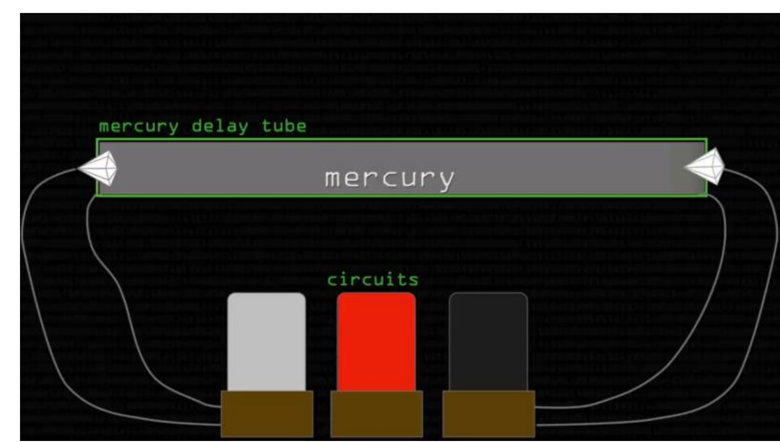
SECTION 4

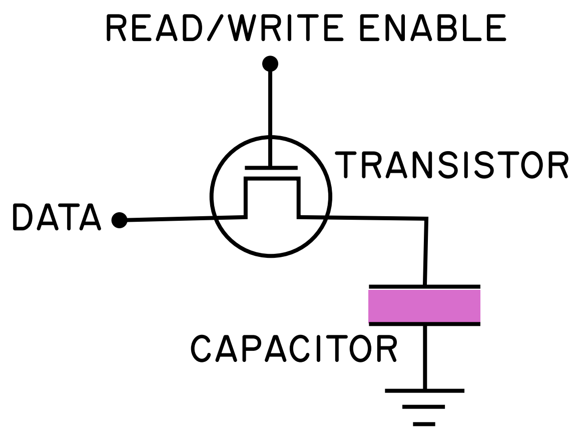
Data Structures

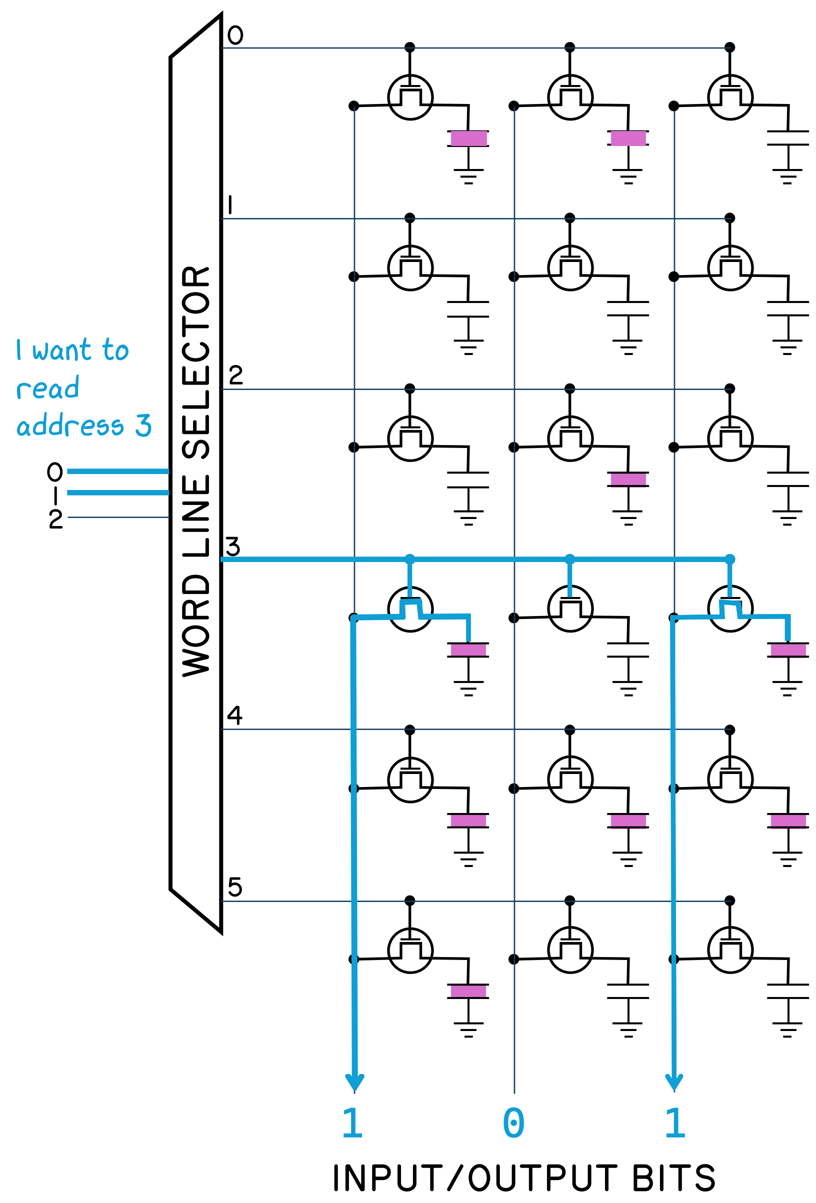
A mercury delay tube from EDSAC 1, made at the University of Cambridge Mathematical Laboratory, 1946-1958.

EDSAC (Electronic Delay Storage Automatic Calculator) was considered to be the world's first fully operational and practical stored program computer. It ran its first program on 6 May 1949.



