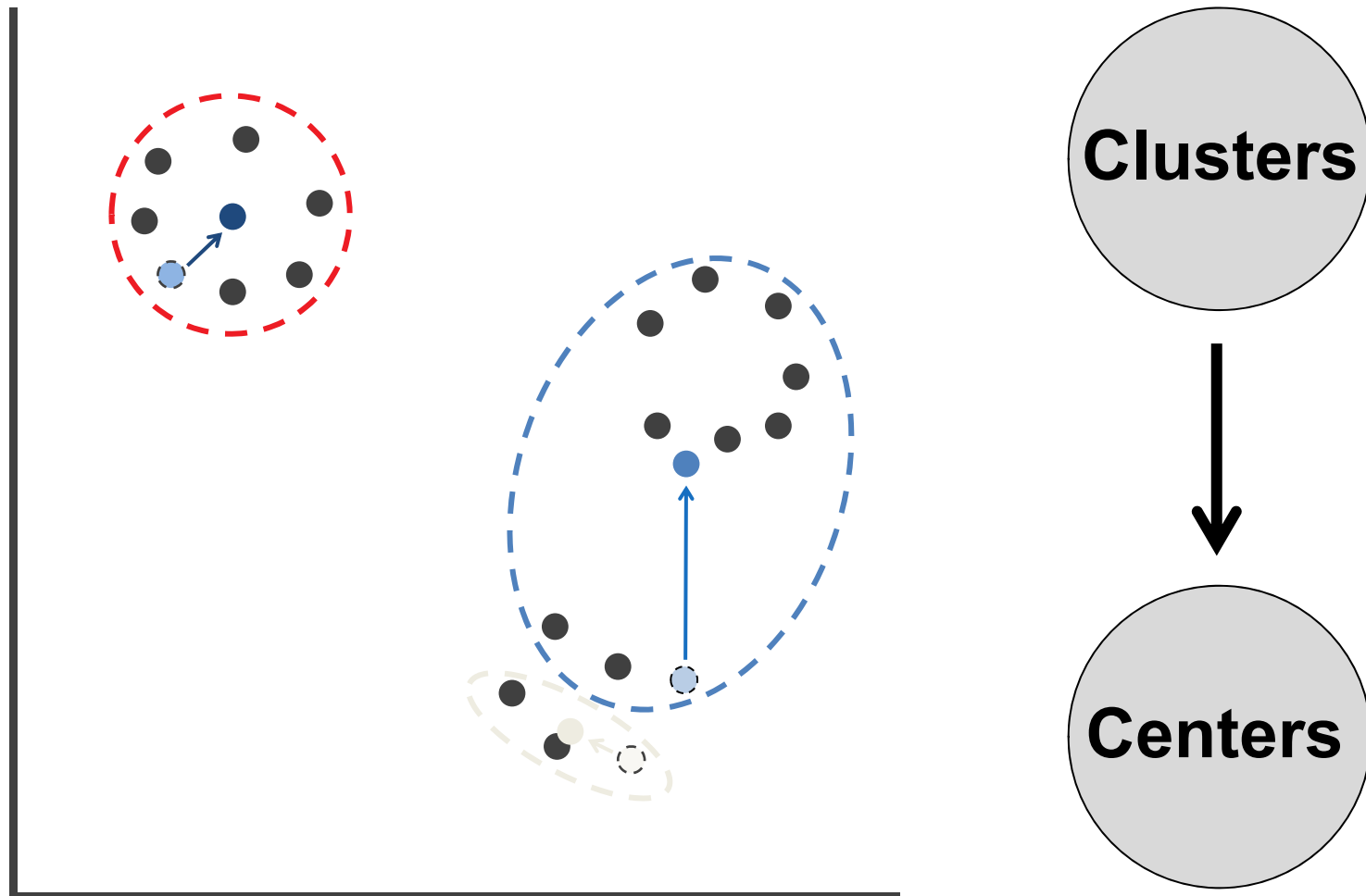
Select k arbitrary data points as *Centers*

The Lloyd Algorithm in Action



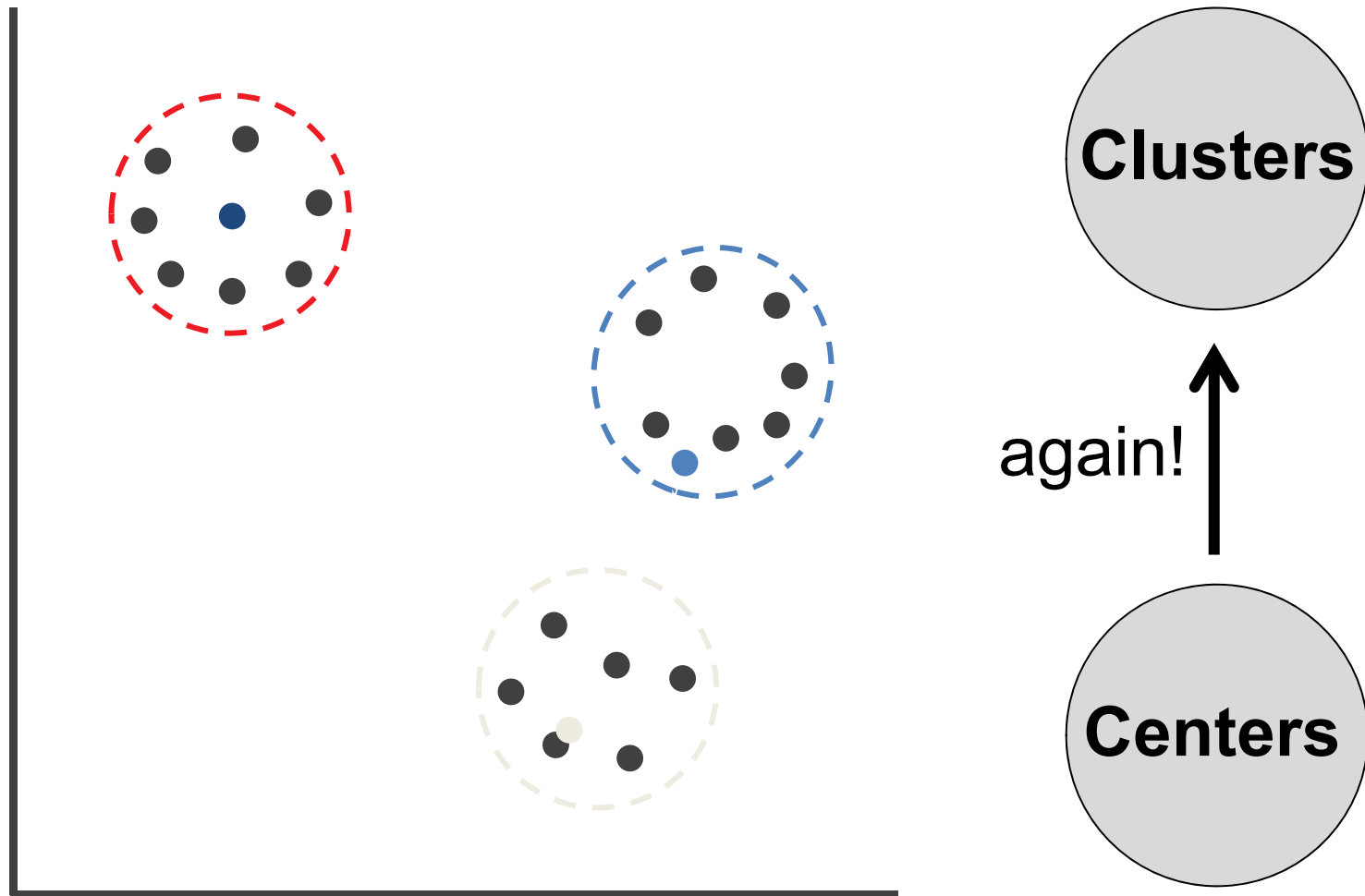
assign each data point to its nearest
center

The Lloyd Algorithm in Action



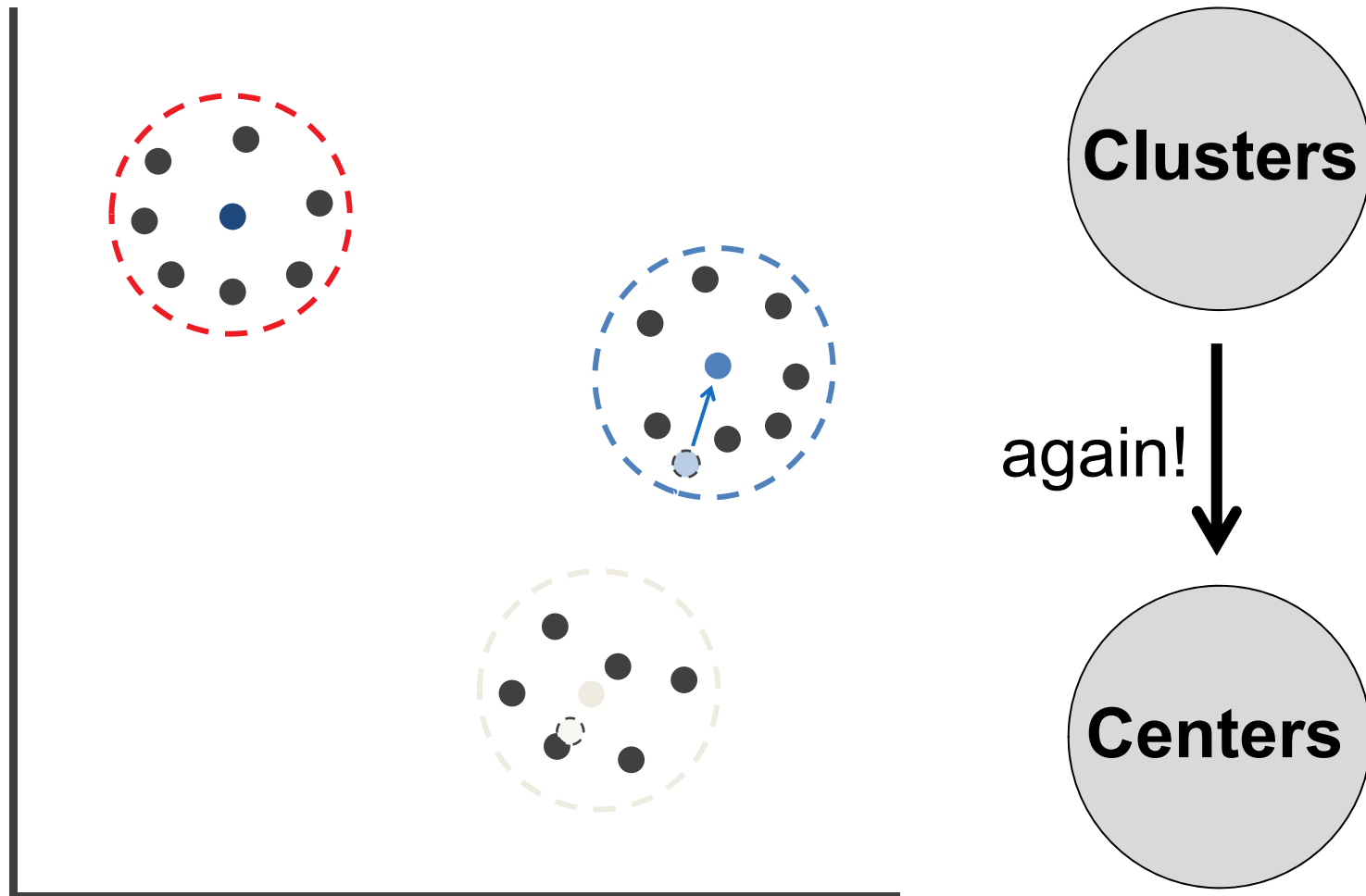
new centers ← clusters' centers of
gravity

The Lloyd Algorithm in Action



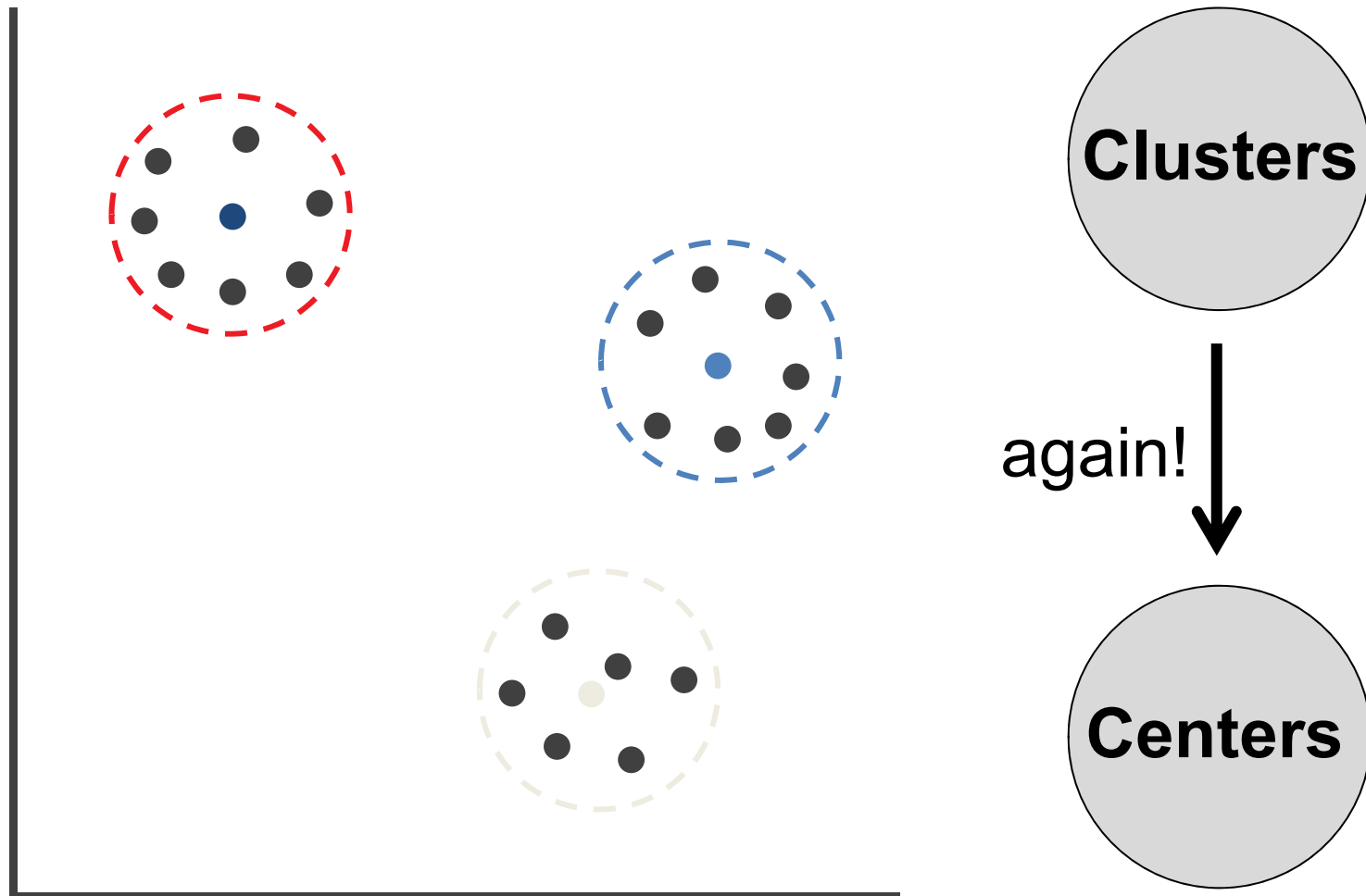
assign each data point to its nearest
center

The Lloyd Algorithm in Action



new centers ← clusters' centers of
gravity

The Lloyd Algorithm in Action



assign each data point to its nearest
center

The Lloyd Algorithm

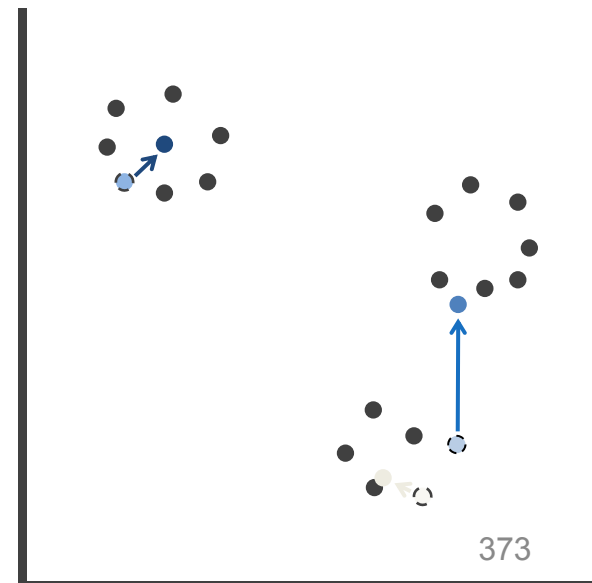
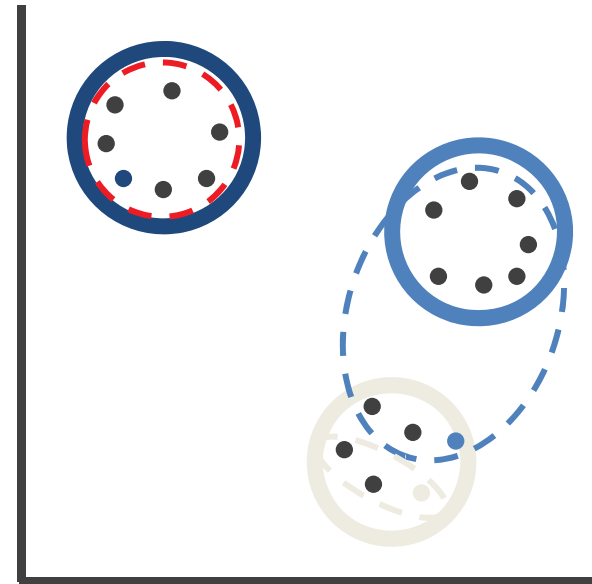
Select k arbitrary data points as *Centers* and then iteratively performs the following two steps:

- **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**).

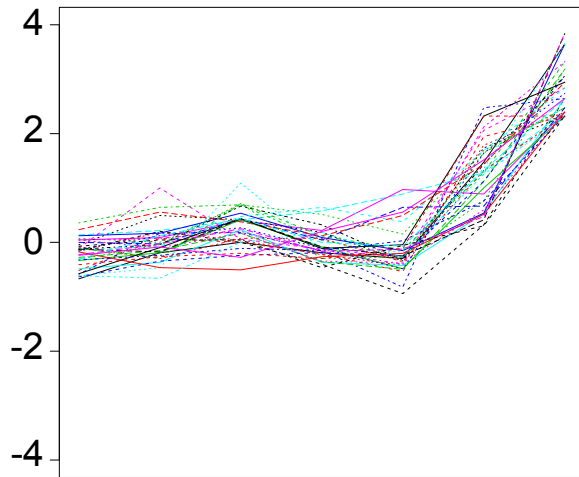
Must the Lloyd Algorithm Converge?

- 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).

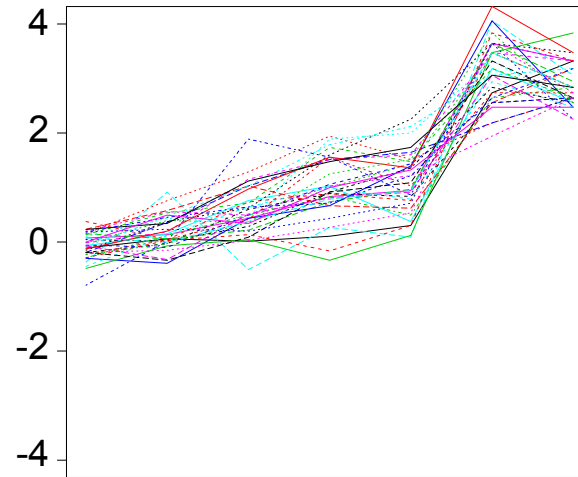


Clustering Yeast Genes

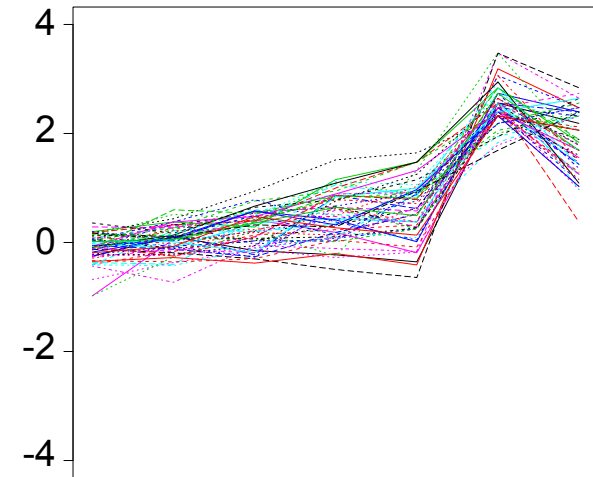
Cluster 1



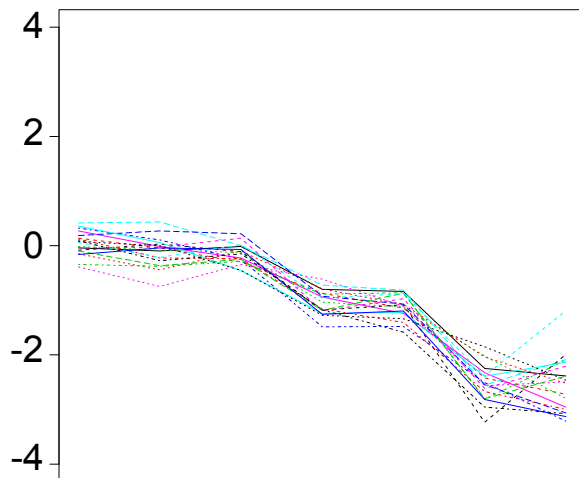
Cluster 2



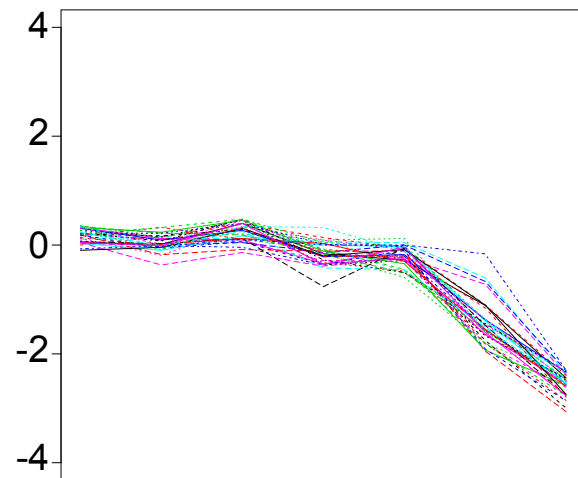
Cluster 3



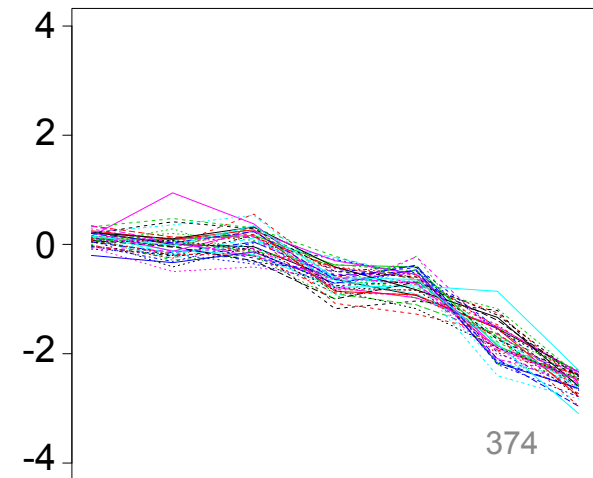
Cluster 4



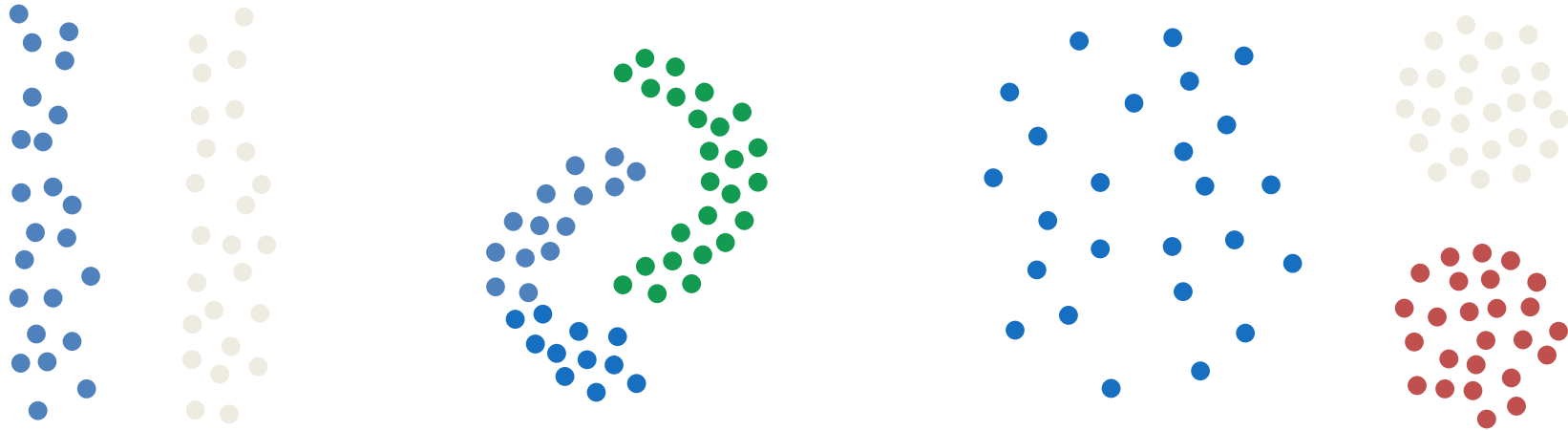
Cluster 5



Cluster 6



k -means Clustering vs. the Human Eye

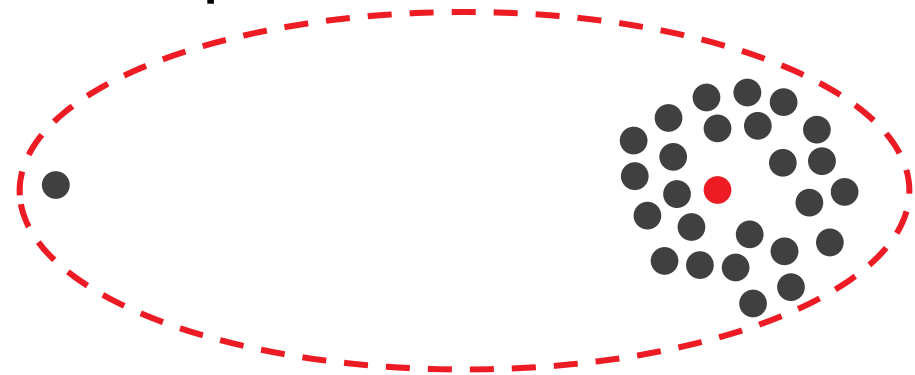
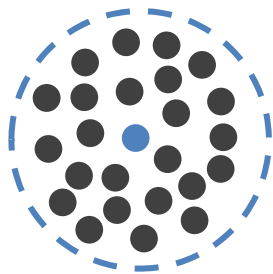


How would the *Lloyd algorithm* cluster these sets of points?



Soft vs. Hard Clustering

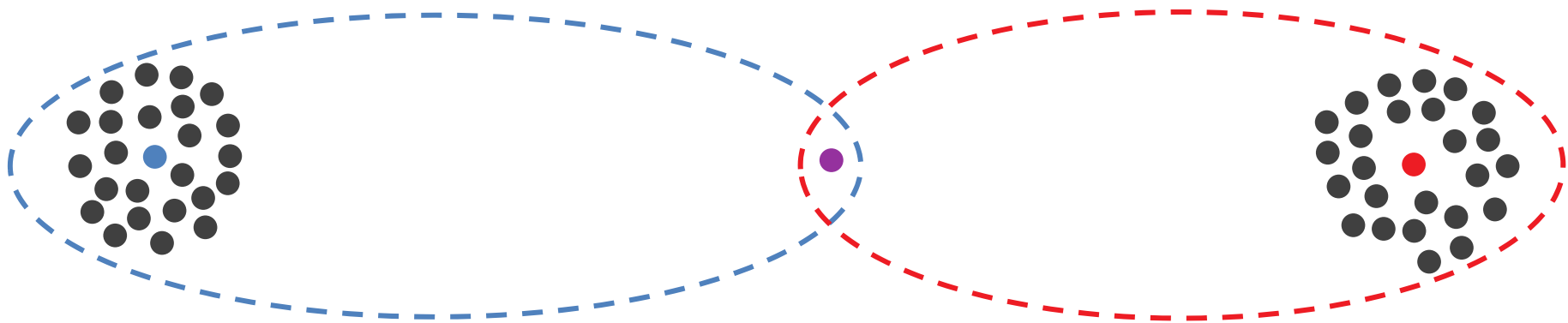
- The Lloyd algorithm assigns the midpoint either to the red or to the blue cluster.
 - “**hard**” assignment of data points to clusters.



Midpoint: A point approximately halfway between two clusters.

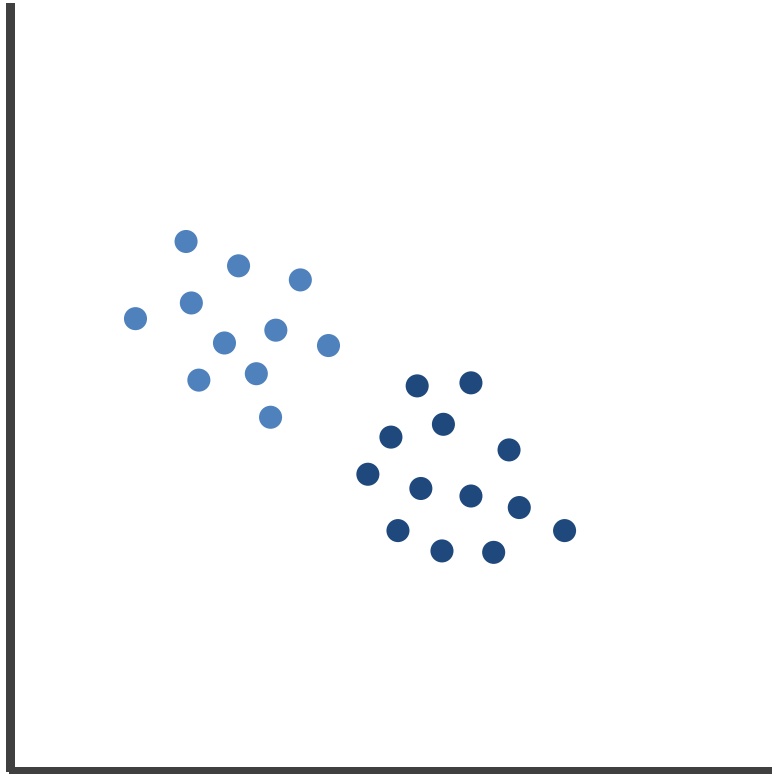
Soft vs. Hard Clustering

- The Lloyd algorithm assigns the midpoint either to the red or to the blue cluster.
 - “**hard**” assignment of data points to clusters.

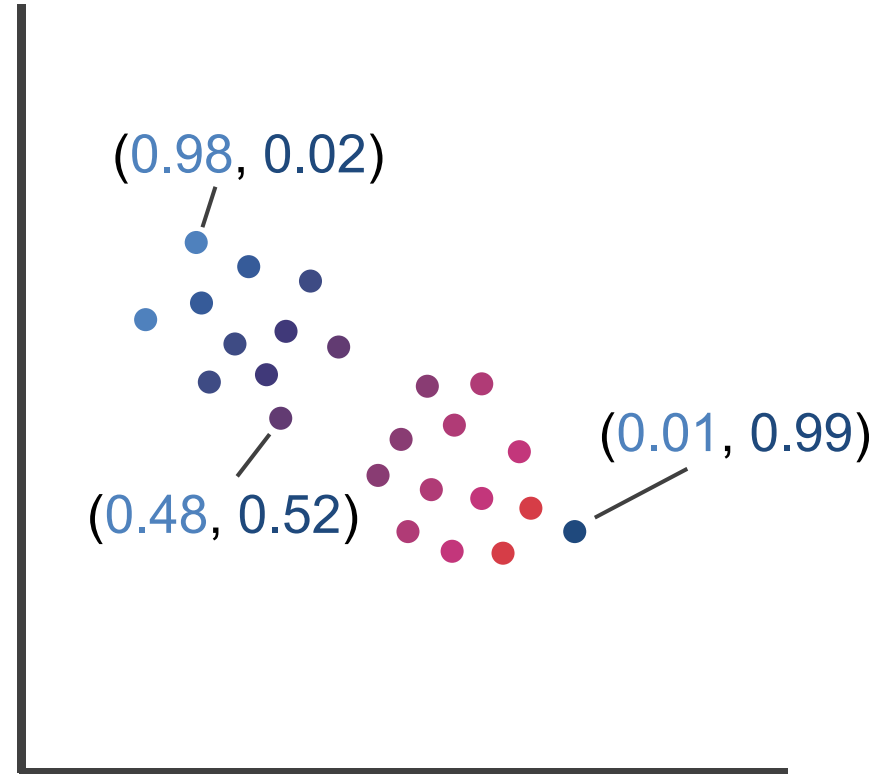


- Can we color the midpoint half-red and half-blue?
 - “**soft**” assignment of data points to clusters.

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} 378
($r_{\text{blue}} + r_{\text{red}} = 1$)

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 minimized at $\theta = i/n$ (most likely bias)



A

Flipping Two Biased Coins



B

Data

HTTTHTTHTH	0.4
HHHHTHHHHH	0.9
HTHHHHHTHH	0.8
HTTTTTTHHTT	0.3
THHHTHHHTH	0.7

Goal: estimate the probabilities θ_A and θ_B



We Knew Which Coin

Used in Each Sequence...



	<i>Data</i>	<i>HiddenVector</i>
HTTTHTTHTH	0.4	1
HHHTHHHHH	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

Used in Each Sequence...



Was

	<i>Data</i>	<i>HiddenVector</i>
HTTTHTTHTH	0.4	1
HHHHTHHHHH	0.9	0
HTHHHHHTHH	0.8	0
HTTTTTTHHTT	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=($\theta_A,$ 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 =
 $= (4+3) / (10+10) = (0.4+0.3) / 2 = 0.35$

$$(0.4*1+0.9*0+0.8*0+0.3*1+0.7*0) / (1+0+0+1+0) = 0.35$$

$$\sum_{\text{all data points } i} \text{Data}_i * \text{HiddenVector}_i / \sum_{\text{all data points } i} \text{HiddenVector}_i = 0.35$$

$$\text{Data} * \text{HiddenVector} / (1, \mathbf{1}, \dots, * 1) * \text{HiddenVector}$$

$\mathbf{1}$ refers to a vector $(1, 1, \dots, 1)$ consisting of all 1s

Parameters as a Dot-Product

	<i>Data</i>	<i>HiddenVector</i>	<i>Parameters</i> =($\theta_A,$
θ_B)			
HTTTHTTHTH	0.4	*	1
HHHTTHHHHH	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
 $= (9+8+7) / (10+10+10) = (0.9+0.8+0.7) / (1+1+1) = 0.80$

$$(0.5*0+0.9*1+0.8*1+0.4*0+0.7*1) / (0+1+1+0+1) = 0.80$$

$$\sum_{\text{all points } i} \text{Data}_i * (1 - \text{HiddenVector}_i) / \sum_{\text{all points } i} (1 - \text{HiddenVector}_i) =$$

$$\text{Data} * (1 - \text{HiddenVector}) / 1 * (1 - \text{HiddenVector}) \quad 384$$

Parameters as a Dot-Product

	<i>Data</i>	<i>HiddenVector</i>	<i>Parameters</i> =($\theta_A,$
θ_B)			
HTTTHTTHTH	0.4	*	1
HHHHTHHHHH	0.9	*	0
HTHHHHHTHH	0.8	*	0
HTTTTTHTTT	0.3	*	1
THHHTHHHTH	0.7	*	0

(0.35, 0.80)

θ_A = fraction of heads generated in all flips with coin *A*
 = $(0.4+0.3)/2=0.35$
 = *Data* * *HiddenVector* / 1 * *HiddenVector*

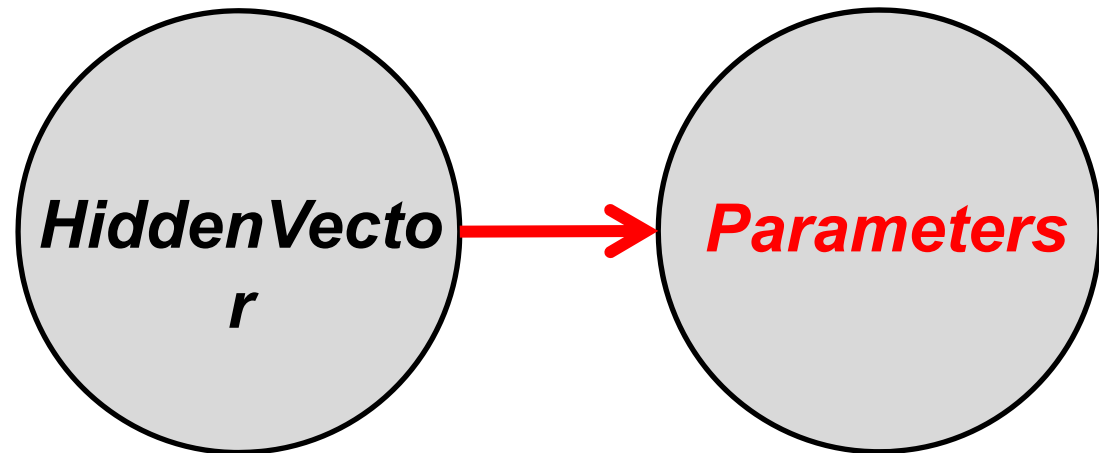
θ_B = fraction of heads generated in all flips with coin *B*
 = $(0.9+0.8+0.7)/3=0.80$
 = *Data* * (1-*HiddenVector*) / 1 * (1 - *HiddenVector*)

Data, HiddenVector, Parameters

θ_B)

Data HiddenVector Parameters= $(\theta_A,$

0.4	1	→ (0.35, 0.80)
0.9	0	
0.8	0	
0.3	1	
0.7	0	



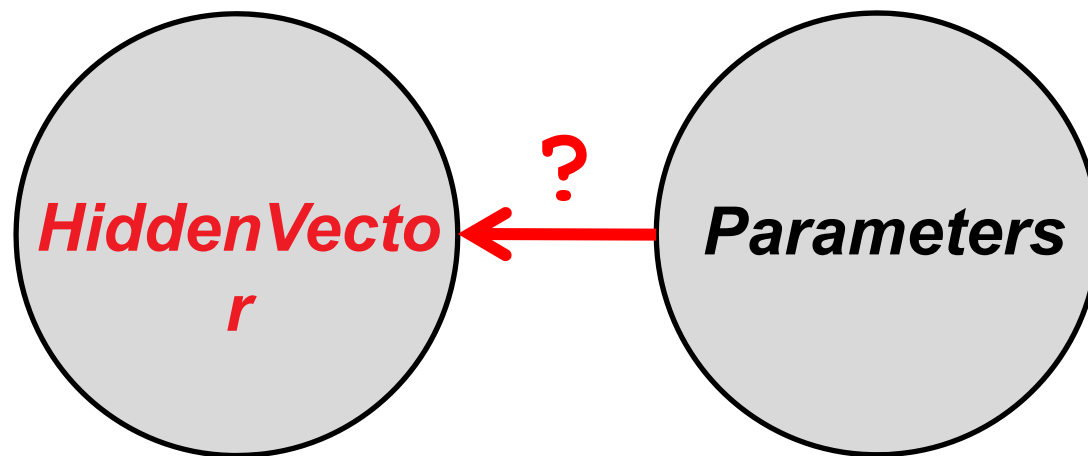
Data, *HiddenVector*, Parameters

θ_B)


Data *HiddenVector* Parameters= $(\theta_A,$

0.4	?
0.9	?
0.8	?
0.3	?
0.7	?

$(0.35, 0.80)$




From *Data & Parameters* to *HiddenVector*

	<i>Data</i>	<i>HiddenVector</i>	<i>Parameters</i> =($\theta_A,$
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)?

$$\begin{aligned} \text{Pr}(1^{\text{st}} \text{ sequence} | \theta_A) &= \theta_A^4 (1-\theta_A)^6 = 0.35^4 \cdot 0.65^6 \approx 0.00113 > \\ \text{Pr}(1^{\text{st}} \text{ sequence} | \theta_B) &= \theta_B^4 (1-\theta_B)^6 = 0.80^4 \cdot 0.20^6 \approx 0.00003 \end{aligned}$$


From *Data & Parameters* to *HiddenVector*

	<i>Data</i>	<i>HiddenVector</i>	<i>Parameters</i> =($\theta_A,$
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)?

$$\begin{aligned} \text{Pr}(1^{\text{st}} \text{ sequence} | \theta_A) &= \theta_A^4 (1-\theta_A)^6 = 0.35^4 \cdot 0.65^6 \approx 0.00113 > \\ \text{Pr}(1^{\text{st}} \text{ sequence} | \theta_B) &= \theta_B^4 (1-\theta_B)^6 = 0.80^4 \cdot 0.20^6 \approx 0.00003 \end{aligned}$$

From *Data* & *Parameters* to *HiddenVector*

	<i>Data</i>	<i>HiddenVector</i>	<i>Parameters</i> =($\theta_A,$
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)?

$$\begin{aligned} \text{Pr}(2^{\text{nd}} \text{ sequence} | \theta_A) &= \theta_A^9 (1-\theta_A)^1 = 0.35^9 \cdot 0.65^1 \approx 0.00005 < \\ \text{Pr}(2^{\text{nd}} \text{ sequence} | \theta_B) &= \theta_B^9 (1-\theta_B)^1 = 0.80^9 \cdot 0.20^1 \approx 0.02684 \end{aligned}$$

From *Data* & *Parameters* to *HiddenVector*

	<i>Data</i>	<i>HiddenVector</i>	<i>Parameters</i> =($\theta_A,$
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)?

$$\begin{aligned} \Pr(2^{\text{nd}} \text{ sequence} | \theta_A) &= \theta_A^9 (1-\theta_A)^1 = 0.35^9 \cdot 0.65^1 \approx 0.00005 < \\ \Pr(2^{\text{nd}} \text{ sequence} | \theta_B) &= \theta_B^9 (1-\theta_B)^1 = 0.80^9 \cdot 0.20^1 \approx 0.02684 \end{aligned}$$

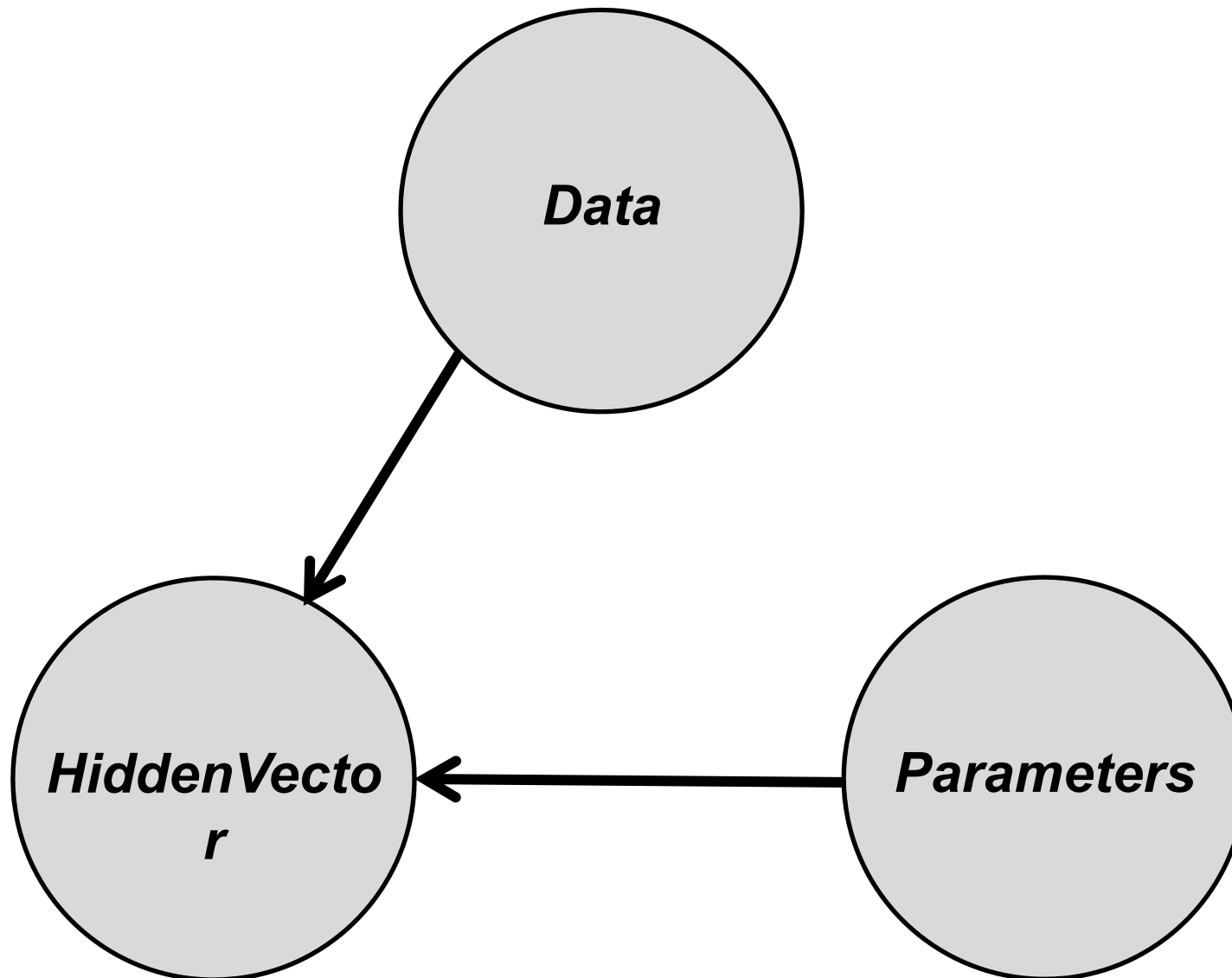
HiddenVector Reconstructed!