Transistor memory (DRAM)





MEMORY DUMP

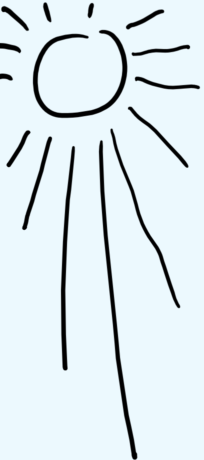
110	000	010	101	111	100

```
djw@mink:$ hexdump -C -n 256 slides08.pptx

00000000  50 4b 03 04 14 00 06 00  08 00 00 00 21 00 73 ca  |PK..........!.s.|
00000010  a8 73 b0 02 00 00 d8 0d  00 00 14 00 00 00 70 70  |.s............pp|
00000020  74 2f 70 72 65 73 65 6e  74 61 74 69 6f 6e 2e 78  |t/presentation.x|
00000030  6d 6c ec 97 dd 6e da 30  14 c7 ef 27 ed 1d 22 df  |ml...n.0...'..".|
00000040  4e 34 df 1f 20 42 55 d6  65 9a c4 24 54 da 07 70  |N4.. BU.e..$T..p|
00000050  13 53 a2 3a 76 64 1b 0a  9d f6 ee 3b 4e 0c 09 50  |.S.:vd.....;N..P|
00000060  4d 7d 80 5c e1 f8 7f 7c  ce f1 8f 13 c7 67 7a bb  |M}.\...|.....gz.|
00000070  af a8 b5 23 42 96 9c a5  c8 bd 71 90 45 58 ce 8b  |...#B.....q.EX..|
00000080  92 bd a4 e8 e9 31 1b 25  c8 92 0a b3 02 53 ce 48  |.....1.%.....S.H|
00000090  8a 0e 44 a2 db d9 d7 2f  d3 7a 52 0b 22 09 53 58  |..D..../.zR.".SX|
000000a0  c1 52 0b dc 30 39 c1 29  da 28 55 4f 6c 5b e6 1b  |.R..09.).(UOl[..|
000000b0  52 61 79 c3 6b c2 40 5b  73 51 61 05 8f e2 c5 2e  |Ray.k.@[sQa.....|
000000c0  04 7e 03 f7 15 b5 3d c7  89 ec 0a 97 0c 99 f5 e2  |.~....=.........|
000000d0  33 eb f9 7a 5d e6 e4 9e  e7 db 0a c2 b7 4e 04 a1  |3..z]........N..|
000000e0  4d 1e 72 53 d6 f2 e8 ad  fe 8c b7 fe 2e ce 53 92  |M.rS..........S.|
000000f0  78 47 56 db 67 49 54 c6  99 92 40 07 cd 60 db 92  |xGV.gIT...@..`..|
```

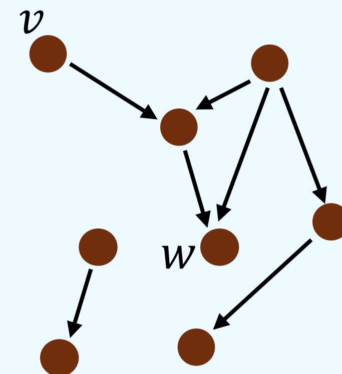
The Cathar Heresy

circa 1100–1400

The Cathars were a religious sect who believed the world consists of two realms, one spiritual and one physical, each with its own deity. And that life is a struggle of the light to escape base matter.

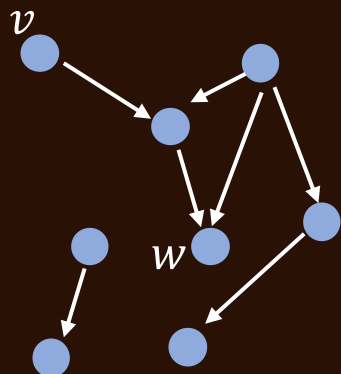


A graph $g = (V, E)$ is an ordered pair,
consisting of a vertex set V and an edge set E that is a relation on V.



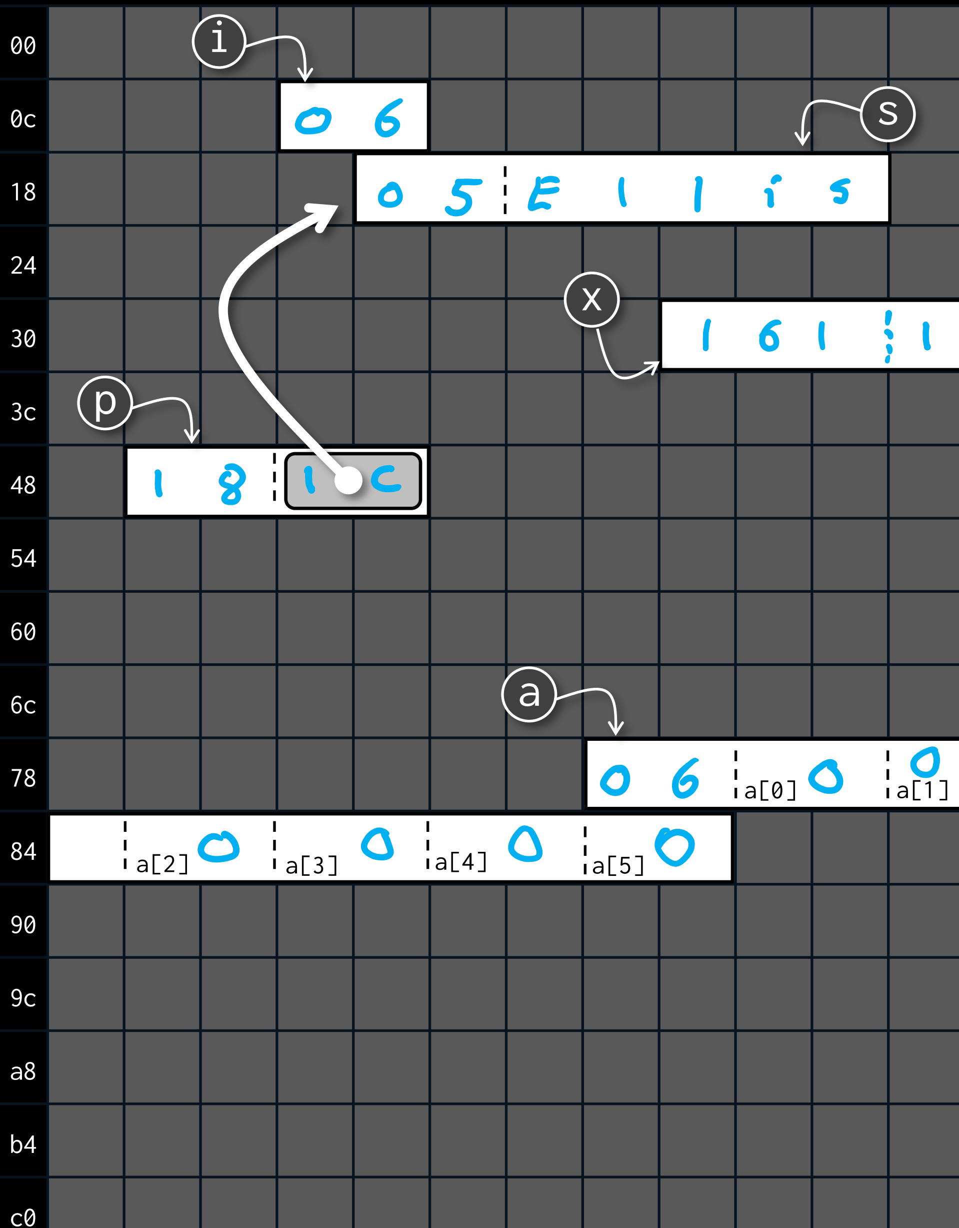
We say $w \in V$ is *reachable* from $v \in V$ if $(v, w) \in E^{\circ+}$.

DISCRETE MATHS EXERCISE 6.2.4



- How do we store a graph in memory?
- What's an algorithm for computing reachability?
- How efficient can we make it?

```
djw@mink:$ hexdump -C -n 256 slides08.pptx

00000000  50 4b 03 04 14 00 06 00  08 00 00 00 21 00 73 ca  |PK..........!.s.|
00000010  a8 73 b0 02 00 00 d8 0d  00 00 14 00 00 00 70 70  |.s............pp|
00000020  74 2f 70 72 65 73 65 6e  74 61 74 69 6f 6e 2e 78  |t/presentation.x|
00000030  6d 6c ec 97 dd 6e da 30  14 c7 ef 27 ed 1d 22 df  |ml...n.0...'..".|
00000040  4e 34 df 1f 20 42 55 d6  65 9a c4 24 54 da 07 70  |N4.. BU.e..$T..p|
00000050  13 53 a2 3a 76 64 1b 0a  9d f6 ee 3b 4e 0c 09 50  |.S.:vd.....;N..P|
00000060  4d 7d 80 5c e1 f8 7f 7c  ce f1 8f 13 c7 67 7a bb  |M}.\...|.....gz.|
00000070  af a8 b5 23 42 96 9c a5  c8 bd 71 90 45 58 ce 8b  |...#B.....q.EX..|
00000080  92 bd a4 e8 e9 31 1b 25  c8 92 0a b3 02 53 ce 48  |.....1.%.....S.H|
00000090  8a 0e 44 a2 db d9 d7 2f  d3 7a 52 0b 22 09 53 58  |..D..../.zR.".SX|
000000a0  c1 52 0b dc 30 39 c1 29  da 28 55 4f 6c 5b e6 1b  |.R..09.).(UOl[..|
000000b0  52 61 79 c3 6b c2 40 5b  73 51 61 05 8f e2 c5 2e  |Ray.k.@[sQa.....|
000000c0  04 7e 03 f7 15 b5 3d c7  89 ec 0a 97 0c 99 f5 e2  |.~....=.........|
000000d0  33 eb f9 7a 5d e6 e4 9e  e7 db 0a c2 b7 4e 04 a1  |3..z]........N..|
000000e0  4d 1e 72 53 d6 f2 e8 ad  fe 8c b7 fe 2e ce 53 92  |M.rS..........S.|
000000f0  78 47 56 db 67 49 54 c6  99 92 40 07 cd 60 db 92  |xGV.gIT...@..`..|
```