θ_B)

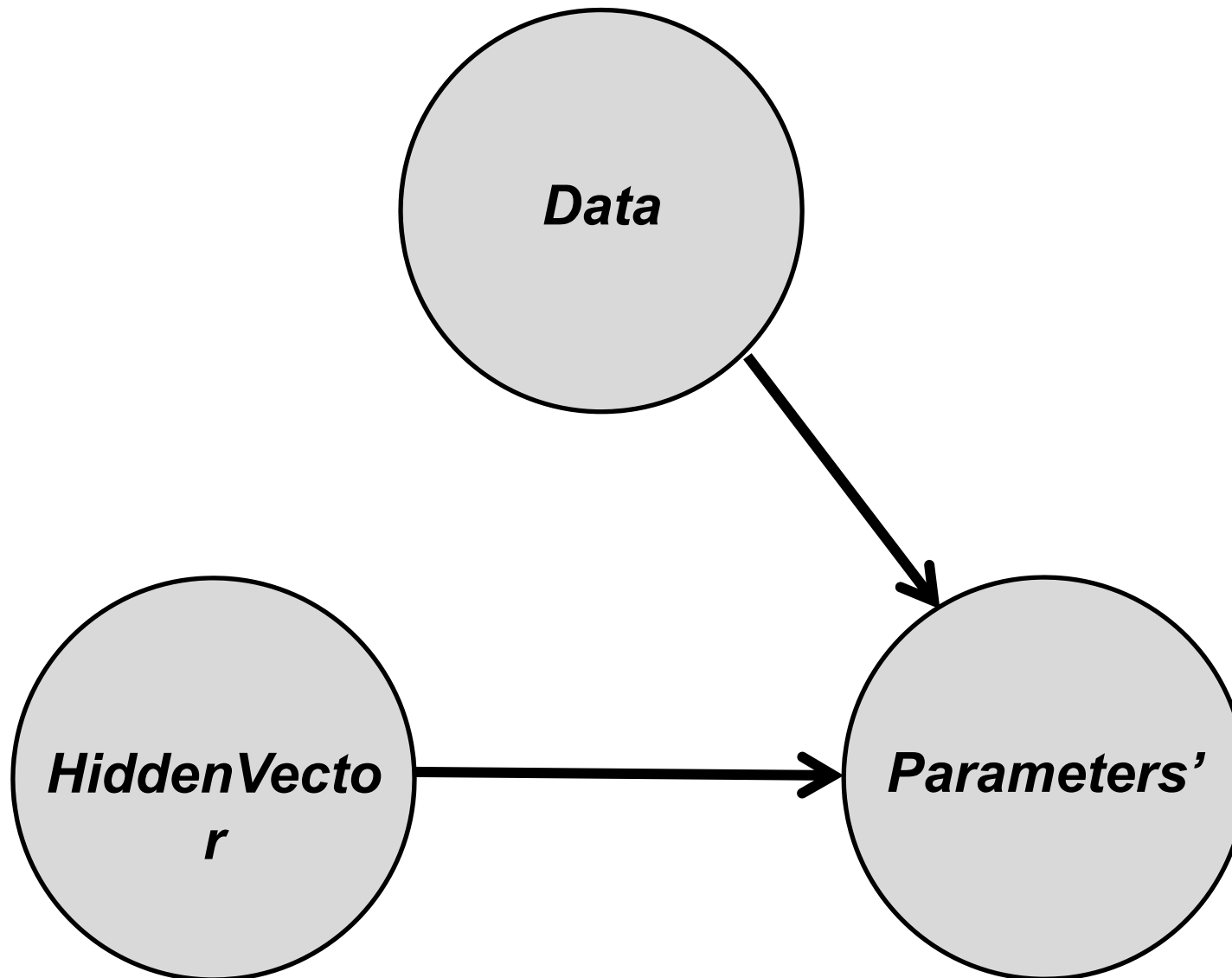
Data ***HiddenVector*** **Parameters**=(θ_A ,

0.4	1	
0.9	0	
0.8	0	← (0.35, 0.80)
0.3	1	
0.7	0	

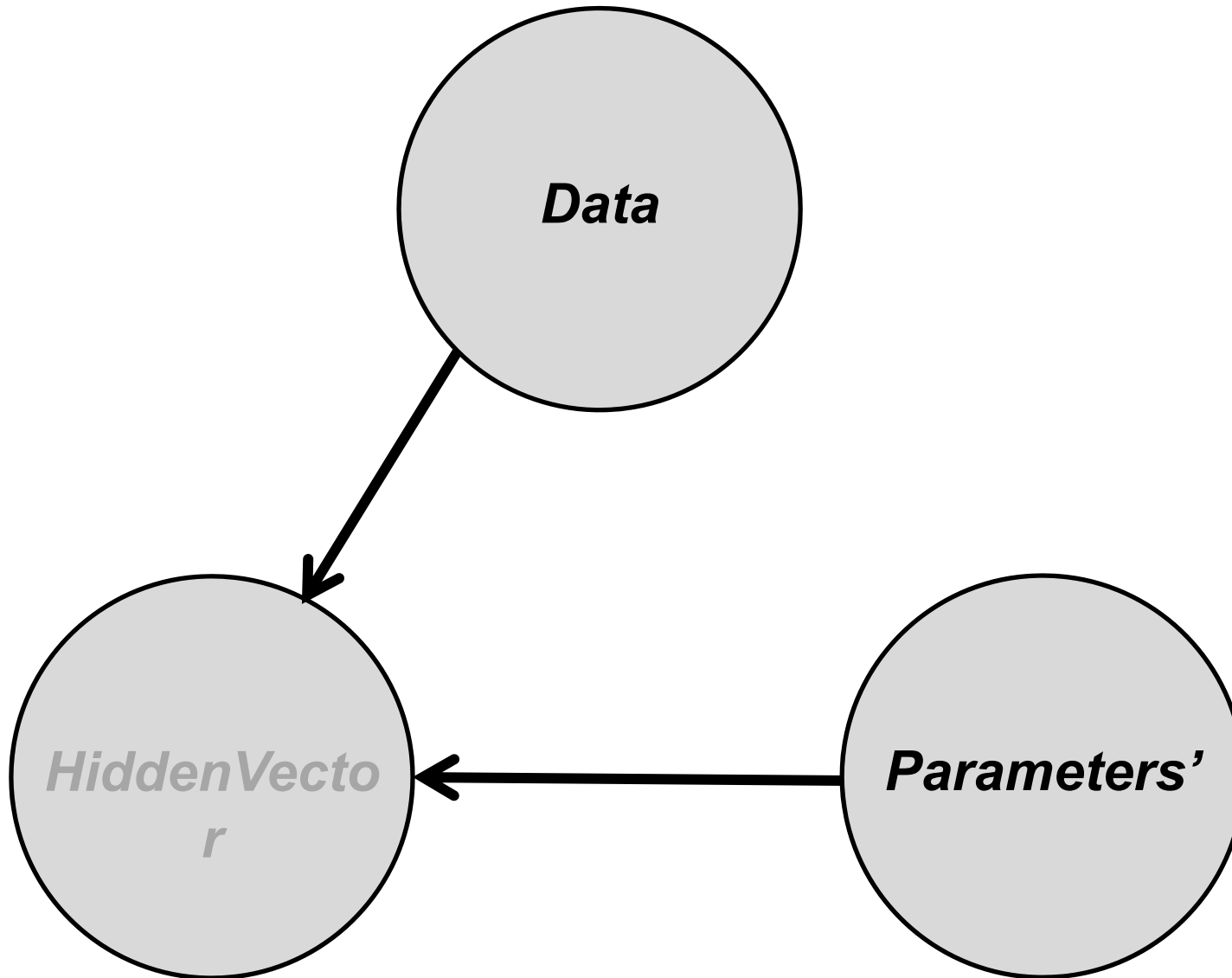
Reconstructing *HiddenVector* and *Parameters*



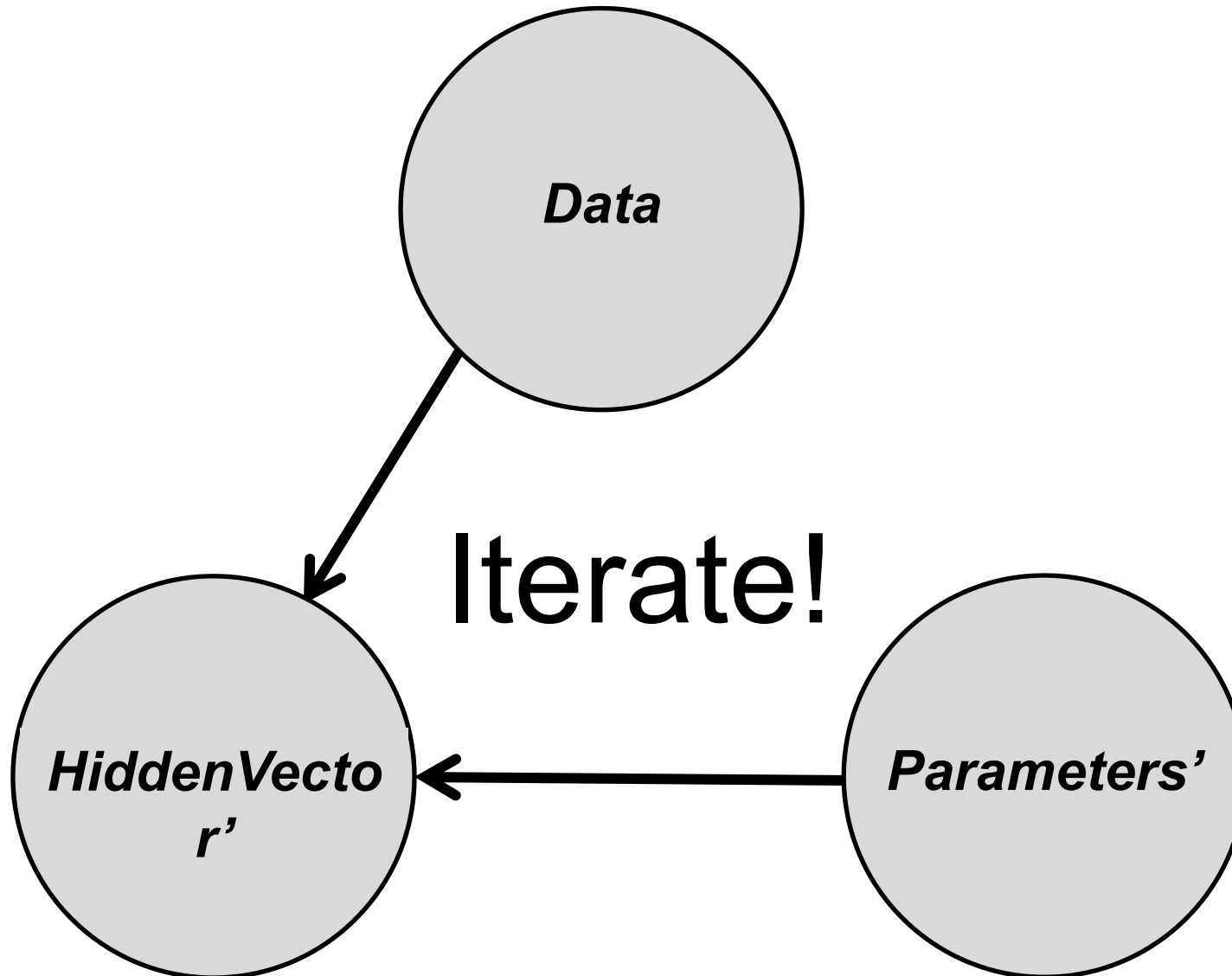
Reconstructing *HiddenVector* and *Parameters*



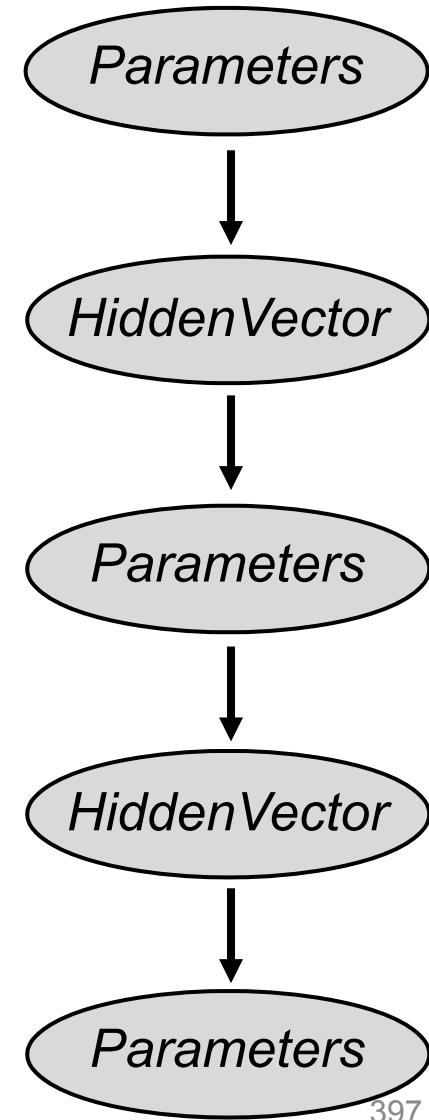
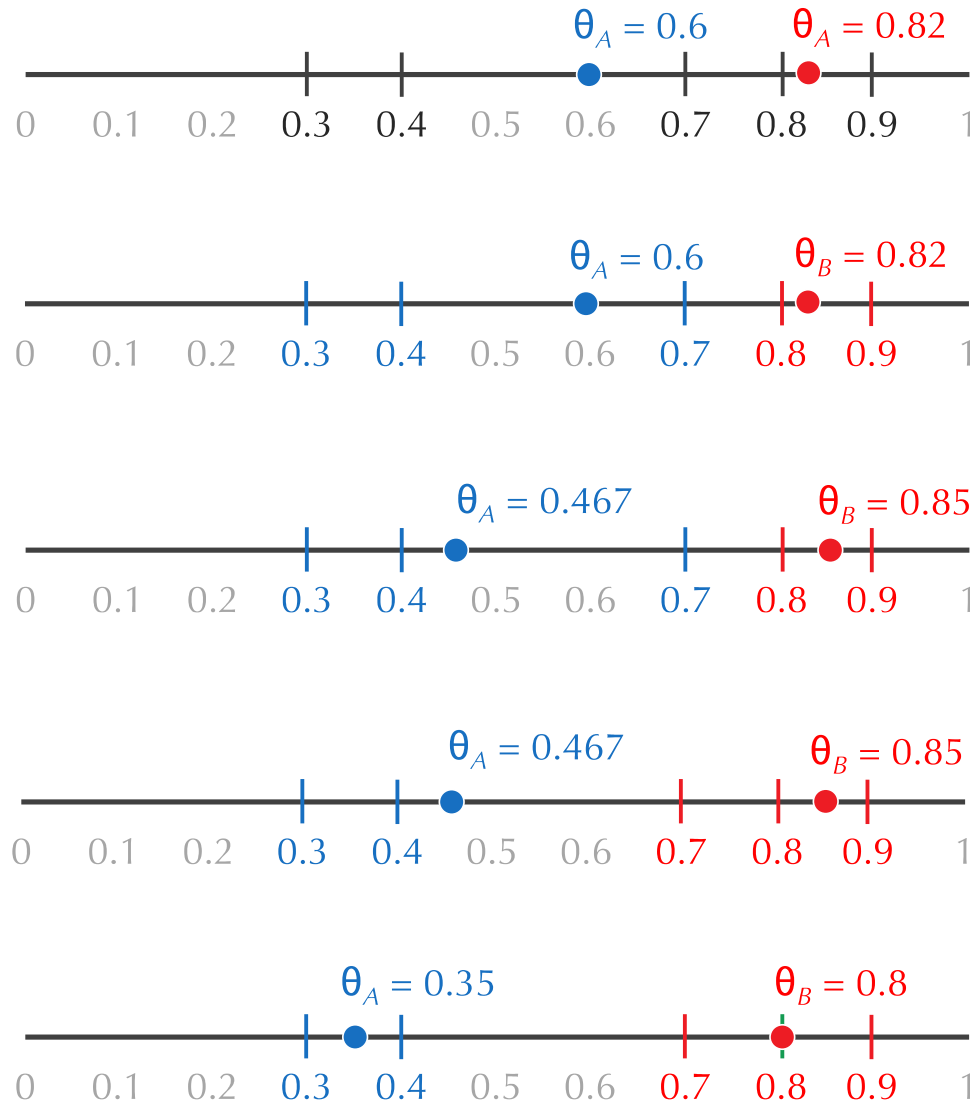
Reconstructing *HiddenVector* and *Parameters*



Reconstructing *HiddenVector* and *Parameters*



What does this algorithm remind you of?

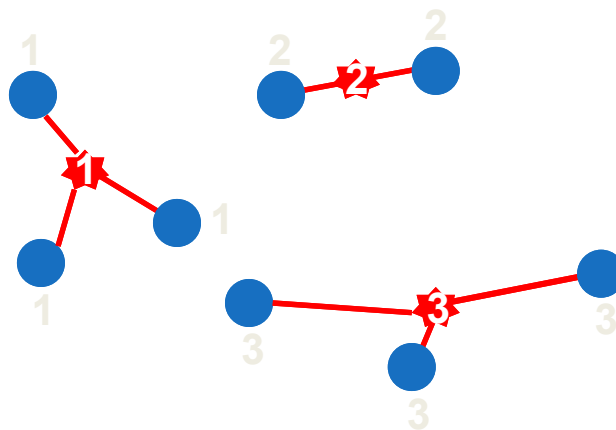


From Coin Flipping to k-means Clustering: Where Are *Data*, *HiddenVector*, and *Parameters*?

Data: data points $Data = (Data_1, \dots, Data_n)$

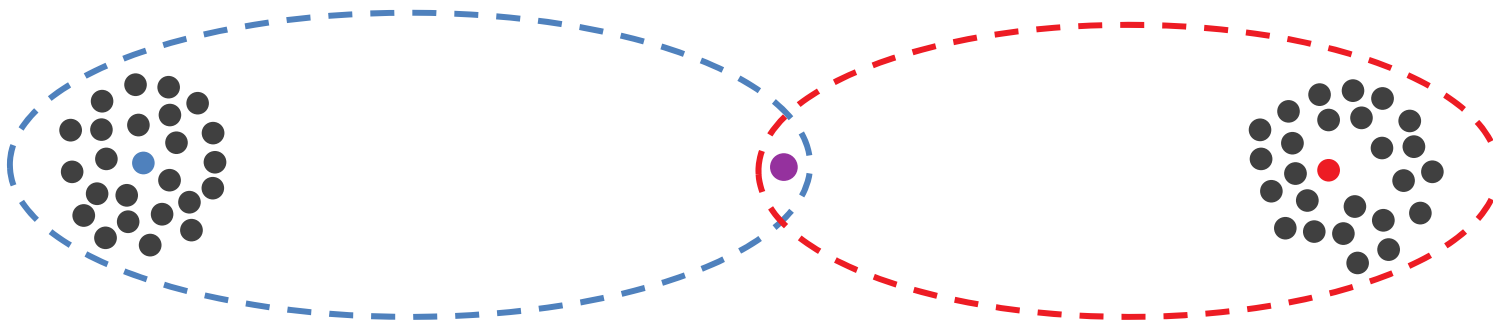
Parameters: $Centers = (Center_1, \dots, Center_k)$

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 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.

Memory Flash: From *Data & Parameters* to *HiddenVector*

	<i>Data</i>	<i>HiddenVector</i>	<i>Parameters</i> =
(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)?

$$\Pr(1^{\text{st}} \text{ sequence} | \theta_A) = \theta_A^5 (1 - \theta_A)^5 = 0.60^4 \cdot 0.40^6 \approx 0.000531 >$$
$$\Pr(1^{\text{st}} \text{ sequence} | \theta_B) = \theta_B^5 (1 - \theta_B)^5 = 0.82^4 \cdot 0.18^6 \approx 0.000015$$

Memory Flash:

From *Data & Parameters* to *HiddenVector*


	<i>Data</i>	<i>HiddenVector</i>	<i>Parameters</i> =
(θ_A, θ_B)		1	
	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)?

$$\Pr(1^{\text{st}} \text{ sequence} | \theta_A) = \theta_A^5 (1 - \theta_A)^5 = 0.60^4 \cdot 0.40^6 \approx 0.000531 >$$

$$\Pr(1^{\text{st}} \text{ sequence} | \theta_B) = \theta_B^5 (1 - \theta_B)^5 = 0.82^4 \cdot 0.18^6 \approx 0.000015$$

From *Data & Parameters* to *HiddenMatrix*

(θ_A, θ_B)	Data	HiddenMatrix	Parameters =
	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?

$$\Pr(1^{\text{st}} \text{ sequence} | \theta_A) \approx 0.000531 >$$


$$\Pr(1^{\text{st}} \text{ sequence} | \theta_B) \approx 0.000015$$

$$0.000531 / (0.000531 + 0.000015) \approx 0.97$$

$$0.000015 / (0.000531 + 0.000015) \approx 0.03$$

From *Data & Parameters* to *HiddenMatrix*

(θ_A, θ_B)

	<i>Data</i>	<i>HiddenMatrix</i>	<i>Parameters =</i>
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?

$$\Pr(2^{\text{nd}} \text{ sequence} | \theta_A) \approx 0.0040 <$$

$$\Pr(2^{\text{nd}} \text{ sequence} | \theta_B) \approx 0.0302$$

$$0.0040 / (0.0040 + 0.0302) = 0.12$$

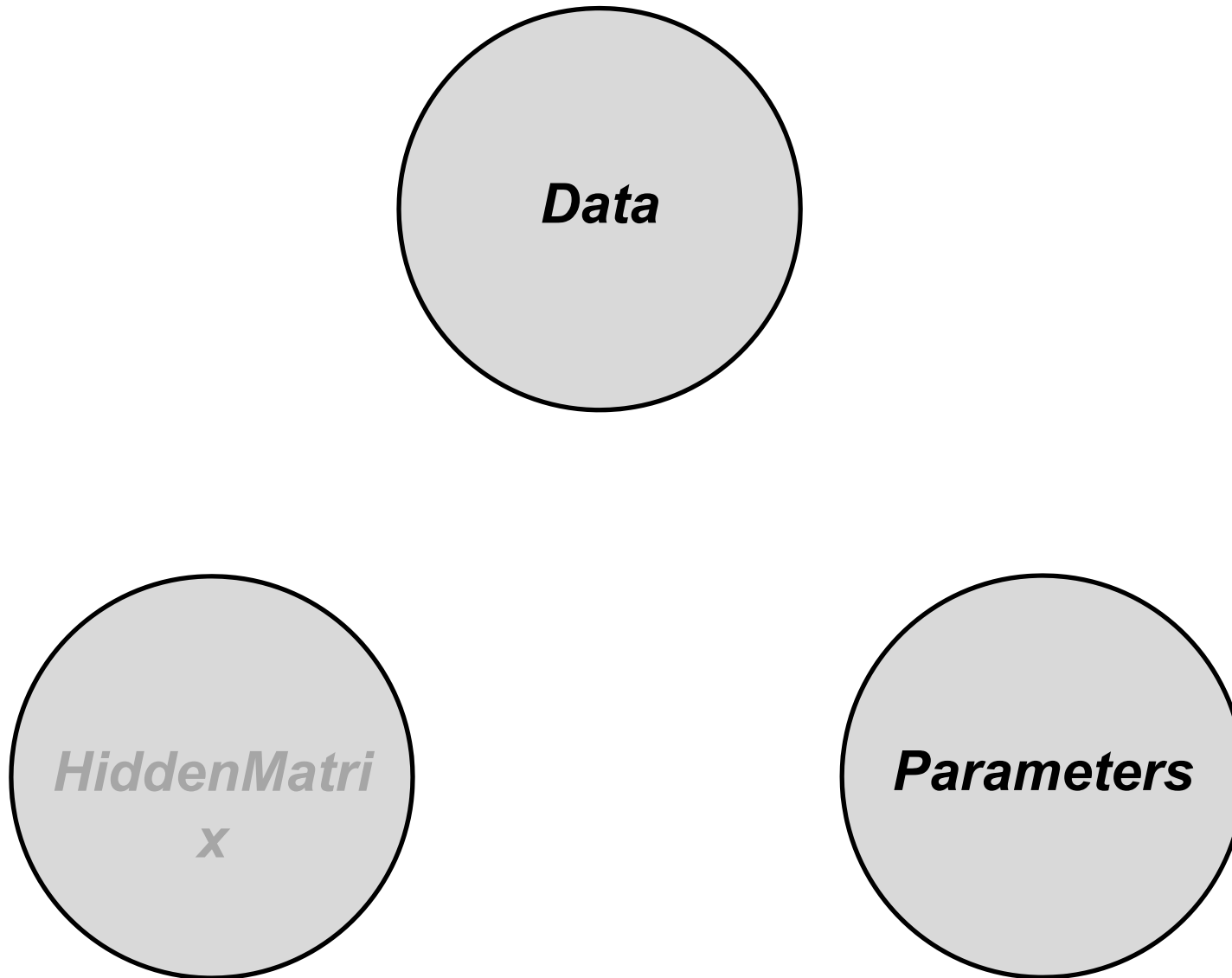
$$0.0342 / (0.0040 + 0.0342) = 0.88$$

HiddenMatrix Reconstructed!

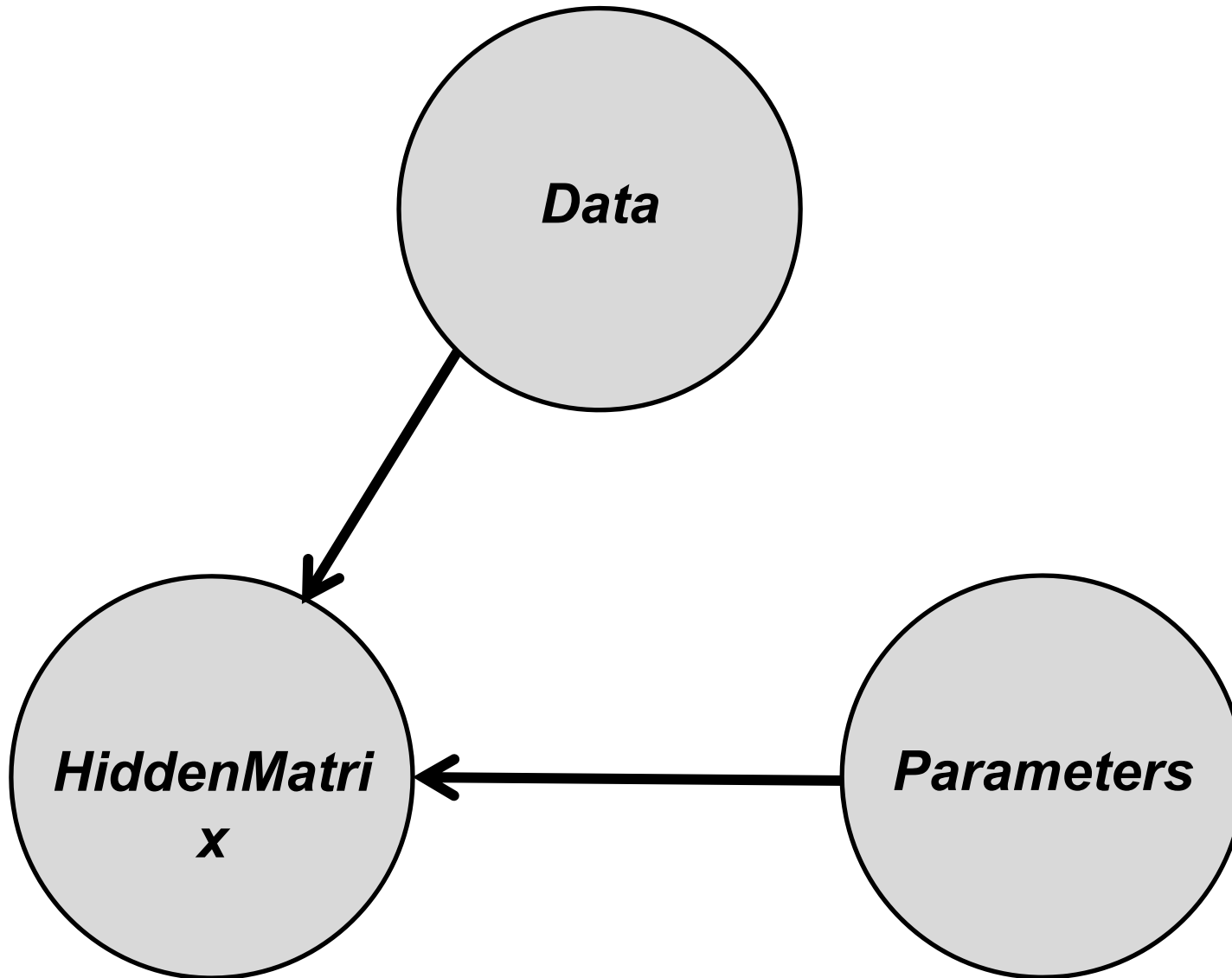
(θ_A, θ_B)

	<i>Data</i>	<i>HiddenMatrix</i>	<i>Parameters =</i>
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	

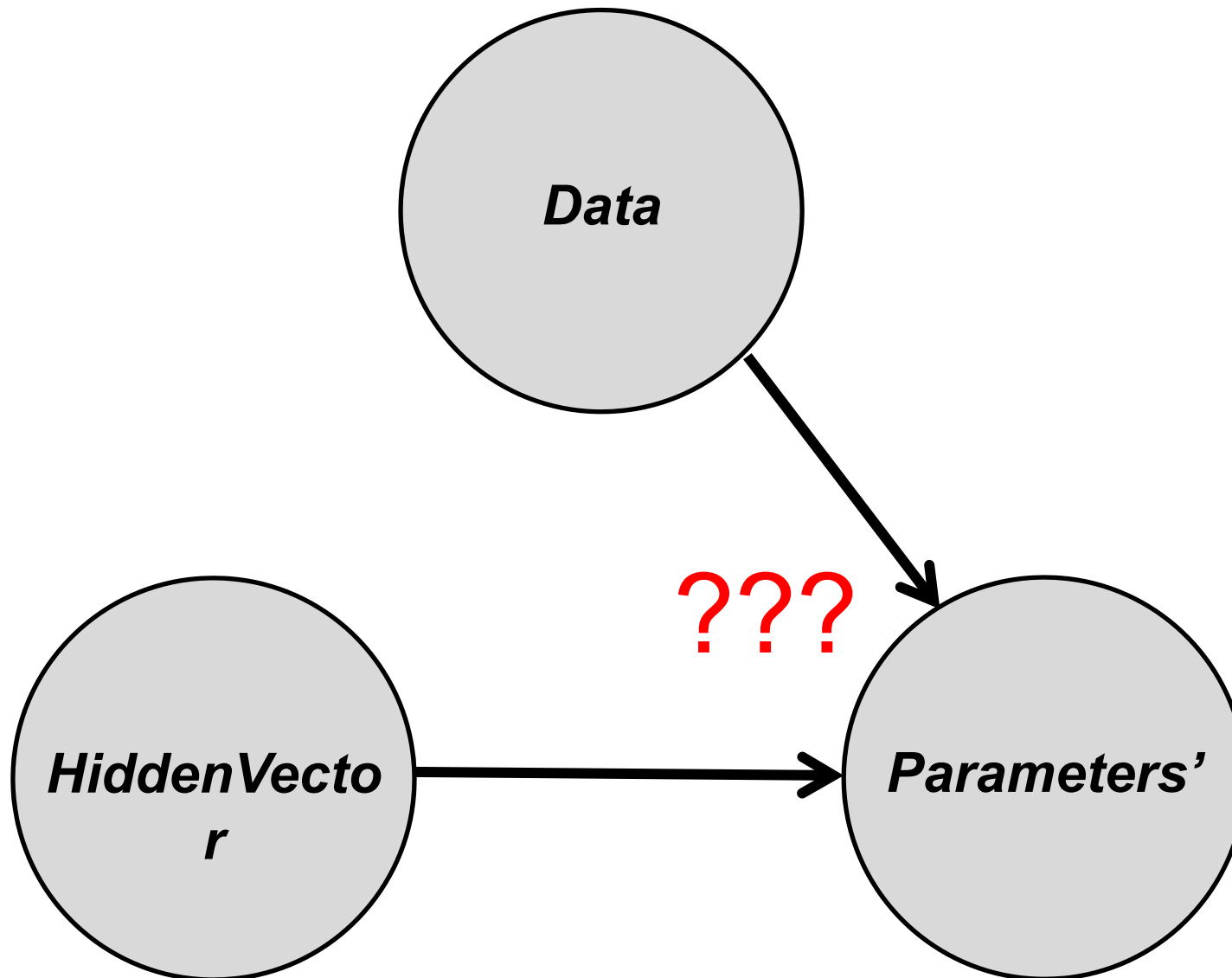
Expectation Maximization Algorithm



E-step



M-step



Memory Flash: Dot Product

Data *HiddenVector* **Parameters**=($\theta_A,$

$\theta_B)$

HTTTHTTHTH

0.4

*

1

HHHTHHHHH

0.9

*

0

HTHHHHHTHH

0.8

*

0

???

HTTTTTHHTT

0.3

*

1

THHHTHHHTH

0.7

*

0

$\theta_A = Data$

*

HiddenVector

/ 1

*

HiddenVector

$\theta_B = Data$

*

(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	
HTTTTTHTTT	0.3	1	
THHHTHHHTH	0.7	0	
$\theta_A = \text{Data} * \text{HiddenVector}$			$/ 1 * \text{HiddenVector}$

$\theta_B = \text{Data} * (1 - \text{HiddenVector}) / 1 * (1 - \text{HiddenVector})$

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

HHHTHHHHH 0.9 0

HTHHHHHTHH 0.8 0

HTTTTTHHTT 0.3 1

THHHTHHHTH 0.7 0

$\theta_A = \text{Data} * \text{HiddenVector} / 1 * \text{HiddenVector}$

$\theta_A = \text{Data} * 1^{\text{st}} \text{ row of HiddenMatrix} / 1 * 1^{\text{st}} \text{ row of HiddenMatrix}$

$\theta_B = \text{Data} * (1 - \text{HiddenVector}) / 1 * (1 - \text{HiddenVector})$

$\theta_B = \text{Data} * 2^{\text{nd}} \text{ row of HiddenMatrix} / 1 * 2^{\text{nd}} \text{ row of HiddenMatrix}$

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

HHHTHHHHH 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

$\theta_A = \text{Data} * \text{HiddenVector} / 1 * \text{HiddenVector}$

$\theta_A = \text{Data} * 1^{\text{st}} \text{ row of HiddenMatrix} / 1 * 1^{\text{st}} \text{ row of HiddenMatrix}$

$\theta_B = \text{Data} * (1 - \text{HiddenVector}) / 1 * (1 - \text{HiddenVector})$

$\theta_B = \text{Data} * 2^{\text{nd}} \text{ row of HiddenMatrix} / 1 * 2^{\text{nd}} \text{ row of HiddenMatrix}$

HiddenVector = (1 0 0 1 0)

Hidden Matrix = $\begin{pmatrix} .97 & .03 & .29 & .99 \\ .55 & & & \end{pmatrix}$

From *HiddenVector* to *HiddenMatrix*

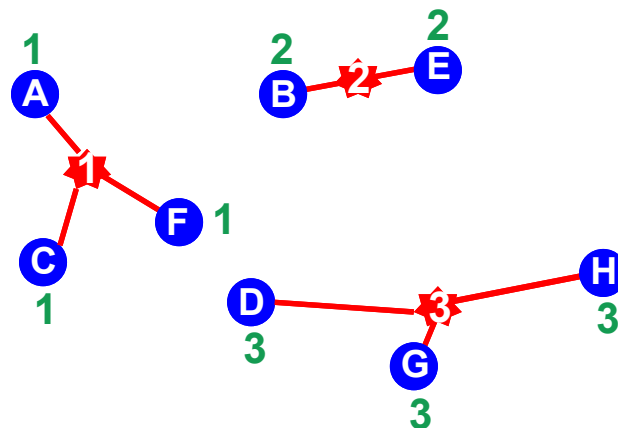
Data: data points $Data = \{Data_1, \dots, Data_n\}$

Parameters: *Centers* = $\{Center_1, \dots, Center_k\}$

HiddenVector: assignments of data points to centers

	A	B	C	D	E	F	G
<i>HiddenVector</i>	1	2	1	3	2	1	3

<i>HiddenMatrix</i>	1	1	0	1	0	0	1	0	0
	2	0	1	0	0	1	0	0	0
	3	0	0	1	0	0	1	1	1



From *HiddenVector* to *HiddenMatrix*

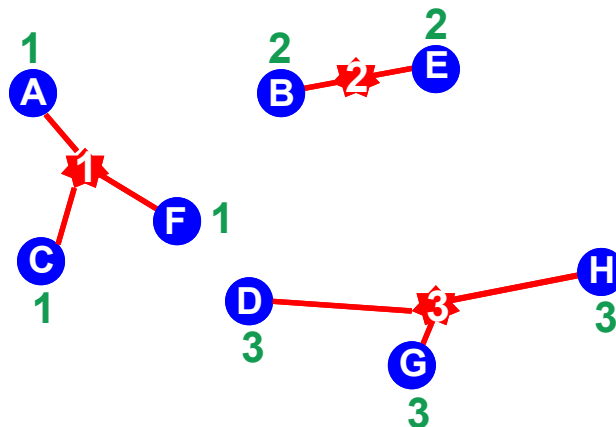
Data: data points $Data = \{Data_1, \dots, Data_n\}$

Parameters: Centers = $\{Center_1, \dots, Center_k\}$

HiddenMatrix $_{i,j}$: responsibility of center i for data point j

HiddenMatrix

	A	B	C	D	E	F	G
1	0.7	0	1	0	0	1	0
2	0.2	1	0	0	1	0	0
3	0.1	0	0	1	0	0	1



From *HiddenVector* to *HiddenMatrix*

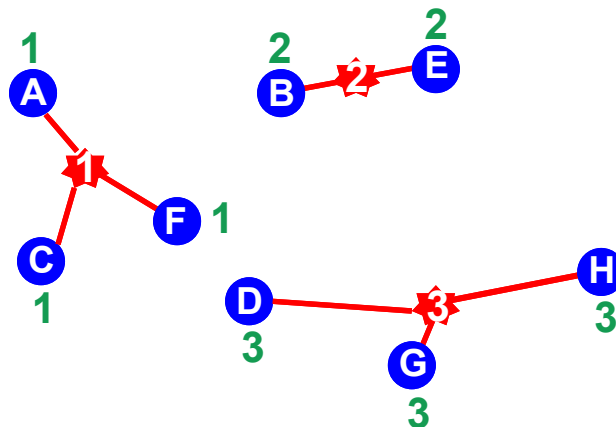
Data: data points $Data = \{Data_1, \dots, Data_n\}$

Parameters: *Centers* = $\{Center_1, \dots, Center_k\}$

*HiddenMatrix*_{*i,j*}: responsibility of center *i* for data point *j*

HiddenMatrix

	A	B	C	D	E	F	G
1	0.70	0.15	0.73	0.40	0.15	0.80	0.05
2	0.20	0.80	0.17	0.20	0.80	0.10	0.20
3	0.10	0.05	0.10	0.40	0.05	0.10	0.90



Responsibilities and the Law of Gravitation



planets

stars

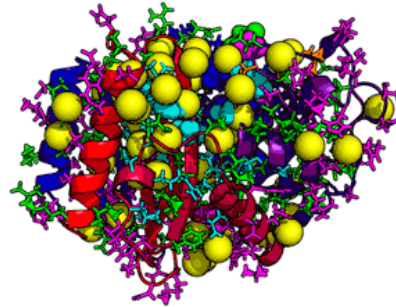
0.70	0.15	0.73	0.40	0.15	0.80	0.05	0.05
0.20	0.80	0.17	0.20	0.80	0.10	0.05	0.20
0.10	0.05	0.10	0.40	0.05	0.10	0.90	0.75

responsibility of star i for a planet j is proportional to the pull (Newtonian law of gravitation):

$$Force_{i,j} = 1 / \text{distance}(\text{Data}_j, \text{Center}_i)^2$$

$$\text{HiddenMatrix}_{ij} := \text{Force}_{i,j} / \sum_{\text{all centers } j} \text{Force}_{i,j}$$

Responsibilities and Statistical Mechanics



data

center
s

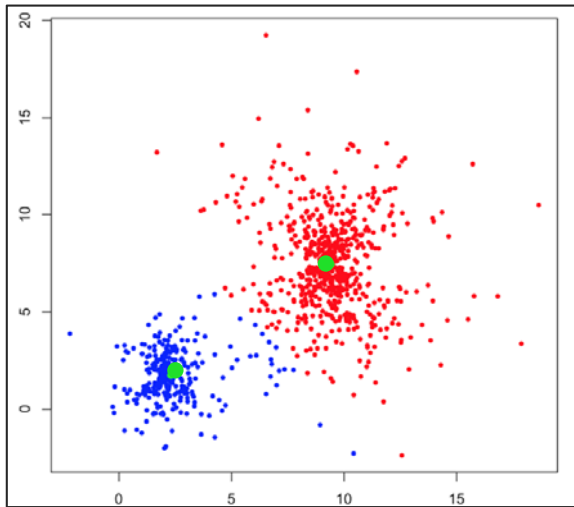
0.70	0.15	0.73	0.40	0.15	0.80	0.05	0.05
0.20	0.80	0.17	0.20	0.80	0.10	0.05	0.20
0.10	0.05	0.10	0.40	0.05	0.10	0.90	0.75

responsibility of center i for a data point j is proportional to

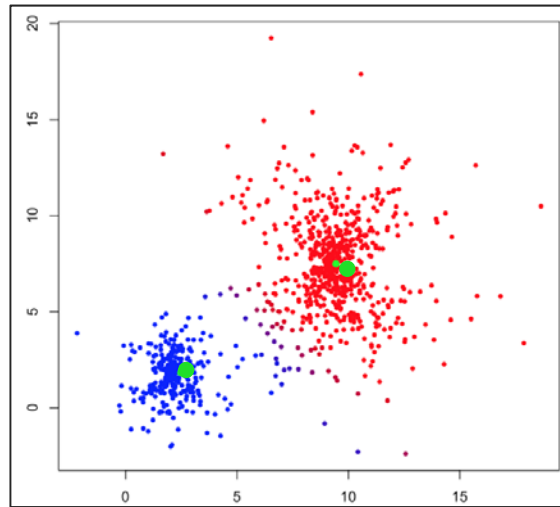
$$Force_{i,j} = e^{-\beta \cdot \text{distance}(\text{Data}_j, \text{Center}_i)}$$

$$\text{HiddenMatrix}_{ij} := \frac{Force_{i,j}}{\sum_{\text{all centers } j} Force_{i,j}}$$

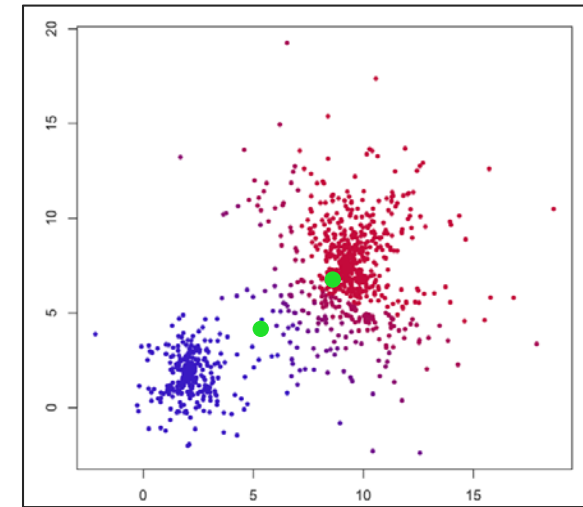
How Does Stiffness Affect Clustering?



Hard k -means
clustering



Soft k -means
clustering
(stiffness $\beta=1$)



Soft k -means
clustering
(stiffness $\beta=0.3$)

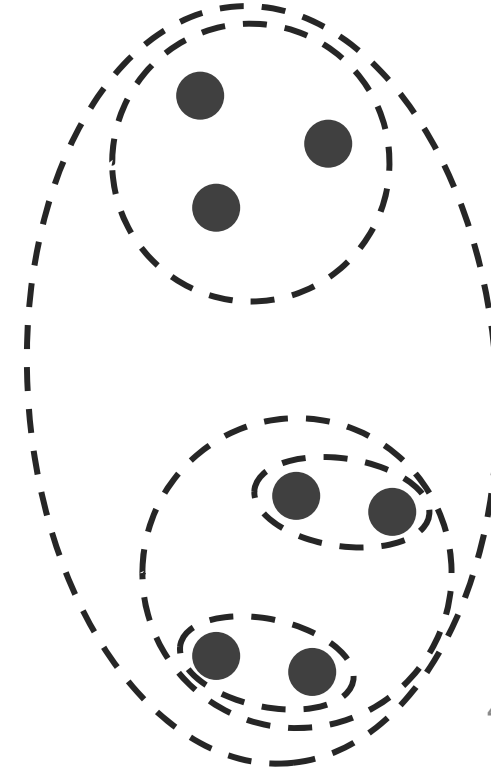
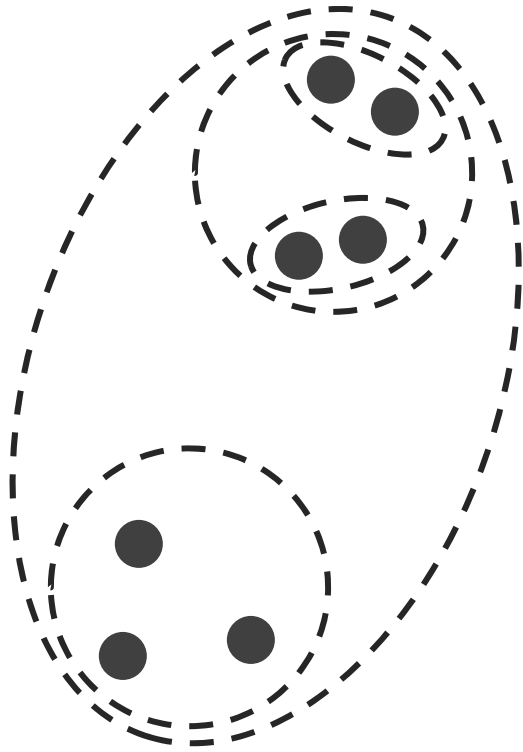
Stratification of Clusters

Clusters often have **subclusters**, which have subsubclusters, and so on.



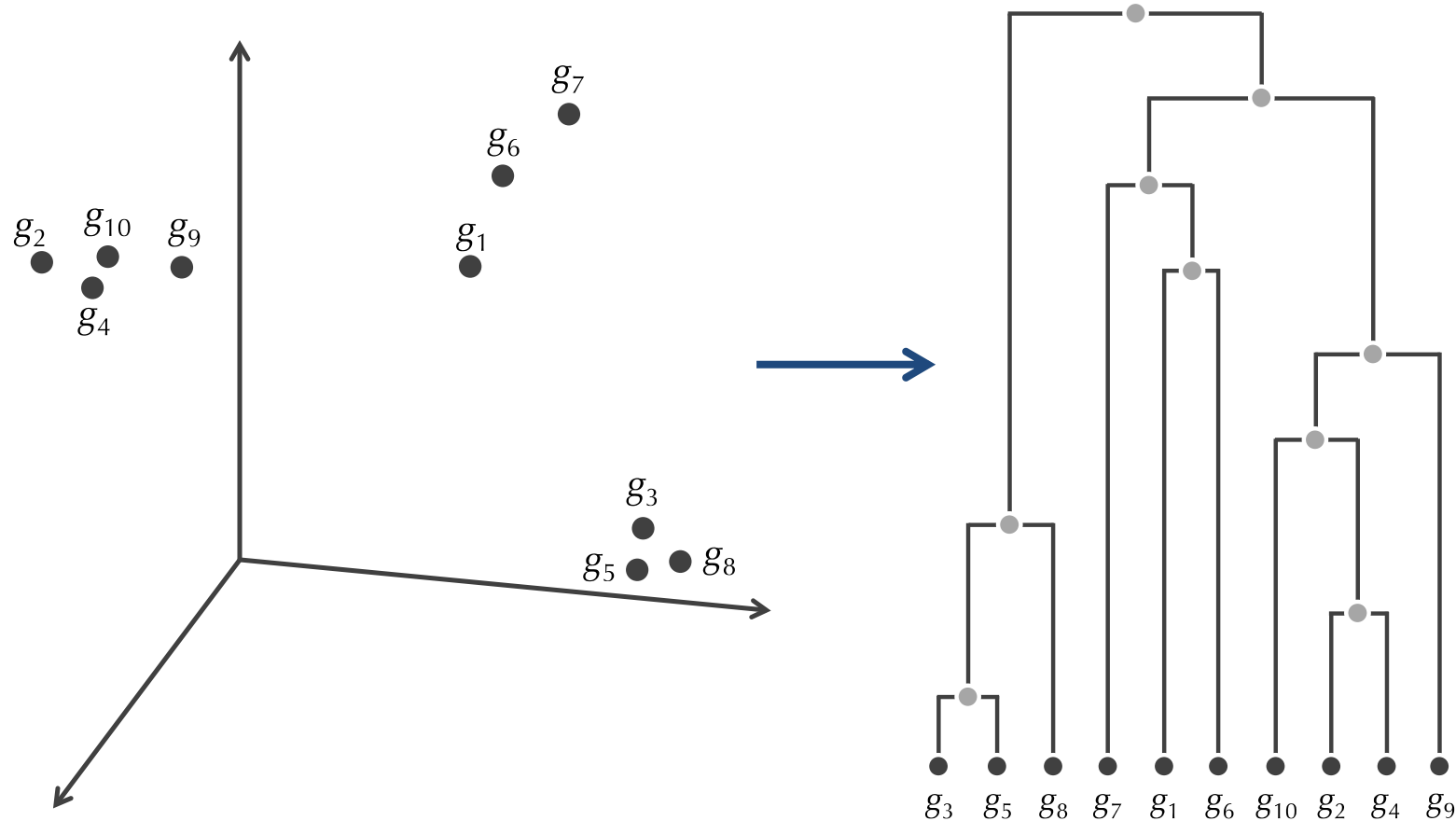
Stratification of Clusters

Clusters often have **subclusters**, which have sub-subclusters, and so on.



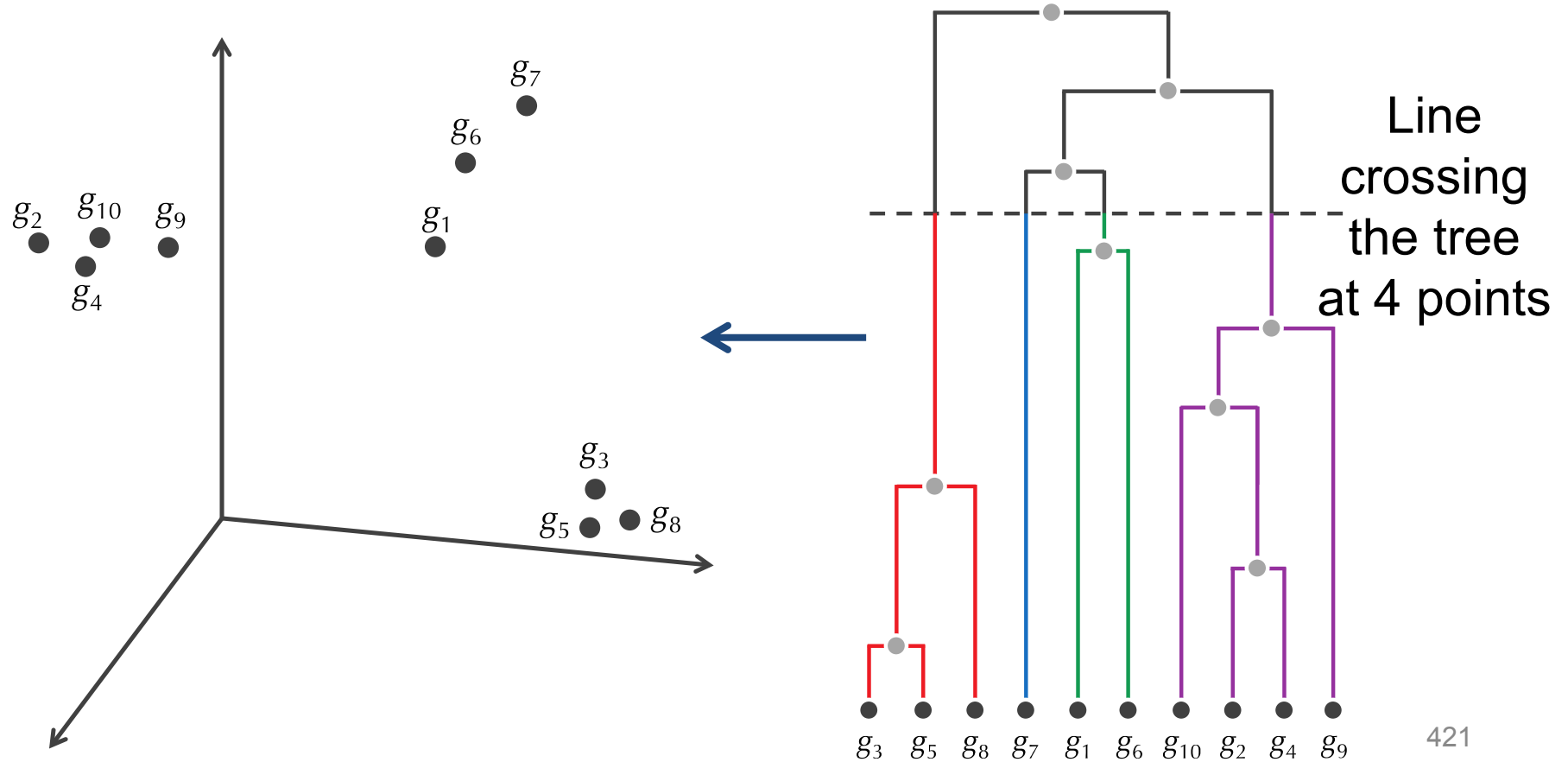
From Data to a Tree

To capture stratification, the **hierarchical clustering** algorithm organizes n data points into a tree.