4.1.1 Machine data types

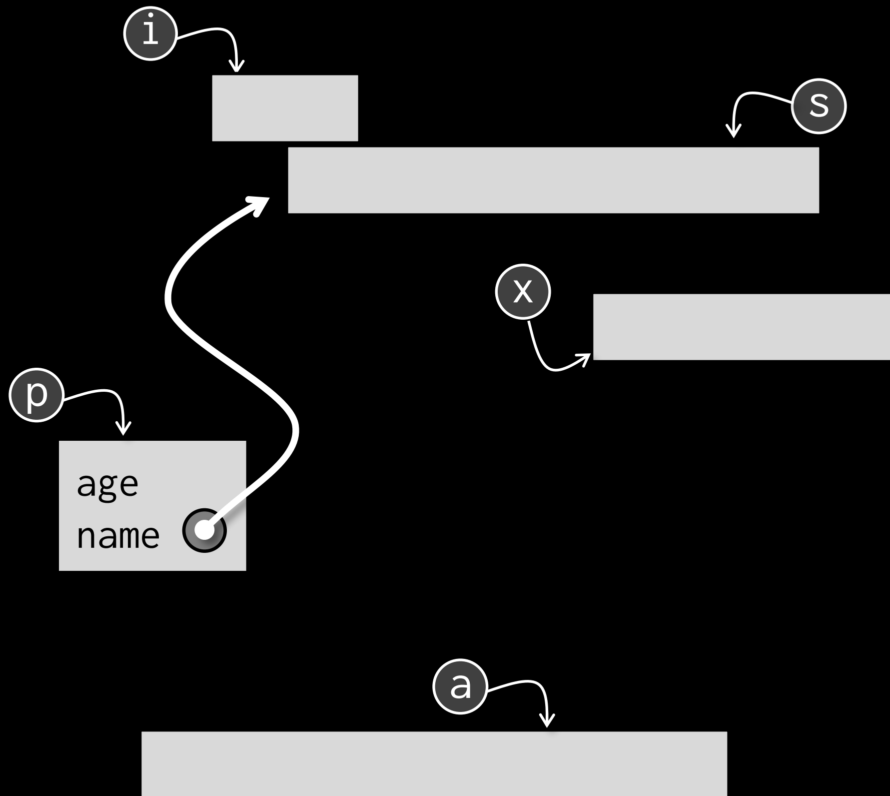
```
int i = 6

float x = 1.618

str s = “Ellis”

struct Person {
    int age
    str *name
}
Person p = new Person(age=18, name=&s)

array a = np.zeros(6, dtype=np.int16)
```

stored as digits and power of 10, e.g. 0.1618×10^1

str* means "store a pointer to a string"

&s means "the address of s"

4.1.1 Machine data types

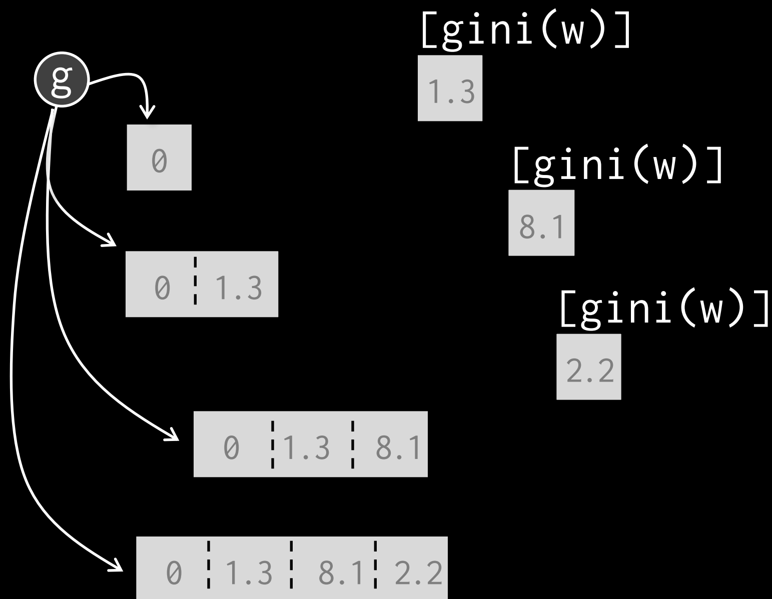


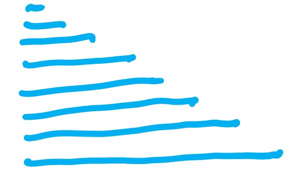

```
int i = 6

float x = 1.618

str s = “Ellis”

struct Person {
   int age
   str *name
}
Person p = new Person(age=18, name=&s)

array a = np.zeros(6, dtype=np.int16)
```

We’ll work with an idealized machine. We won’t worry about exactly how many bytes each of these takes to store. We’ll assume numbers don’t overflow, and there’s space for all our arrays.

4.1.2 Vectors / arrays

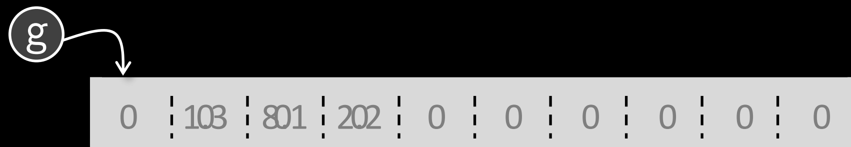
As a coder, it's your responsibility to be aware of the time and memory costs of your code.