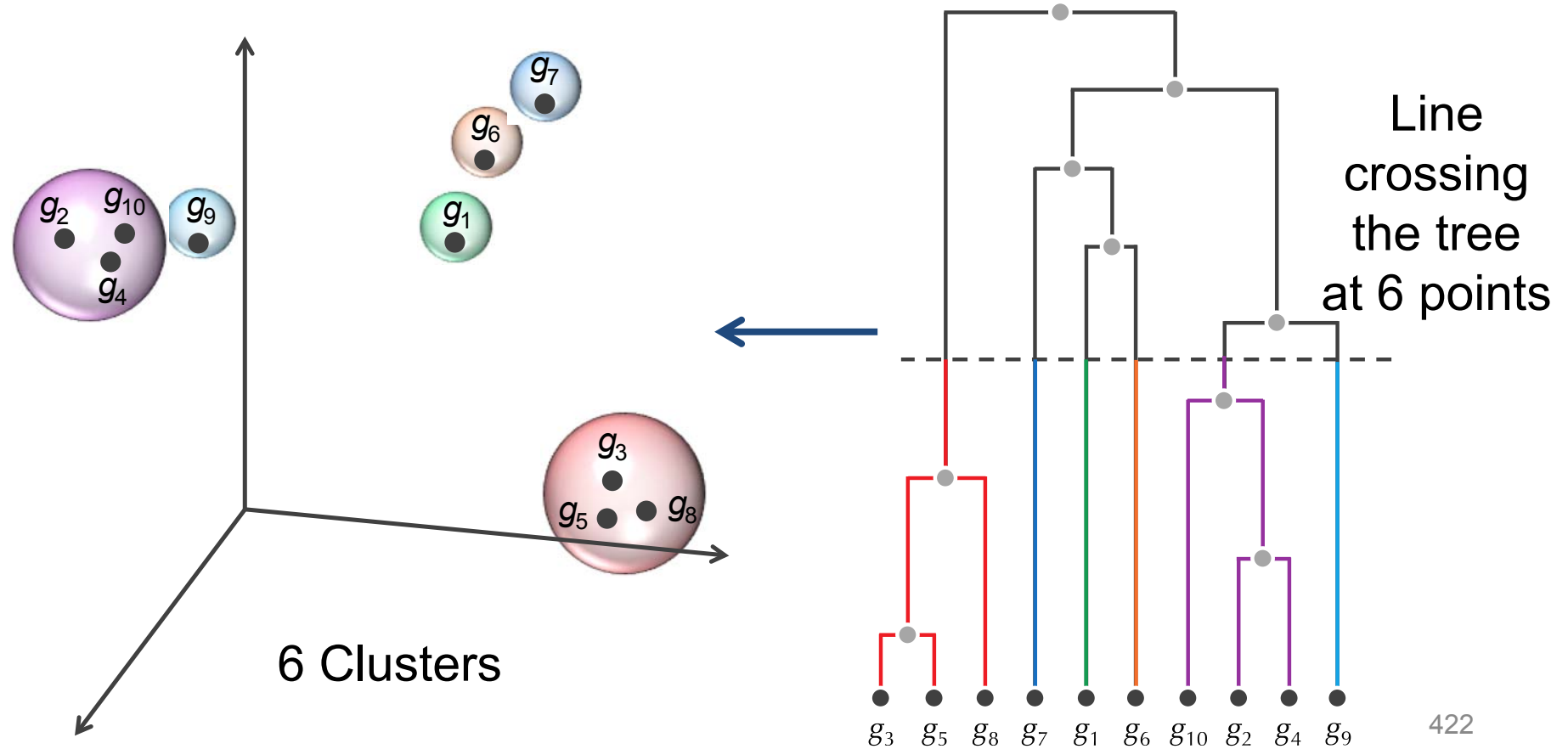
From a Tree to a Partition into 4 Clusters

To capture stratification, the **hierarchical clustering** algorithm organizes n data points into a tree.



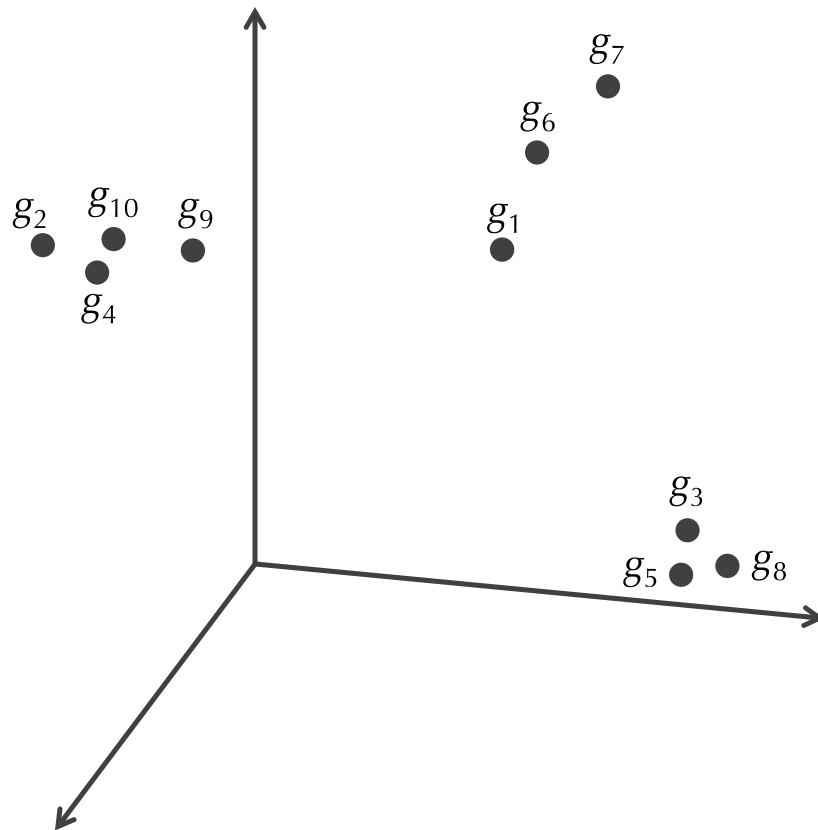
From a Tree to a Partition into 6 Clusters

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 $n \times m$ expression matrix into $n \times n$ **similarity matrix** or **distance matrix**.

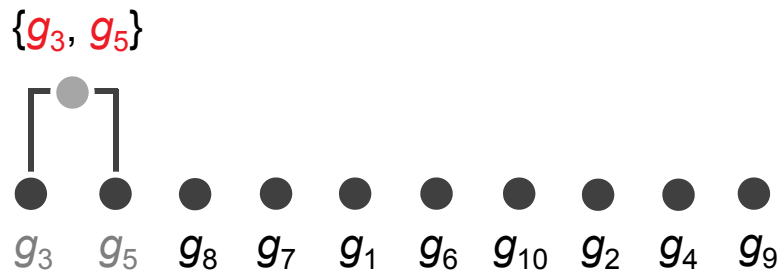


Distance Matrix

	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

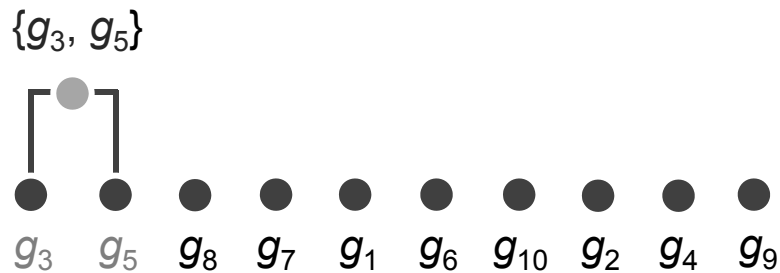
Identify the two **closest** clusters and merge them.



	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

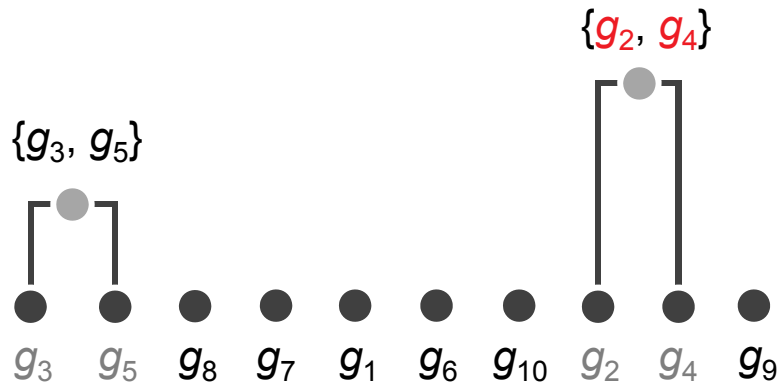
Recompute the distance between two clusters as average distance between elements in the cluster.



	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

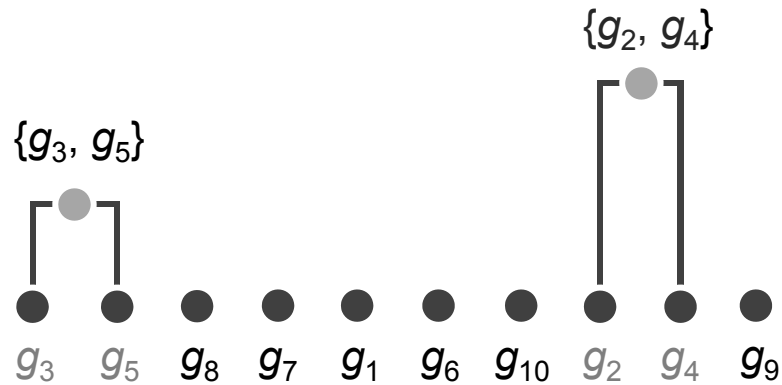
Identify the two **closest** clusters and merge them.



	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

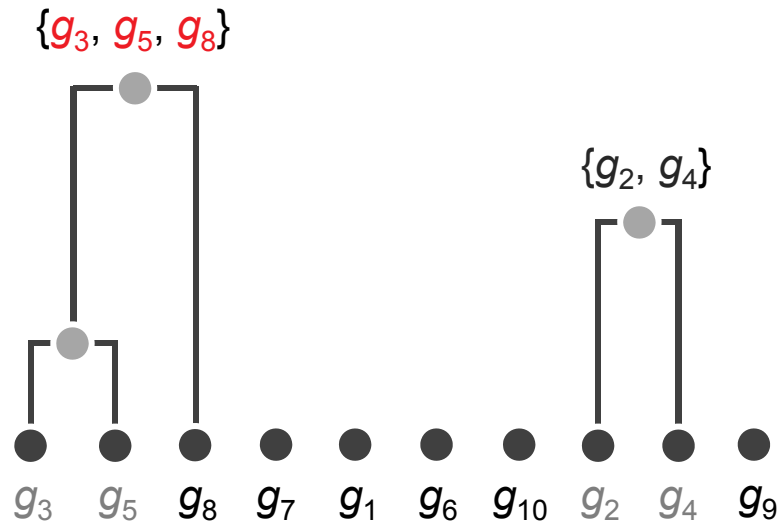
Recompute the distance between two clusters (as average distance between elements in the cluster).



	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

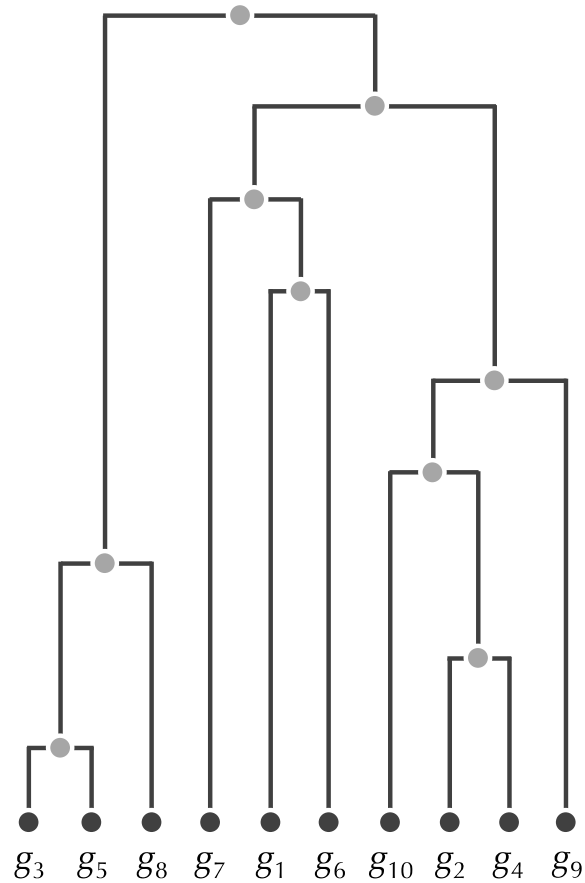
Identify the two **closest** clusters and merge them.



	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)

$Clusters \leftarrow n$ single-element clusters labeled 1 to n

$T \leftarrow$ 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

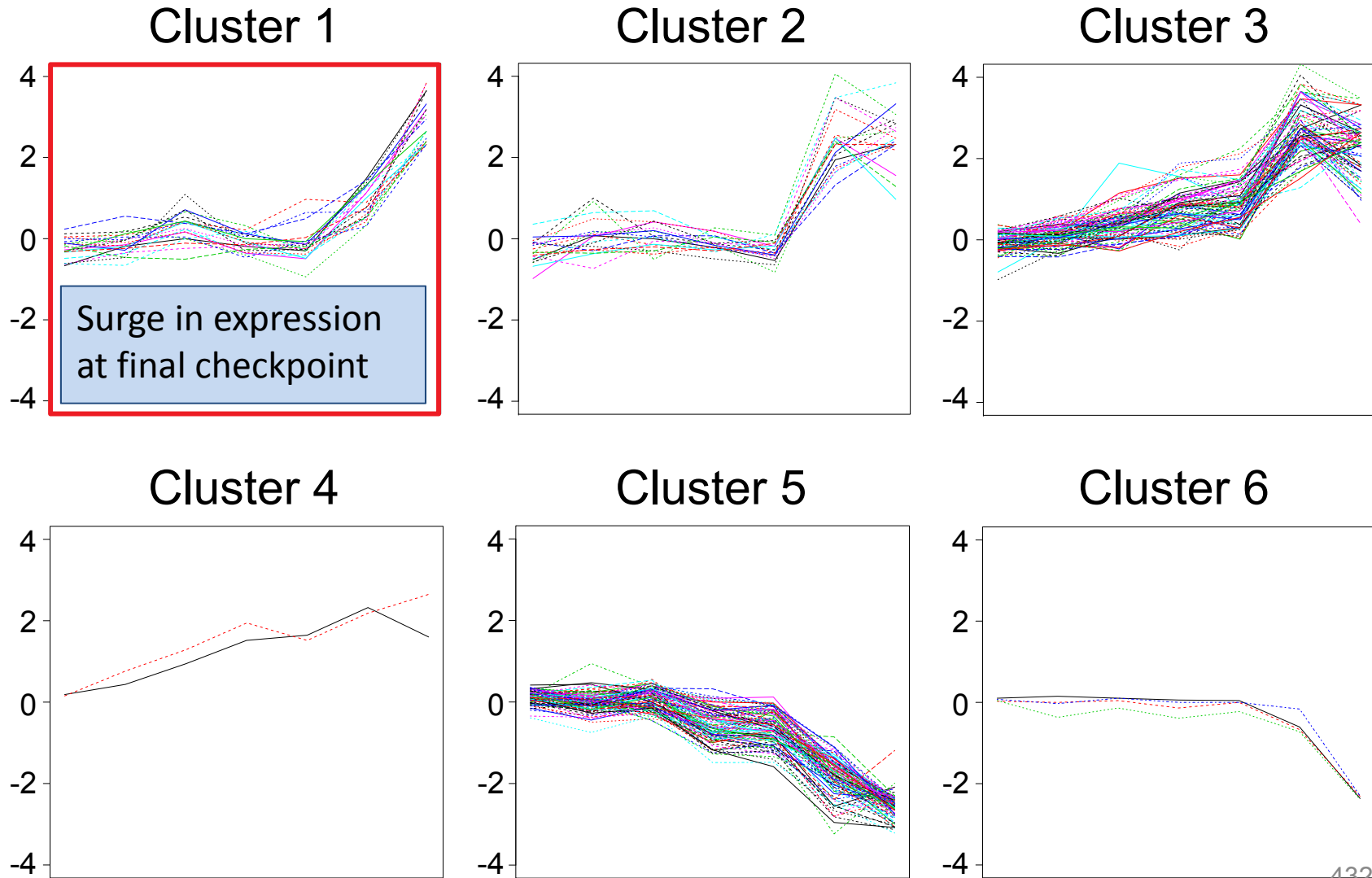
Average distance between elements of two clusters:

$$D_{\text{avg}}(C_1, C_2) = (\sum_{\text{all points } i \text{ and } j \text{ in clusters } C_1 \text{ and } C_2, \text{ respectively}} D_{i,j}) / (|C_1| * |C_2|)$$

Minimum distance between elements of two clusters:

$$D_{\text{min}}(C_1, C_2) = \min_{\text{all points } i \text{ and } j \text{ in clusters } C_1 \text{ and } C_2, \text{ respectively}} D_{i,j}$$

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.

Material and code at micans.org/mcl

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.

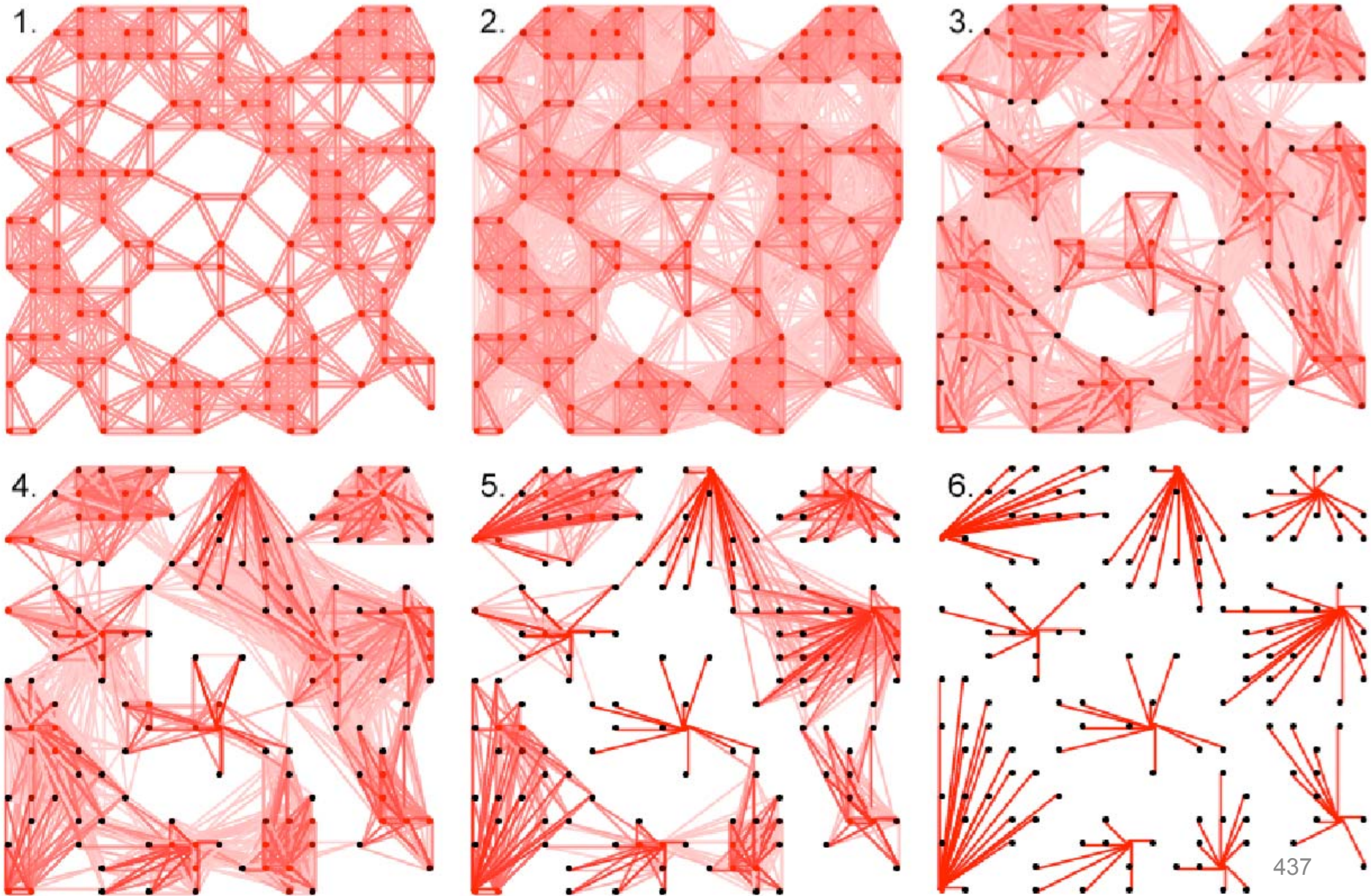
A random walk has a higher probability to stay inside the cluster than to leave it soon. The crucial point lies in boosting this effect by an iterative alternation of expansion and inflation steps. An inflation parameter is responsible for both strengthening and weakening of current.

(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

- 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

Genome Assembly Outline

- 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

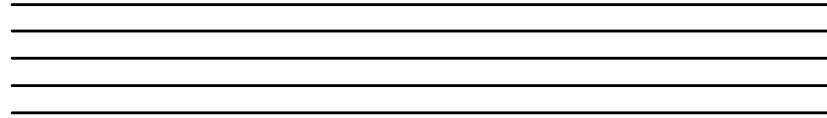
Individual

CTGAGGATGGACTACGCTACTACTGATAGCTGTTT

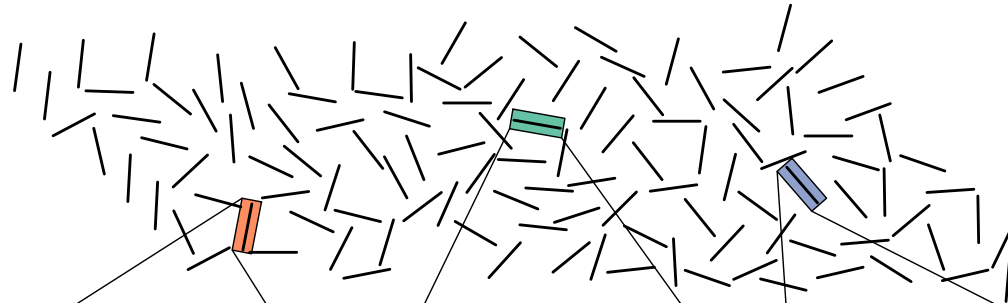
Reference

Why Not Use Assembly?

Multiple copies of a genome



Shatter the genome into reads



Sequence the reads

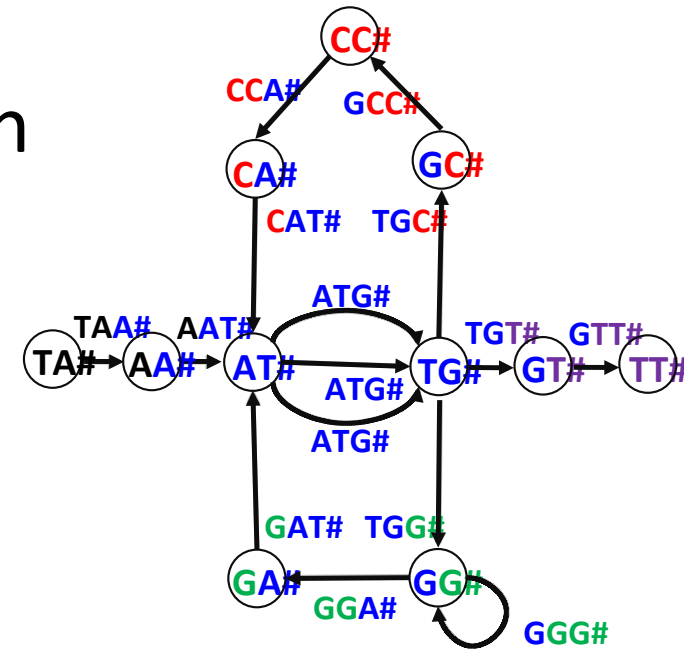


Assemble the genome with overlapping reads



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.

CTGAGGATGGACTACGCTACTACTGATAGCTGTTT Reference
GAGGA C**C**ACG TGA - A Reads

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*.

p a n a m a b a **n a n a** s *Genome*
 n a n a *Pattern*

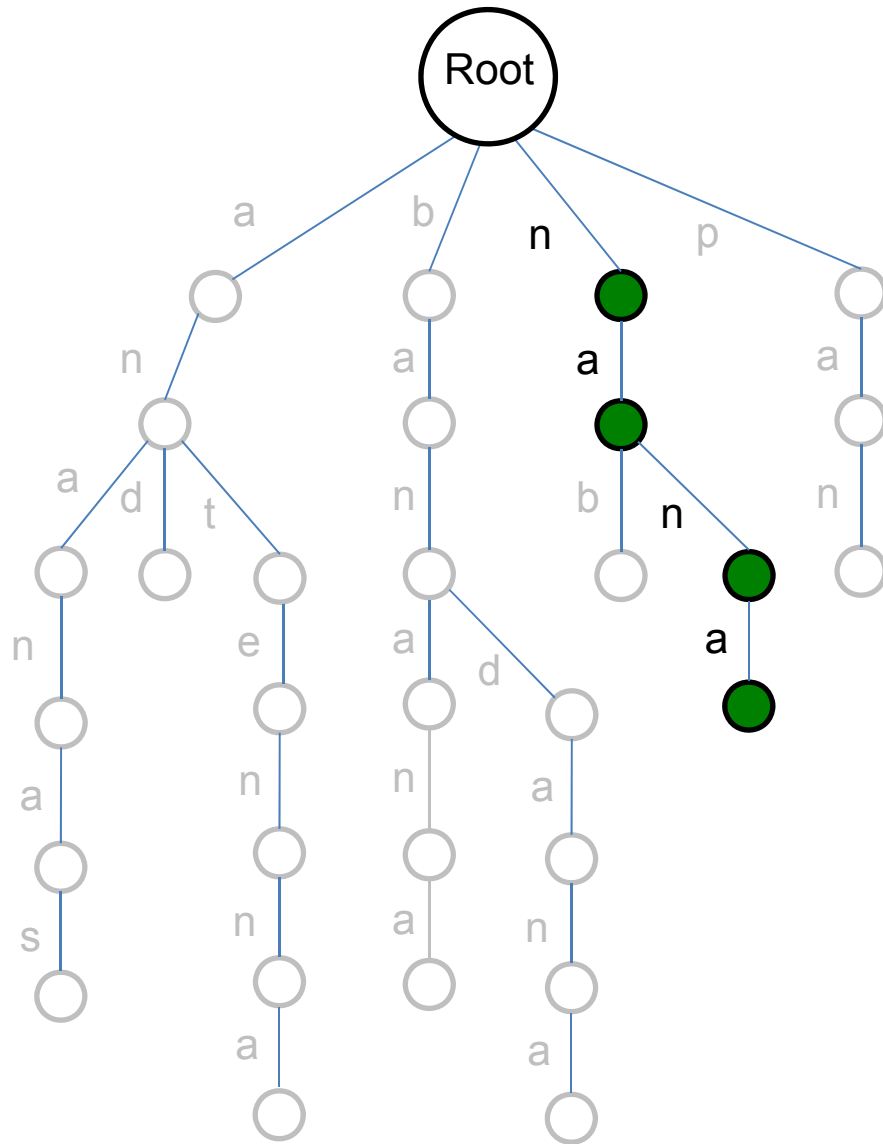
- **Note:** we use words instead of DNA strings for convenience.

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

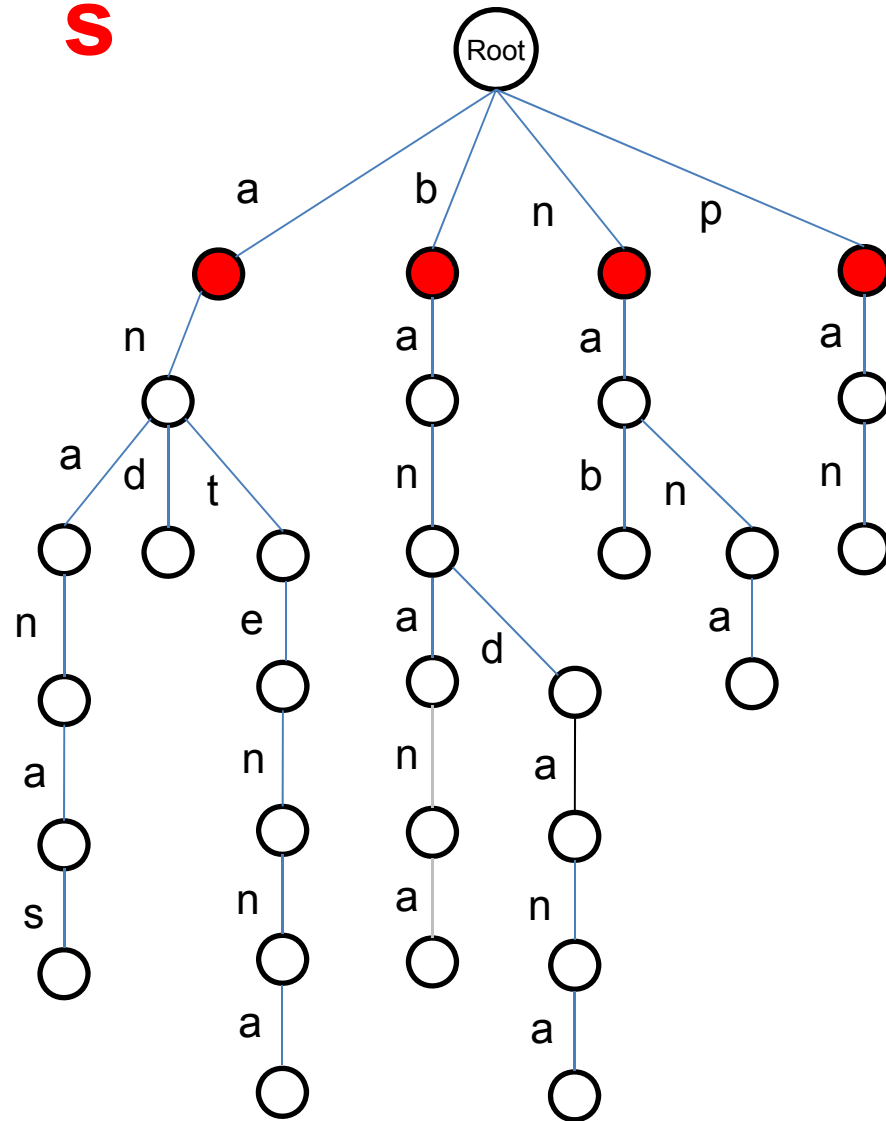
- banana
- pan
- and
- nab
- antenna
- bandana
- ananas
- nana**

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

panamabana

S

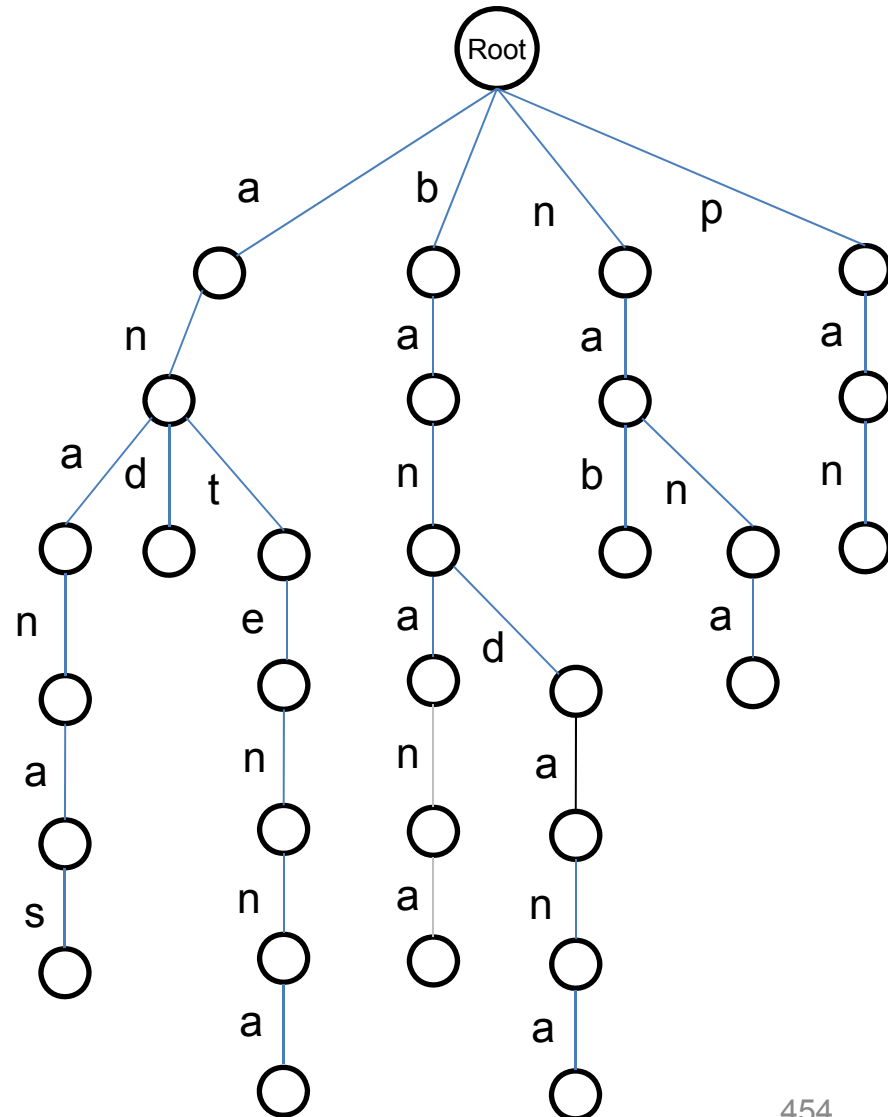


Success!

- Runtime of Brute Force:
 - Total: $O(|Genome| * |Patterns|)$
- Runtime of Trie Matching:
 - Trie Construction: $O(|Patterns|)$
 - Pattern Matching: $O(|Genome| * |LongestPattern|)$

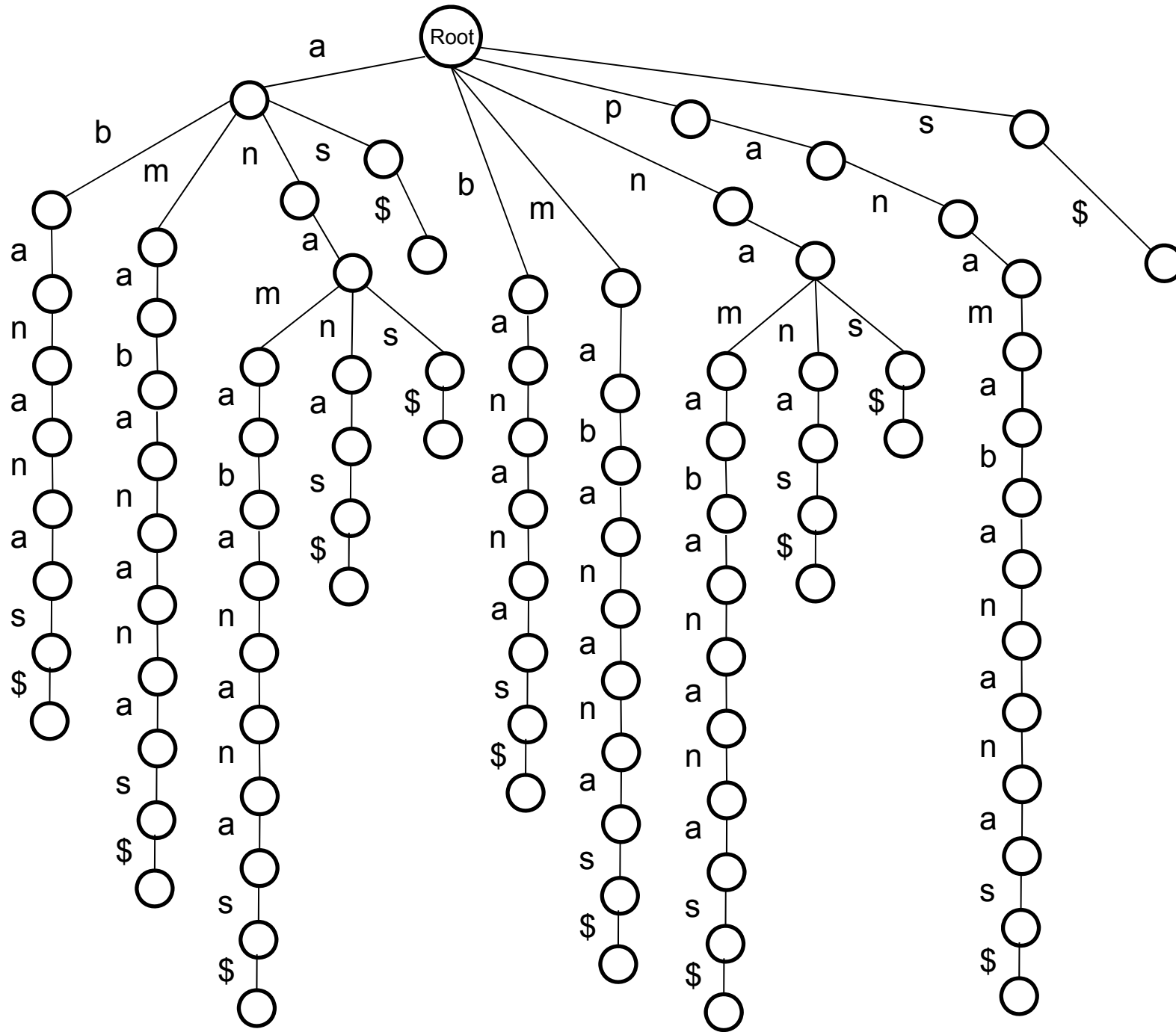
Memory Analysis of TrieMatching

- Son completely forgot about memory!
- Our trie: 30 edges, $|Patterns| = 39$
- Worst case: # edges = $O(|Patterns|)$



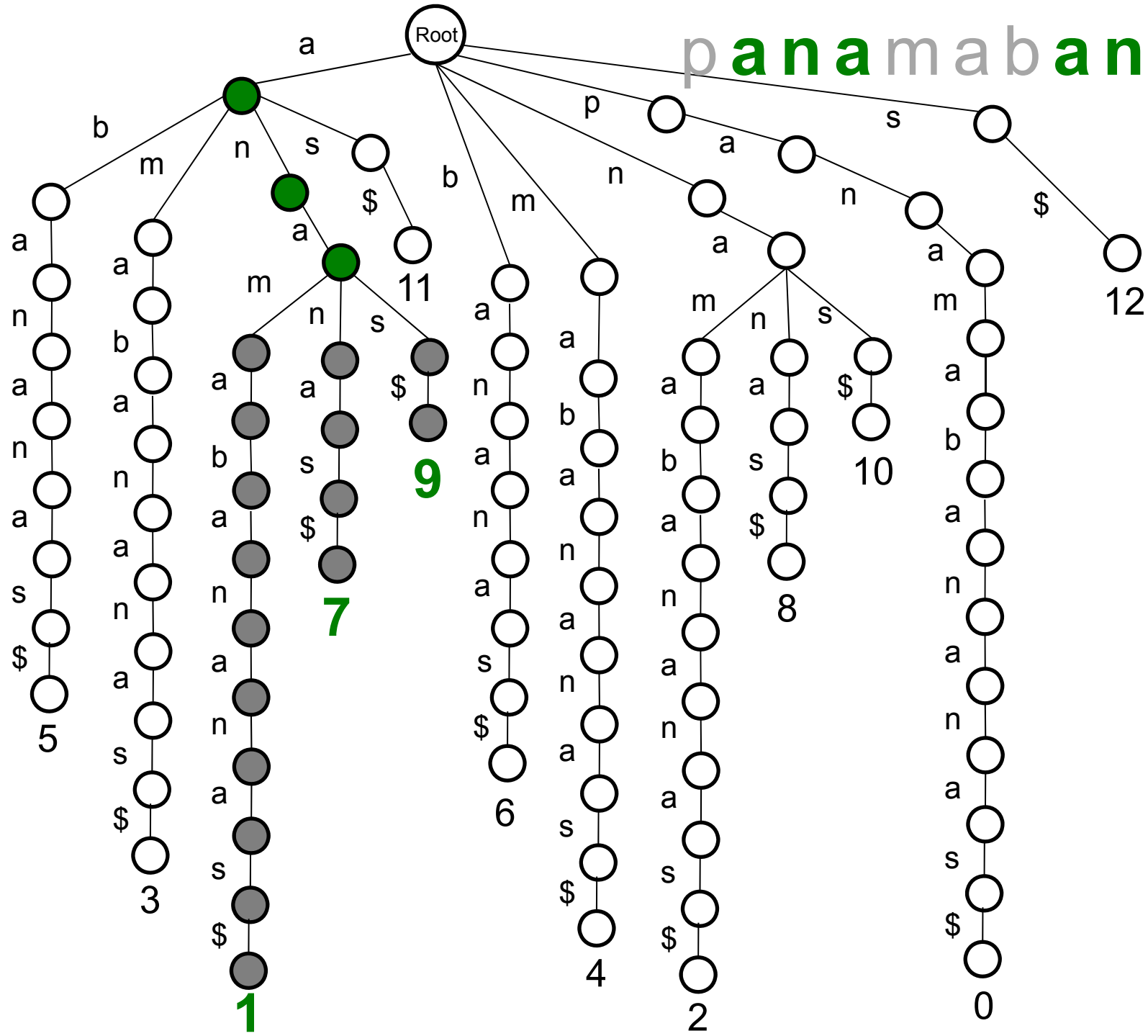
Preprocessing the Genome

- 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!



The Suffix Trie and Pattern Matching

- For each *Pattern*, see if *Pattern* can be spelled out from the root downward in the suffix trie.



panamabanna

s \$

Memory Trouble Once Again

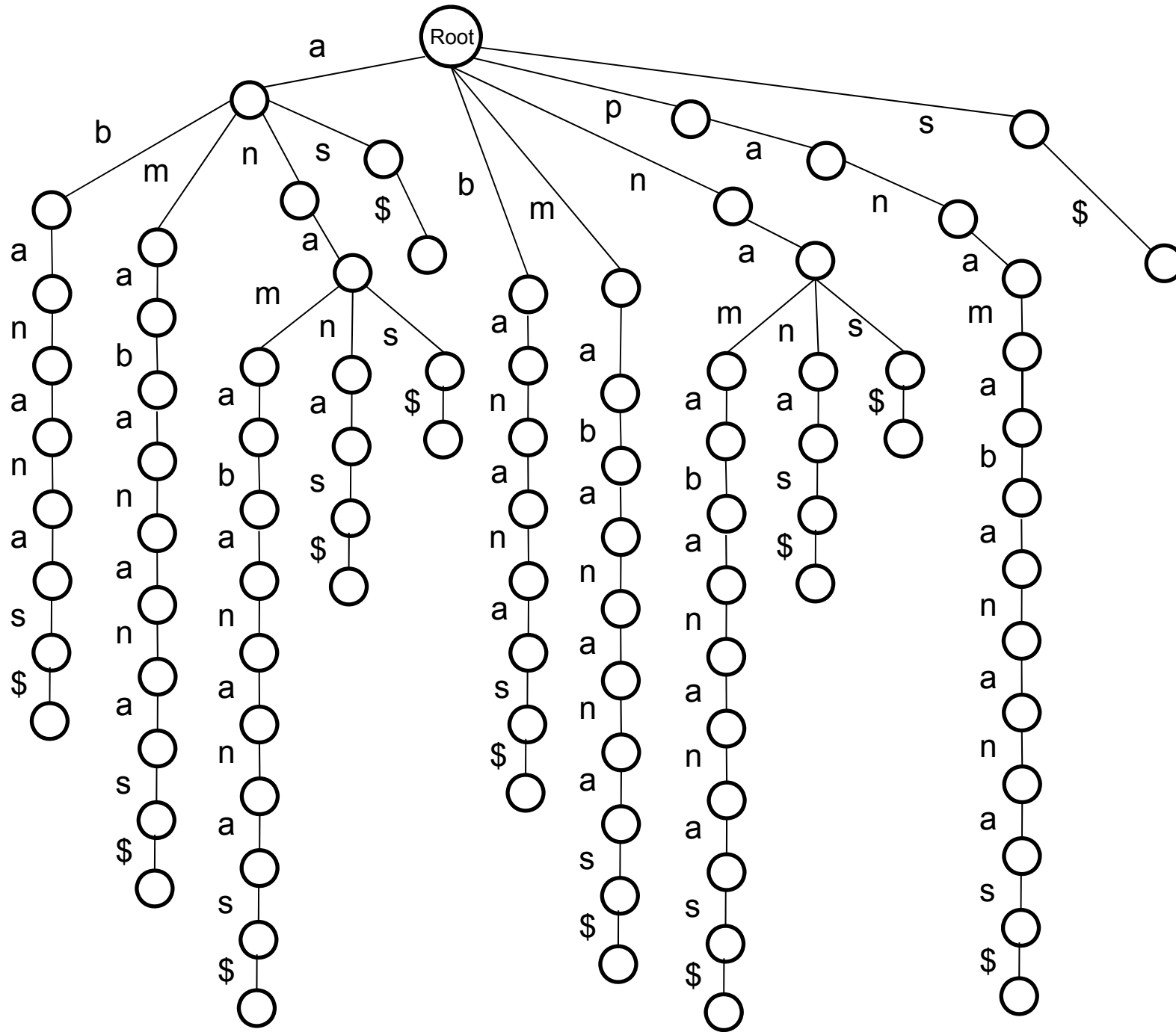
- Worst case: the suffix trie holds $O(|\textit{Suffixes}|)$ nodes.
- For a *Genome* of length n ,
 $|\textit{Suffixes}| = n(n - 1)/2 = O(n^2)$

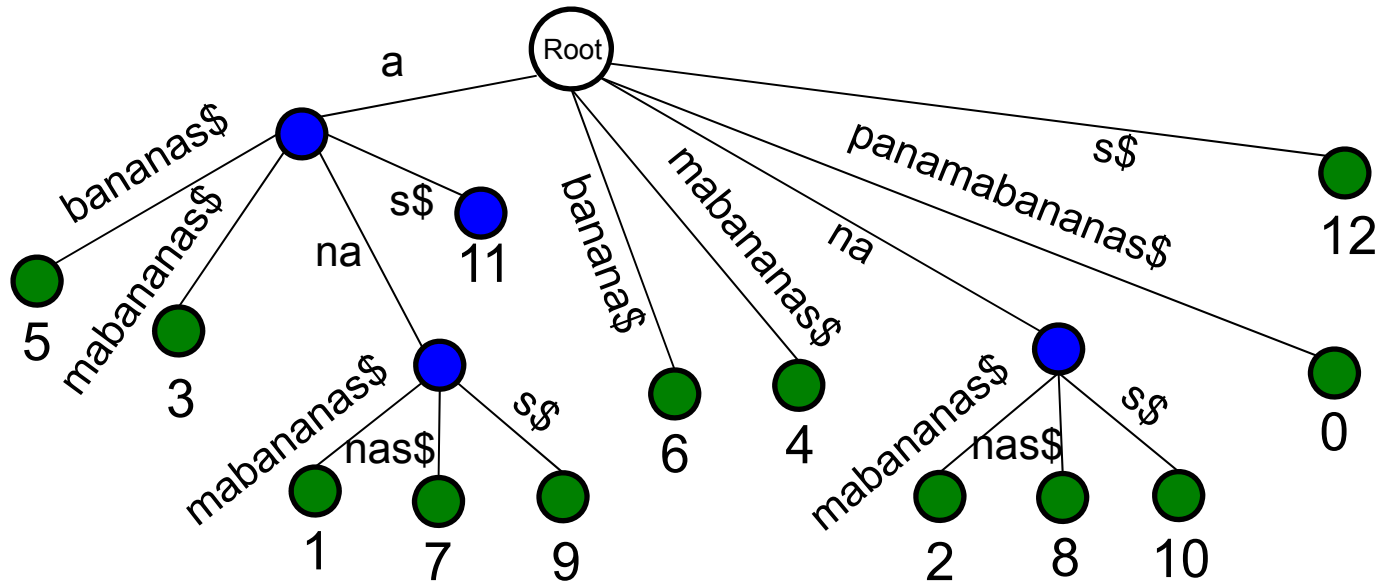
Suffixes

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.





- 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

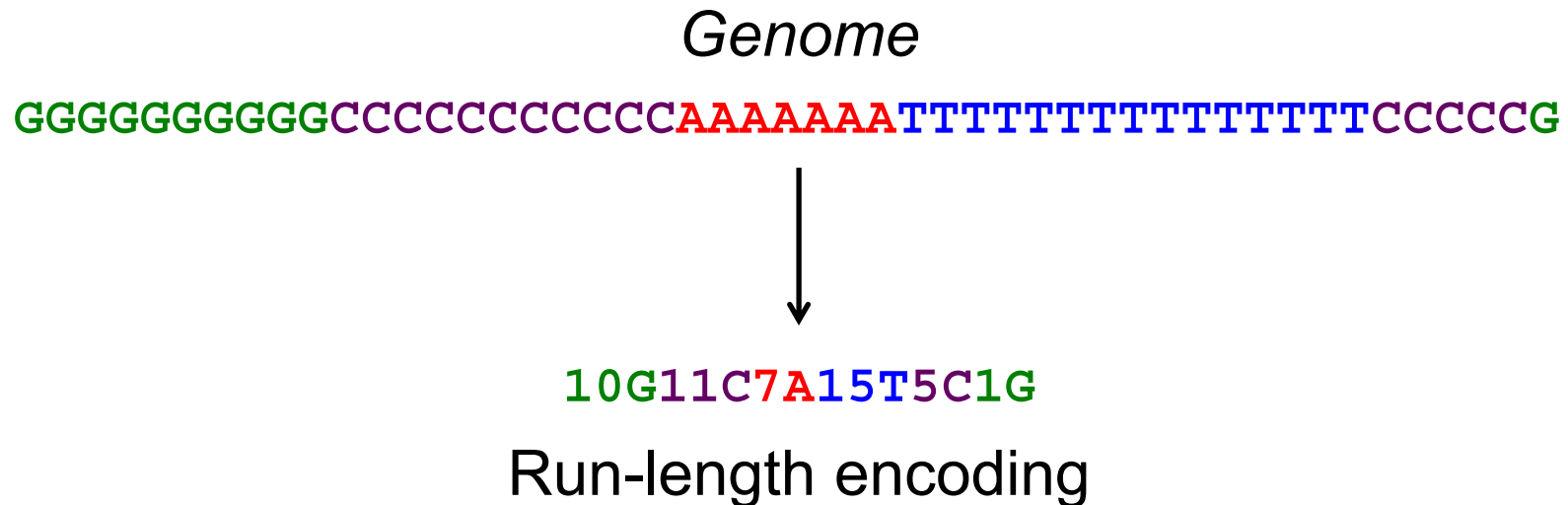
- I am happy with the suffix tree, but I am not completely satisfied.
 - Runtime: $O(|Genome| + |Patterns|)$
 - Memory: $O(|Genome|)$
- However, big-O notation ignores constants!
 - The best known suffix tree implementations require ~ 20 times the length of $|Genome|$.
 - Can we reduce this constant 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

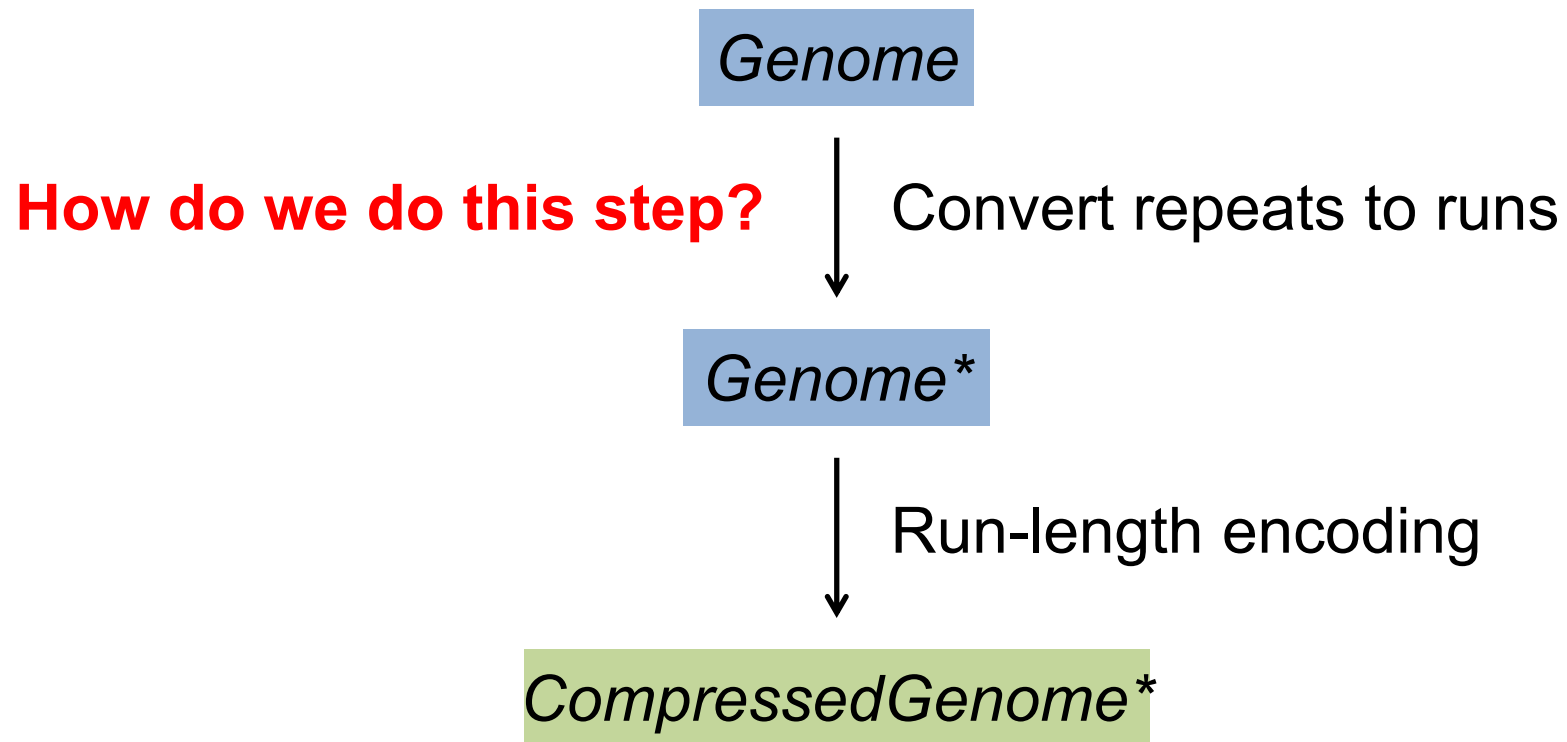
- **Run-length encoding:** compresses a run of n identical symbols.



- Problem: Genomes don't have lots of runs...

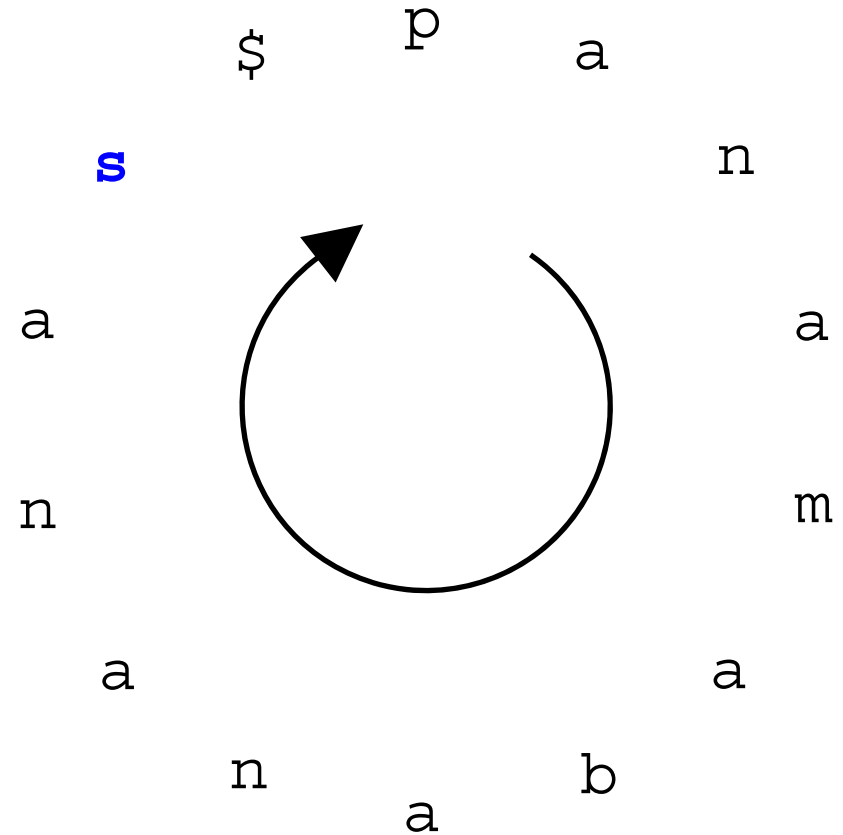
Converting Repeats to Runs

- ...but they do have lots of repeats!



The Burrows-Wheeler Transform

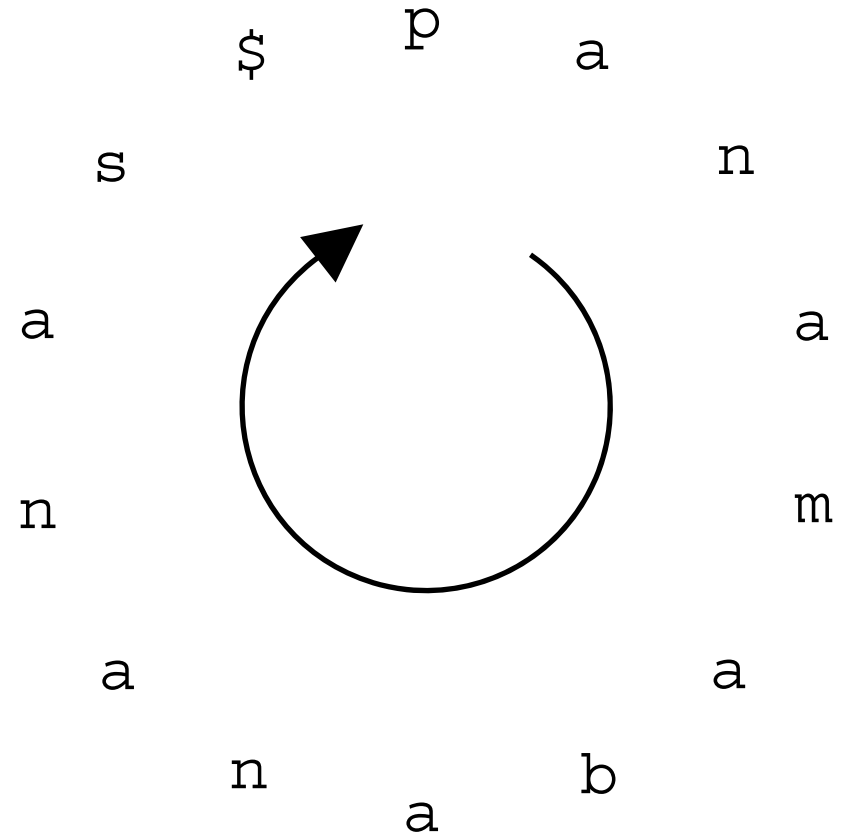
panamabananas\$
\$panamabananas
s\$panamabanana



Form all cyclic rotations of
"panamabananas\$"

The Burrows-Wheeler Transform

panamabananas\$
\$panamabananas
s\$panamabanana
as\$panamabanana
nas\$panamabana
anas\$panamaba
nanas\$panamaba
ananas\$panamab
bananas\$panama
abananas\$panam
mabananas\$pana
amabananas\$pan
namabananas\$pa
anamabananas\$pa



Form all cyclic rotations of
“panamabananas\$”

The Burrows-Wheeler Transform

panamabananas\$
\$panamabananas
s\$panamabanana
as\$panamabanana
nas\$panamabana
anas\$panamaban
nanas\$panamaba
ananas\$panamab
bananas\$panama
abananas\$panam
mabananas\$pana
amabananas\$pan
namabananas\$pa
anamabananas\$p



\$panamabananas
abananas\$panam
amabananas\$pan
anamabananas\$p
ananas\$panamab
anas\$panamaban
as\$panamaban
bananas\$panama
mabananas\$pana
namabananas\$pa
nanas\$panamaba
nas\$panamabana
panamabananas\$
s\$panamabanana

Form all cyclic rotations of
“panamabananas\$”

Sort the strings
lexicographically
(\$ comes first)

The Burrows-Wheeler Transform

panamabananas\$
 \$panamabananas
 s\$panamabanana
 as\$panamabanana
 nas\$panamabana
 anas\$panamaban
 nanas\$panamaba
 ananas\$panamab
 bananas\$panama
 abananas\$panam
 mabananas\$pana
 amabananas\$pan
 namabananas\$pa
 anamabananas\$pa



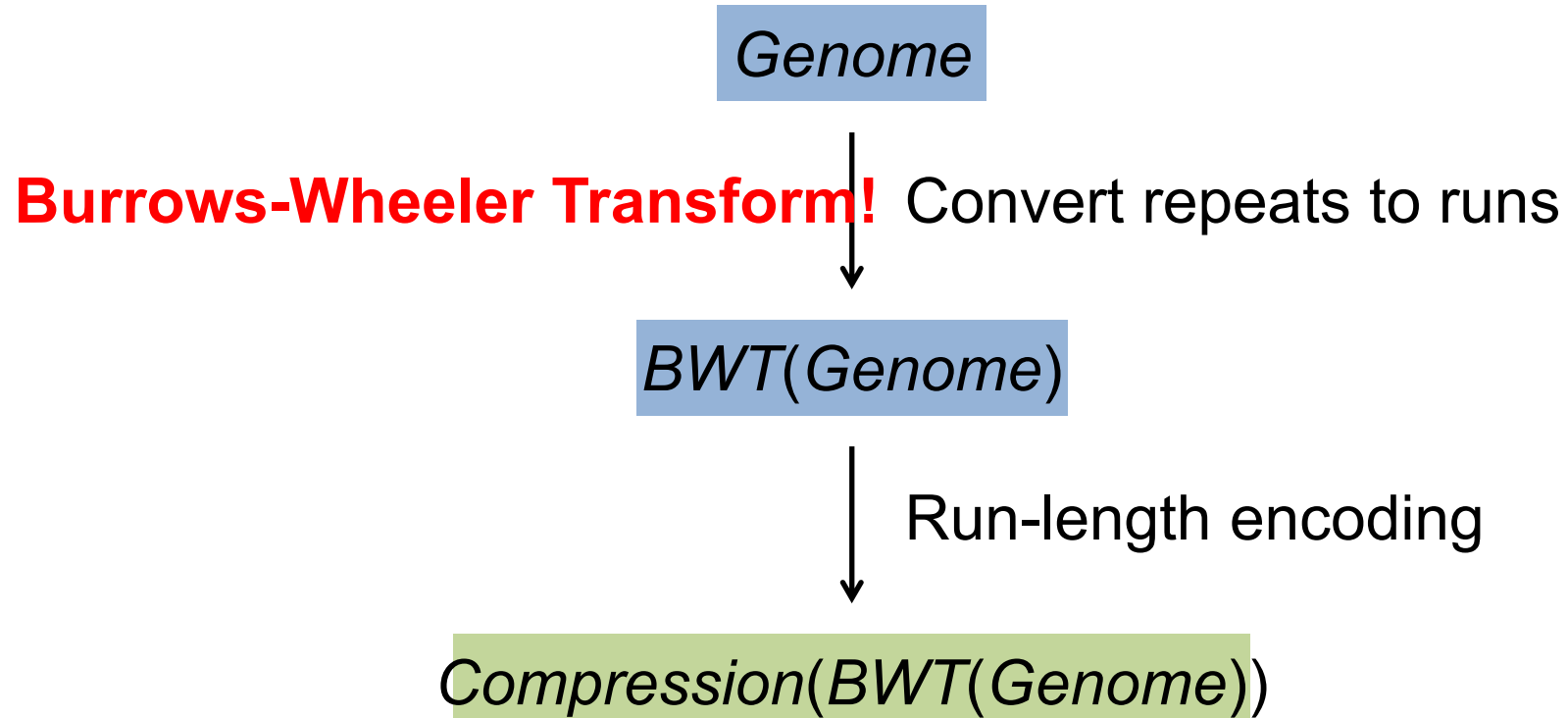
\$panamabanana**s**
 abananas\$pana**m**
 amabananas\$pan**n**
 anamabananas\$**p**
 ananas\$panama**b**
 anas\$panamaba**n**
 as\$panamabana**n**
 bananas\$panama**a**
 mabananas\$pana**a**
 namabananas\$pa**a**
 nanas\$panamaba**a**
 nas\$panamabana**a**
 panamabananas\$**s**
 s\$panamabanana**a**

Form all cyclic rotations of
“panamabananas\$”

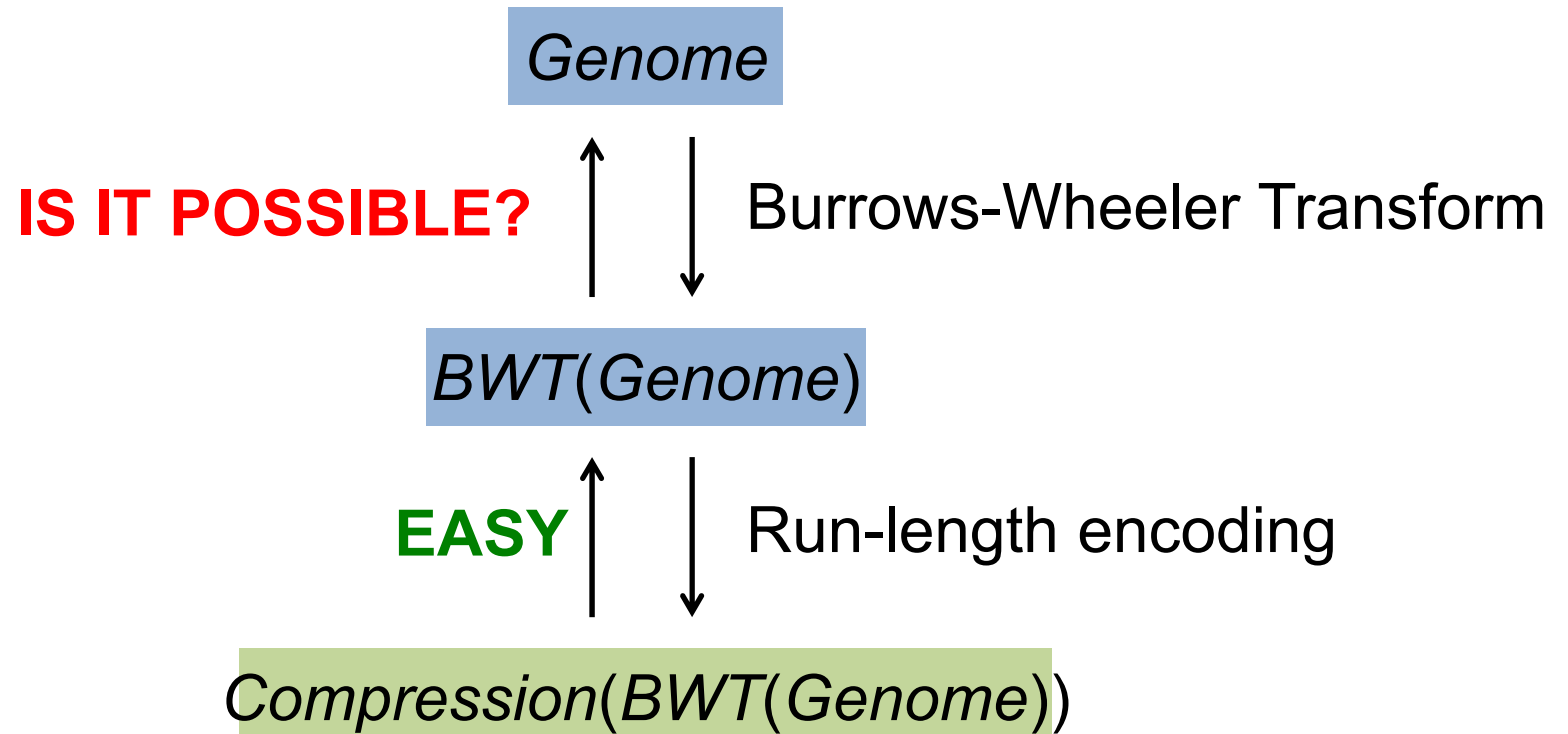
**Burrows-Wheeler
Transform:**

Last column =
smnpbnnaaaaa\$a

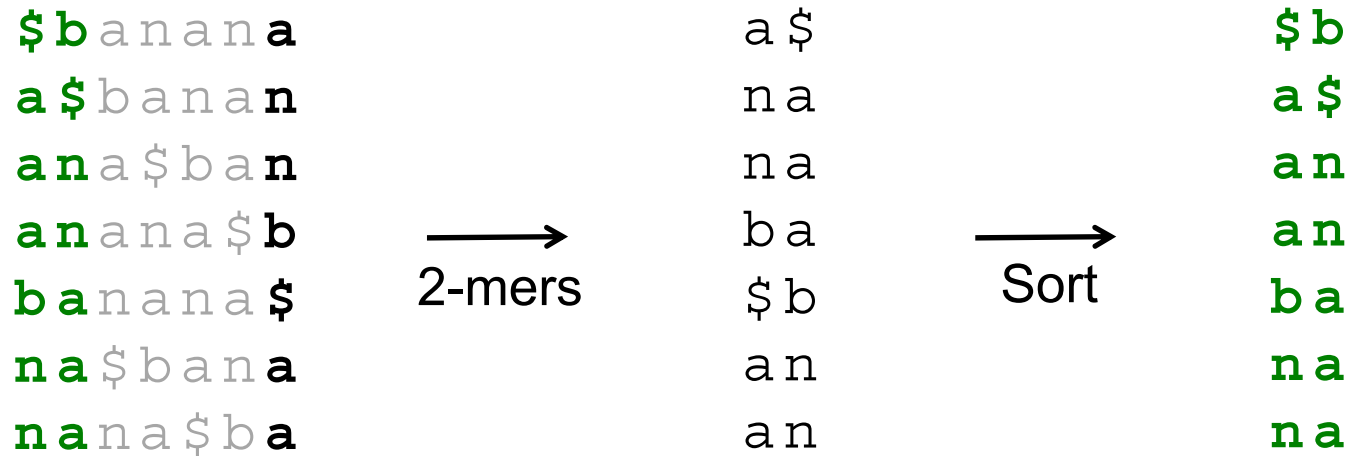
BWT: Converting Repeats to Runs



How Can We Decompress?

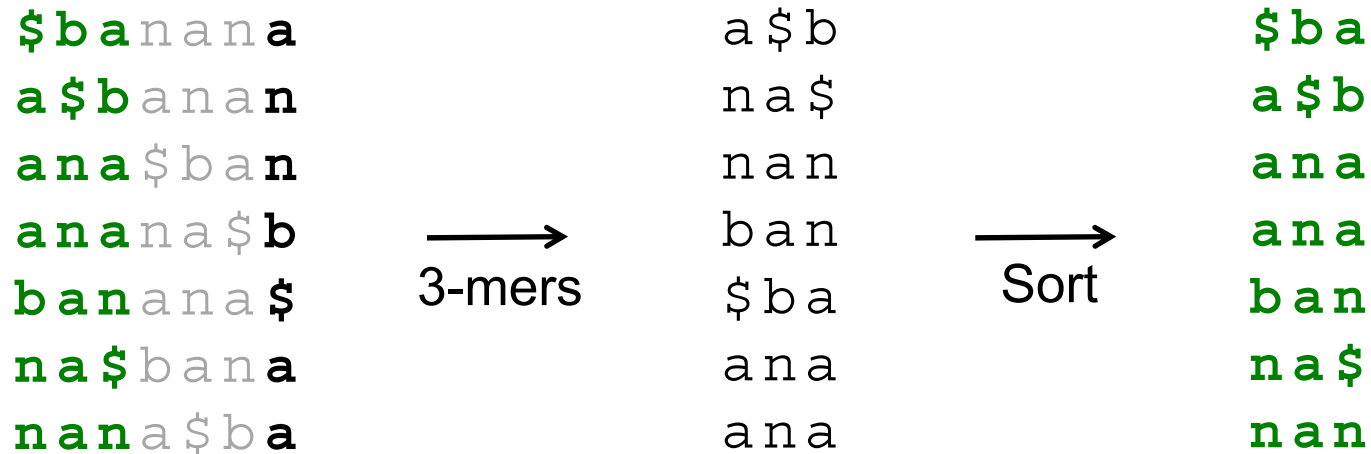


Reconstructing banana



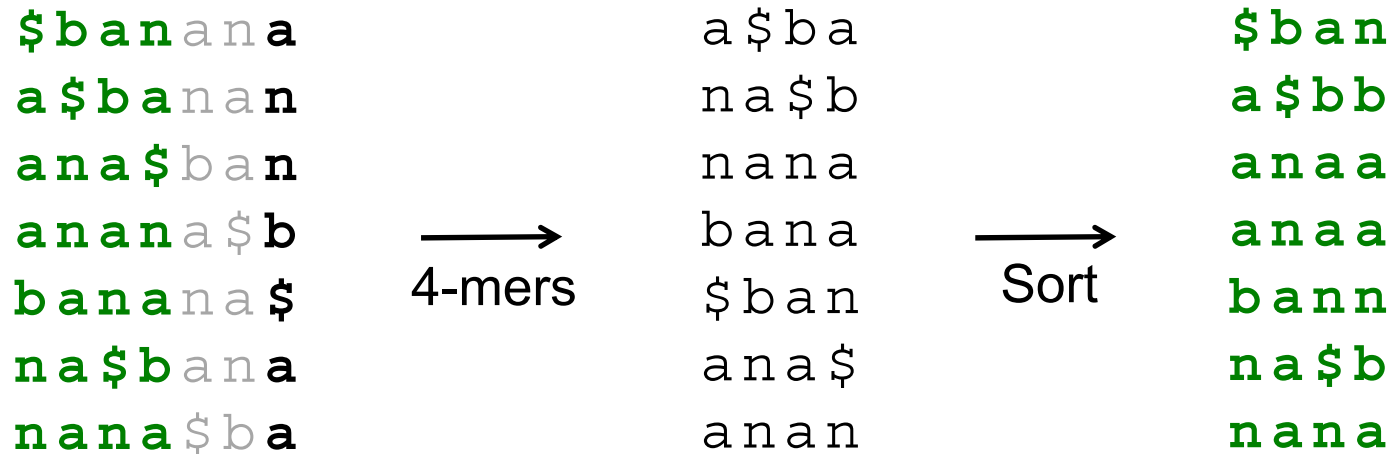
- We now know 2-mer composition of the circular string banana\$
- Sorting gives us the first 2 columns of the matrix.

Reconstructing banana



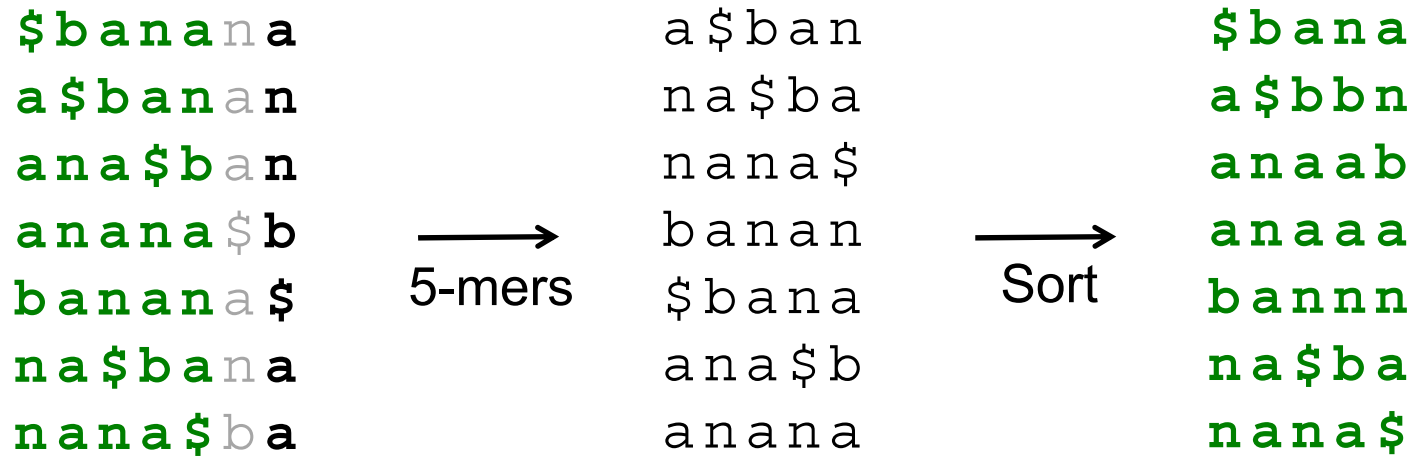
- We now know 3-mer composition of the circular string banana\$
- Sorting gives us the first 3 columns of the matrix.

Reconstructing banana



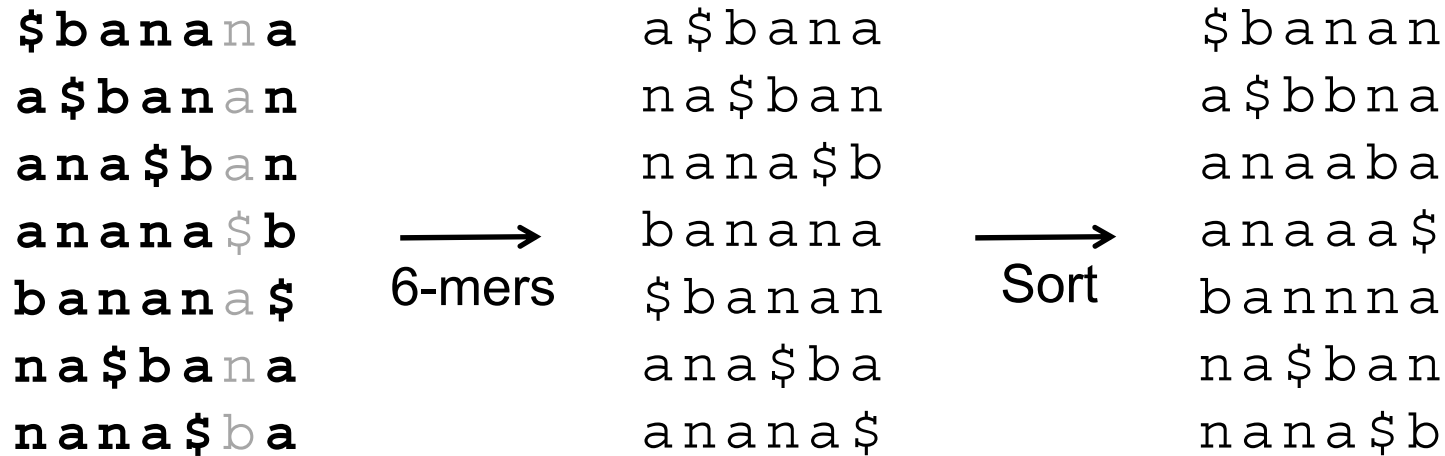
- We now know 4-mer composition of the circular string banana\$
- Sorting gives us the first 4 columns of the matrix.

Reconstructing banana



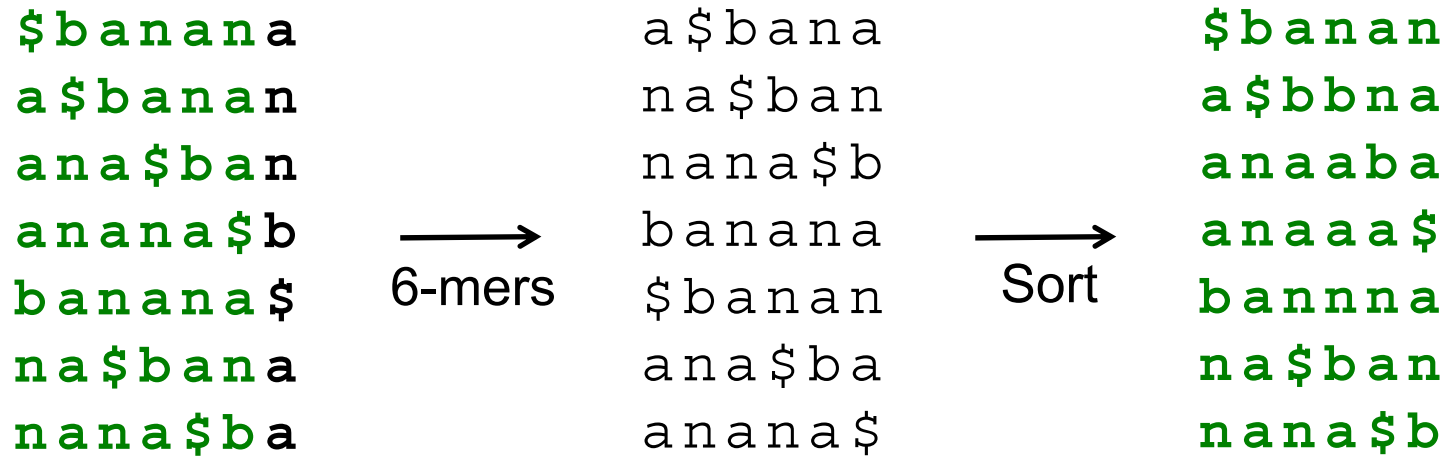
- We now know 5-mer composition of the circular string `banana$`
- Sorting gives us the first 5 columns of the matrix.

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

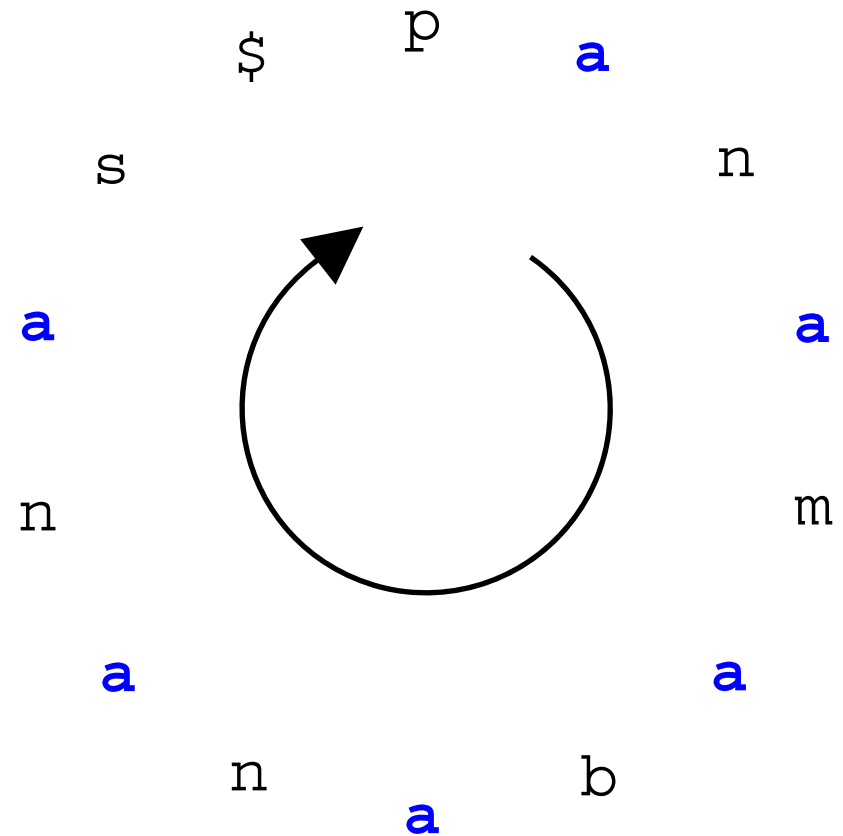
- Reconstructing *Genome* from *BWT(Genome)* required us to store $|Genome|$ copies of $|Genome|$.

```
$banana  
a$banan  
ana$ban  
anana$b  
banana$  
na$bana  
nana$ba
```

- Can we invert BWT with less space?

A Strange Observation

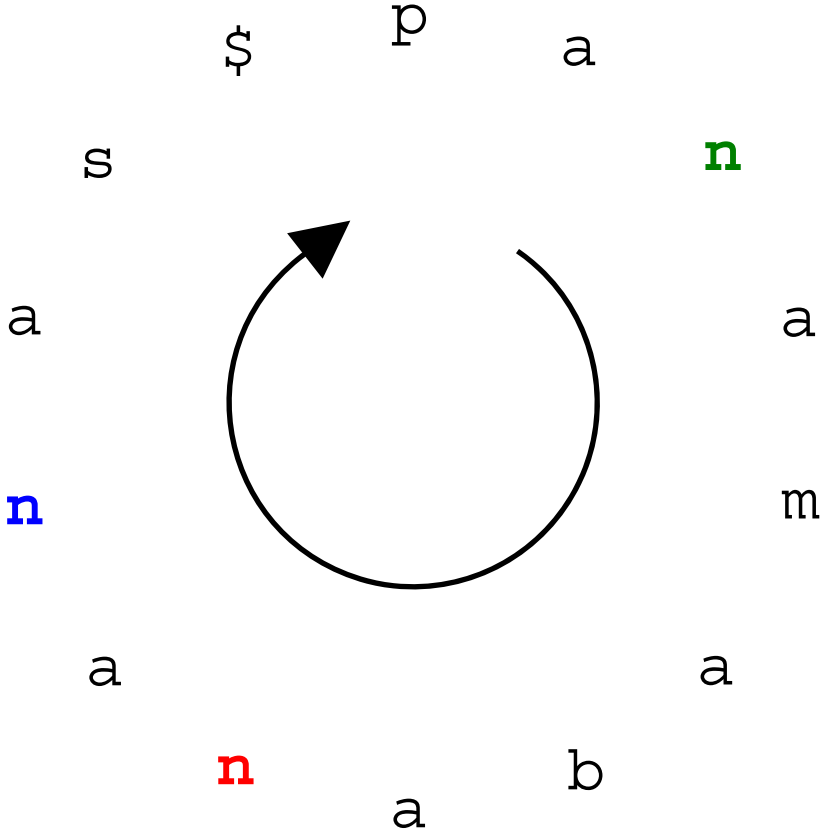
\$panamabananas
abananas\$panam
amabananas\$pan
anamabananas\$p
ananas\$panamab
anas\$panamaban
as\$panamabanan
bananas\$panama
mabananas\$pana**a**
namabananas\$pa**a**
nanas\$panamaba**a**
nas\$panamabana**a**
panamabananas\$
s\$panamabanana**a**



A Strange Observation


```

$panamabananas
abananas$panam
amabananas$pan
anamabananas$p
ananas$panamab
anas$panamaba
as$panamabana
bananas$panama
mabananas$pana
namabananas$pa
nanas$panamaba
nas$panamabana
panamabananas$
s$panamabanana
    
```



Is It True in General?

\$panamabananas
1 a bananasspanam
2 amabananas\$pan
3 anabananas\$pan
4 ananasspanamab
5 anasspanamaban
6 as\$panamabanan
bananasspanama
mabananas\$pana
namabananas\$pan
nanasspanamaba
nasspanamabana
panamabananas\$
spanamabana



Chop off a

bananasspanam
mabananas\$pan
namabananas\$pan
nanasspanamab
nasspanamaban
spanamabanan

These strings are sorted

Is It True in General?

\$panamabananas
1 a bananasspanam
2 amabananas\$pan
3 anabananas\$pan
4 ananasspanamab
5 anasspanamaban
6 as\$panamabanan
bananasspanama
mabananas\$pana
nabananas\$pana
nanasspanamaba
nasspanamabana
panamabananas\$
spanamabana


Chop off a

bananasspanam
mabananas\$pan
nabananas\$pan
nanasspanamab
nasspanamaban
spanamabanan

Still
sorted

These strings are sorted

Is It True in General?

\$panamabananas
1 a b a n a n a s \$ p a n a m
2 a m a b a n a n a s \$ p a n
3 a n a m a b a n a n a s \$ p
4 a n a n a s \$ p a n a m a b
5 a n a s \$ p a n a m a b a n
6 a s \$ p a n a m a b a n a n
b a n a n a s \$ p a n a m a
m a b a n a n a s \$ p a n a 1
n a m a b a n a n a s \$ p a 2
n a n a s \$ p a n a m a b a 3
n a s \$ p a n a m a b a n a 4
p a n a m a b a n a n a s \$ 5
s \$ p a n a m a b a n a n a 6

Chop off **a**

Ordering
doesn't
change!

b a n a n a s \$ p a n a m
m a b a n a n a s \$ p a n
n a m a b a n a n a s \$ p
n a n a s \$ p a n a m a b
n a s \$ p a n a m a b a n
s \$ p a n a m a b a n a n

Still
sorted

Add **a**
to end

b a n a n a s \$ p a n a m a
m a b a n a n a s \$ p a n a
n a m a b a n a n a s \$ p a
n a n a s \$ p a n a m a b a
n a s \$ p a n a m a b a n a
s \$ p a n a m a b a n a n a

Still
sorted

These strings are sorted

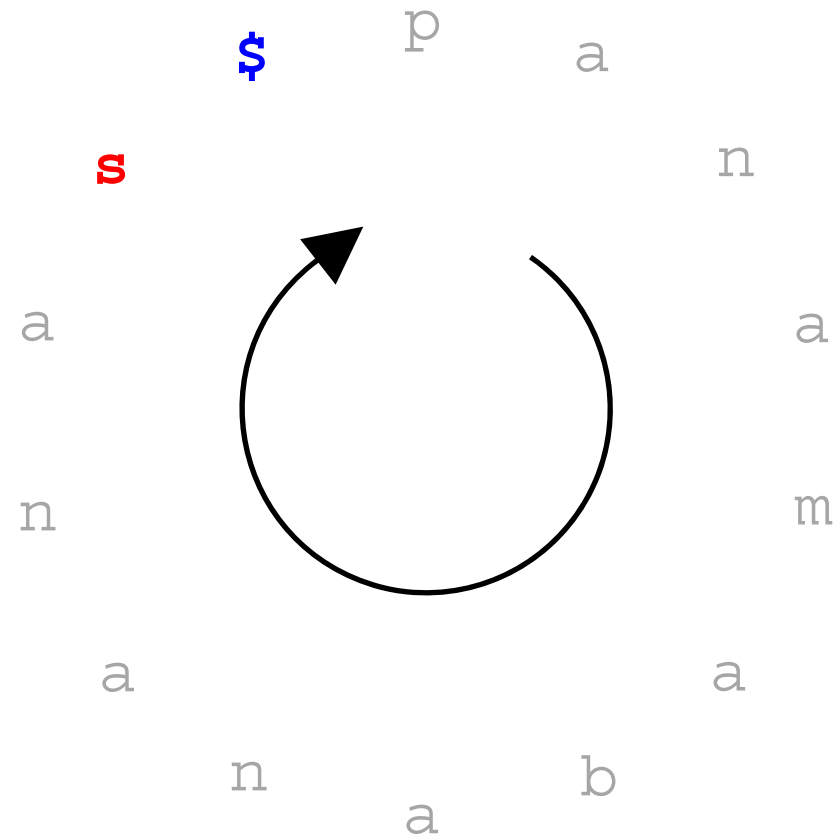
Is It True in General?

- **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_s₁
a₁bananas\$panam_m₁
a₂mabananas\$pan_n₁
a₃namabananas\$p_p₁
a₄nanas\$panamab_b₁
a₅nas\$panamaban_n₂
a₆s\$panamabanan_n₃
b₁ananas\$panama_a₁
m₁abananas\$pana_a₂
n₁amabananas\$pa_a₃
n₂anas\$panamaba_a₄
n₃as\$panamabana_a₅
p₁anamabananas\$_s₁
s₁\$panamabanana_a₆

More Efficient BWT Decompression

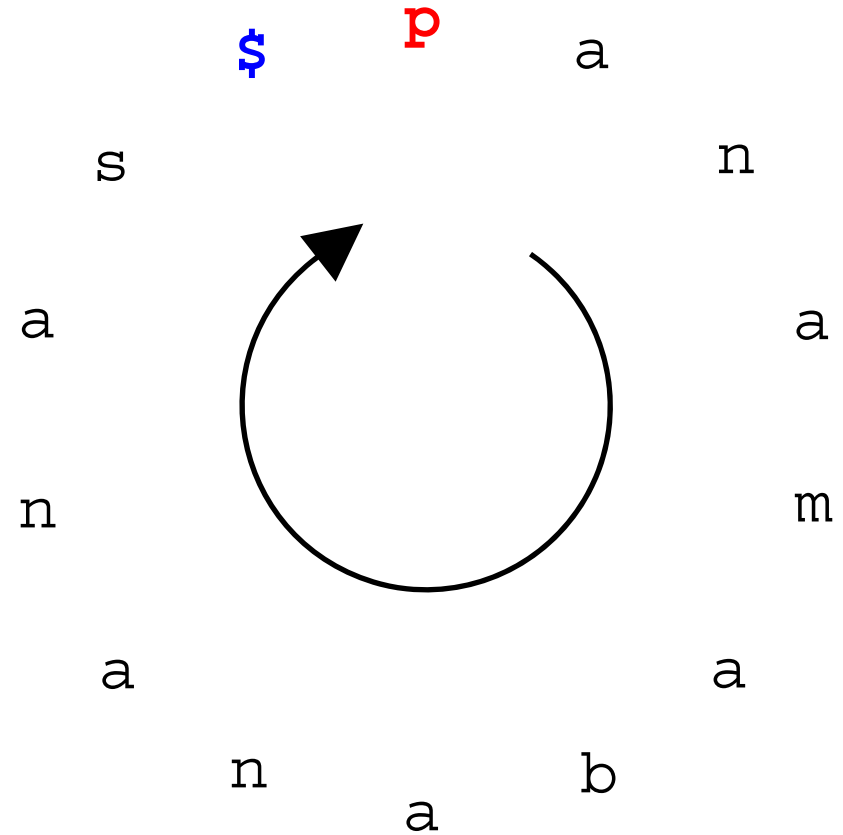
\$₁panamabanana**s**₁
 a₁bananas\$panam₁
 a₂mabananas\$pan₁
 a₃namabananas\$p₁
 a₄nanas\$panamab₁
 a₅nas\$panamaban₂
 a₆s\$panamabanana₃
 b₁ananas\$panama₁
 m₁abananas\$pana₂
 n₁amabananas\$pa₃
 n₂anas\$panamaba₄
 n₃as\$panamabana₅
 p₁anamabananas\$₁
 s₁\$panamabanana₆



More Efficient BWT Decompression

```

$1panamabananas1
a1bananas$panam1
a2mabananas$pan1
a3namabananas$p1
a4nanas$panamab1
a5nas$panamaban2
a6s$panamaban3
b1ananas$panama1
m1abananas$pana2
n1amabananas$pa3
n2anas$panamaba4
n3as$panamabana5
p1anamabananas$1
s1$panamabanana6
  
```

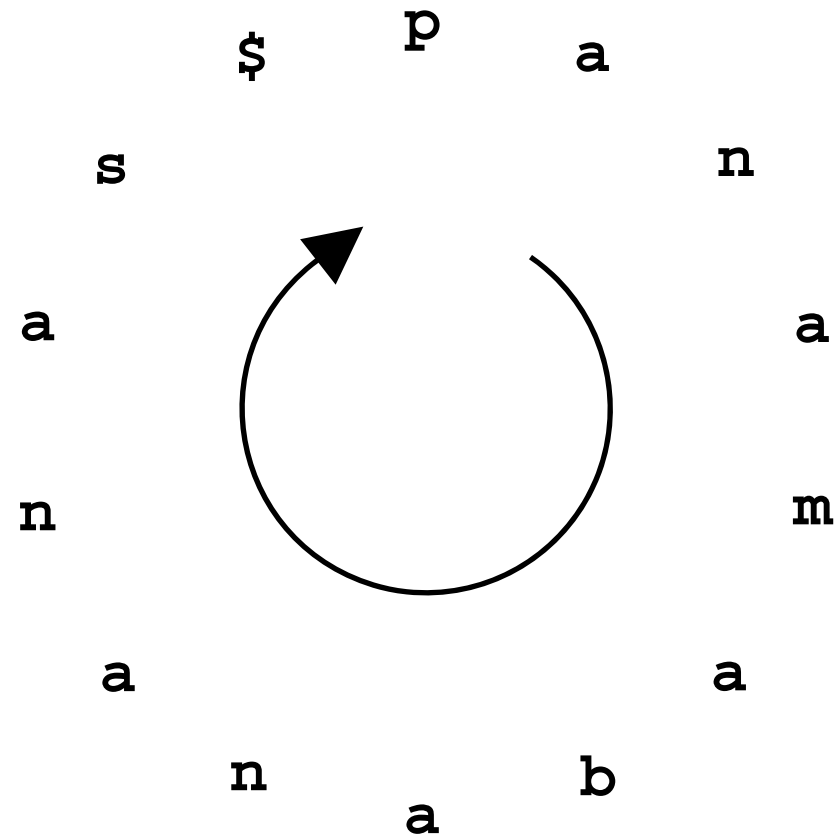


More Efficient BWT Decompression

```

$1panamabananas1
a1bananas$panam1
a2mabananas$pan1
a3namabananas$p1
a4nanas$panamab1
a5nas$panamaban2
a6s$panamaban3
b1ananas$panama1
m1abananas$pana2
n1amabananas$pa3
n2anas$panamaba4
n3as$panamabana5
p1anamabananas$1
s1$panamabanana6

```



- Memory: $2|Genome| = O(|Genome|)$.

Recalling Our Goal

- Suffix Tree Pattern Matching:
 - Runtime: $O(|Genome| + |Patterns|)$
 - Memory: $O(|Genome|)$
 - Problem: suffix tree takes $20 \times |Genome|$ space
- Can we use $BWT(Genome)$ as our data structure instead?