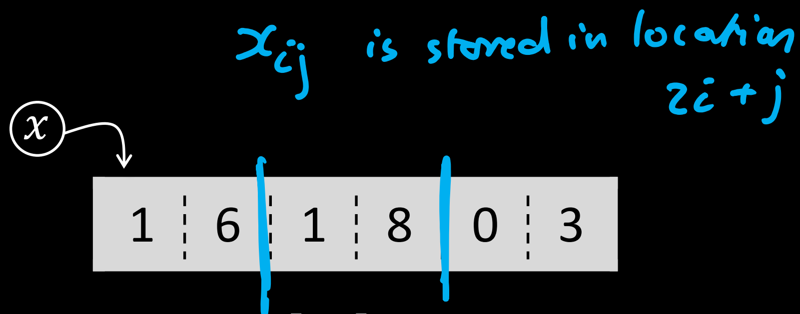


```
w = np.ones(N)
g = np.array([0])
for t in range(T):
    w = sim_step(w)
    g = np.concatenate(g, [gini(w)])
```

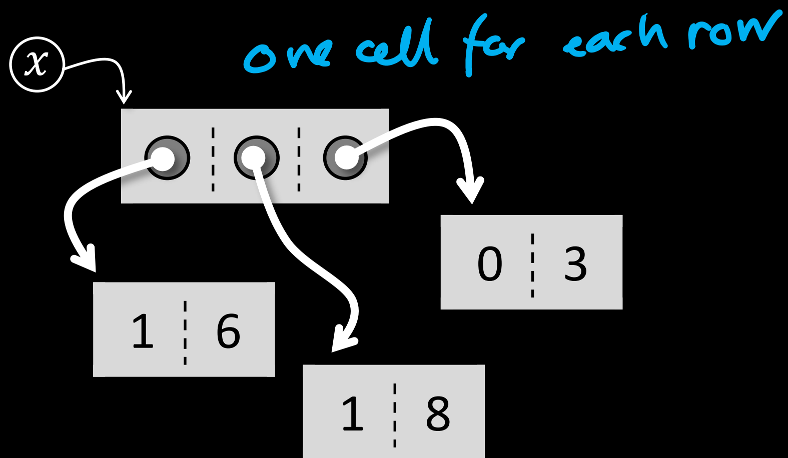
This has $\Theta(T^2)$ running time, since it copies an array of size 1, then 2, then 3, then 4, ..., then T−1.



```
w = np.ones(N)
g = np.zeros(T))
for t in range(T):
    w = sim_step(w)
    g[t+1] = gini(w)
```



To fetch $x[i, j]$ the cost is:
1 multiplication and 1 addition to compute $2i + j$, then 1 lookup



To fetch $x[i, j]$ the cost is:
2 lookups

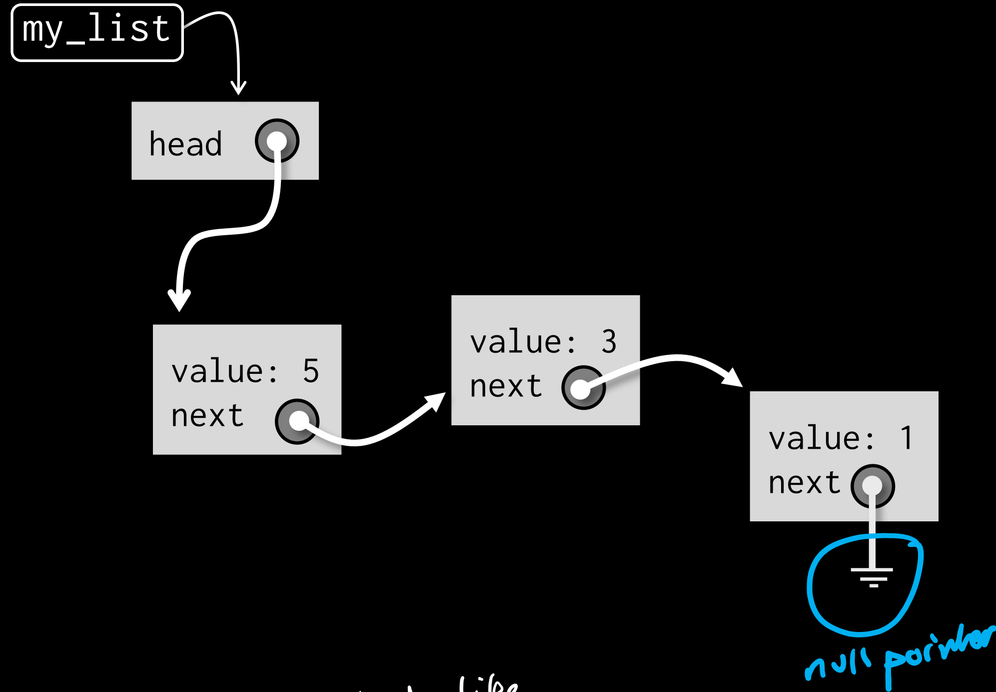
4.1.2 Vectors / arrays

How might we represent a 2-dimensional array?

$$x = \begin{bmatrix} 1 & 6 \\ 1 & 8 \\ 0 & 3 \end{bmatrix}$$

x_{ij} x_{RC}
row column.

4.1.3 Lists



this is where methods like is_empty() belong.

```
class List<T>:
    ListItem<T>? head

class ListItem<T>:
    T value
    ListItem<T>? next
```

?: next is either a pointer to a ListItem, or a null pointer

4.2.2 Lists

This notation means "generic type". E.g. List<int> is a list of integers, List<str *> is a list of pointers to strings.

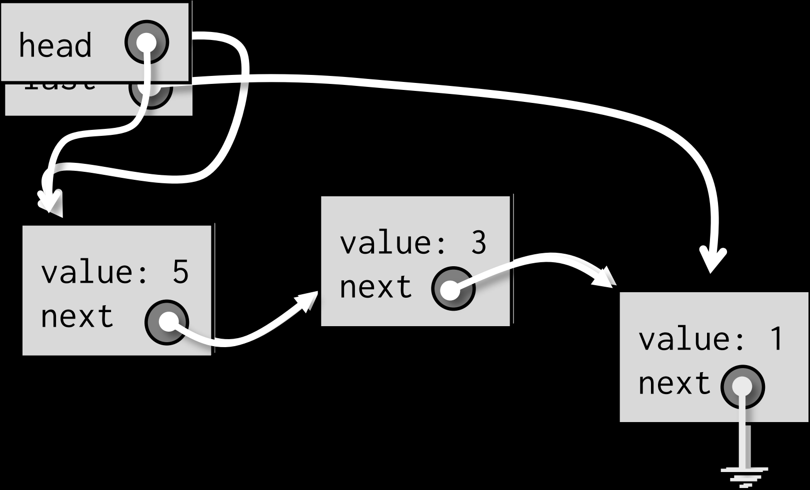
```
interface List<T>:

    # Returns true iff list is empty.
    boolean is_empty()

    # If list is non-empty, returns first item
    # (without removing it). If list is empty,
    # behaviour is undefined.
    T head()

    # Returns all items except the first,
    # in order.
    List<T> tail()

    # Adds item x to the beginning of the list.
    prepend(T x)
```

Out of these operations above, we can implement iterating through the list by using head, is_empty, and tail. So it's cleaner (from a theory POV) not to include list iteration. Of course, in practice, the users of our List class would probably find it helpful.

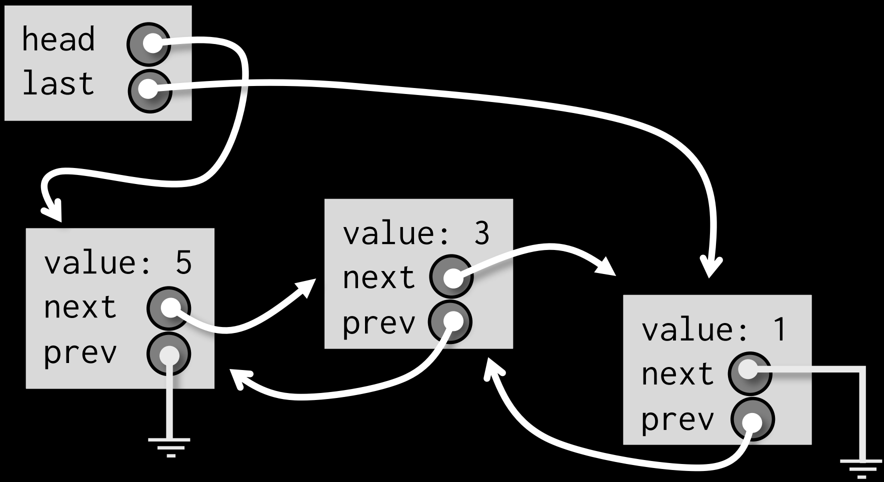
It clarifies our thinking to minimally *specify* the behaviour we need from a data structure, independent of its implementation.

If we have enough discrete maths to formally specify not only the type signatures but also the behaviour, we call it an *abstract data type*.