Finding Pattern Matches Using BWT

- Searching for **ana** in **p**ana**mab**ana**nas**

```
$1panamabananas1
a1bananas$panam1
a2mabananas$pan1
a3namabananas$p1
a4nanas$panamab1
a5nas$panamaban2
a6s$panamaban3
b1ananas$panama1
m1abananas$pana2
n1amabananas$pa3
n2anas$panamaba4
n3as$panamabana5
p1anamabananas$1
s1$panamabanana6
```

Finding Pattern Matches Using BWT

- Searching for **ana** in panamabananas

```
$1panamabananas1  
a1bananas$panam1  
a2mabananas$pan1  
a3namabananas$p1  
a4nanas$panamab1  
a5nas$panamaban2  
a6s$panamaban3  
b1ananas$panama1  
m1abananas$pana2  
n1amabananas$pa3  
n2anas$panamaba4  
n3as$panamabana5  
p1anamabananas$1  
s1$panamabana6
```

Finding Pattern Matches Using BWT

- Searching for **ana** in panamabananas

```
$1panamabananas1  
a1bananas$panam1  
a2mabananas$pann1  
a3namabananas$p1  
a4nanas$panamab1  
a5nas$panamaban2  
a6s$panamabanan3  
b1ananas$panama1  
m1abananas$pana2  
n1amabananas$pa3  
n2anas$panamaba4  
n3as$panamabana5  
p1anamabananas$1  
s1$panamabana6
```

Finding Pattern Matches Using BWT

- Searching for **ana** in panamabananas

```
$1panamabananas1  
a1bananas$panam1  
a2mabananas$pan1  
a3namabananas$p1  
a4nanas$panamab1  
a5nas$panamaban2  
a6s$panamaban3  
b1ananas$panama1  
m1abananas$pana2  
n1amabananas$pa3  
n2anas$panamaba4  
n3as$panamabana5  
p1anamabananas$1  
s1$panamabanana6
```


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 p a n a m a b a n a n a s₁
a₁ b a n a n a s \$ p a n a m₁
a₂ m a b a n a n a s \$ p a n₁
a₃ n a m a b a n a n a s \$ p₁
a₄ n a n a s \$ p a n a m a b₁
a₅ n a s \$ p a n a m a b a n₂
a₆ s \$ p a n a m a b a n a n₃
b₁ a n a n a s \$ p a n a m a₁
m₁ a b a n a n a s \$ p a n a₂
n₁ a m a b a n a n a s \$ p a₃
n₂ a n a s \$ p a n a m a b a₄
n₃ a s \$ p a n a m a b a n a₅
p₁ a n a m a b a n a n a s \$₁
s₁ \$ p a n a m a b a n a n a₆

Using the Suffix Array to Find Matches

- **Suffix array:** holds starting position of each suffix beginning a row.

```
$1 p a n a m a b a n a n a s1
a1 b a n a n a s $ p a n a m1
a2 m a b a n a n a s $ p a n1
a3 n a m a b a n a n a s $ p1
a4 n a n a s $ p a n a m a b1
a5 n a s $ p a n a m a b a n2
a6 s $ p a n a m a b a n a n3
b1 a n a n a s $ p a n a m a1
m1 a b a n a n a s $ p a n a2
n1 a m a b a n a n a s $ p a3
n2 a n a s $ p a n a m a b a4
n3 a s $ p a n a m a b a n a5
p1 a n a m a b a n a n a s $1
s1 $ p a n a m a b a n a n a6
```

Using the Suffix Array to Find Matches

- **Suffix array:** holds starting position of each suffix beginning a row.

panamabananas\$

1	3	\$ ₁ panamabananas ₁
		a ₁ bananas\$panam ₁
		a ₂ mabananas\$pan ₁
		a ₃ namabananas\$p ₁
		a ₄ nanas\$panamab ₁
		a ₅ nas\$panamaban ₂
		a ₆ s\$panamaban ₃
		b ₁ ananas\$panama ₁
		m ₁ abananas\$pana ₂
		n ₁ amabananas\$pa ₃
		n ₂ anas\$panamaba ₄
		n ₃ as\$panamabana ₅
		p ₁ anamabananas\$ ₁
		s ₁ \$panamabanna ₆

Using the Suffix Array to Find Matches

- **Suffix array:** holds starting position of each suffix beginning a row.

panam**ab**ananas\$

1	3	\$ ₁ panamabananas ₁
	5	a ₁ bananas \$panam ₁
		a ₂ mabananas\$pan 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 amabananas\$pa ₃
		n ₂ anas\$panamaba ₄
		n ₃ as\$panamabana ₅
		p ₁ anamabananas\$ ₁
		s ₁ \$panamabanna ₆

Using the Suffix Array to Find Matches

- **Suffix array:** holds starting position of each suffix beginning a row.

pan**amabananas**\$

1	3	\$ ₁ pan amabananas ₁
5		a ₁ bananas\$panam ₁
3		a ₂ mabanas\$pan 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 amabananas\$pa ₃
		n ₂ anas\$panamaba ₄
		n ₃ as\$panamabana ₅
		p ₁ anamabananas\$ ₁
		s ₁ \$panamabanna ₆

Using the Suffix Array to Find Matches

- **Suffix array:** holds starting position of each suffix beginning a row.

pan**a**mab**a**nan**a**s**\$**

13	\$ ₁ p	an	ab	an	anas	\$ ₁
5	a ₁	ban	anas	\$	p	anam
3	a ₂	m	aban	anas	\$	pan
1	a ₃	n	am	ab	an	anas
	a ₄	nan	as	\$	p	anam
	a ₅	nas	\$	p	anam	ab
	a ₆	s	\$	p	anam	aban
	b ₁	an	anas	\$	p	ana
	m ₁	a	ban	anas	\$	pan
	n ₁	a	m	ab	an	anas
	n ₂	a	nas	\$	p	anam
	n ₃	a	s	\$	p	anam
	p ₁	a	n	a	m	ab
	s ₁	\$	p	a	n	a

Using the Suffix Array to Find Matches

- **Suffix array:** holds starting position of each suffix beginning a row.

panamab**ananas**\$

1	3	\$ ₁ panamab ananas s ₁
5		a ₁ bananas \$panam ₁
3		a ₂ mab ananas \$pan ₁
1		a ₃ namab ananas \$p ₁
7		a ₄ nanas\$bpanamab ₁
		a ₅ nas\$bpanamaban ₂
		a ₆ s\$bpanamabanan ₃
		b ₁ ananas\$bpanama ₁
		m ₁ ab ananas \$pana ₂
		n ₁ amab ananas \$pa ₃
		n ₂ anas\$bpanamaba ₄
		n ₃ as\$bpanamabana ₅
		p ₁ anamab ananas \$ ₁
		s ₁ \$panamab anana s ₆

Using the Suffix Array to Find Matches

- **Suffix array:** holds starting position of each suffix beginning a row.

panamabanas\$

13	\$ ₁ panamabanas ₁
5	a ₁ bananas\$panam ₁
3	a ₂ mabanas\$pan ₁
1	a ₃ namabanas\$pa ₁
7	a ₄ nanas\$panamab ₁
9	a ₅ nas\$panamaban ₂
	a ₆ s\$panamabanan ₃
	b ₁ anas\$panama ₁
	m ₁ abanas\$pana ₂
	n ₁ amabanas\$pa ₃
	n ₂ anas\$panamaba ₄
	n ₃ as\$panamabana ₅
	p ₁ anamabanas\$ ₁
	s ₁ \$panamabanna ₆

Using the Suffix Array to Find Matches

- **Suffix array:** holds starting position of each suffix beginning a row.

panamabananas\$

1 3	\$ 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
1 1	a ₆ s\$panamaban ₃
	b ₁ ananas\$panama ₁
	m ₁ abananas\$pana ₂
	n ₁ amabananas\$pa ₃
	n ₂ anas\$panamaba ₄
	n ₃ as\$panamabana ₅
	p ₁ anamabananas\$ ₁
	s ₁ \$panamabanna ₆

Using the Suffix Array to Find Matches

- **Suffix array:** holds starting position of each suffix beginning a row.

panam**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\$panamaban ₃
6	b ₁ ananas\$panama ₁
	m ₁ abananas\$pana ₂
	n ₁ amabananas\$pa ₃
	n ₂ anas\$panamaba ₄
	n ₃ as\$panamabana ₅
	p ₁ anamabananas\$ ₁
	s ₁ \$panamabanna ₆

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
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 \$panamabanna 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
7	a ₄ nanas\$panamab ₁
9	a ₅ nas\$panamaban ₂
11	a ₆ s\$panamaban 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
	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\$panamaban 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 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\$panamaban 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.

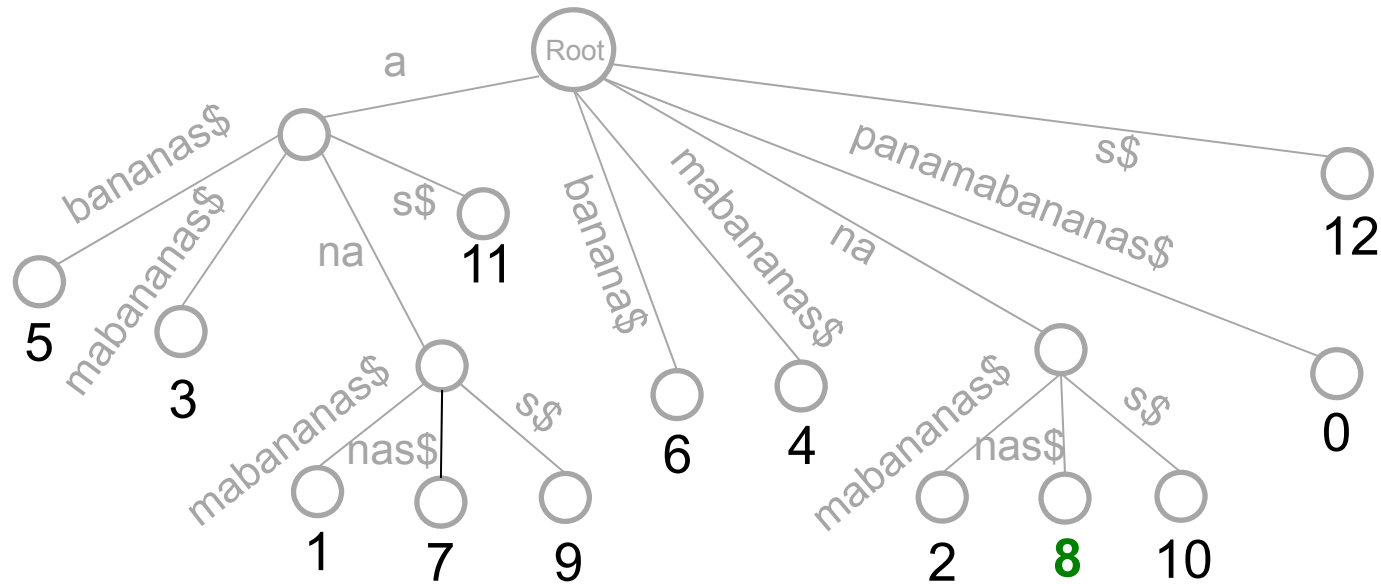
- Thus, **ana** occurs at positions **1, 7, 9** of **p**ana**mab**anana**s\$**.


```

1 3  $1panamabananas1
   5  a1bananas$panam1
   3  a2mabananas$pan1
   1  a3namabananas$p1
   7  a4nanas$panamab1
   9  a5nas$panamaban2
  11  a6s$panamaban3
   6  b1ananas$panama1
   4  m1abananas$pana2
   2  n1amabananas$pa3
   8  n2anas$panamaba4
  10  n3as$panamabana5
   0  p1anamabananas$1
  12  s1$panamabanana6
    
```


The Suffix Array: Memory Once Again

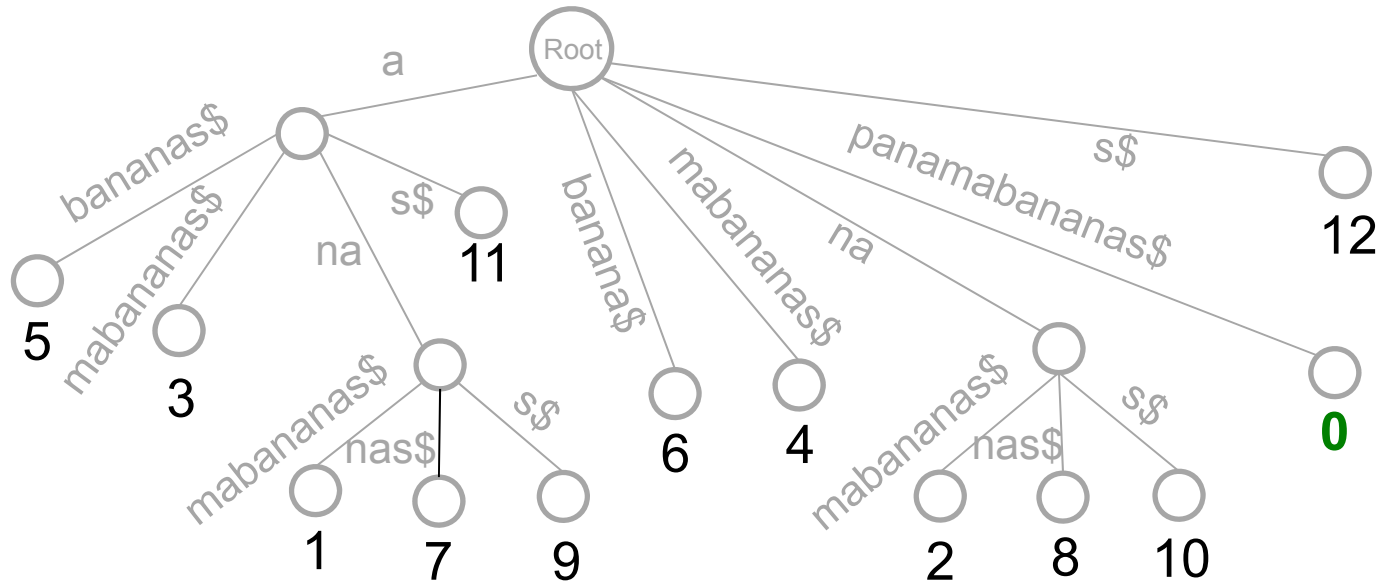
- Memory: $\sim 4 \times |Genome|$.



[13 5 3 1 7 9 11 6 4 2 8 10 0 1

The Suffix Array: Memory Once Again

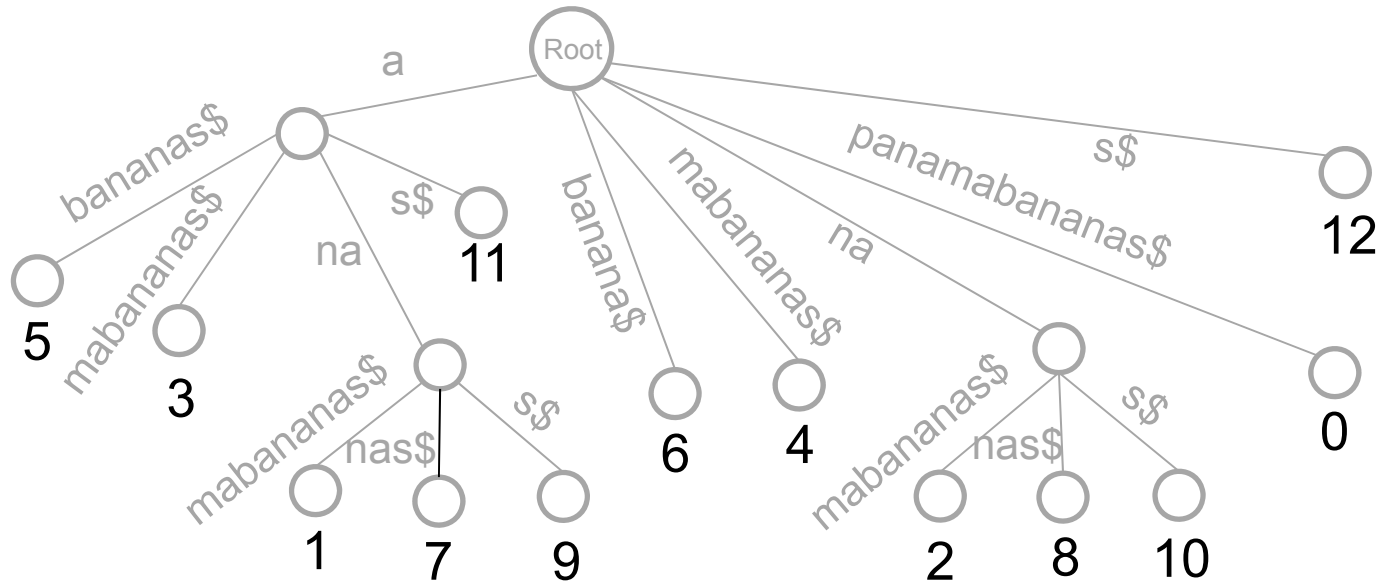
- Memory: $\sim 4 \times |Genome|$.



[13 5 3 1 7 9 11 6 4 2 8 10 0 1

The Suffix Array: Memory Once Again

- Memory: $\sim 4 \times |Genome|$.



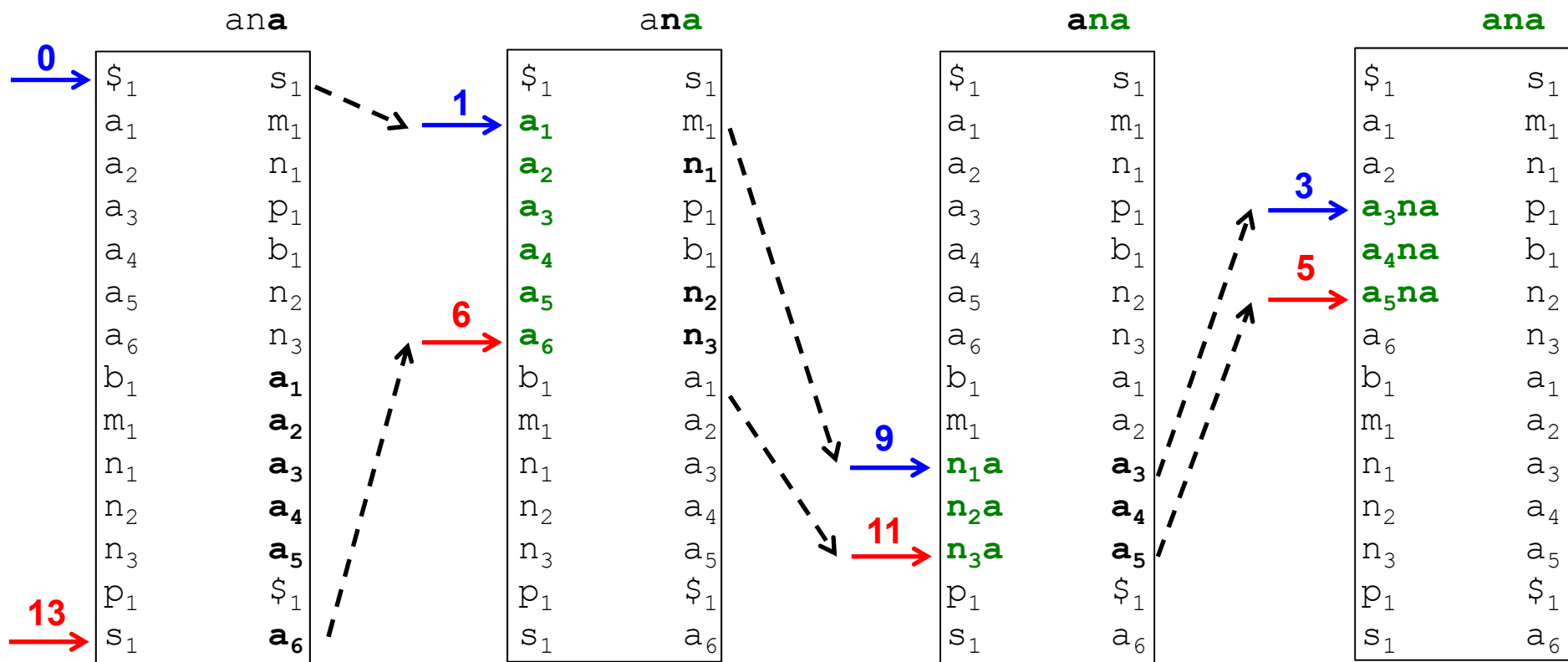
[13 5 3 1 7 9 11 6 4 2 8 10 0 1

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.



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:

<i>Pattern</i>	act	t	ggct
<i>Genome</i>	...ggc	a	ggctcc...

Method 1: Seeding

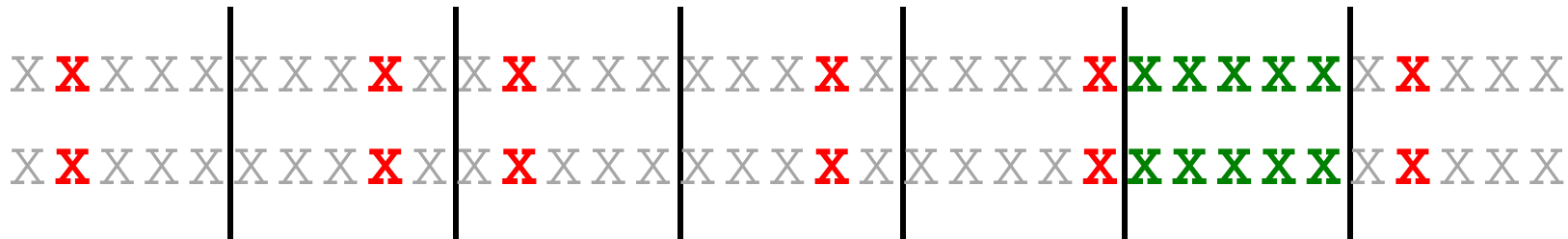
- Say that *Pattern* appears in *Genome* with 1 mismatch:

<i>Pattern</i>	a c t t g g c t
<i>Genome</i>	...g g c a c a c t a g g c t c c...

- 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 **ana** in panamabananas

Now we extend
all strings with at
most 1 mismatch.

	# Mismatches
\$ ₁ panamabananas ₁	
a ₁ bananas\$pana m ₁	1
a ₂ mabananas\$pan n ₁	0
a ₃ namabananas\$p p ₁	1
a ₄ nanas\$panama b ₁	1
a ₅ nas\$panamaba n ₂	0
a ₆ s\$panamabana n ₃	0
b ₁ ananas\$panama ₁	
m ₁ abananas\$pana ₂	
n ₁ amabananas\$pa ₃	
n ₂ anas\$panamaba ₄	
n ₃ as\$panamabana ₅	
p ₁ anamabananas\$ ₁	
s ₁ \$panamabana ₆	

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 panamabananas ₁	
a ₁ bananas\$panam ₁	
a ₂ mabananas\$pan ₁	
a ₃ namabananas\$p ₁	
a ₄ nanas\$panamab ₁	
a ₅ nas\$panamaban ₂	
a ₆ s\$panamaban ₃	
b ₁ a nanas\$panam a ₁	1
m ₁ a bananas\$pan a ₂	1
n ₁ a mabananas\$pa a ₃	0
n ₂ a nas\$panamab a ₄	0
n ₃ a s\$panamaban a ₅	0
p ₁ a namabananas\$ s ₁	2
s ₁ \$panamabanana 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 panamabananas ₁	
a ₁ b ananas\$panam ₁	1
a ₂ m abananas\$pan ₁	1
a ₃ n amabananas\$p ₁	0
a ₄ n anas\$panamab ₁	0
a ₅ n a\$\$panamaban ₂	0
a ₆ \$panamaban ₃	
b ₁ ananas\$panam a ₁	
m ₁ abananas\$pan a ₂	
n ₁ amabananas\$p a ₃	
n ₂ anas\$panamab a ₄	
n ₃ as\$panamaban a ₅	
p ₁ anamabananas\$ s ₁	
s ₁ \$panamabanana ₆	

Method 2: BWT Saves the Day Again

- Recall: searching for **ana** in panamab**an**anas

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 mabananas\$pan ₁	3
a ₃ namabananas\$p ₁	1
a₄nanas\$panamab₁	7
a ₅ nas\$panamaban ₂	9
a ₆ s\$panamaban ₃	
b ₁ ananas\$panama ₁	
m ₁ abananas\$pana ₂	
n ₁ amabananas\$pa ₃	
n ₂ anas\$panamaba ₄	
n ₃ as\$panamabana ₅	
p ₁ anamabananas\$ ₁	
s ₁ \$panamabanana ₆	

Method 2: BWT Saves the Day Again

- Recall: searching for **ana** in panamaban**anas**

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 mabananas\$pan ₁	3
a ₃ namabananas\$p ₁	1
a ₄ nanas\$panamab ₁	7
a₅nas\$panamaban₂	9
a ₆ s\$panamaban ₃	
b ₁ ananas\$panama ₁	
m ₁ abananas\$pana ₂	
n ₁ amabananas\$pa ₃	
n ₂ anas\$panamaba ₄	
n ₃ as\$panamabana ₅	
p ₁ anamabananas\$ ₁	
s ₁ \$panamabanana ₆	

Related work

- Accommodate mismatches in string using e.g. a seed and extend approach
 - Heuristic approaches
 - BatMis
<https://doi.org/10.1093/bioinformatics/bts339>
- Lots of interest in processing k-mers of sequence data
 - E.g. MinHash for comparing two sets of reads without alignment
- Next, we will consider how we can compare more divergent sequences

Hidden Markov Models Outline

- From a Crooked Casino to a Hidden Markov Model
- Decoding Problem
- The Viterbi Algorithm
- Profile HMMs for Sequence Alignment
- Classifying proteins with profile HMMs
- Viterbi Learning
- Soft Decoding Problem
- Baum-Welch Learning

The Crooked Casino

A crooked dealer may use one of two identically looking coins:

- The **fair** coin (F) gives heads with probability $1/2$
 $\Pr_F(\text{“Head”}) = 1/2$ $\Pr_F(\text{“Tail”}) = 1/2$
- The **biased** coin (B) gives heads with probability $3/4$:



What coin is **more likely** if **63** out of **100** flips resulted in heads?

Hint: 63 is closer to 75 than to 50!

Fair or Biased?

- Given a sequence of n flips with k “Heads”:

$$X = X_1 X_2 \dots X_n$$

- The probability this sequence was generated by the **fair** coin:

$$\Pr(x|F) = \Pr_F(x_1) * \dots * \Pr_F(x_n) = (1/2)^n$$

- The probability that it was generated by the **biased** coin:

$$\Pr(x|B) = \Pr_B(x_1) * \dots * \Pr_B(x_n) = (3/4)^k * (1/4)^{n-k}$$

$\Pr(x|F) > \Pr(x|B) \rightarrow$ **fair** is more likely

$\Pr(x|F) < \Pr(x|B) \rightarrow$ **biased** is more

likely

Equilibrium: $P(x|F) = P(x|B)$

$$(1/2)^n = (3/4)^k * (1/4)^{n-k} \rightarrow 2^n = 3^k \rightarrow k - \log_2 3 * n \rightarrow k \approx 0.632 * n$$

Even though 63 is closer to 75 than to 50, **fair** coin is more likely to result in 63 heads!

Two Coins Up the Dealer Sleeve

- Now the dealer has *both* **fair** and **biased** coins and can change between them at any time (without you noticing) with probability 0.1.

After watching a sequence of flips, can you tell when the dealer was using **fair** coin and when he was using **biased** coin?

Reading the Dealer's Mind

Casino Problem: *Given a sequence of coin flips, determine when the dealer used a fair coin and a biased coin.*

- **Input:** A sequence $x = x_1 x_2 \dots x_n$ of flips made by coins F (fair) and B (biased).
- **Output:** A sequence $\pi = \pi_1 \pi_2 \dots \pi_n$, with each π_i being equal to either F or B and indicating that x_i is the result of flipping the fair or biased coin, respectively.

The Problem with the Casino Problem

- Any outcome of coin tosses could have been generated by any combination of **fair** and **biased** coins!
 - HHHHH could be generated by **BBBBB**, **FFFFF**, **FBFBF**, etc.

We need to **grade** different scenarios:

BBBBB, **FFFFF**, **FBFBF**, etc.

differently, depending on how likely they are.

How can we explore and grade 2^n possible scenarios?

Reading the Dealer's Mind (one window at a time)

HHHTHHHT

BBBBB

FFFFF

$\Pr(x|F)/\Pr(x|B) <$

$\Pr(x|F)/\Pr(x|B) >$

1

Log-odds ratio of sequence $x = \log_2 \Pr(x|F) / \Pr(x|B)$

$$\log_2 (1/2)^n / (3/4)^k \cdot (1/4)^{n-k} = \text{\#Tosses} - \log_2 3^*$$

\#Heads

Reading the Dealer's Mind (one window at a time)

HHHTHHTHHHT

BBBBB

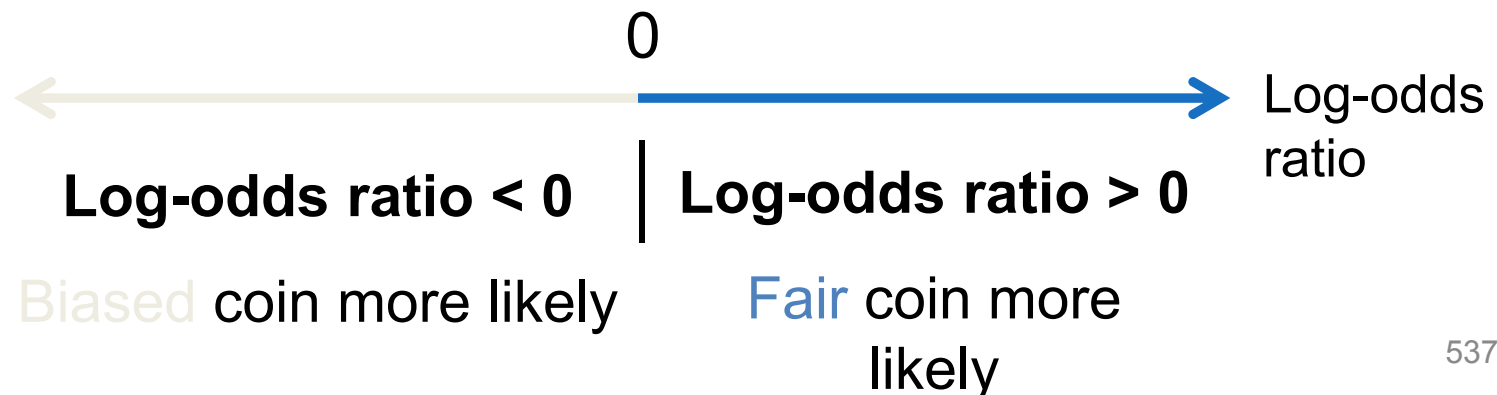
FFFFF

Log-odds < 0

Log-odds > 0

Log-odds ratio of sequence $x = \log_2 \Pr(x|F) / \Pr(x|B)$
 $= \#Tosses - \log_2 3^*$

#Heads



Reading the Dealer's Mind

HHHTH **HHHT**

BBBBB

FFFFF

FFFFF

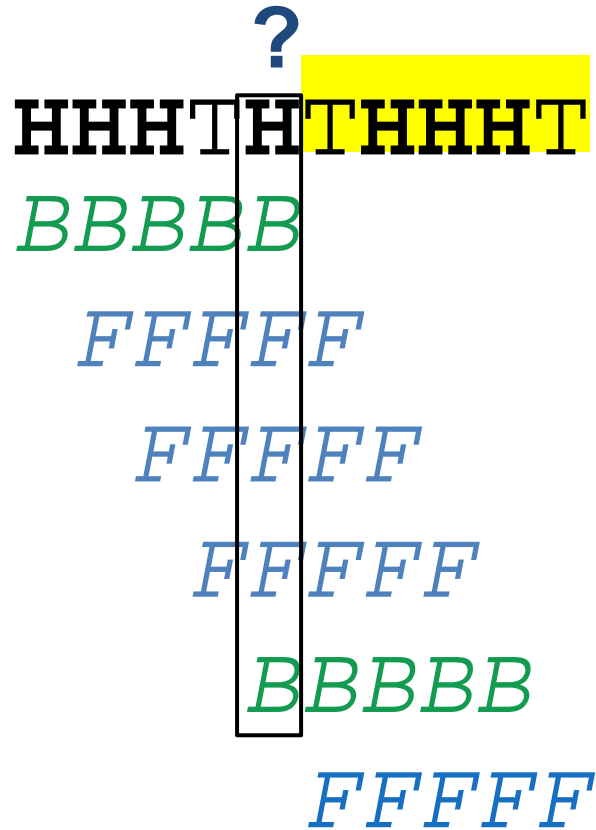
FFFFF

BBBBB

FFFFF

What are the disadvantages of this approach?

Disadvantages of the Sliding Window Approach



Different windows may classify the same coin flip

differently

The results depend on the window length. How to choose

it?

Why Are **CG** Dinucleotides More Rare than **GC** Dinucleotides in Genomic Sequences?

- Different species have widely varying **GC-content** (percentages of G+C nucleotides in the genome).



46% for gorilla and human



58% for platypus

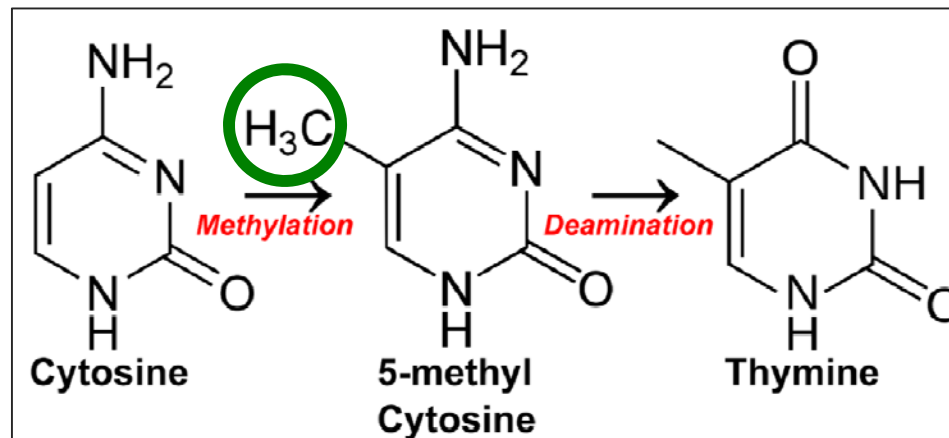
- Each of the dinucleotides **CC**, **CG**, **GC**, and **GG** is expected to occur in the human genome with frequency $0.23 * 0.23 = 5.29\%$.

But the frequency of **CG** in the human genome is only 1%!

Methylation

Methylation: adds a methyl (CH_3) group to the cytosine nucleotide (often within a **CG** dinucleotide).

- The resulting **methylated cytosine** has the tendency to *deaminate* into thymine.



- As a result of methylation, **CG** is the least frequent dinucleotide in many genomes.

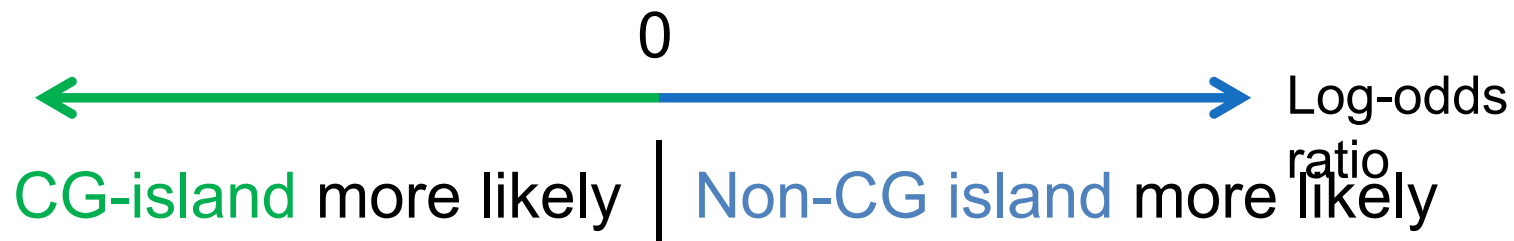
Looking for CG-islands

Methylation is often suppressed around genes in areas called **CG-islands** (CG appears frequently).

ATTTCTTCTCGTCGACGCCTAATTTCTTGGAAATATCATTAT

In a first attempt to find genes, how would you search for CG-islands?

Looking for CG-islands



- Different windows may classify the same position in the genome differently.
- It is not clear how to choose the length of the window for detecting CG-islands.
- Does it make sense to choose the same window length for all regions in the genome?

Turning the Dealer into a Machine

- Think of the dealer as a machine with k **hidden states** (F and B) that proceeds in a sequence of steps.
- In each step, it emits a symbol (H or T) while being in one of its hidden states.
- While in a certain state, the machine makes two decisions:
 - Which *symbol* will I emit?
 - Which *hidden state* will I move to next?



Hidden Markov Model (HMM)

Σ : an **alphabet** of emitted symbols

H and T

States : a set of **hidden states**

F and *B*

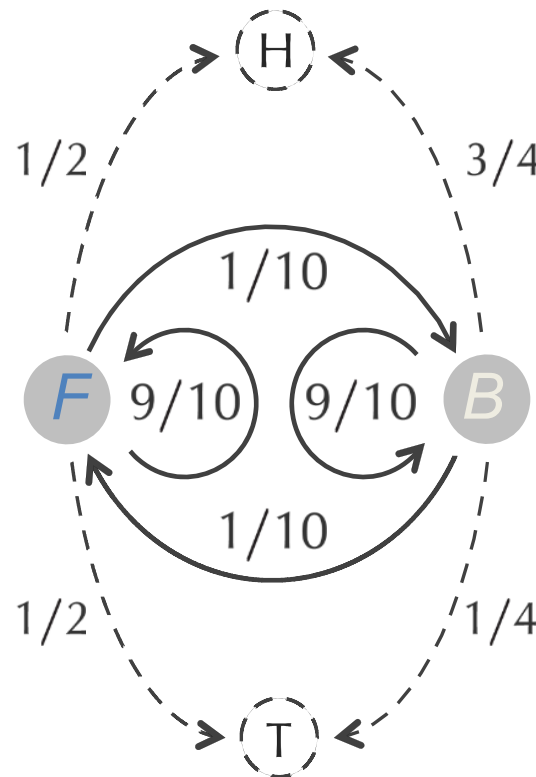
Transition = ($transition_{l,k}$): a $|States| \times |States|$
matrix of **transition probabilities** (of
changing from state l to state k)

	<i>F</i>	<i>B</i>
<i>F</i>	0.9	0.1
<i>B</i>	0.1	0.9

Emission = ($emission_k(b)$): a $|States| \times |\Sigma|$
matrix of **emission probabilities** (of
emitting symbol b when the HMM is in state k)

	H	T
<i>F</i>	0.50	0.50
<i>B</i>	0.75	0.25

HMM Diagram



Transition

	F	B
F	0.9	0.1
B	0.1	0.9

Emission

	H	T
F	0.50	0.50
B	0.75	0.25

Hidden Path

Hidden path: the sequence $\pi = \pi_1 \dots \pi_n$ of states that the HMM passes through.

- $\Pr(x, \pi)$: the probability that an HMM follows the hidden path π and emits the string $x = x_1 x_2 \dots x_n$.

x:	T	H	T	H	H	H	T	H	T	T	H
π :	F	F	F	B	B	B	B	B	F	F	F

$$\sum_{\text{all possible emitted strings } x} \sum_{\text{all possible hidden paths } \pi} \Pr(x, \pi) = 1$$

- $\Pr(x|\pi)$: the **conditional probability** that an HMM emits the string x after following the hidden path π .

$$\sum_{\text{all possible emitted strings } x} \Pr(x|\pi) = 1$$

$$\Pr(x, \pi) = \Pr(x|\pi) *$$

- $\Pr(x, \pi)$: the probability that an HMM $\Pr(\pi)$ follows the hidden path π and emits the string x . $\Pr(x_i|\pi_i)$ – probability that x_i was emitted from the state π_i (equal to *emission* _{π_i} (x_i)).
- $\Pr(\pi_{i-1} \rightarrow \pi_i)$ – probability that the HMM moved from $\pi_{i-1} \rightarrow \pi_i$ (equal to *transition* _{π_{i-1}, π_i}).

x												
π		T	H	T	H	H	H	T	H	T	T	H
$\Pr(\pi_{i-1} \rightarrow \pi_i)$		F	F	F	B	B	B	B	B	F	F	F
$\Pr(x_i \pi_i)$.5	.9	.9	.1	.9	.9	.9	.1	.9	.9	
		1/2	1/2	1/2	3/4	3/4	3/4	1/4	3/4	1/2	1/2	1/2

$$\Pr(\pi) = \prod_{i=1, n} \Pr(\pi_{i-1} \rightarrow \pi_i) = \prod_{i=1, n} \text{transition}_{\pi_{i-1}, \pi_i}$$

$$\Pr(x|\pi) = \prod_{i=1, n} \Pr(x_i|\pi_i) = \prod_{i=1, n} \text{emission}_{\pi_i}(x_i)$$

Computing Probability of a Hidden Path $\Pr(\pi)$ and Conditional Probability of an Outcome $\Pr(x|\pi)$

Probability of a Hidden Path Problem. *Compute the probability of an HMM's hidden path.*

- **Input:** A hidden path π in an HMM $(\Sigma, States, Transition, Emission)$.
- **Output:** The probability of this path, $\Pr(\pi)$.

Probability of an Outcome Given a Hidden Path Problem. *Compute the probability that an HMM will emit a given string given its hidden path.*

- **Input:** A string $x=x_1, \dots, x_n$ emitted by an HMM $(\Sigma, States, Transition, Emission)$ and a hidden path $\pi = \pi_1, \dots, \pi_n$.
- **Output:** The conditional probability $\Pr(x|\pi)$ that x will be emitted given that the HMM follows the hidden path π .

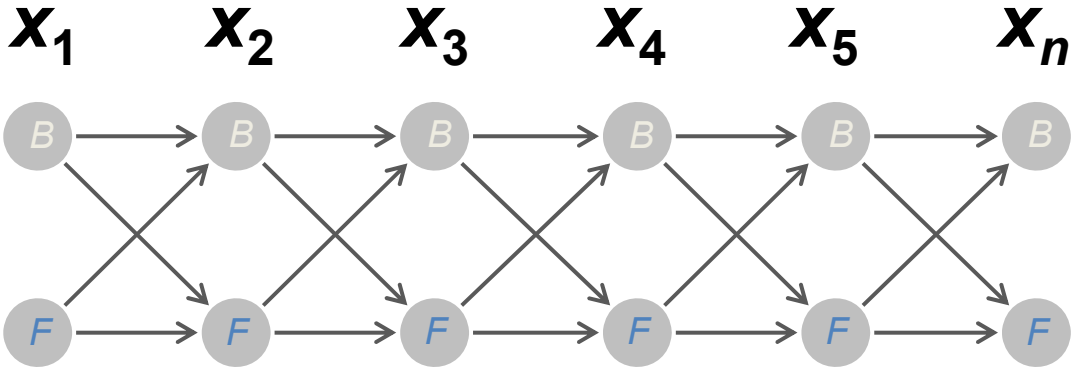
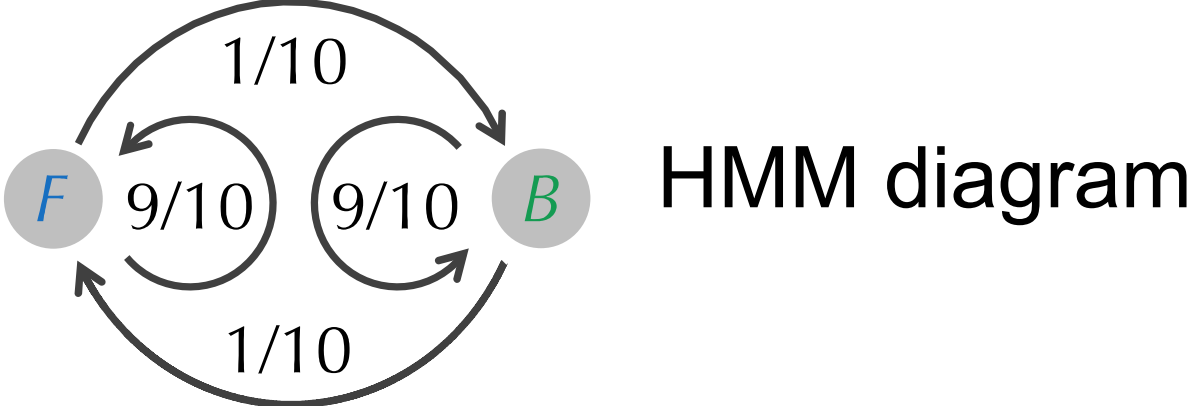
Decoding Problem

Decoding Problem: Find an optimal hidden path in an HMM given its emitted string.

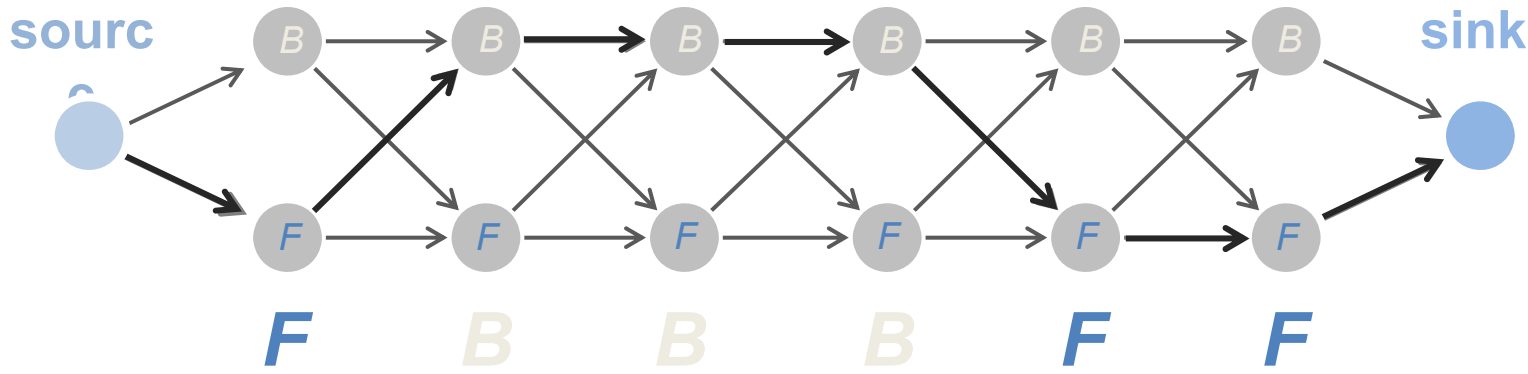
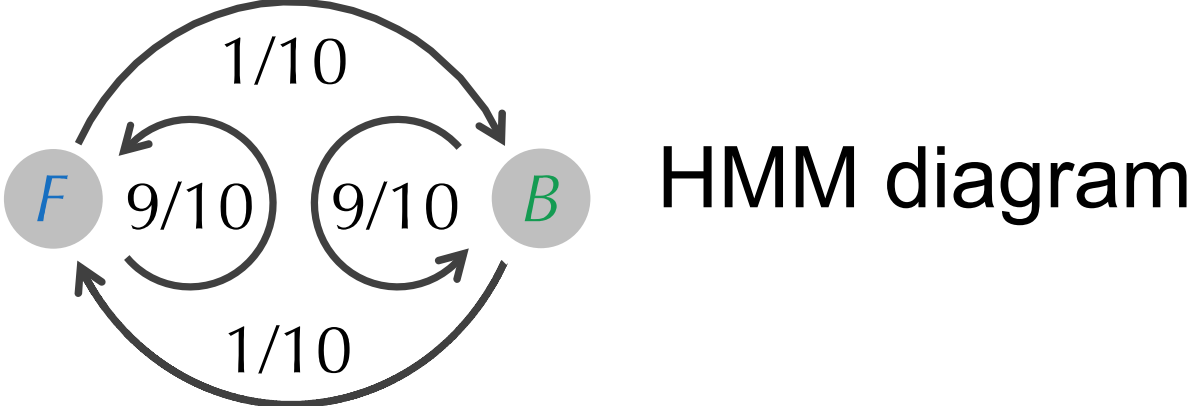
- **Input:** A string $x = x_1 \dots x_n$ emitted by an HMM $(\Sigma, \text{States}, \text{Transition}, \text{Emission})$.
- **Output:** A path π that maximizes the probability $\text{Pr}(x, \pi)$ over all possible paths through this HMM.

$$\begin{aligned}\text{Pr}(x, \pi) &= \text{Pr}(x|\pi) * \text{Pr}(\pi) \\ &= \prod_{i=1, n} \text{Pr}(x_i|\pi_i) * \text{Pr}(\pi_{i-1} \rightarrow \pi_i) \\ &= \prod_{i=1, n} \text{emission}_{\pi_i}(x_i) * \text{transition}_{\pi_{i-1}, \pi_i}\end{aligned}$$

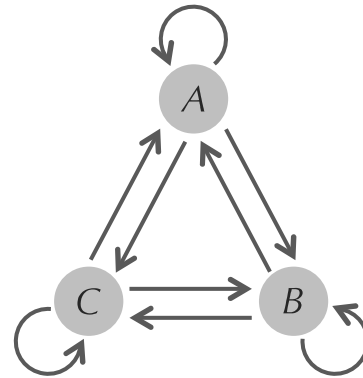
Building Manhattan for the Crooked Casino



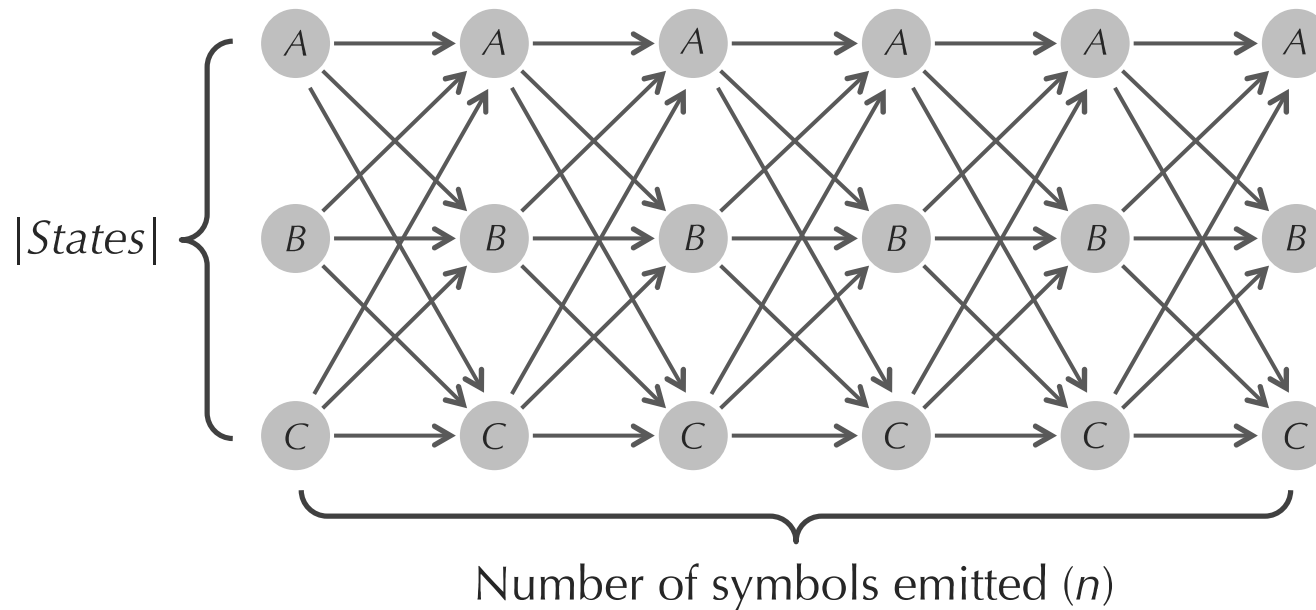
Building Manhattan for the Crooked Casino