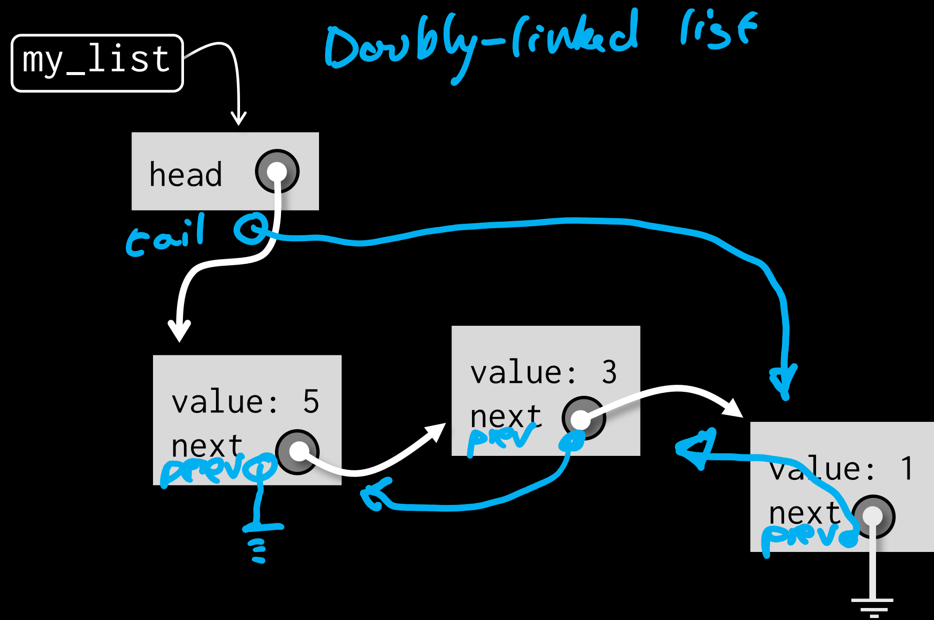
SINGLY-LINKED LIST



DOUBLY-LINKED LIST



4.1.3 Lists




```
class List<T>:
    ListItem<T>? head

class ListItem<T>:
    T value
    ListItem<T>? next
```

4.2.2 Lists

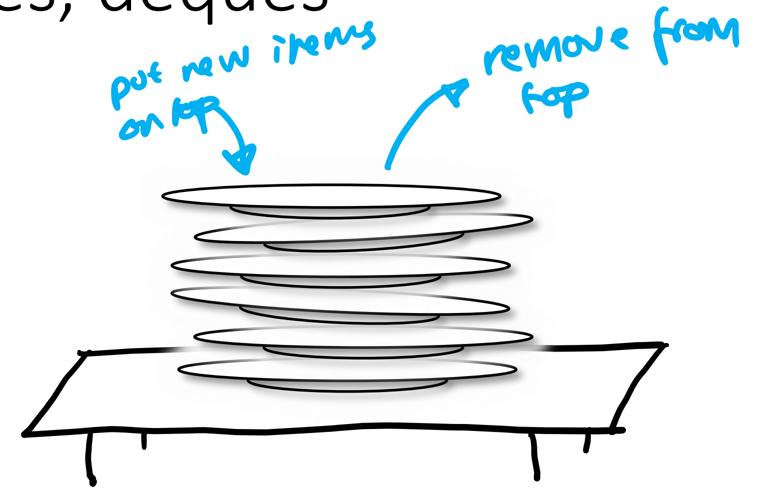
```
interface List<T>:

    # Returns true iff list is empty.
    boolean is_empty()

    # If list is non-empty, returns first item
    # (without removing it). If list is empty,
    # behaviour is undefined.
    T head()

    # Returns all items except the first,
    # in order.
    List<T> tail()

    # Adds item x to the beginning of the list.
    prepend(T x)
```

append(T x)   delete_last()

element_at(int i)

4.2.1, 4.2.3 Stacks, queues, deques

```
interface Stack<T>:

    # reading operations
    boolean is_empty()
    T top()

    # mutating operations
    push(T x)   # add x to top
    T pop()     # remove from top
```

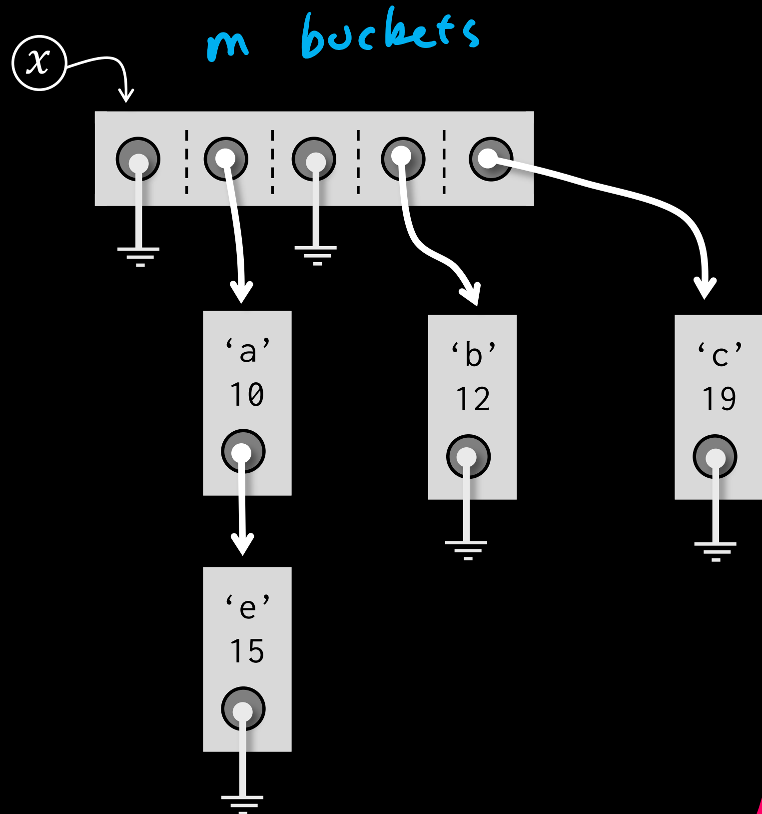



```
interface Queue<T>:

    # reading operations
    boolean is_empty()
    T front()

    # mutating operations
    push(T x)   # add x to back
    T pop()     # remove from front
```

A *deque* lets us add and remove at both ends.

These are all easy to implement. We might as well just use a doubly-linked list for all of them.

4.7 Hash tables



- Choose a number of buckets m
- Choose a *hash function* $h : \text{Keys} \to \{0, 1, \ldots, m-1\}$
- In each bucket, store a linked list of items that hash to that bucket

4.2.4 Dictionary

```
x = {'a': 10, 'c': 19, 'b': 12, 'e': 15}

# dictionaries support:
# is_empty(), contains(k), get(k),
# set(k,v), del(k)
```

e.g. we could choose m similar to Bucket Sort:

1. Fix a, e.g. $a = 2.5$
2. Let n = max # items I'll want to store in my dictionary
3. Let $m = \lceil \frac{n}{a} \rceil$

Then # items / bucket $= \frac{n}{m} = \frac{n}{\lceil n/a \rceil} \approx \frac{n}{n/a} = a$

So it doesn't take much time to scan through the linked list for a single bucket.

hash function h: Keys $\to \{0, \ldots, m-1\}$

Simple example: let keys be strings, and define $h(s)$ to compute the ascii codes for each character, add them up, and compute the answer mod m.

IMPLEMENTATION ONE

We could implement Set using buckets + linked lists, just like a dictionary. The only difference is that the list items only need to store keys, not keys+values.