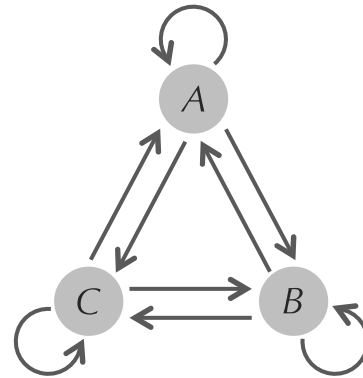
Building Manhattan for Decoding Problem



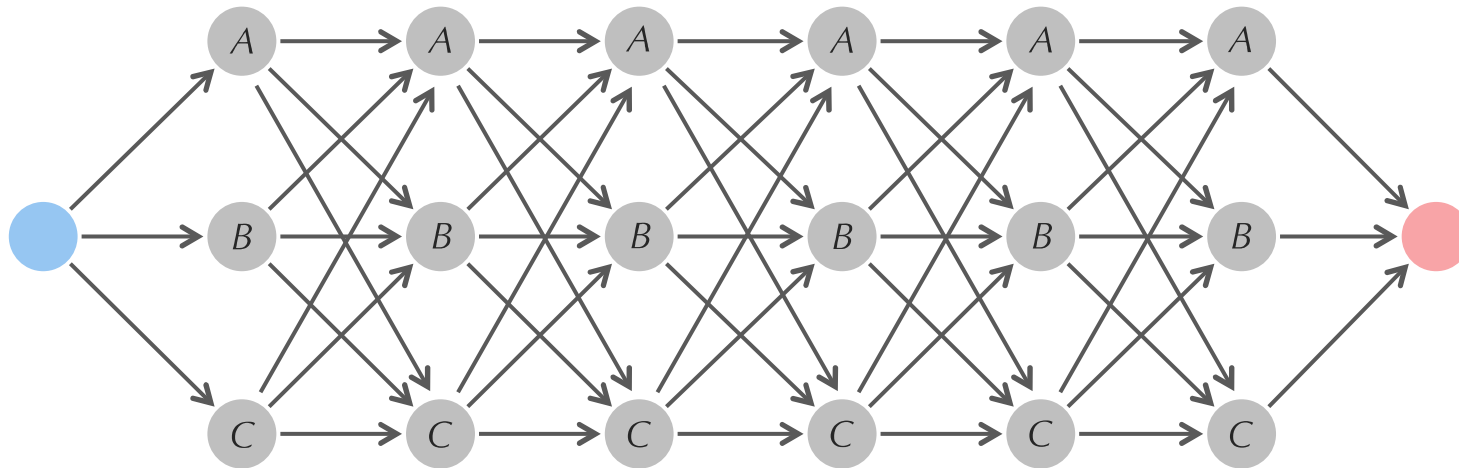
HMM diagram



Building Manhattan for Decoding Problem



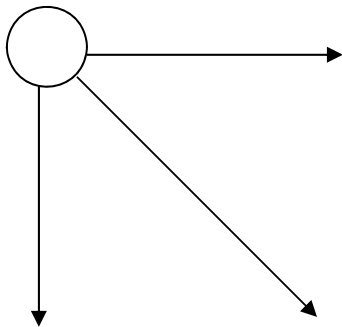
HMM diagram



Alignment Manhattan vs. Decoding Manhattan

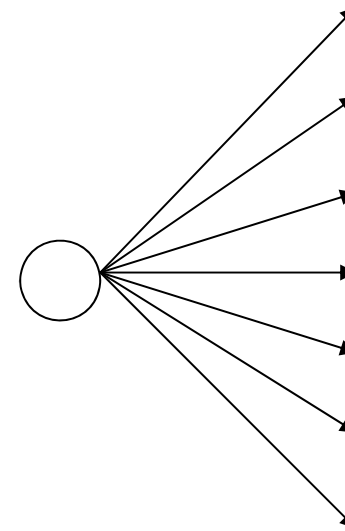
Alignment

three valid directions

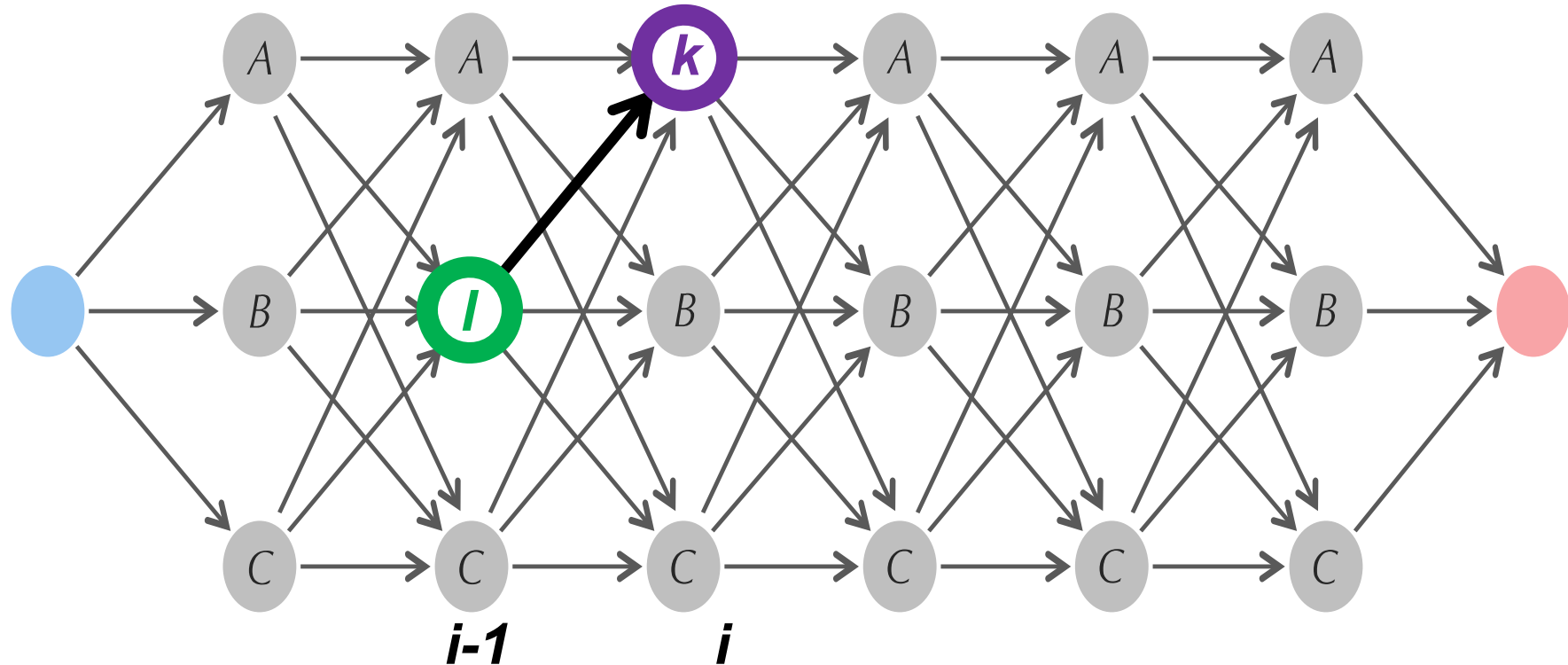


Decoding

many valid directions



Edge Weights in the HMM Manhattan



Edge $(l, k, i-1)$ from node $(l, i-1)$ to node (k, i) :

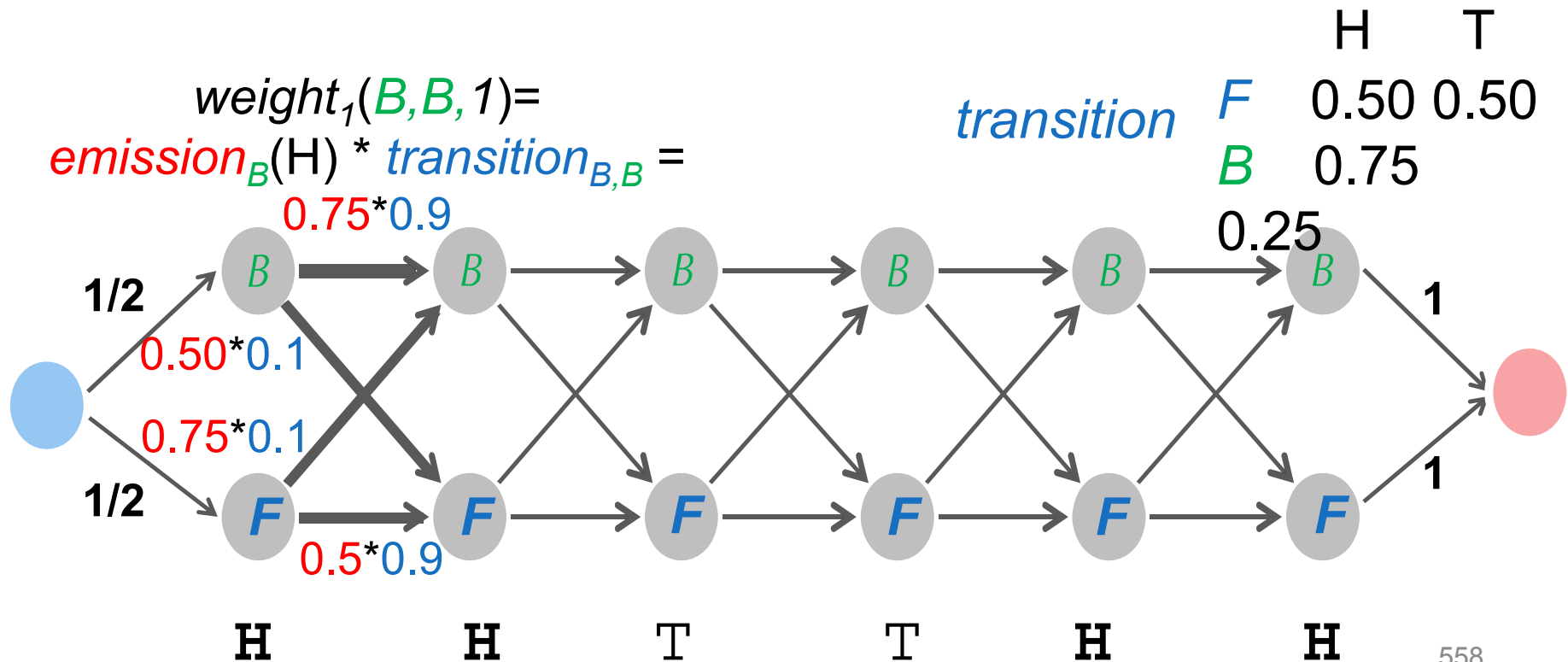
- transitioning from state l to state k (with probability $transition_{l,k}$)
- emitting symbol x_i (with probability $emission_k(x_i)$)

$$weight(l, k, i-1) = emission_k(x_i) * transition_{l,k}$$

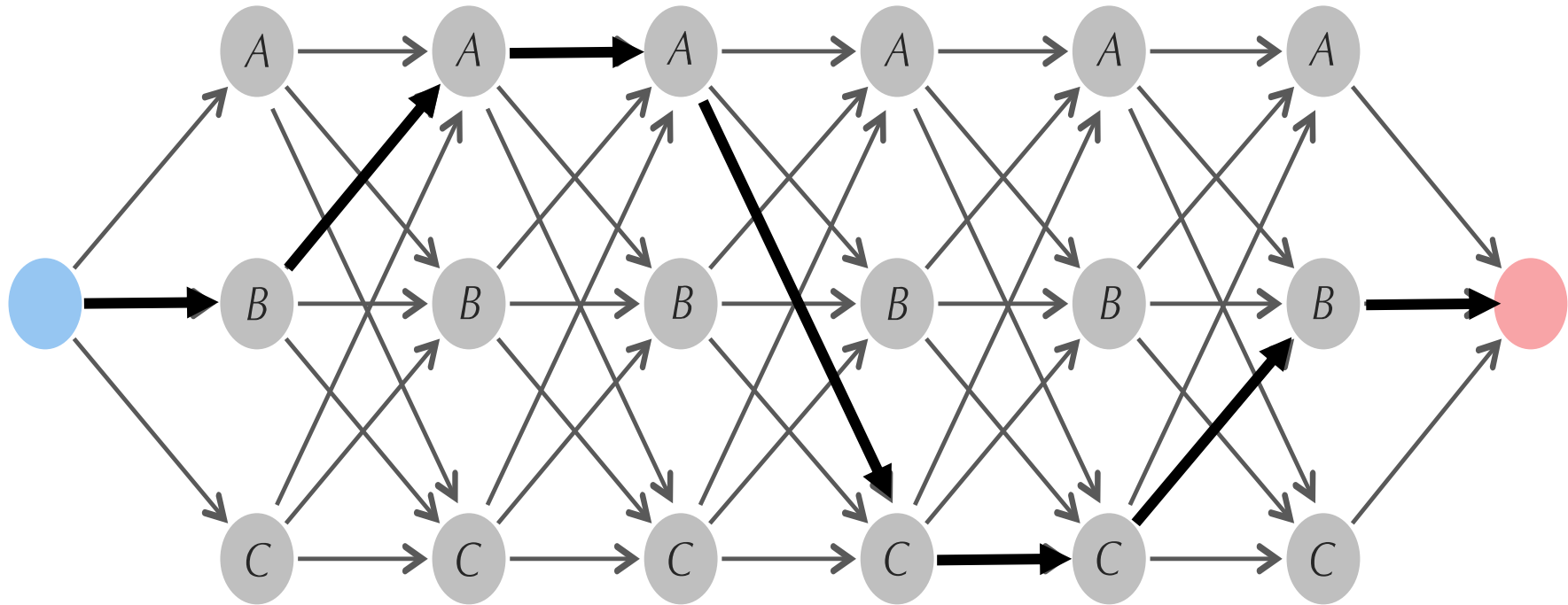
Edge Weights for the Crooked Casino

$$weight(l, k, i-1) = emission_k(x_i) * transition_{l,k}$$

		<i>F</i>	<i>B</i>
<i>emission</i>	<i>F</i>	0.9	0.1
	<i>B</i>	0.1	
<i>n</i>		0.9	



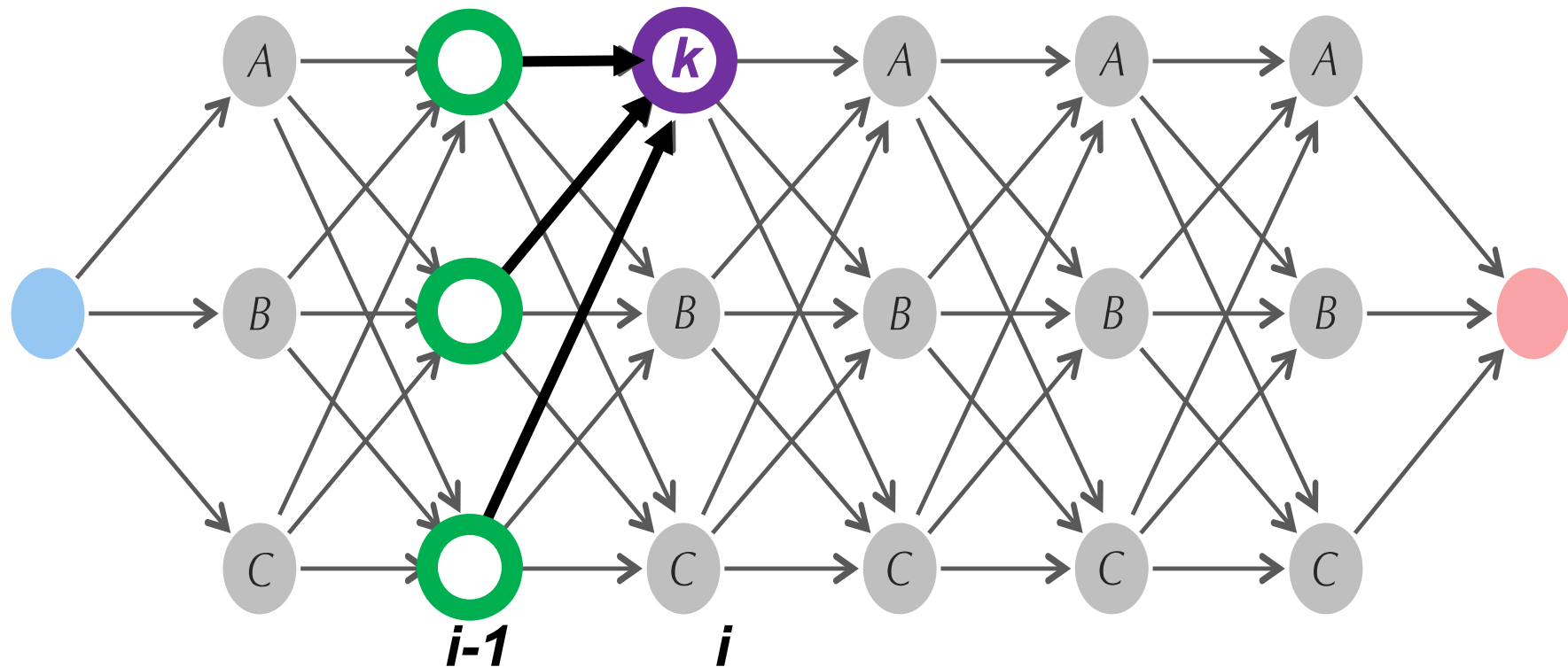
Product Weight of a Hidden Path



$$\begin{aligned} \Pr(x, \pi) &= \prod_{i=1, n} \text{emission}_{\pi_i}(x_i) * \text{transition}_{\pi_{i-1}, \pi_i} \\ &= \prod_{i=1, n} \text{weight of the } i\text{-th edge in path } \pi \\ &= \prod_{i=1, n} \text{weight}(\pi_{i-1}, \pi_i, i-1) \end{aligned}$$

Dynamic Programming for Decoding Problem

$score_{k,i}$: the maximum product weight among all paths from *source* to node (k, i) :



$$score_{k,i} = \max_{\text{all states } l} \{score_{l,i-1} \cdot \text{weight of edge from } (l, i-1) \text{ to } (k, i)\}$$

Recurrence for Viterbi Algorithm

- Recurrence:

$$score_{k,i} = \max_{\text{all states } l} \{score_{l,i-1} \cdot weight(l,k,i-1)\}$$

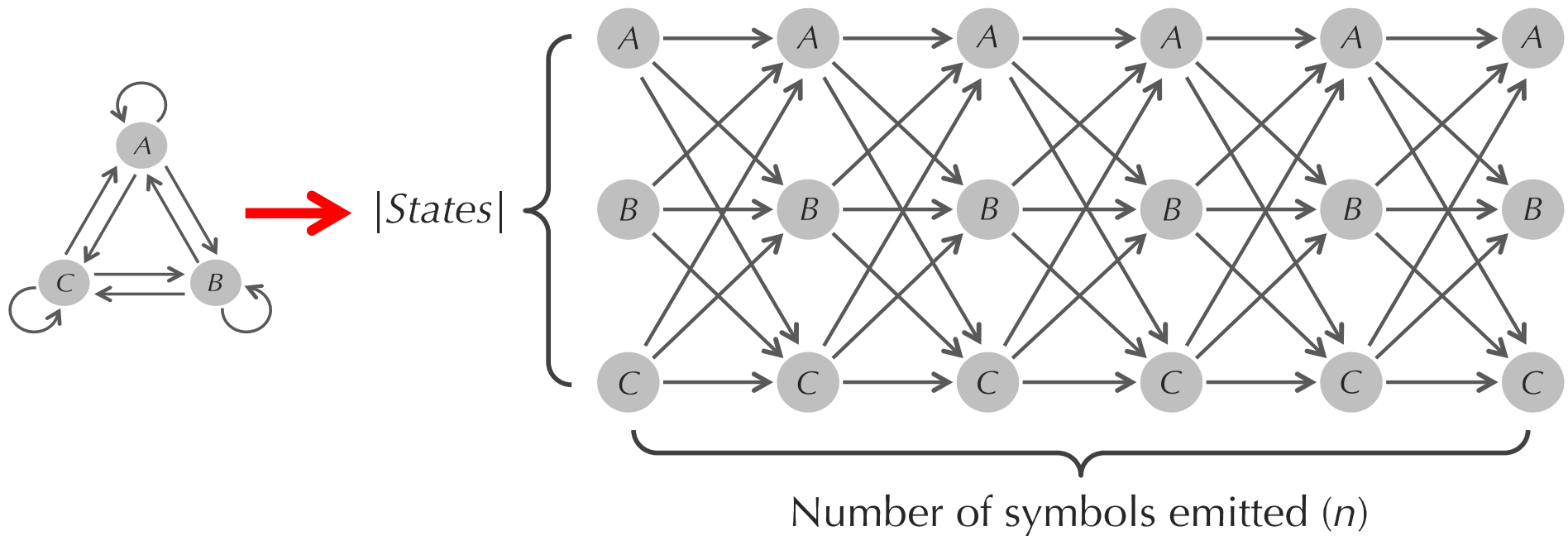
- Initialization:

$$score_{source} = 1$$

- The maximum product weight over all paths from *source* to *sink*:

$$score_{sink} = \max_{\text{all states } l} score_{l,n}$$

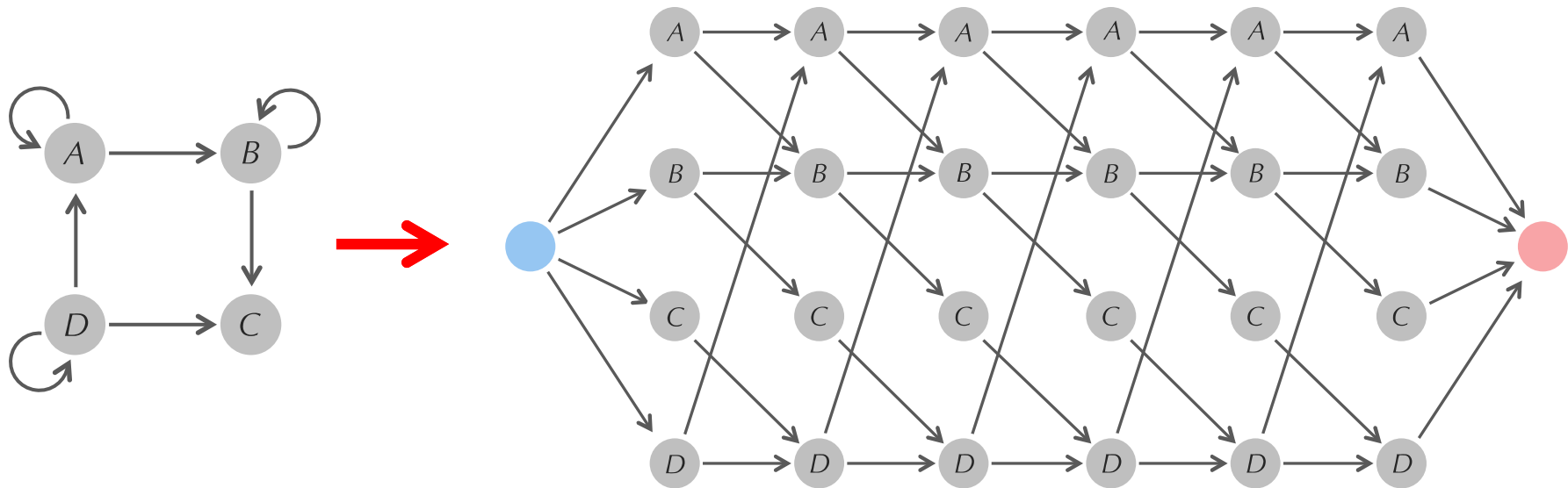
Running Time of the Viterbi Algorithm



Running time \sim #edges in the Viterbi graph
 $\sim O(|States|^2 \cdot n)$

Running Time of the Viterbi Algorithm

Forbidden transition: an edge not represented in the HMM diagram.



Running time \sim #edges in the Viterbi graph
 $\sim O(\text{\#edges in the HMM diagram} \cdot n)$

From Product of Weights to Sum of Their Logarithms

Since $score_{k,i}$ may become small (danger of underflow), biologists prefer to work with logarithms of scores:

$$score_{k,i} = \max_{\text{all states } l} \{ score_{l,i-1} \cdot weight(l,k,i-1) \}$$



$$\log(score_{k,i}) = \max_{\text{all states } l} \{ \log(score_{l,i-1}) + \log(weight(l,k,i-1)) \}$$

This transformation substitutes weights of edges by their logarithms:

product of weights \rightarrow **sum** of weights

Computing $\Pr(\pi)$ Versus Computing $\Pr(x)$

- $\Pr(x, \pi)$: the probability that an HMM follows the hidden path π and emits the string $x = x_1 x_2 \dots x_n$.

x : T H T H H H T H T T H
 π : F F F B B B B B F F F
 .5 .9 .9 .1 .9 .9 .9 .9 .1 .9 .9

$$\Pr(\pi) = \sum_{\text{all possible emitted strings } x} \Pr(x, \pi) = \prod_{i=1, n} \text{transition}_{\pi_i}$$

$$\Pr(x) = \sum_{\text{all possible hidden paths } \pi} \Pr(x, \pi) =$$

$$\Pr(x) = \sum_{\text{all possible hidden paths } \pi} \text{product weight of } \pi$$

$$\text{score}_{\text{sink}} = \max_{\text{all possible hidden paths } \pi} \text{product weight of } \pi$$

What is the Most Likely Outcome of an HMM?

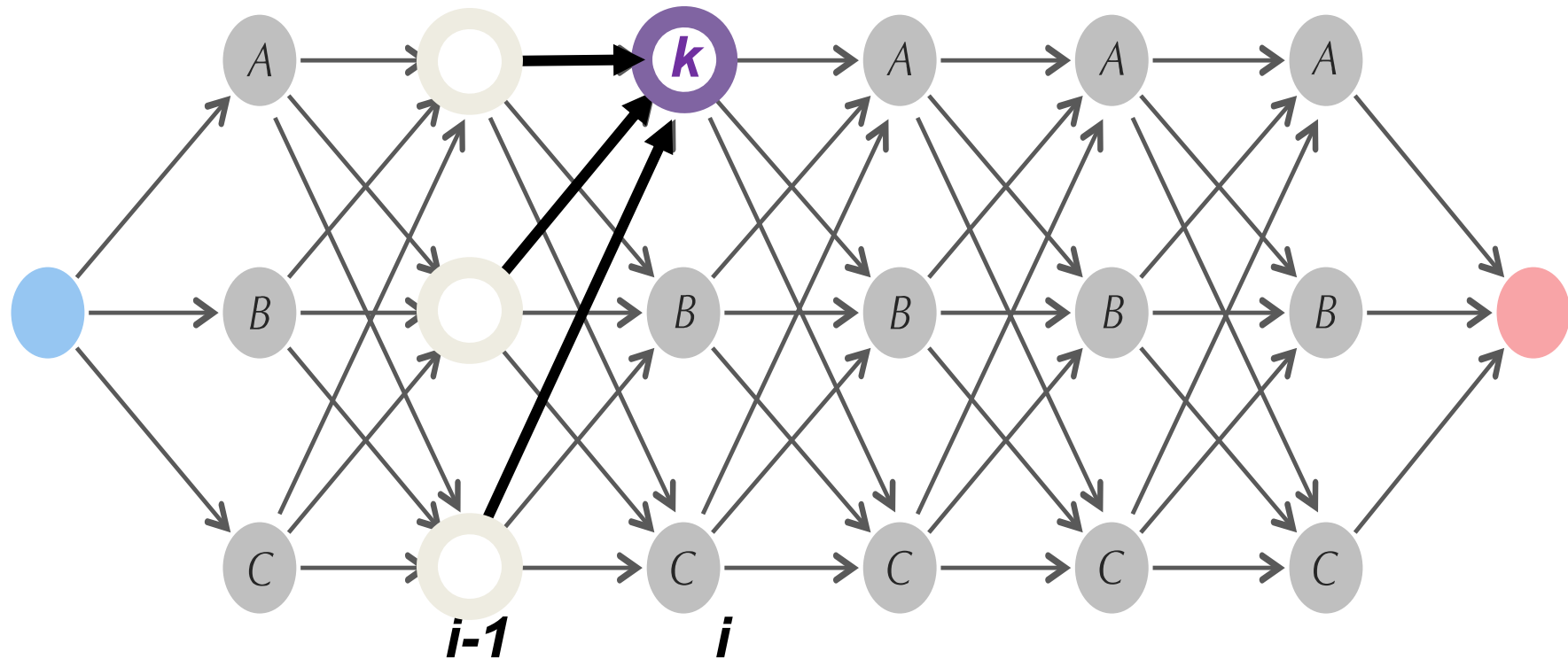
- **Outcome Likelihood Problem.** *Find the probability that an HMM emits a given string.*
- **Input:** A string $x = x_1 \dots x_n$ emitted by an HMM (Σ , States, Transition, Emission).
- **Output:** The probability $\Pr(x)$ that the HMM emits x .

Can you solve the Outcome Likelihood Problem by making a *single change* in the Viterbi recurrence

$$\text{score}_{k,i} = \max_{\text{all states } l} \{ \text{score}_{l,i-1} \cdot \text{weight}(l,k,i-1) \} ?$$

Viterbi Algorithm: From MAX to Σ

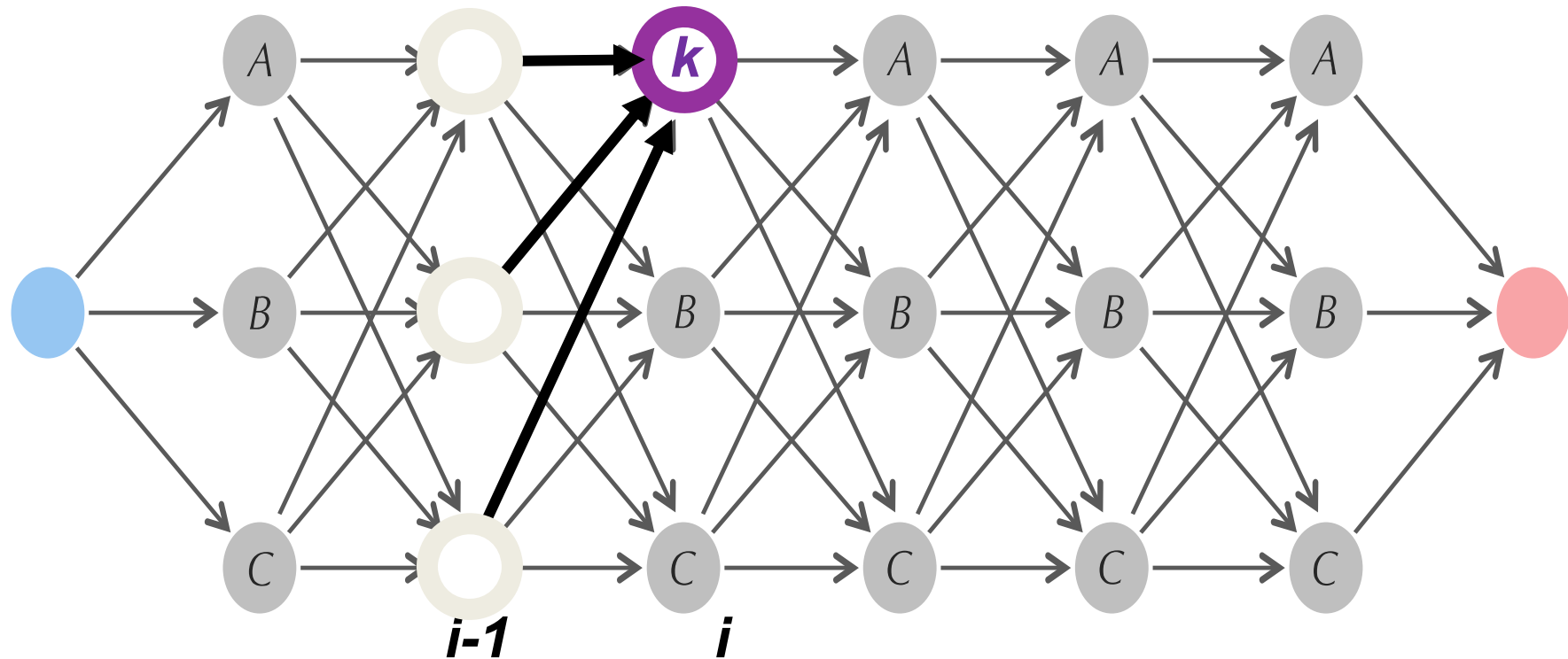
- $forward_{k,i}$: **total** product weight of all paths to (k,i) :



$$\begin{aligned}
 forward_{k,i} &= \sum_{\text{all states } l} \{ forward_{l,i-1} \cdot \text{weight of } (l,i-1) \rightarrow (k,i) \} \\
 &= \sum_{\text{all states } l} \{ forward_{l,i-1} \cdot \text{weight}(l, k, i-1) \}
 \end{aligned}$$

Viterbi Algorithm: From MAX to Σ

- $forward_{k,i}$: **total** product weight of all paths to (k,i) :

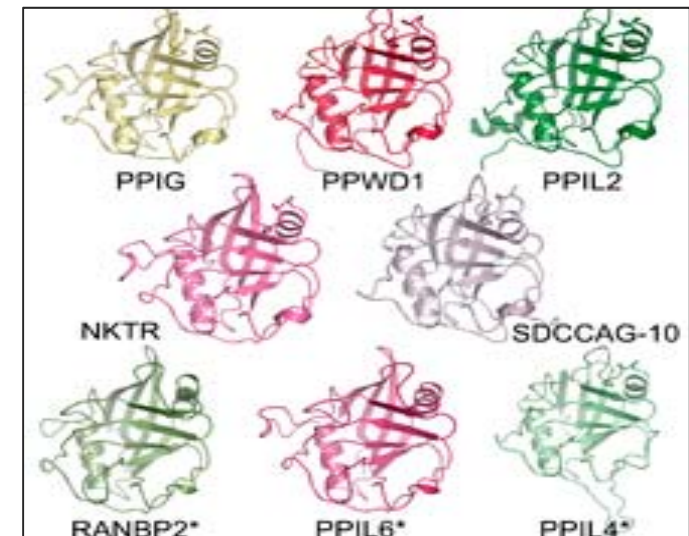


$$score_{k,i} = \mathbf{max}_{\text{all states } l} \{ score_{l,i-1} \cdot weight(l, k, i-1) \}$$

Classifying Proteins into Families

- Proteins are organized into **protein families** represented by multiple alignments.

- A distant cousin may have weak *pairwise* similarities with family members failing a significance test.



- However, it may have weak similarities with *many* family members, indicating a relationship.

From Alignment to Profile

	1	2	3	4	5	6	7	8	
<i>Alignment</i>	A	C	D	E	F	A C	A	D	F
	A	F	D	A	-	- - -	C	C	F
	A	-	-	E	F	D -	F	D	C
	A	C	A	E	F	- - -	A	-	C
	A	D	D	E	F	A A	A	D	F

Remove columns if the fraction of space symbols (“-”) exceeds θ , **the maximum fraction of insertions threshold.**

From Alignment to Profile

		1	2	3	4	5	6	7	8	
<i>Alignment</i>		A	C	D	E	F	A C	A	D	F
		A	F	D	A	-	- - -	C	C	F
		A	-	-	E	F	D -	F	D	C
		A	C	A	E	F	- - -	A	-	C
		A	D	D	E	F	A A	A	D	F
<i>Alignment*</i>		A	C	D	E	F		A	D	F
		A	F	D	A	-		C	C	F
		A	-	-	E	F		F	D	C
		A	C	A	E	F		A	-	C
		A	D	D	E	F		A	D	F
PROFILE(<i>Alignment*</i>)	A	1	0	0	1/5	0	3/5	0	0	
	C	0	2/4	0	0	0	1/5	1/4	2/5	
	D	0	1/4	3/4	0	0	0	3/4	0	
	E	0	0	0	4/5	0	0	0	0	
	F	0	1/4	0	0	1	1/5	0	3/5	

From Profile to HMM

	1	2	3	4	5	6	7	8	
<i>Alignment</i>	A	C	D	E	F	A C	A	D	F
	A	F	D	A	-	- - -	C	C	F
	A	-	-	E	F	D -	F	D	C
	A	C	A	E	F	- - -	A	-	C
	A	D	D	E	F	A A	A	D	F

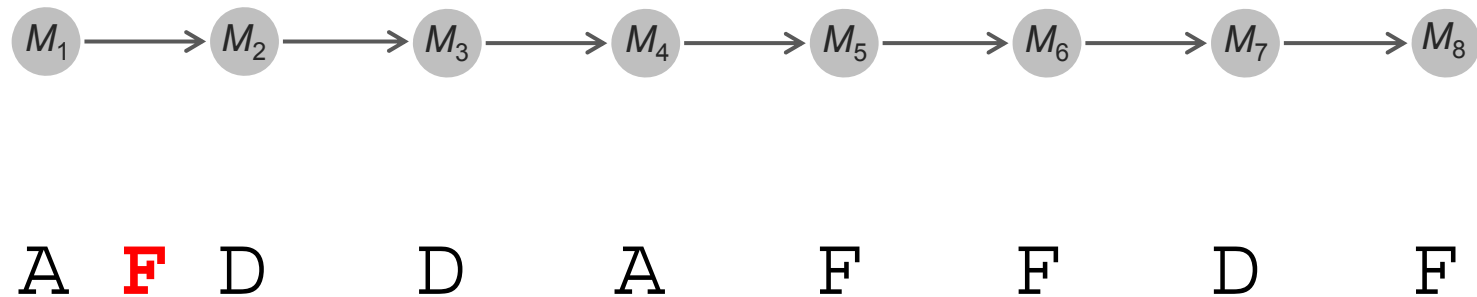
<i>Alignment*</i>	A	C	D	E	F	A	D	F
	A	F	D	A	-	C	C	F
	A	-	-	E	F	F	D	C
	A	C	A	E	F	A	-	C
	A	D	D	E	F	A	D	F

	1	2	3	4	5	6	7	8
A	1	0	0	1/5	0	3/5	0	0
C	0	2/4	0	0	0	1/5	1/4	2/5
D	0	1/4	3/4	0	0	0	3/4	0
E	0	0	0	4/5	0	0	0	0
F	0	1/4	0	0	1	1/5	0	3/5

HMM diagram

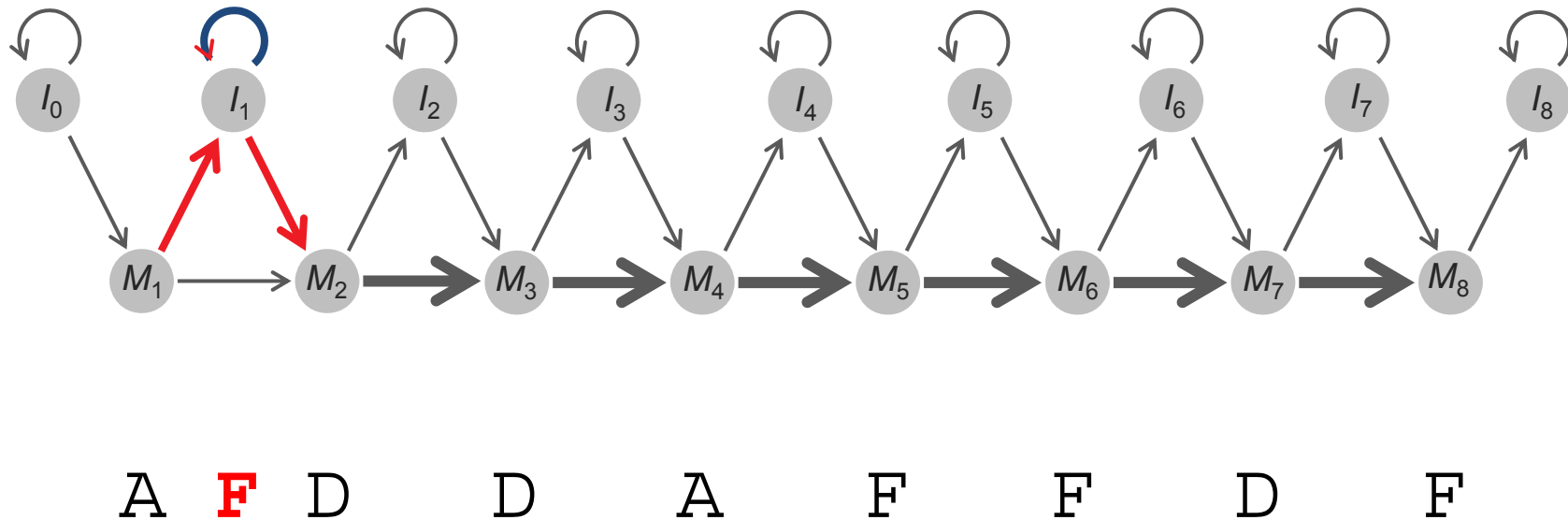
A 1 * D .25 * D .75 * A .20 * F 1 * F .20 * D .75 * F .60

Toward a Profile HMM



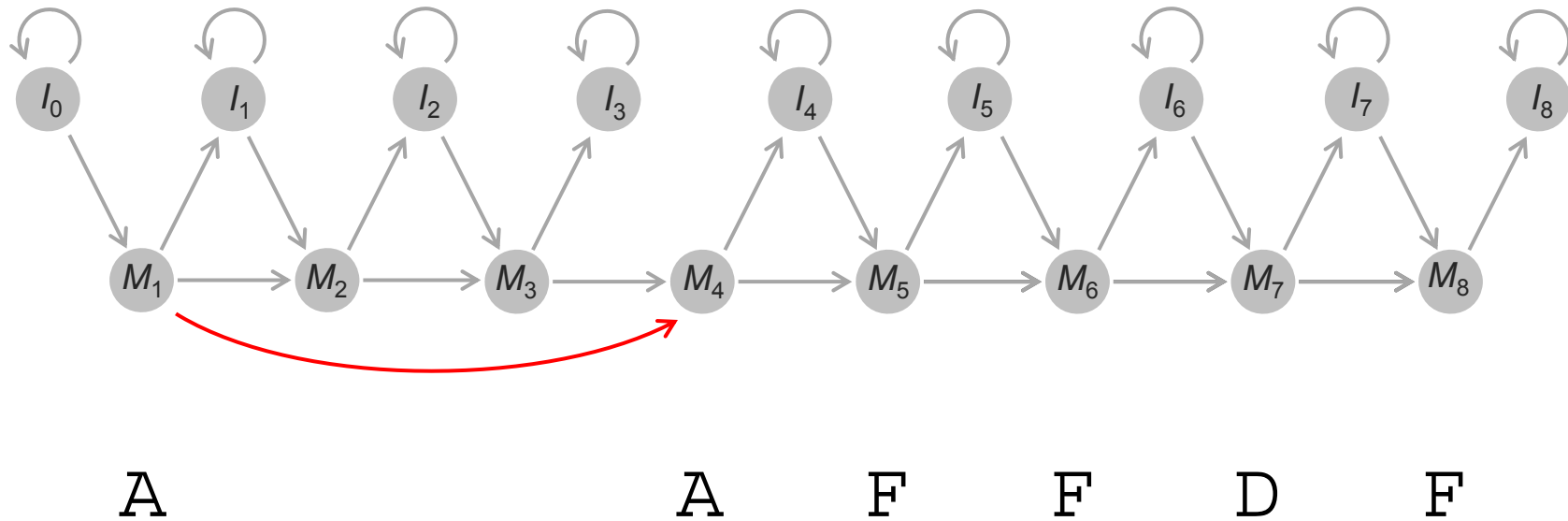
How do we model insertions?

Toward a Profile HMM: Insertions

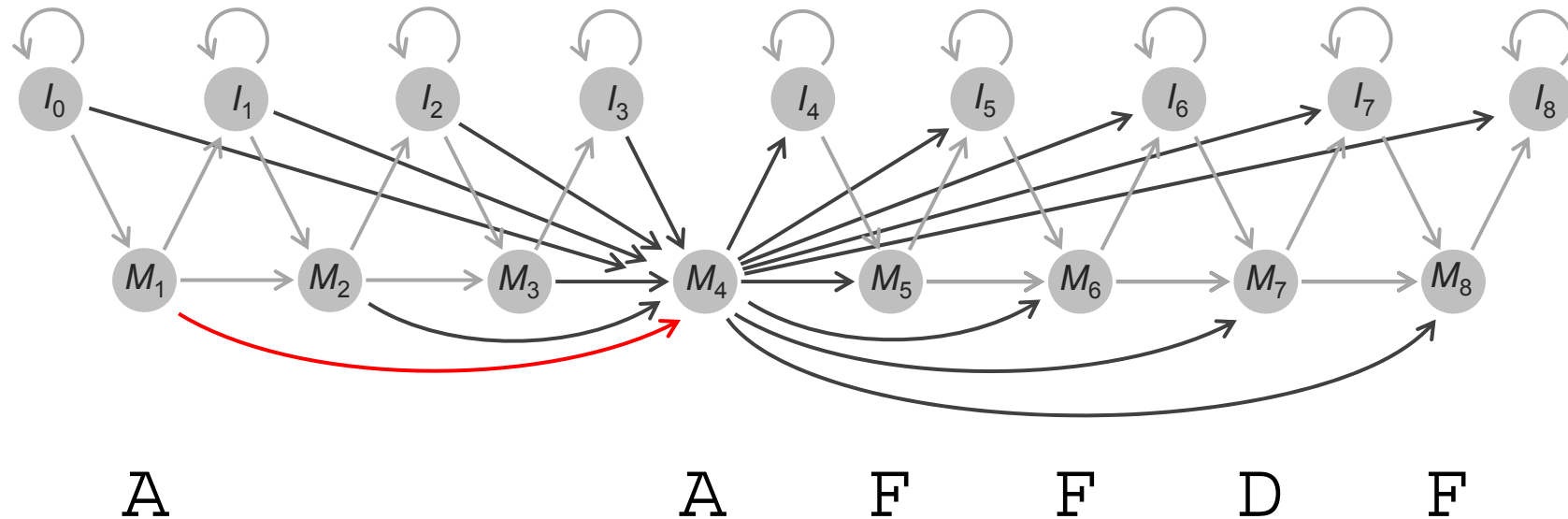


How do we model deletions?

Toward a Profile HMM: Deletions

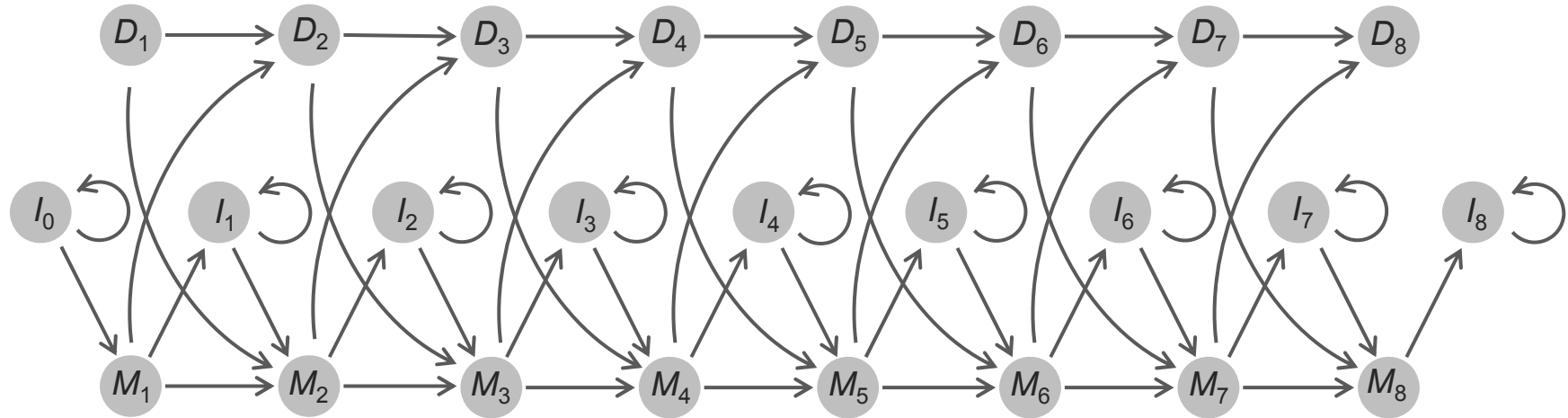


Toward a Profile HMM: Deletions



How many edges are in this HMM diagram?

Adding "Deletion States"



A

A

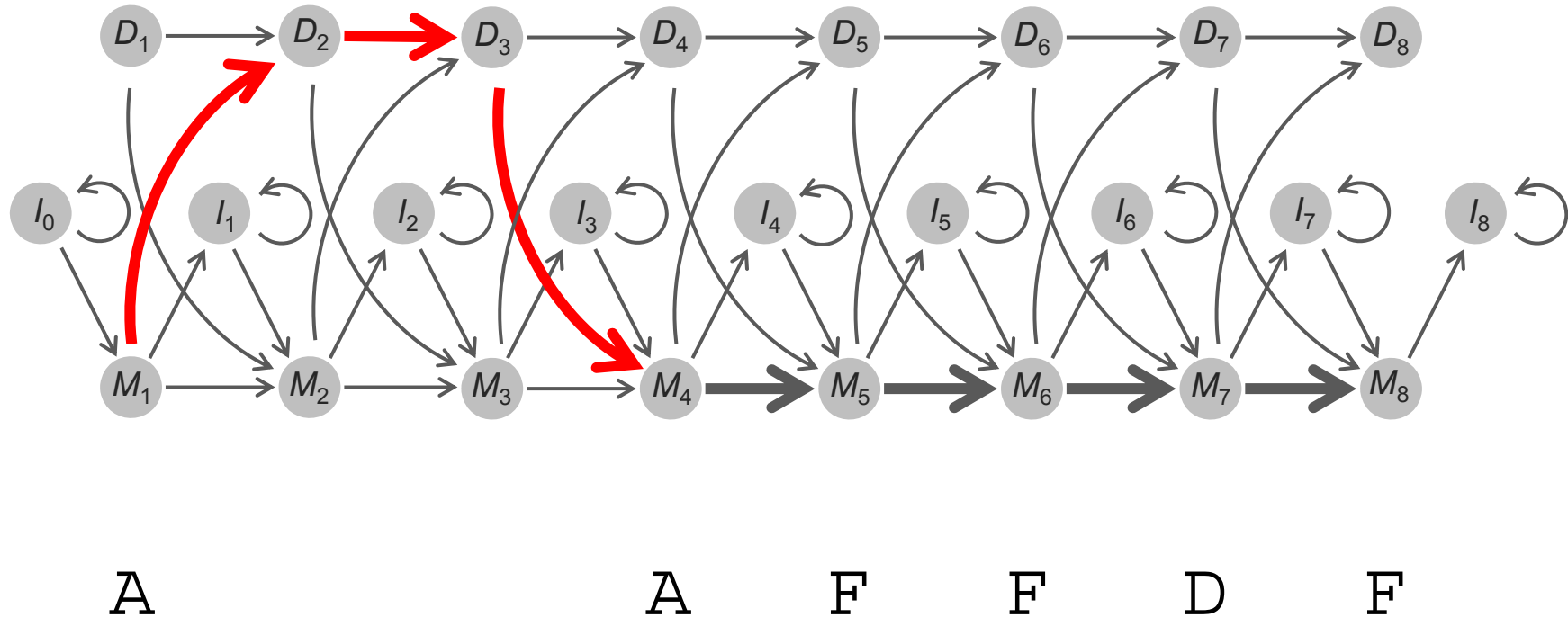
F

F

D

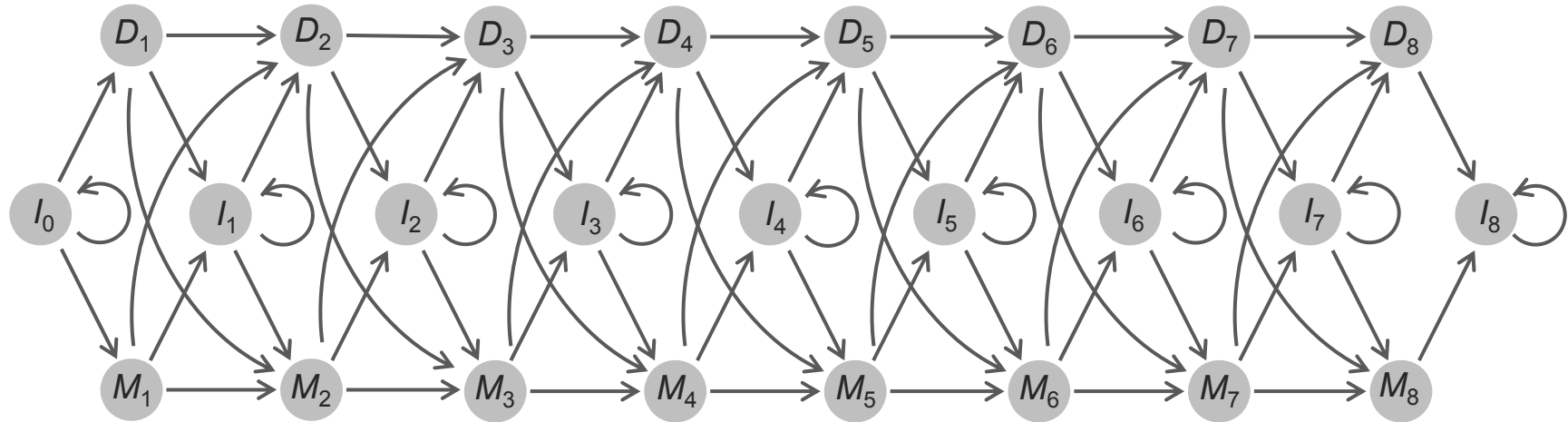
F

Adding “Deletion States”

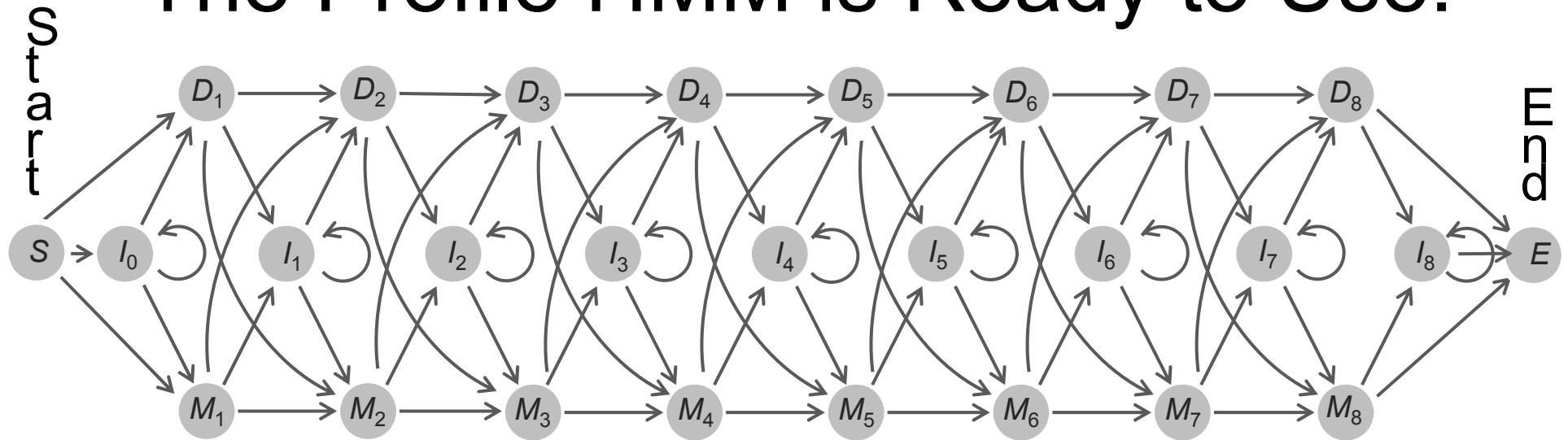


Are any edges still missing in this HMM diagram?

Adding Edges Between Deletion/Insertion States



The Profile HMM is Ready to Use!

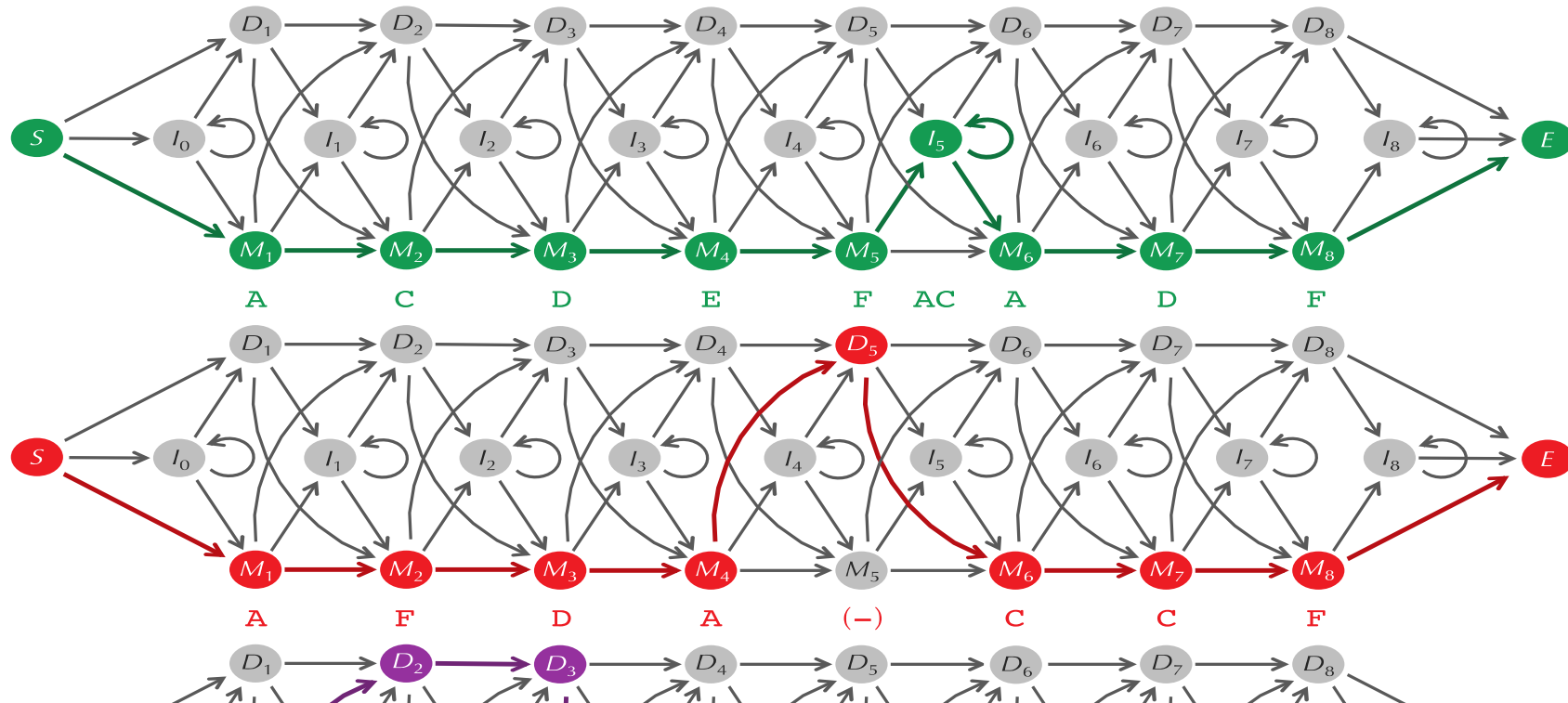


Profile HMM Problem: *Construct a profile HMM from a multiple alignment.*

- **Input:** A multiple alignment *Alignment* and a threshold θ (maximum fraction of insertions per column).
- **Output:** Transition and emission matrices of the profile HMM $HMM(Alignment, \theta)$.

Hidden Paths Through Profile HMM

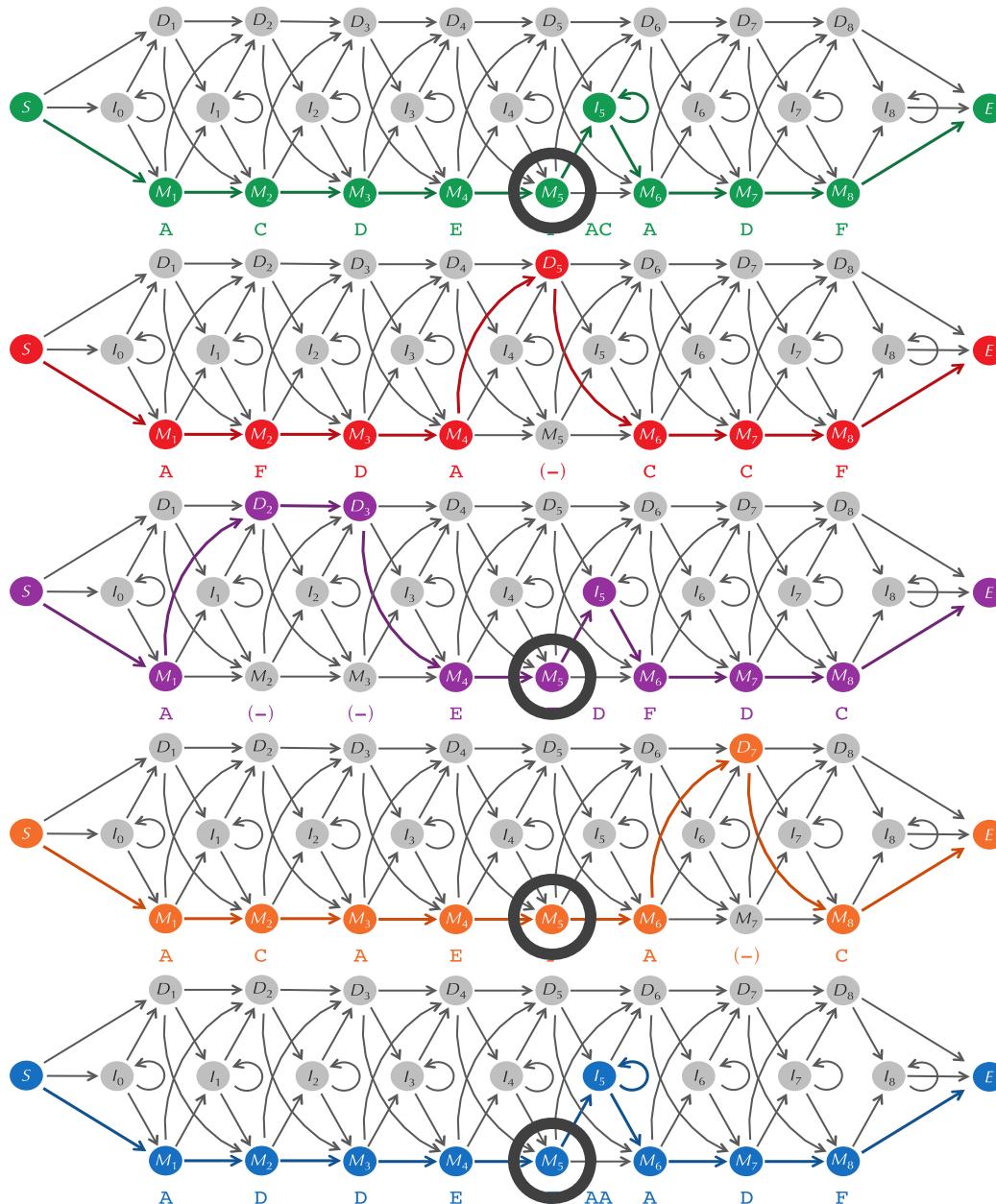
A	C	D	E	F	AC	A	D	F
A	F	D	A	-	-	C	C	F
A	-	-	E	F	D	F	D	C
A	C	A	E	F	-	A	-	C
A	D	D	E	F	AA	A	D	F



Note: this is a hidden path in an *HMM diagram* (not in a *Viterbi graph*).

A (-) (-) E F D F D C

Transition Probabilities of Profile HMM

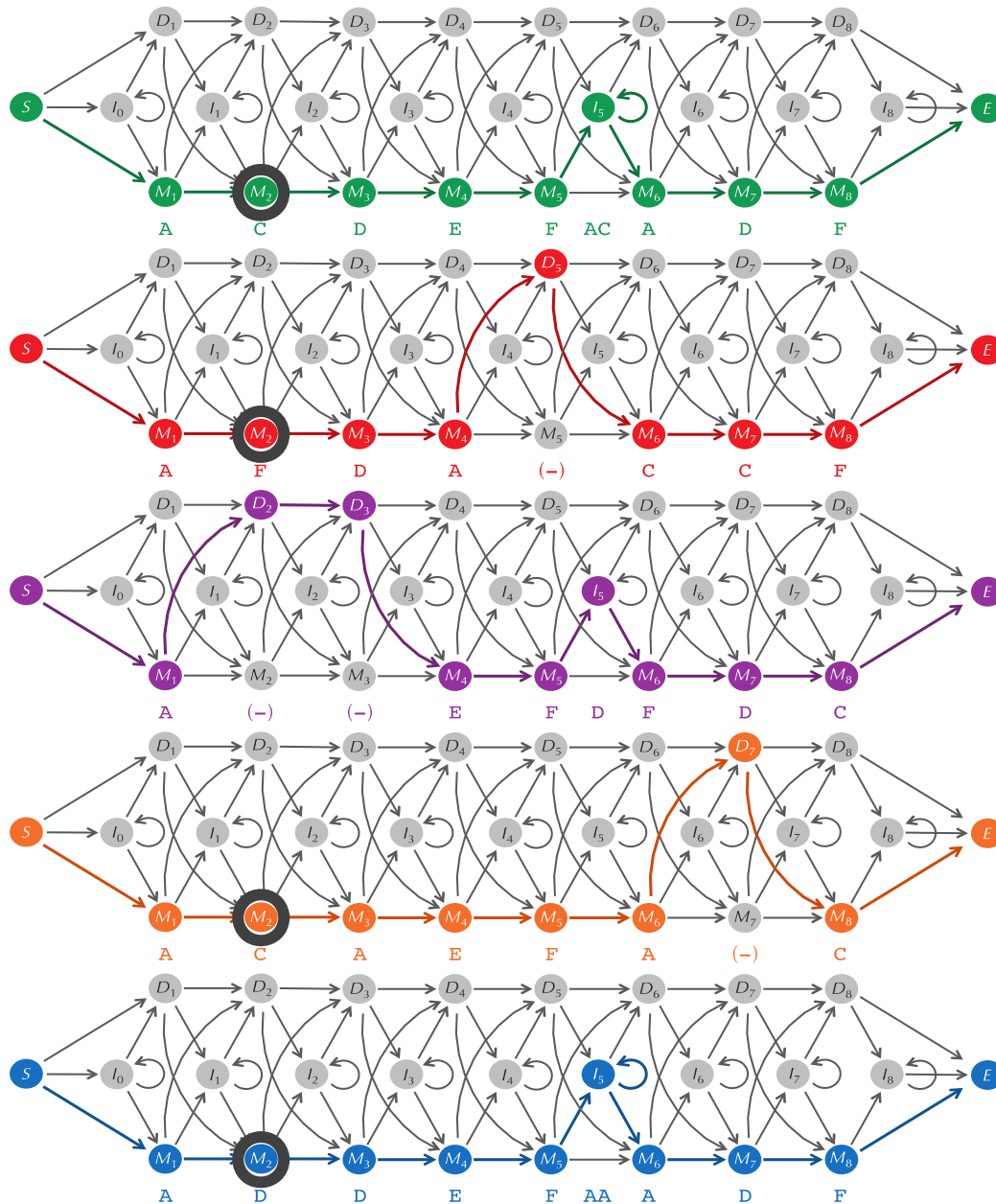


4 transitions from M_5 :

1 + 1 + 1 = 3 into I_5
 1 into M_6
 0 into D_6

$transition_{Match(5), Insertion(5)} = 3/4$
 $transition_{Match(5), Match(6)} = 1/4$
 $transition_{Match(5), Deletion(6)} = 0$

Emission Probabilities of Profile HMM

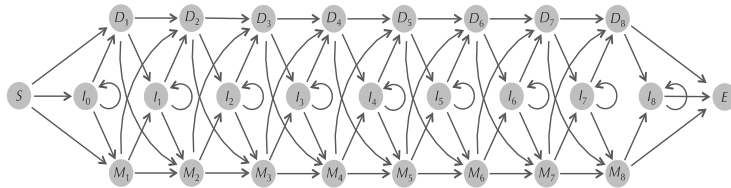


symbols emitted from M_2 :

C, F, C, D

- $emission_{Match(2)}(A) = 0$
- $emission_{Match(2)}(C) = 2/4$
- $emission_{Match(2)}(D) = 1/4$
- $emission_{Match(2)}(E) = 0$
- $emission_{Match(2)}(F) = 1/4$

Forbidden Transitions



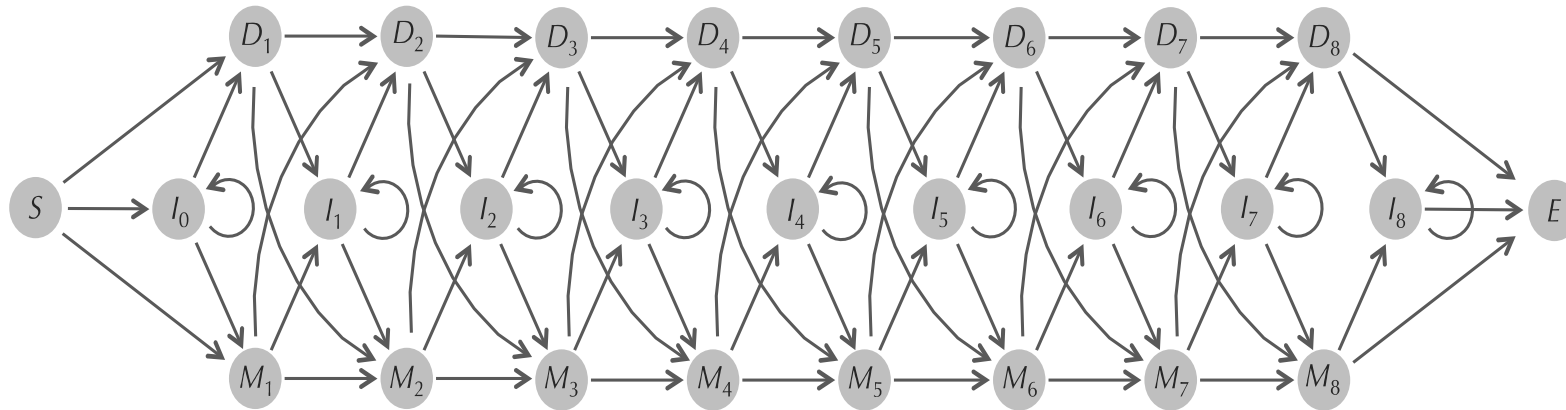
	S	I ₀	M ₁	D ₁	I ₁	M ₂	D ₂	I ₂	M ₃	D ₃	I ₃	M ₄	D ₄	I ₄	M ₅	D ₅	I ₅	M ₆	D ₆	I ₆	M ₇	D ₇	I ₇	M ₈	D ₈	I ₈	E
S			1																								
I ₀																											
M ₁						.8	.2																				
D ₁																											
I ₁																											
M ₂																											
D ₂																											
I ₂																											
M ₃																											
D ₃																											
I ₃																											
M ₄																											
D ₄																											
I ₄																											
M ₅																											
D ₅																											
I ₅																											
M ₆																											
D ₆																											
I ₆																											
M ₇																											
D ₇																											
I ₇																											
M ₈																											
D ₈																											
I ₈																											
E																											

Gray cells:
edges in the
HMM diagram.

Clear cells:
forbidden
transitions.

Don't forget
pseudocounts:
 $HMM(\text{Alignment}, \theta, \sigma)$

Aligning a Protein Against a Profile HMM



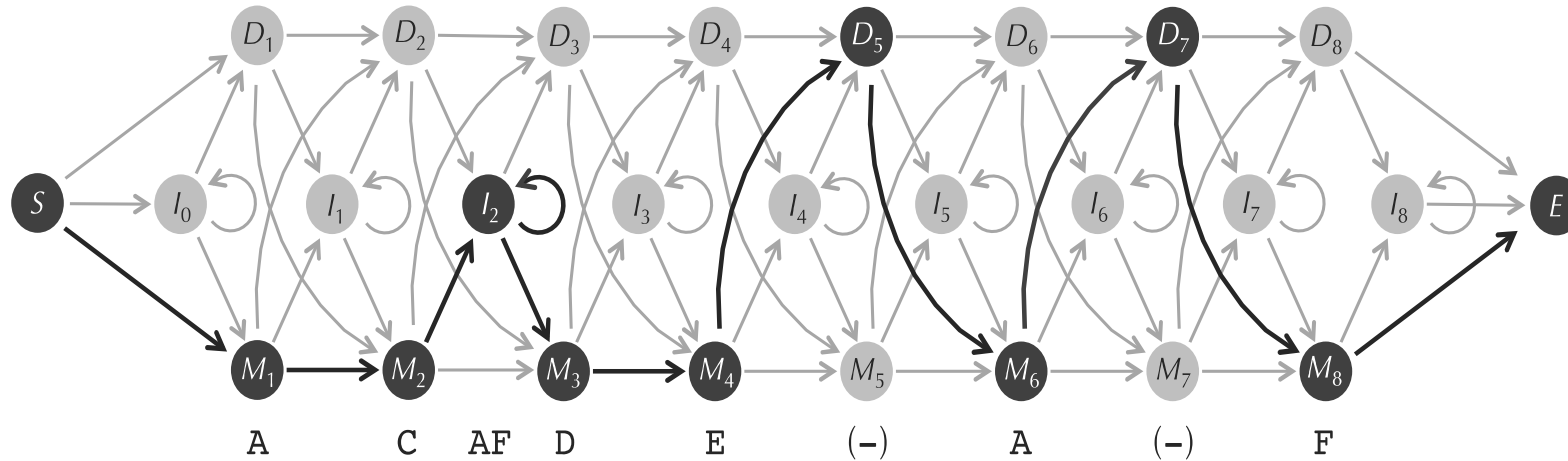
Alignment

A	C	---	D	E	F	AC	A	D	F
A	F	---	D	A	-	---	C	C	F
A	-	---	-	E	F	D-	F	D	C
A	C	---	A	E	F	---	A	-	C
A	D	---	D	E	F	AA	A	D	F

Protein

ACAFDEAF

Aligning a Protein Against a Profile HMM



Alignment

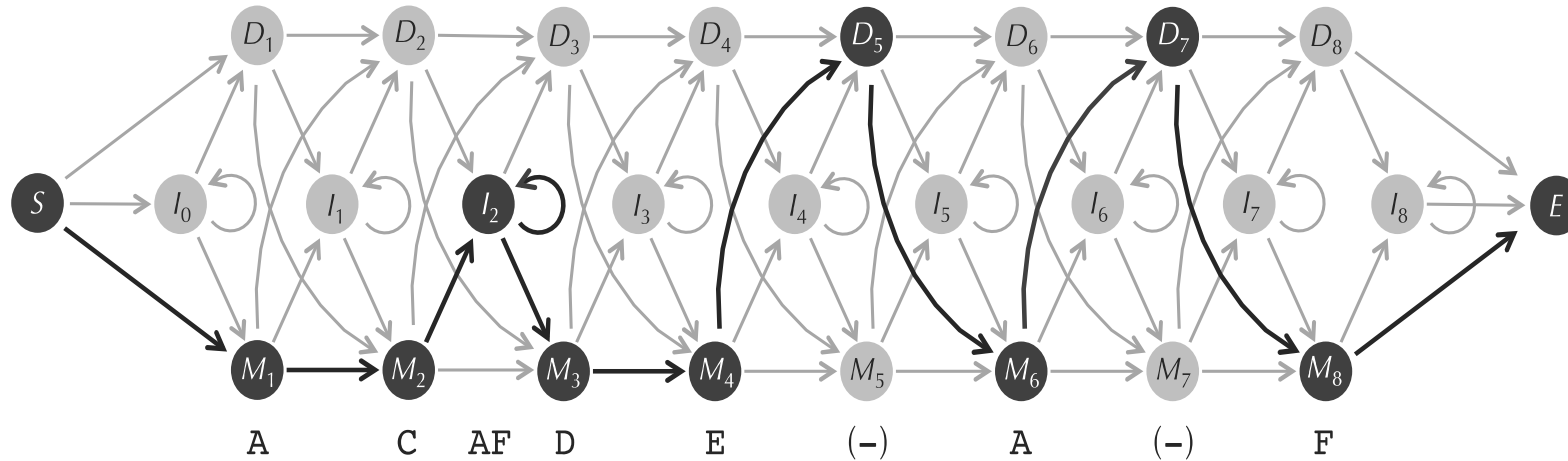
A	C	---	D	E	F	AC	A	D	F
A	F	---	D	A	-	---	C	C	F
A	-	---	-	E	F	D-	F	D	C
A	C	---	A	E	F	---	A	-	C
A	D	---	D	E	F	AA	A	D	F

Protein

ACAFDEAF

Apply Viterbi algorithm to find optimal hidden path!

Aligning a Protein Against a Profile HMM



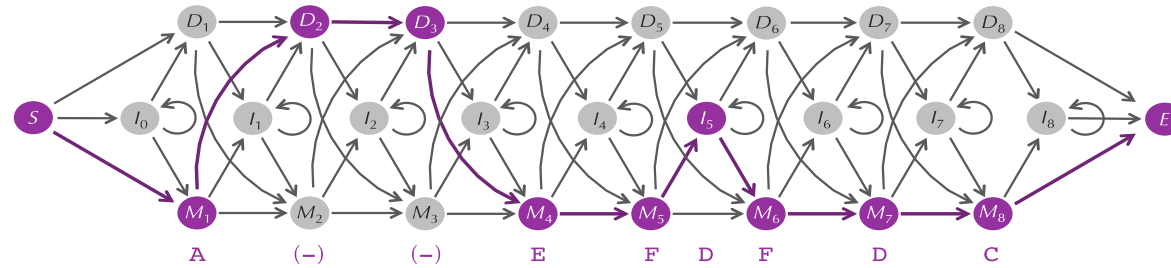
Alignment

Protein

A	C	--	D	E	F	AC	A	D	F
A	F	--	D	A	-	--	C	C	F
A	-	--	-	E	F	D-	F	D	C
A	C	--	A	E	F	--	A	-	C
A	D	--	D	E	F	AA	A	D	F
A	C	AF	D	E	-	--	A	-	F

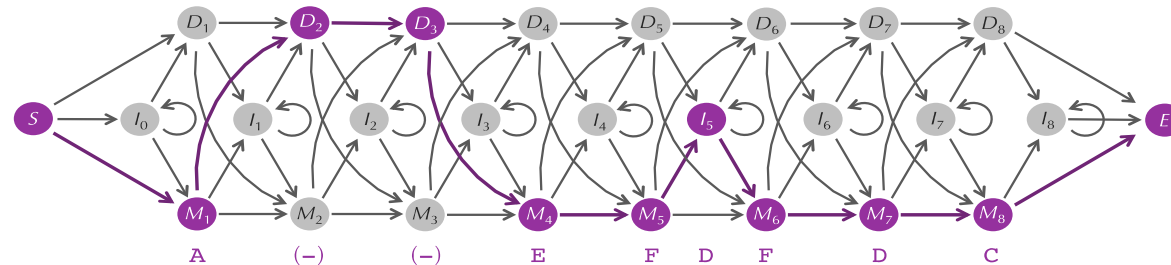
Apply Viterbi algorithm to find optimal hidden path!

Profile HMM diagram



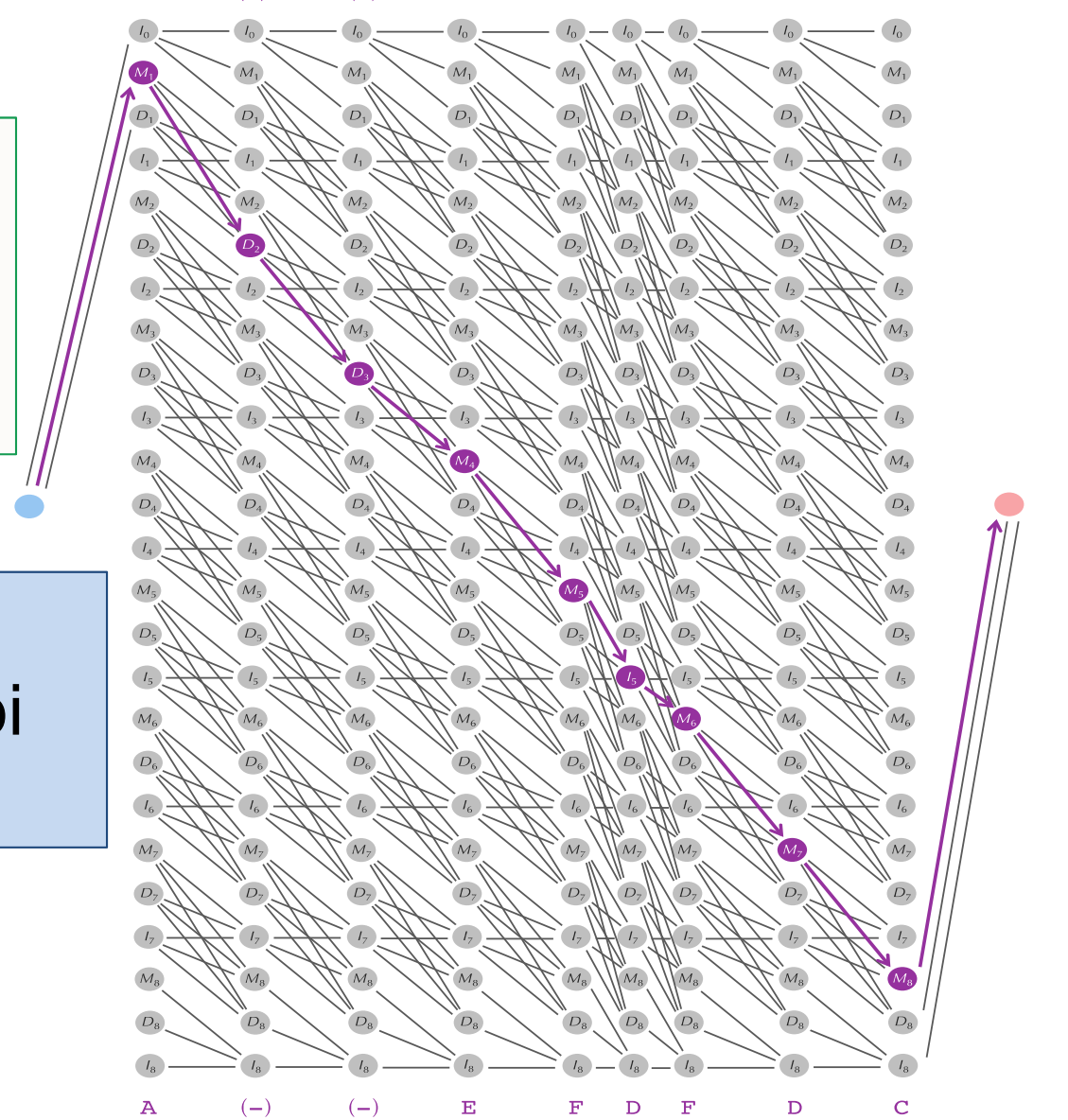
How many rows and columns does the Viterbi graph of this profile HMM have?

Profile HMM diagram

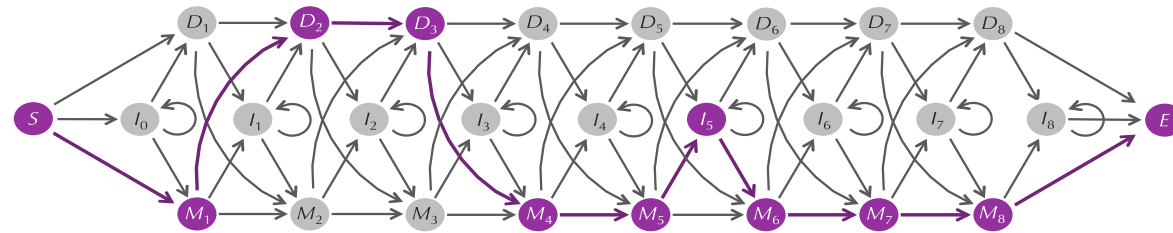


Viterbi graph of profile HMM:
#columns=
#visited states

What is wrong with this Viterbi graph?

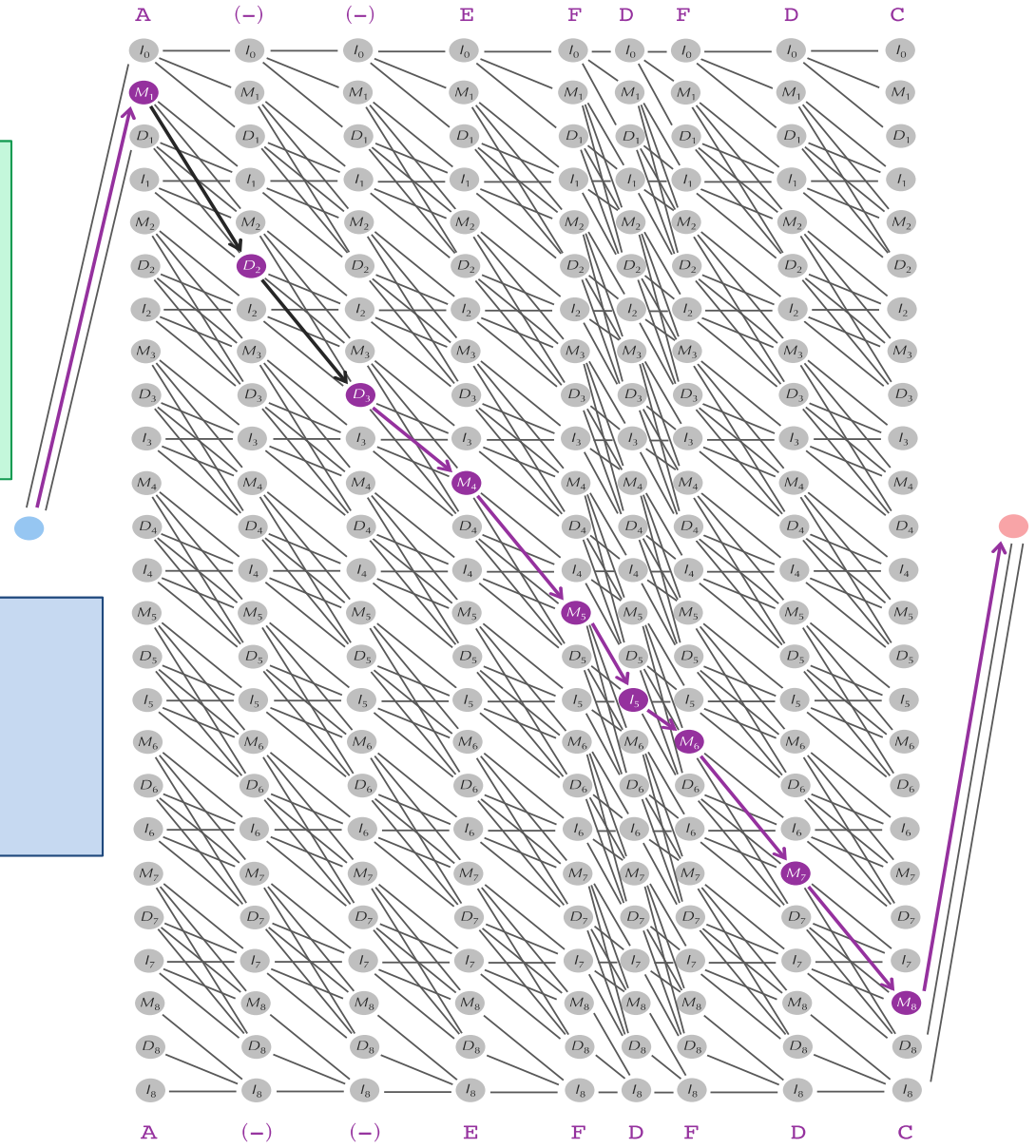


Profile HMM diagram

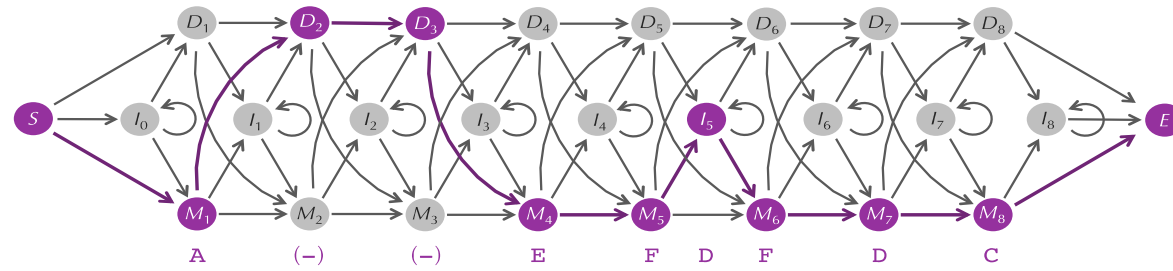


Viterbi graph of profile HMM:
 #columns=
 #visited states

By definition,
 #columns =
 #emitted
 symbols

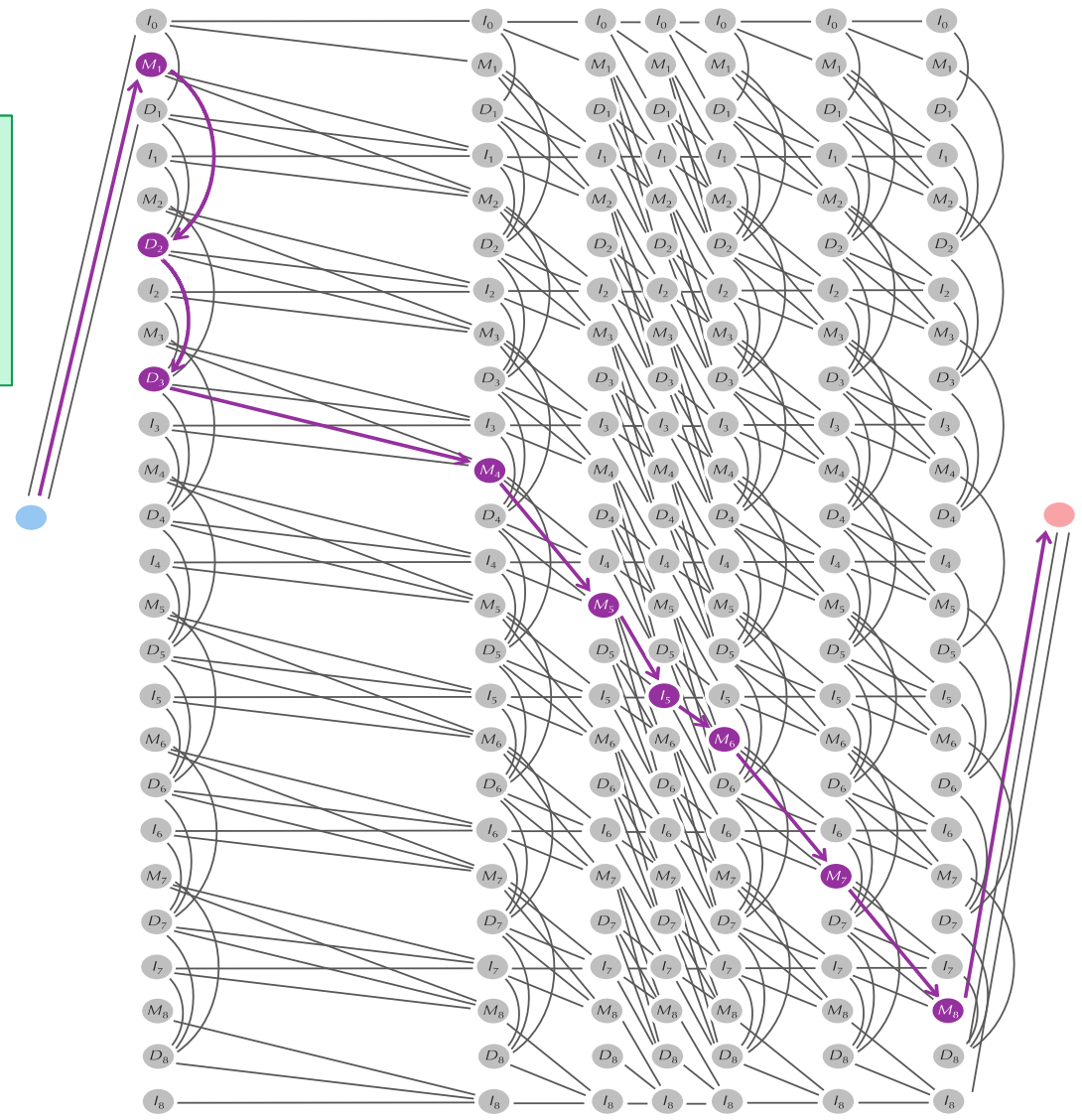


Profile HMM diagram

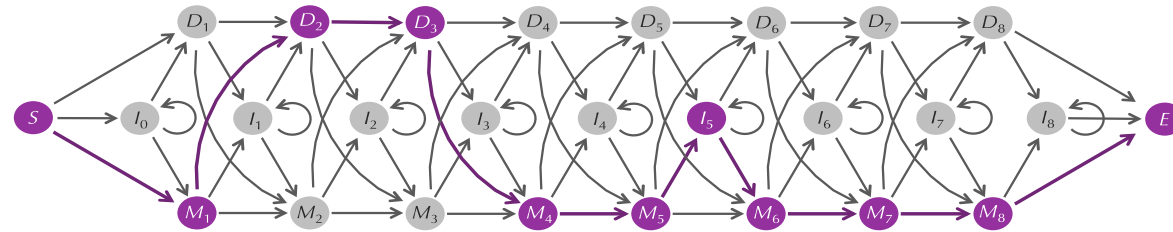


Nearly correct Viterbi graph of profile HMM:

Vertical edges enter "silent" deletion states

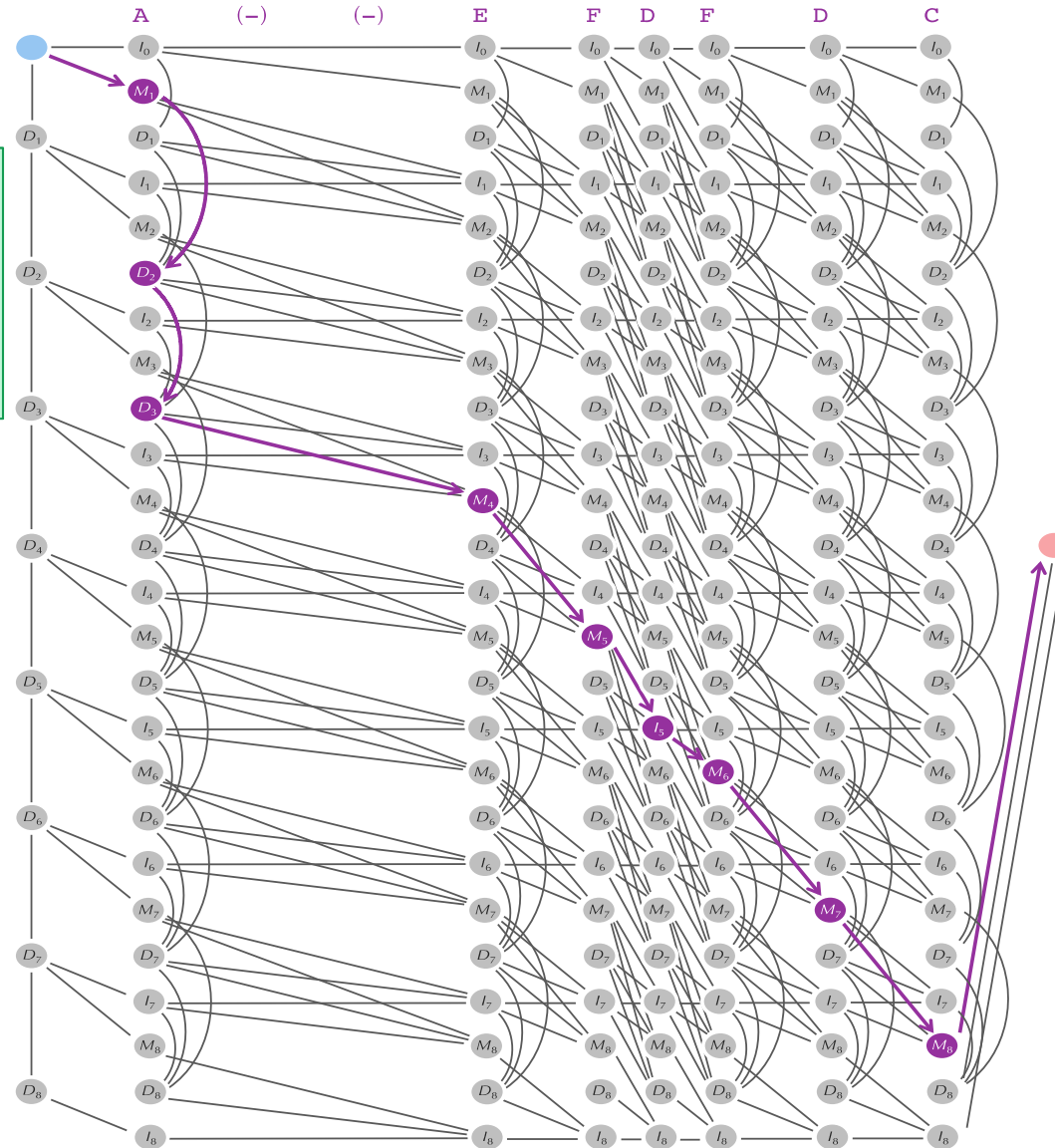


Profile HMM diagram

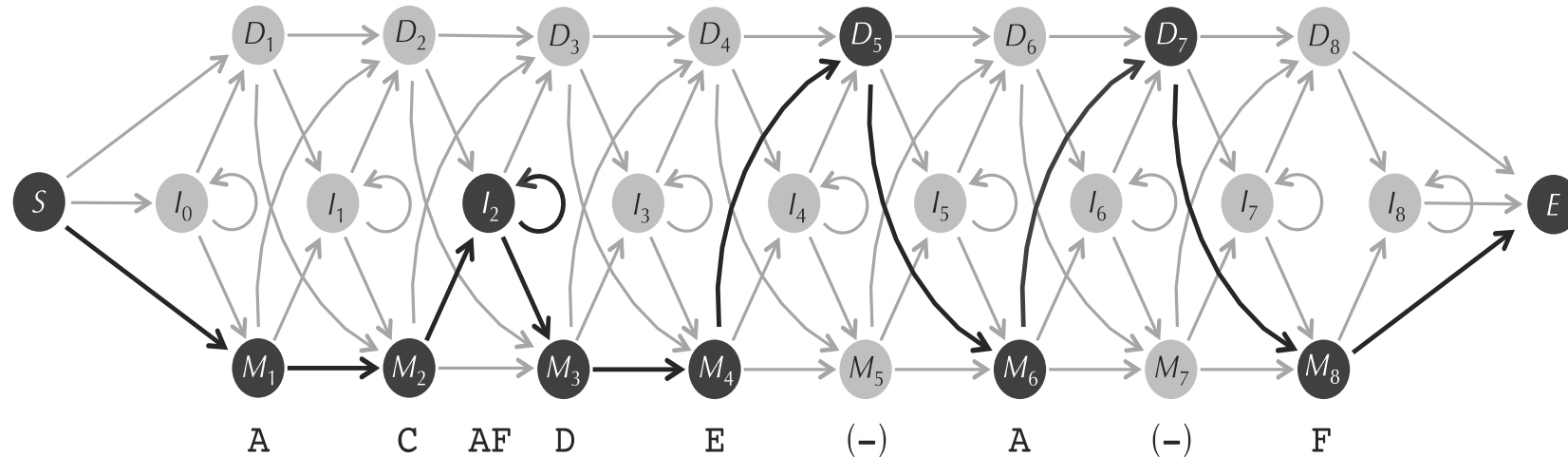


Correct Viterbi graph of profile HMM:

Adding 0-th column that contains only silent states



Alignment with a Profile HMM

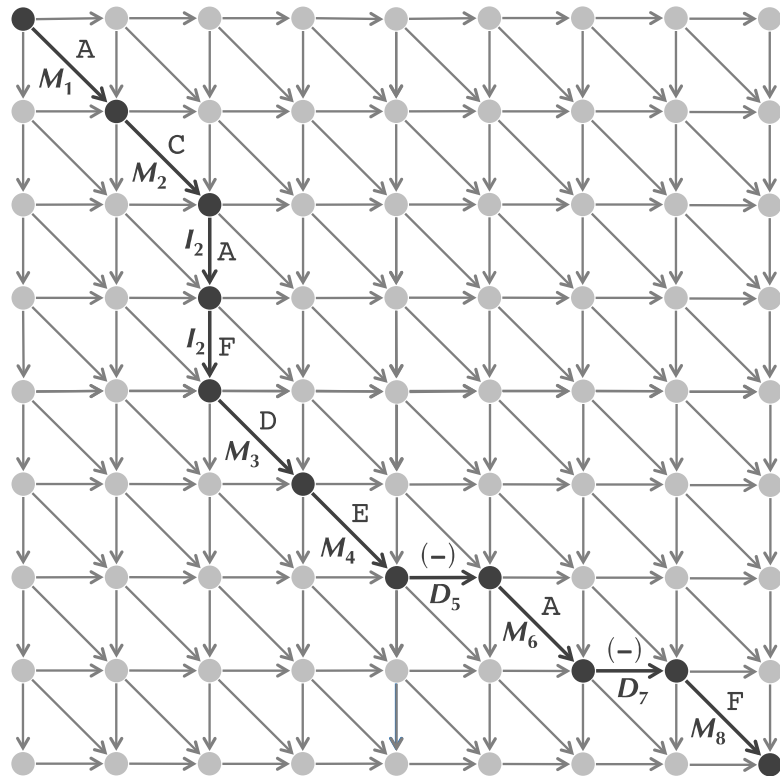
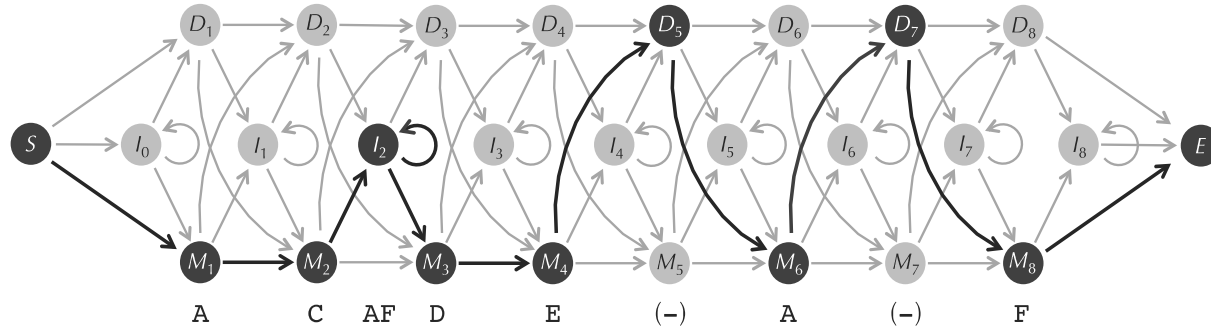


Sequence Alignment with Profile HMM Problem:

Align a new sequence to a family of aligned sequences using a profile HMM.

- **Input:** A multiple alignment *Alignment*, a string *Text*, a threshold θ (maximum fraction of insertions per column), and a pseudocount σ .
- **Output:** An optimal hidden path emitting *Text* in the profile HMM $HMM(\text{Alignment}, \theta, \sigma)$.