IMPLEMENTATION TWO

If our language gives us a Dictionary, we can use it as a set; we just store something completely arbitrary in the value field, and make sure the rest of our code ignores the values.

```
# To store the set {'london', 'paris', 'cambridge'}
x = {'london': true,
     'paris': true,
     'cambridge': true}
```

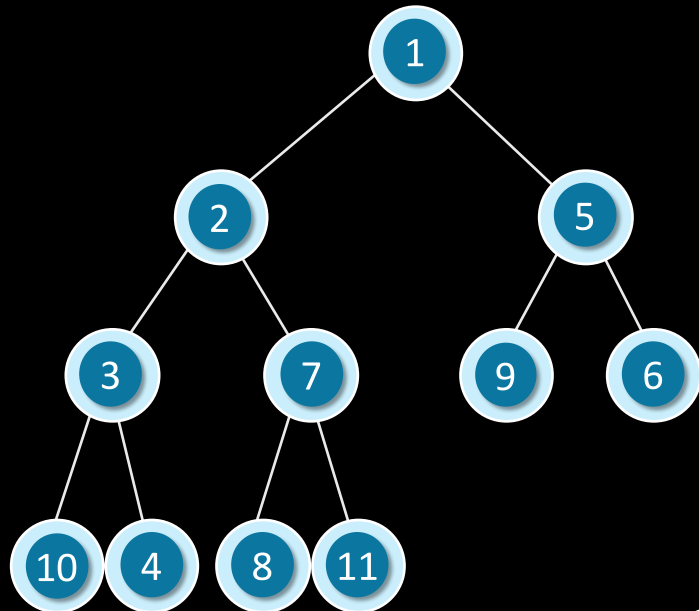
4.2.4 … and set

```
interface Set<T>:
    # reading operations
    boolean is_empty()
    boolean has(T k)
    T get_any()
    # mutating operations
    add(T k)
    del(T k)
    union_with(Set<T> s)
```

4.8.1 Binary heap

A *binary heap* is an almost-full binary tree that satisfies the heap property (everywhere in the tree, parent key $\leq$ child keys).

It supports $O(\log n)$ operations, where n is the size of the heap.



key: 6
value: ...
parent:
left-child:
right-child:

4.8 Priority queue

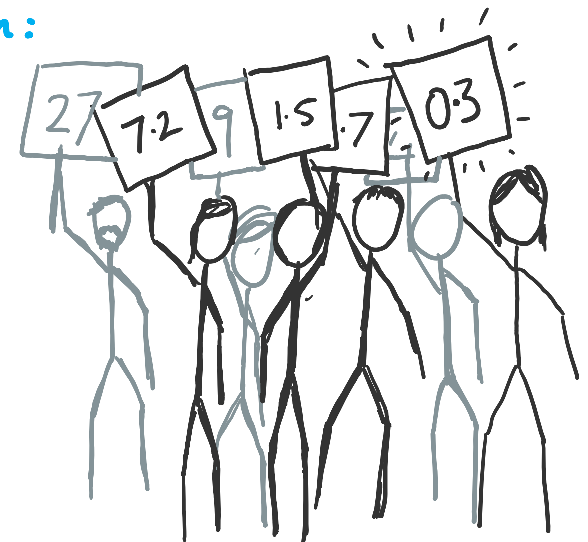
A *priority queue* holds a dynamic collection of items. Each item has a value v, and a key/priority k.

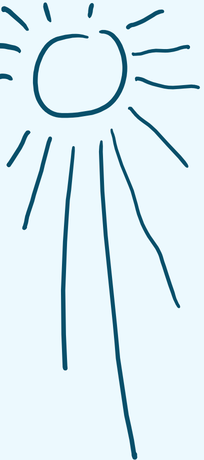
```
interface PriorityQueue<K,V>:

    boolean is_empty()

    # extract the item with the smallest key
    Pair<K,V> popmin()

    # add a new item, and set its key
    push(K key, V value)

    # for an existing item, give it a new (lower) key
    decreasekey(V value, K newkey)
```

e.g. for programming a continuous-time simulation:

keys: wake-up times for each agent in my sim.

Also used in graph algorithms.





“A graph $g = (V, E)$ is an ordered pair, consisting of a vertex set V and an edge set E which is a relation on V.”

If you look deep enough, it’s all just sets.

“A graph can be stored as <something cunning with arrays and linked lists and pointers/>.”

If you look deep enough, it’s all just bytes.

The Cathars were eradicated by the Inquisition, as their beliefs are heretical.

NOTE: HERESY SHOULD BE DEFEATED BY REASON, NOT FORCE. VIOLENCE IS AN OFFENSE AGAINST HUMAN DIGNITY, AND VIOLENCE AGAINST PEOPLE FOR THEIR BELIEFS IS A CRIME AGAINST HUMANITY.

ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

By A. M. TURING.

[Received 28 May, 1936.—Read 12 November, 1936.]

Computing is normally done by writing certain symbols on paper. We may suppose this paper is divided into squares like a child's arithmetic book. In elementary arithmetic the two-dimensional character of the paper is sometimes used. But such a use is always avoidable, and I think that it will be agreed that the two-dimensional character of paper is no essential of computation. I assume then that the computation is carried out on one-dimensional paper, *i.e.* on a tape divided into squares. I shall also suppose that the number of symbols which may be printed is finite. If we were to allow an infinity of symbols, then there would be symbols differing to an arbitrarily small extent†.

Keywords:
❋ finite state machines ❋ mathematical formalization of computation and memory ❋ countability ❋ the Halting Problem

WHAT'S NEXT

Graph algorithms

[8 lectures]

Algorithms for data structures

[8 lectures]