Have I Wasted Your Time?

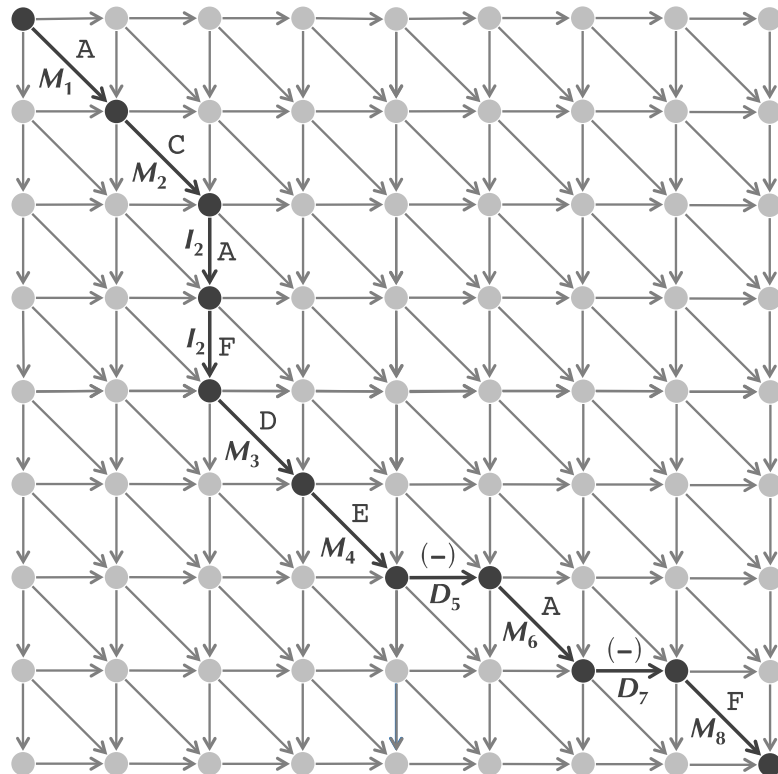
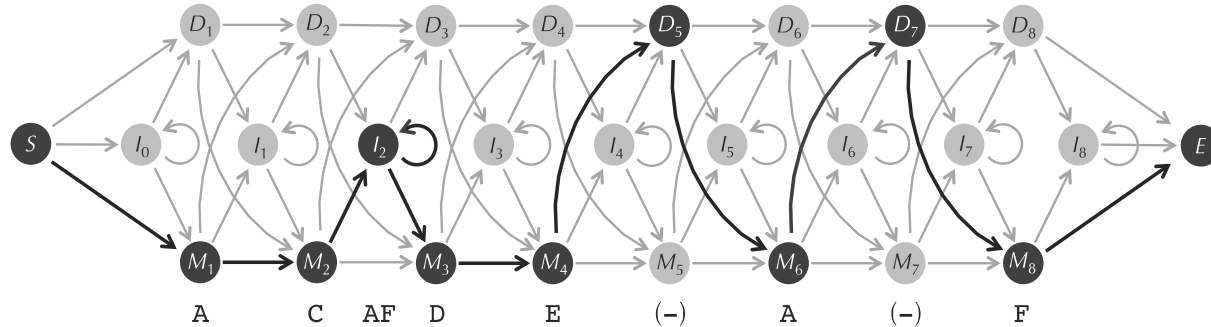


$$s_{M(j),i} = \max \begin{cases} s_{I(j-1),i-1} * \text{weight}(I(j-1), M(j), i-1) \\ s_{D(j-1),i-1} * \text{weight}(D(j-1), M(j), i-1) \\ s_{M(j-1),i-1} * \text{weight}(M(j-1), M(j), i-1) \end{cases}$$

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

The choice of alignment path is now based on varying transition and emission probabilities!

I Have Not Wasted Your Time!



$$s_{M(j),i} = \max \begin{cases} s_{I(j-1),i-1} * \text{weight}_{i-1}(I(j-1), M(j)) \\ s_{D(j-1),i-1} * \text{weight}_{i-1}(D(j-1), M(j)) \\ s_{M(j-1),i-1} * \text{weight}_{i-1}(M(j-1), M(j)) \end{cases}$$

Individual scoring parameters for each edge in the alignment graph capture subtle similarities that evade traditional alignments.



HMM Parameter Estimation

- Thus far, we have assumed that the transition and emission probabilities are known.
- Imagine that you only know that the crooked dealer is using two coins and observe



HHTHHHTHHHTTTTHTTTTH

What are the biases of the coins and how often the dealer switches coins?

Can we develop an algorithm for parameter estimation for an *arbitrary* HMM?

If Dealer Reveals the Hidden Path...

HMM Parameter Estimation Problem: *Find optimal parameters explaining the emitted string and the hidden path.*

- **Input:** A string $x = x_1 \dots x_n$ emitted by a k -state HMM with unknown transition and emission probabilities following a **known** hidden path $\pi = \pi_1 \dots \pi_n$.
- **Output:** *Transition and Emission matrices that maximize $\Pr(x, \pi)$ over all possible matrices of transition and emission probabilities.*

HHTHHHTHHHTTTTHTTTTH

BFFBFFFBFFFBFFBFFBFF

If the Hidden Path is Known...

- $T_{l,k}$: #transitions from state l to state k in path π .

$$\begin{aligned} \text{transition}_{l,k} &= \\ \text{\#transitions from state } l \text{ to state } k &/ \text{\# all transitions from } l \\ &= T_{l,k} / \text{\#visits to state } l \end{aligned}$$

HHTHHHTHHHTTTTHTTTTH
BFFBFFFBBFFBFFB BBBFF

$$T_{B,F} = \frac{5}{9}$$

If the Hidden Path is Known...

- $T_{l,k}$: # transitions from state l to state k in path π .
- $E_k(b)$: # times symbol b is emitted when path π is in state k .

$$\begin{aligned} \text{transition}_{l,k} &= \\ & \# \text{transitions from state } l \text{ to state } k / \# \text{ all transitions from } l \\ & = T_{l,k} / \# \text{visits to state } l \end{aligned}$$

$$\begin{aligned} \text{emission}_k(b) &= \\ & \# \text{times symbol } b \text{ is emitted in state } k / \# \text{ all symbols emitted in state } \\ & \quad k \end{aligned}$$

= $E_k(b) / \# \text{visits to state } l$

HHTHHHTHHHTTTTHTTTH
↑ ↑ ↑ ↑ ↑ ↑ ↑

BFFBFFFBBFFBFFBBBFFF

$$E_F(\text{T}) = 6/11$$

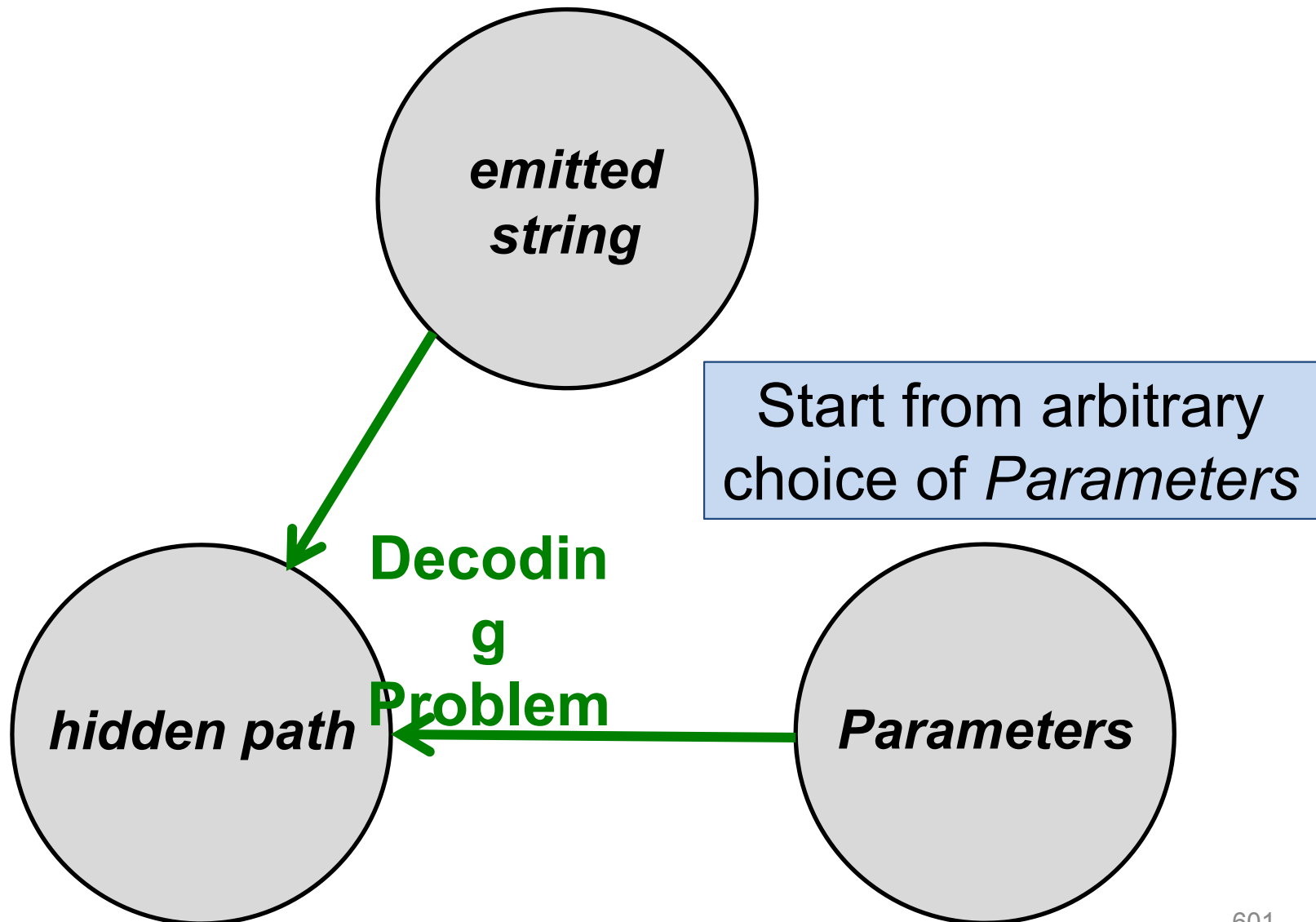
$$T_{B,F} = 5/599$$

When BOTH *HiddenPath* and *Parameters* Are Unknown

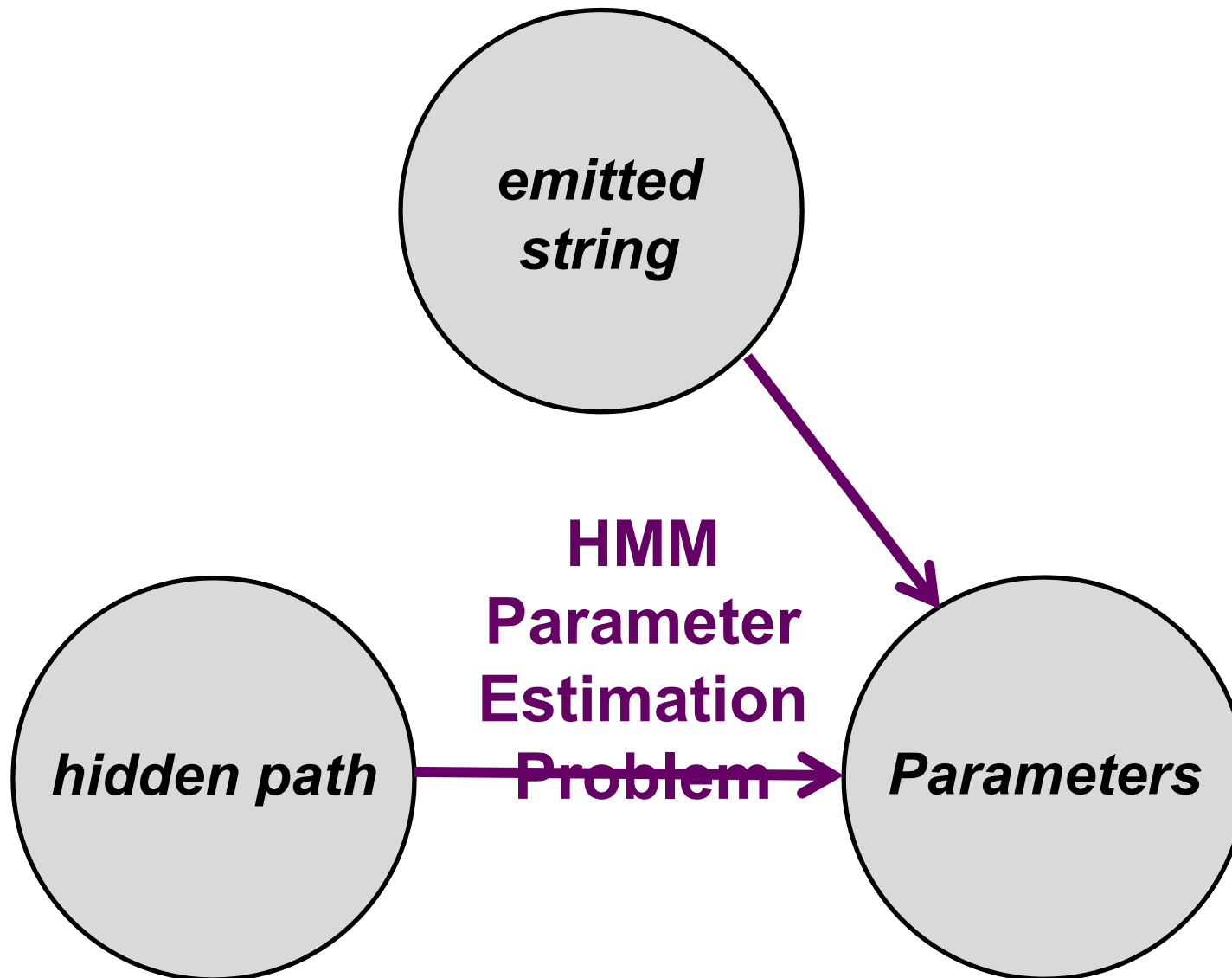
HMM Parameter Learning Problem. *Estimate the parameters of an HMM explaining an emitted string.*

- **Input:** A string $x = x_1 \dots x_n$ emitted by a k -state HMM with unknown transition and emission probabilities.
- **Output:** Matrices *Transition* and *Emission* that maximize $\Pr(x, \pi)$ over all possible transition and emission matrices and over all hidden paths π .

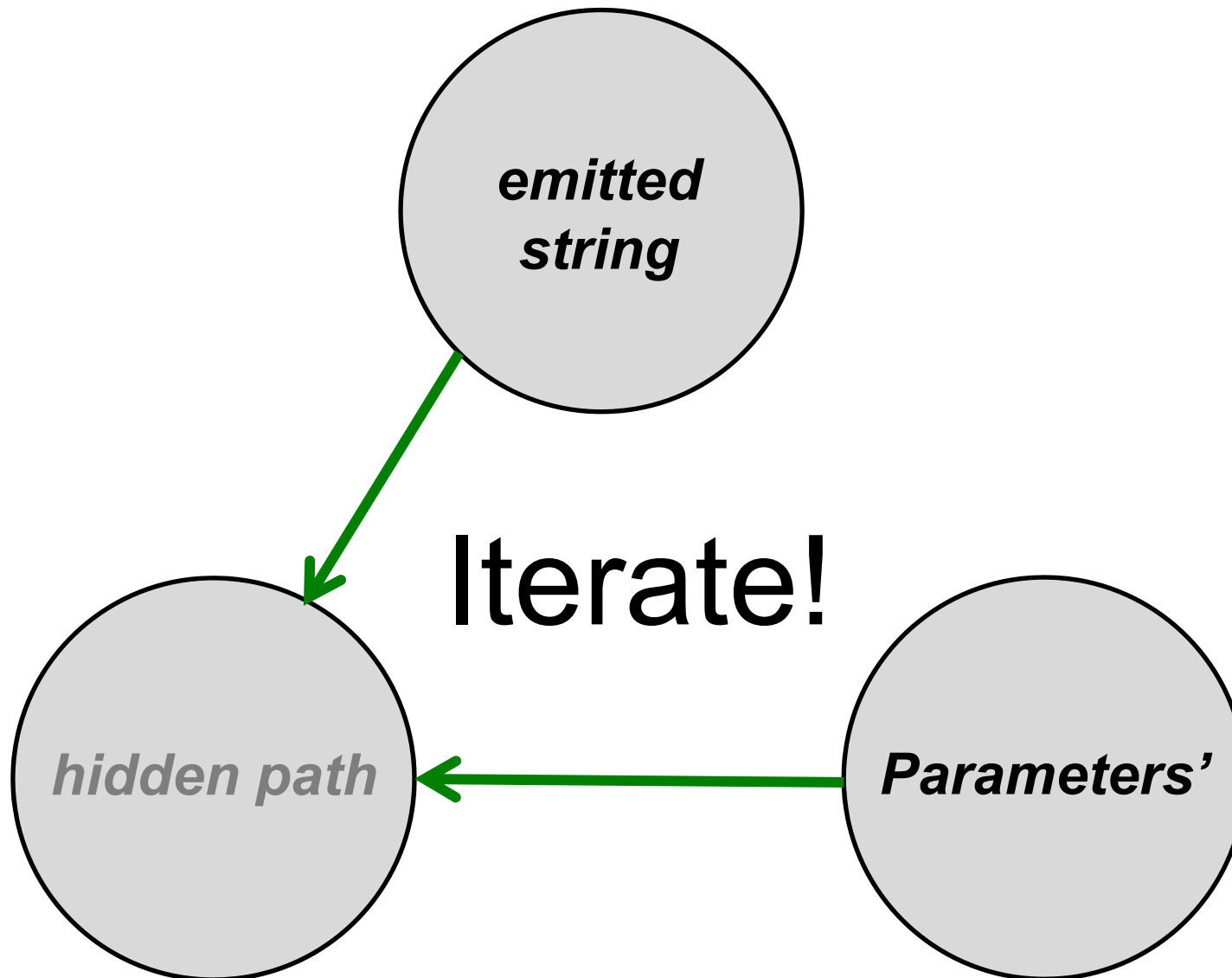
Reconstructing *HiddenPath* AND *Parameters*



Reconstructing *HiddenPath* AND *Parameters*



Viterbi Learning



Changing the Question

- The Viterbi algorithm gives a “yes” or “no” answer to the question: *“Was the HMM in state k at time i given that it emitted string x ?”*

This question fails to account for how certain we are in the “yes”/“no” answer. How can we change this **hard** question into a **soft** one?

What Is $\Pr(\pi_i=k, x)$?

$\Pr(\pi_i=k, x)$: the *unconditional* probability $\Pr(\pi_i=k, x)$ that a hidden path will pass through state k at time i and emit x .

What is the probability that the dealer was using the Fair coin at the 5th flip *given* that he generated a sequence of flips **HHTHTHHHTT**?

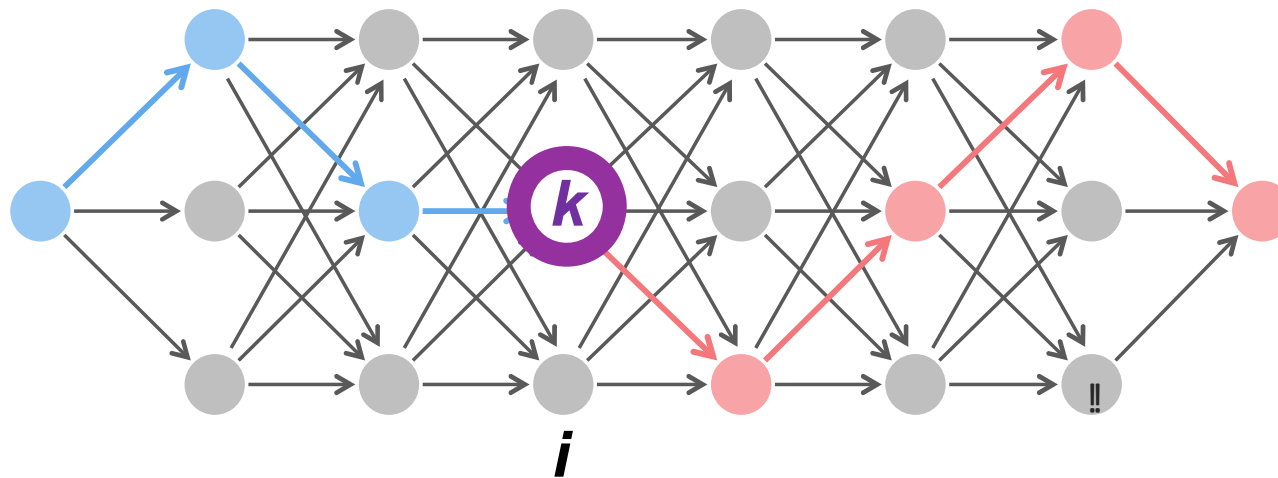
$$\Pr(\pi_i=k, x):$$

Total Product Weight of All Paths Through



$\Pr(\pi_i=k, x)$: the *unconditional* probability that a hidden path will pass through state k at time i and emit x .

$$\Pr(\pi_i=k, x) = \sum_{\text{all paths } \pi \text{ with } \pi_i=k} \Pr(x, \pi)$$



$$\sum_{\text{all possible states } k, \text{ all possible paths } x} \Pr(\pi_i=k, x) = 1$$

What Is $\Pr(\pi_i = k | x)$?

$\Pr(\pi_i = k | x)$: the *conditional* probability that the HMM was in state k at time i *given* that it emitted string x .

What is the probability that the dealer was using the Fair coin at the 5th flip *given* that he generated a sequence of flips **HHTHTHHHTT**?

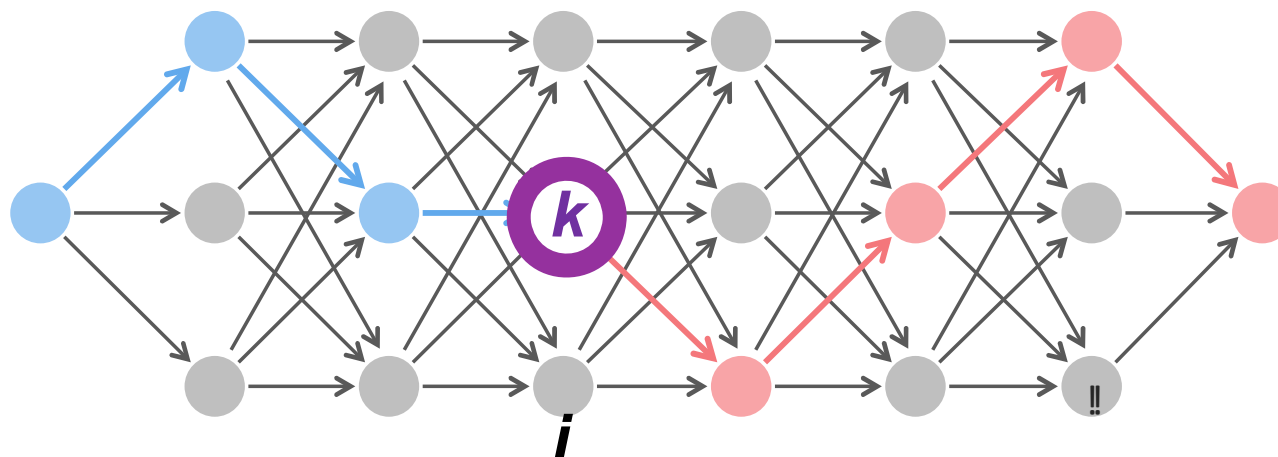
Compare with:

$\Pr(\pi_i = k, x)$: the *unconditional* probability that a hidden path will pass through state k at time i *and* emit x .

What is the probability that the dealer will generate a sequence of flips **HHTHTHHHTT**?
while using the Fair coin at the 5th flip?

What Is $\Pr(\pi_i = k | x)$?

$\Pr(\pi_i = k | x)$: the *conditional* probability that the HMM was in state k at time i given that it emitted string x .



$\Pr(\pi_i = k | x)$: the fraction of the product weight of paths visiting k over the weight of all paths:

$$\Pr(\pi_i = k | x) = \Pr(\pi_i = k, x) / \Pr(x)$$

$$= \sum_{\text{all paths } \pi \text{ with } \pi_i = k} \Pr(x, \pi) / \sum_{\text{all paths } \pi} \Pr(x, \pi)$$

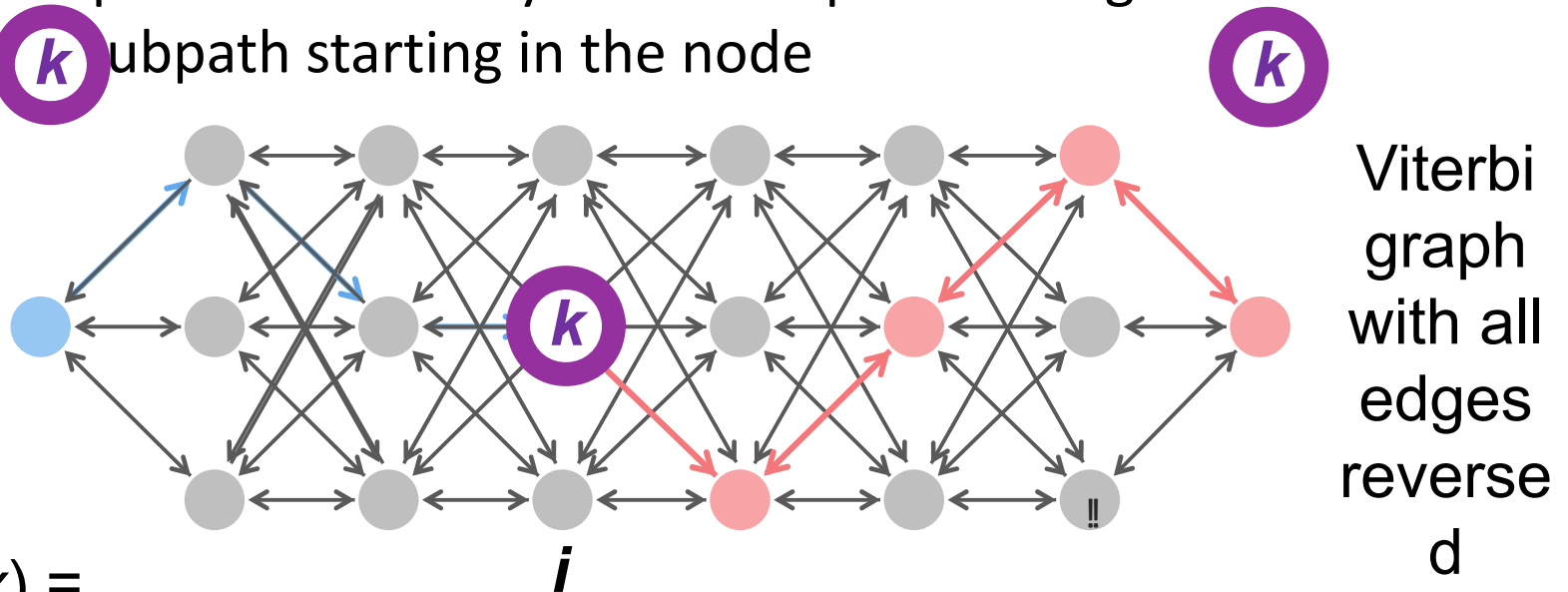
Soft Decoding Problem

Soft Decoding Problem: *Find the probability that an HMM was in a particular state at a particular moment, given its output.*

- **Input:** A string $x = x_1 \dots x_n$ emitted by an HMM (Σ , States, Transition, Emission).
- **Output:** The conditional probability $\Pr(\pi_i = k | x)$ that the HMM was in state k at step i , given x .

Computing $\Pr(\pi_i=k, x)$

- $\Pr(\pi_i=k, x)$ = total product weights of all paths through the Viterbi graph for x that pass through the node (k, i) .
- Each such path is formed by a blue subpath ending in the node and a red subpath starting in the node



$$\Pr(\pi_i=k, x) =$$

\sum product weights of all blue paths * \sum product weights of all red paths

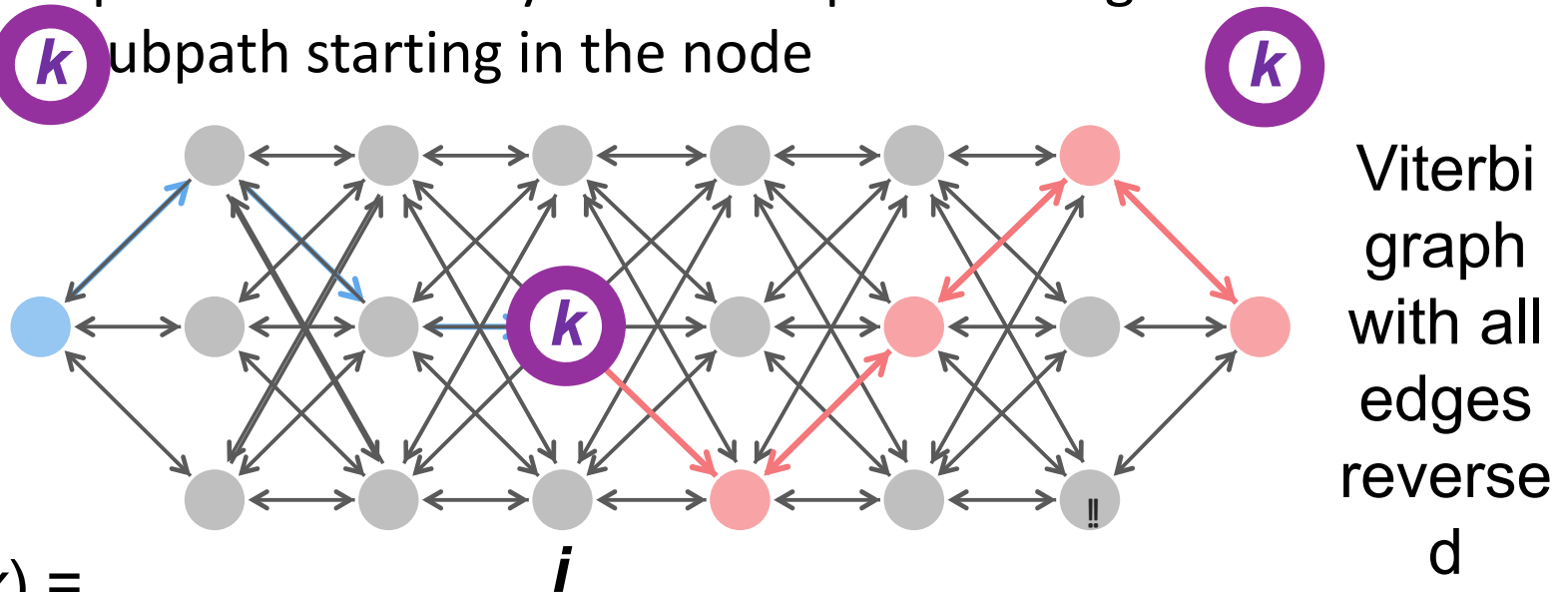
$$\boxed{\text{forward}_{k,i}}$$

*

$$\boxed{???$$

Computing $\Pr(\pi_i=k, x)$

- $\Pr(\pi_i=k, x)$ = total product weights of all paths through the Viterbi graph for x that pass through the node (k, i) .
- Each such path is formed by a blue subpath ending in the node and a red subpath starting in the node



$$\Pr(\pi_i=k, x) =$$

\sum product weights of all blue paths * \sum product weights of all red paths

$$\boxed{\text{forward}_{k,i}}$$

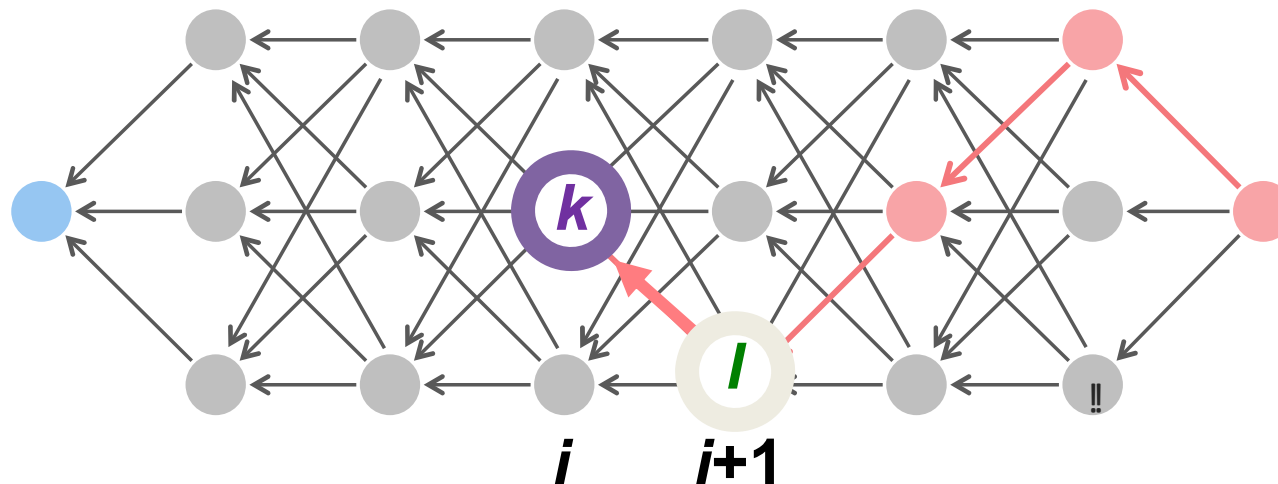
*

$$\boxed{\text{backward}_{k,i}}$$

Forward-Backward Algorithm

- Since the reverse edge connecting node $(l, i+1)$ to node (k, i) in the reversed graph has weight $weight(k, l, i)$:

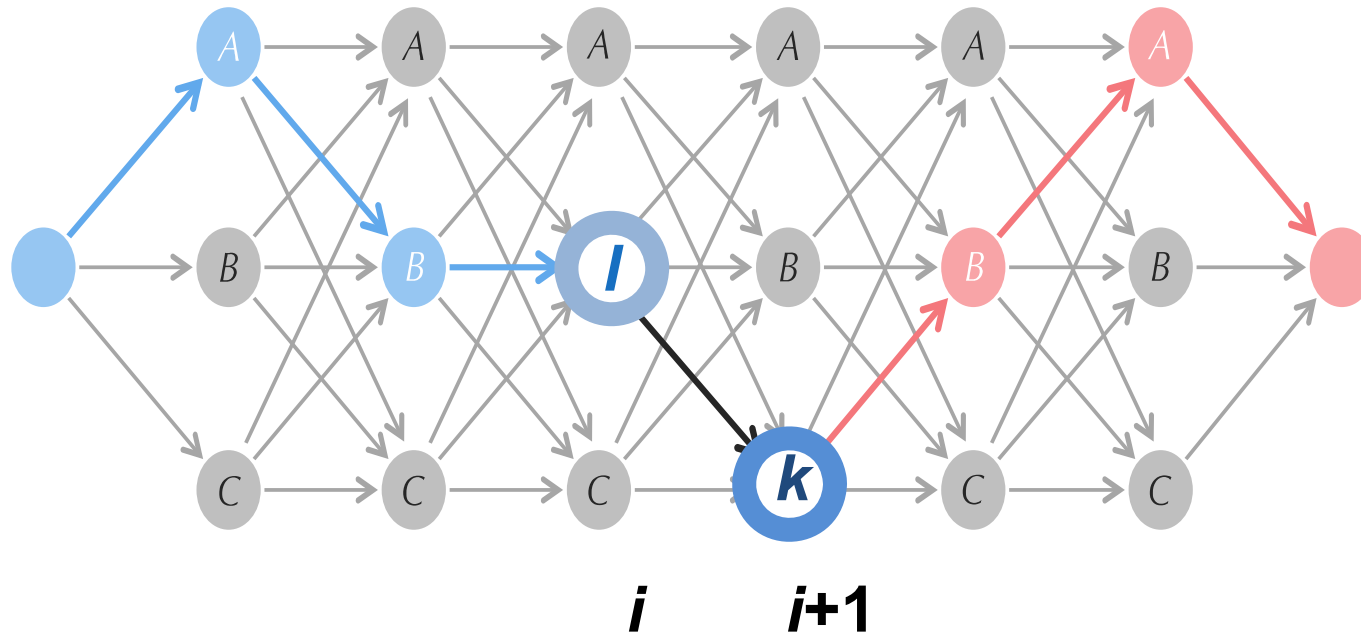
$$backward_{k,i} = \sum_{\text{all states } l} backward_{l,i+1} \cdot weight(k, l, i)$$



- Combining the forward-backward algorithm with the solution to the Outcome Likelihood Problem yields

$$\Pr(\pi_i = k | x) = \Pr(\pi_i = k, x) / \Pr(x) = \frac{Forward_{k,i} * backward_{k,i}}{forward(sink)}$$

The Conditional Probability $\Pr(\pi_i=l, \pi_{i+1}=k | x)$ that the HMM Passes Through an **Edge** in the Viterbi Graph



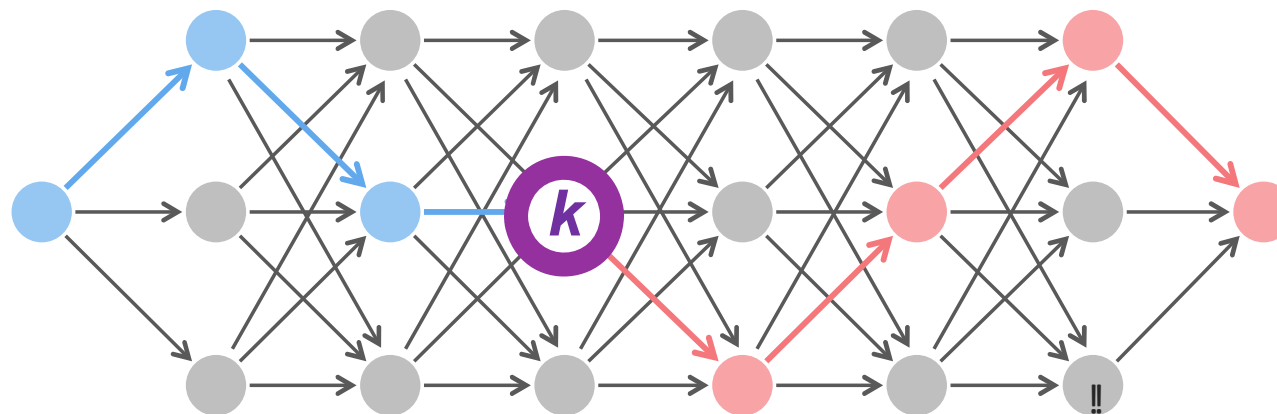
\sum weights of blue paths * weight of black edge * \sum weights of red paths

$$\Pr(\pi_i=l, \pi_{i+1}=k | x) = \frac{\text{forward}_{l,i} * \text{weight}(l, k, i) * \text{backward}_{k,i+1}}{\text{forward}(\text{sink})}$$

Node Responsibility Matrix

- Node responsibility matrix $\Pi^* = (\Pi^*_{k,i})$:

$$\Pi^*_{k,i} = \Pr(\pi_i = k | \mathbf{x})$$



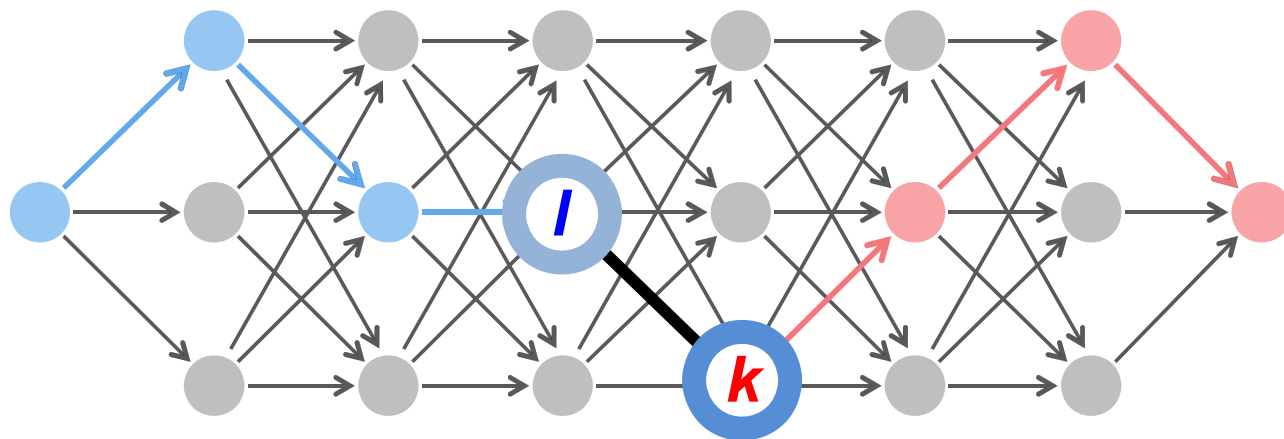
Node responsibility matrix for the crooked casino

	T	H	T	H	H	H	T	H	T	T	H
F	0.636	0.593	0.600	0.533	0.515	0.544	0.627	0.633	0.692	0.686	0.609
B	0.364	0.407	0.400	0.467	0.485	0.456	0.373	0.367	0.308	0.314	0.391

Edge Responsibility Matrix

- Edge responsibility matrix

$$\Pi^{**}_{l,k,i} = \Pr(\pi_i = l, \pi_{i+1} = k | x)$$



Edge responsibility matrix for the crooked

Every element of Π^* (Π^{**}) corresponds to a **node** (**edge**) in the Viterbi graph. Define $\Pi = (\Pi^*, \Pi^{**})$

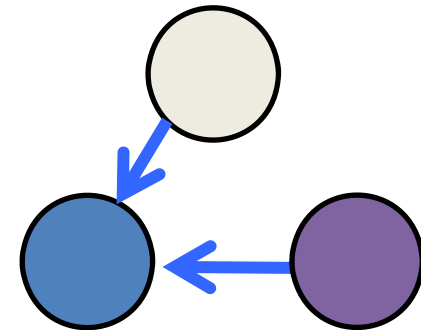
	0.574	0.426	0.418	0.351	0.322	0.282	0.265	0.222
A								
B								

Baum-Welch Learning

Baum-Welch learning alternates between two steps:

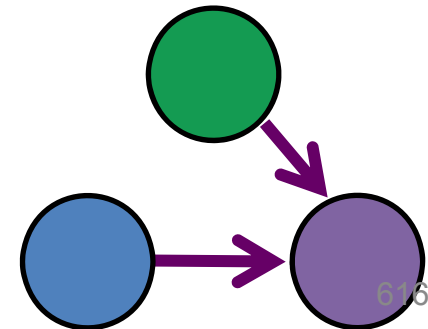
- Re-estimating the responsibility profile Π given the current HMM parameters (the **E-step**):

(emitted string, ?, *Parameters*) $\rightarrow \Pi$



- Re-estimating the HMM parameters given the current responsibility profile (the **M-step**):

(emitted string, Π , ?) \rightarrow *Parameters*



Using a Responsibility Matrix to Compute *Parameters*

- We have defined a transformation

$$(x, \pi, ?) \rightarrow \textit{Parameters}$$

that uses estimators $T_{l,k}$ and $E_k(b)$ based on a path π .

- We now want to define a transformation:

$$(x, \Pi, ?) \rightarrow \textit{Parameters}$$

but the **path is unknown**.

Idea: Use *expected* values $T_{l,k}$ and $E_k(b)$ over *all* possible paths.

Redefining Estimators for *Parameters* (for a known path π)

- $T_{l,k}$: #transitions from state l to state k in path π

$$T_{l,k}^i = \begin{cases} 1 & \text{if } \pi_i = l \text{ and } \pi_{i+1} = k \\ 0 & \text{otherwise} \end{cases}$$

Rewriting
estimators: $T_{l,k} = \sum_{i=1, n-1} T_{l,k}^i$

HHHTHHHTHHHTTTTHTTTTH

BFF
BFF
FFB
BFF
BFF
BBB
BFF

$T_{B,F}^i = 100100001001000000100$

$T_{B,F} = 5$

Redefining the Estimators $T_{l,k}^i$ and $E_k^i(b)$ When the Path is Unknown

- $T_{l,k}$: #transitions from state l to state k in path π
- $E_k(b)$: #times b is emitted when the path π is in state k
- We now define

$$T_{l,k}^i = \begin{cases} 1 & \text{if } \pi_i = l \text{ and } \pi_{i+1} = k \\ 0 & \text{otherwise} \end{cases}$$

$$T_{l,k}^i = \Pr(\pi_i = l, \pi_{i+1} = k | x)$$

$$T_{l,k}^i = \Pi_{l,k,i}^{**}$$

$$E_k^i(b) = \begin{cases} 1 & \text{if } \pi_i = k \text{ and } x_i = b \\ 0 & \text{otherwise} \end{cases}$$

$$E_k^i(b) = \begin{cases} \Pr(\pi_i = k | x) & \text{if } x_i = b \\ 0 & \text{otherwise} \end{cases}$$

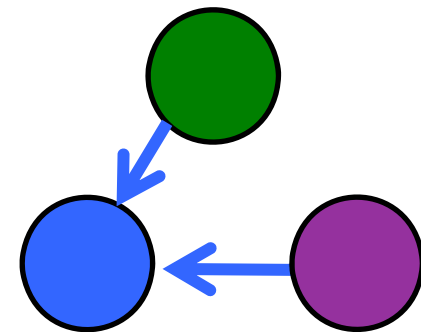
$$E_k^i(b) = \begin{cases} \Pi_{k,i}^* & \text{if } x_i = b \\ 0 & \text{otherwise} \end{cases}$$

Baum-Welch Learning

Baum-Welch learning alternates between two steps:

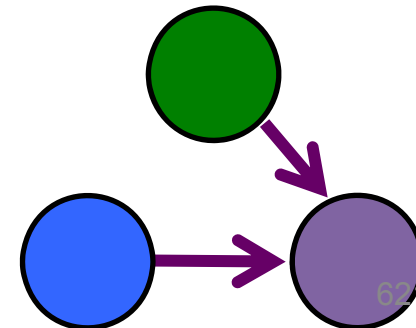
- Re-estimating the responsibility profile Π given the current HMM parameters (the **E-step**):

(emitted string, ?, *Parameters*) $\rightarrow \Pi$



- Re-estimating the HMM parameters given the current responsibility profile (the **M-step**):

(emitted string, Π , ?) \rightarrow *Parameters*



Stopping Rules for the Baum-Welch Learning

- Compute the probability that the HMM emits the string x under current *Parameters*:

$$\Pr(\textit{emitted string} \mid \textit{Parameters})$$

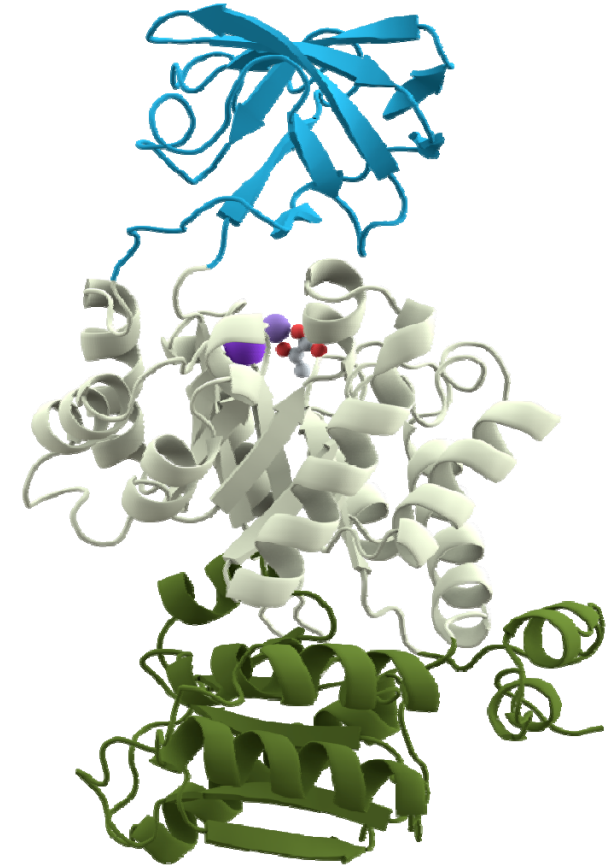
- Compare with the probability for previous values of *Parameters* and stop if the difference is small.
- Stop after a certain number of iterations.

Nature is a Tinkerer and Not an Inventor

Protein domain: a conserved part of a protein that often can function independently.

Nature uses domains as building blocks, shuffling them to create **multi-domain proteins**.

Goal: classify domains into families even though sequence similarities between domains from the same family can be low.



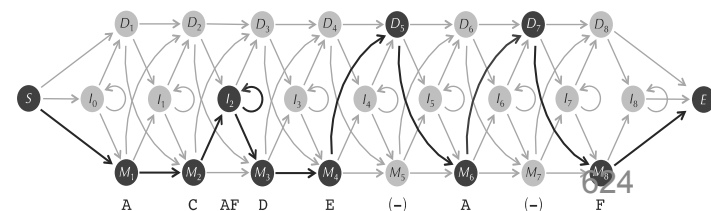
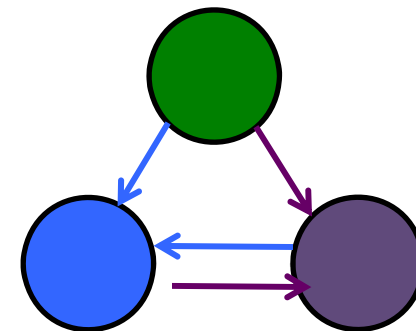
A multi-domain protein

Searching for Protein Domains with Profile HMMs

1. Use alignments to break proteins into domains.
2. Construct alignment of domains from a given family (starting from highly similar domains whose attribution to a family is non-controversial).
3. For each family, construct a profile HMM and estimate its parameters.
4. Align the new sequence against each such HMM to find the best fitting HMM.

ABCDEFGHKLMP
 ERGHKLNPA^BTD
 KLSNPACDEFTH

ABCD	KLMNP	EFGH
ABTD	KL-NP	ERGH
AC-D	KLSNP	EFTH



Pfam: Profile HMM Database

Each domain family in Pfam has:

- **Seed alignment:** Initial multiple alignment of domains in this family.
- **HMM:** Built from seed alignment for new searches.
- **Full alignment:** Enlarged multiple alignment generated by aligning new domains against the seed HMM.

Final Challenge: Using the Pfam HMM for gp120 (constructed from a seed alignment of just 24 gp120 HIV proteins), construct alignments of all known gp120 proteins and identify the “most diverged” gp120 sequence.

Related work

- Mixture Hidden Markov Models (MHMMs) can be used to cluster data into homogenous subsets on the basis of latent characteristics