

Algorithms - Spring '25

Greedy:
Huffman
Stable Matching
Graphs Intro

Recap

- Oral grading (HW4)
 - Join a canvas group
 - Sign up for calendar slot
 - What to expect:
see webpage!
- Midterm exam
 - ↳ Sample posted *(after class!)*
- ✗ ◦ Last HW: group issue
 - for some → fixed, please go update today
- Regrades: Apologies for delay

Example: Huffman trees

Many of you saw this in data structures.

Why?

- cool use of trees
- non-trivial use of other data structure

Really - it's greedy!

Idea: Want to compress data, to use fewest possible bits.

Goal: Minimize Cost

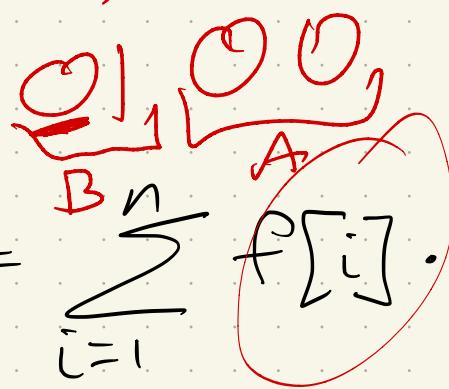
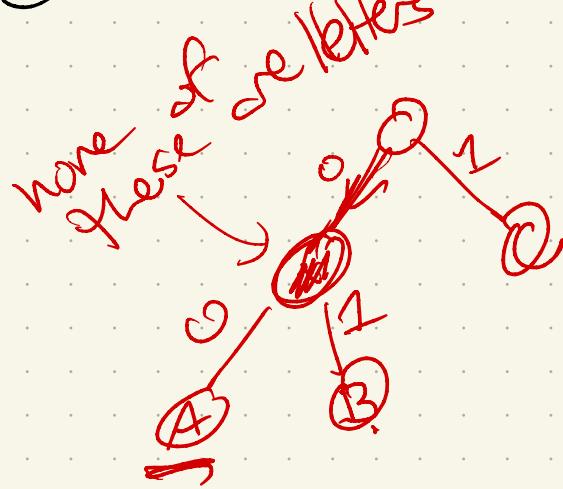
↳ here, minimize total length of encoded message :

Input: Frequency counts $f[1..n]$

one per letter

Compute: binary tree

Leaves: are letters

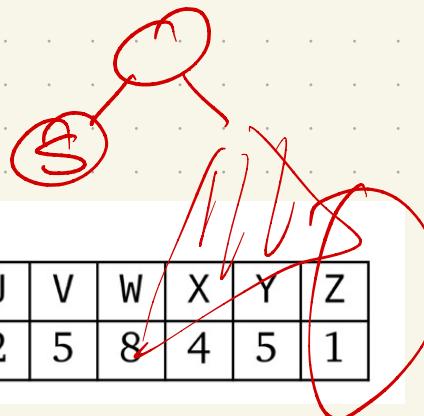


$$\text{cost}(T) = \sum_{i=1}^n f[i] \cdot \text{depth}(i)$$

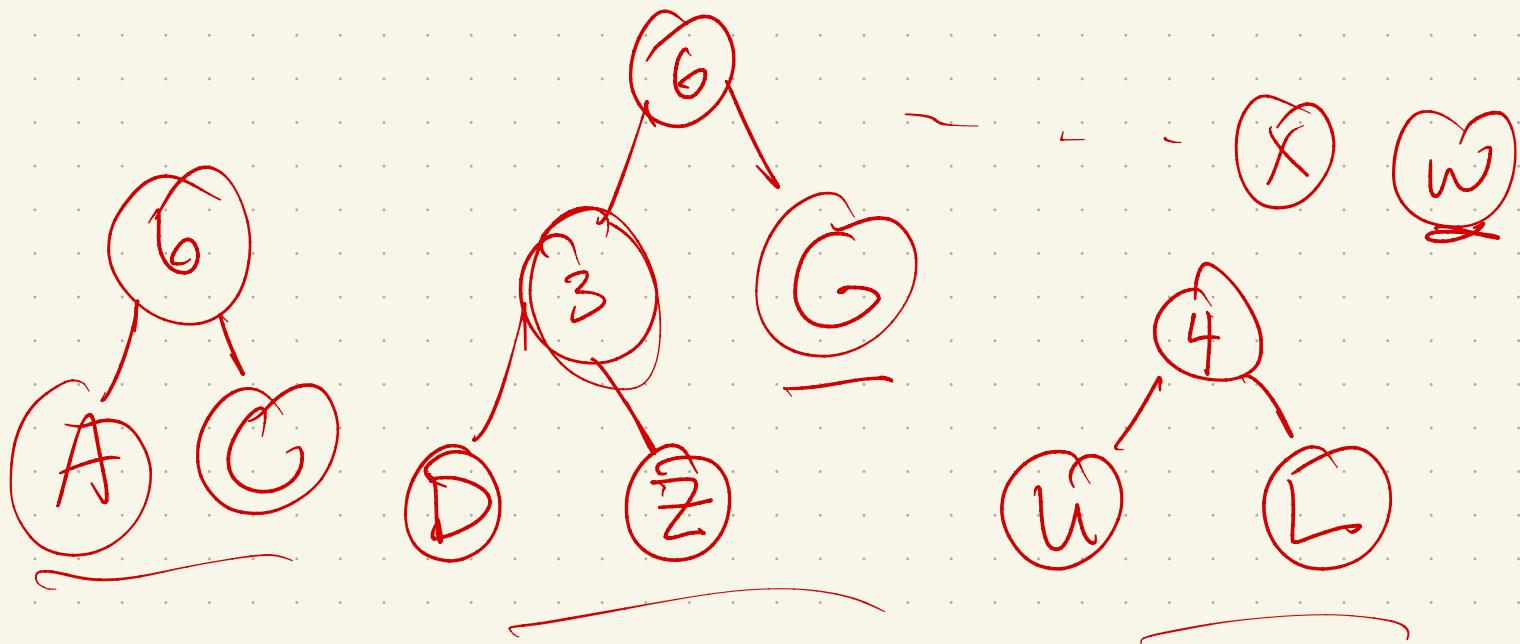
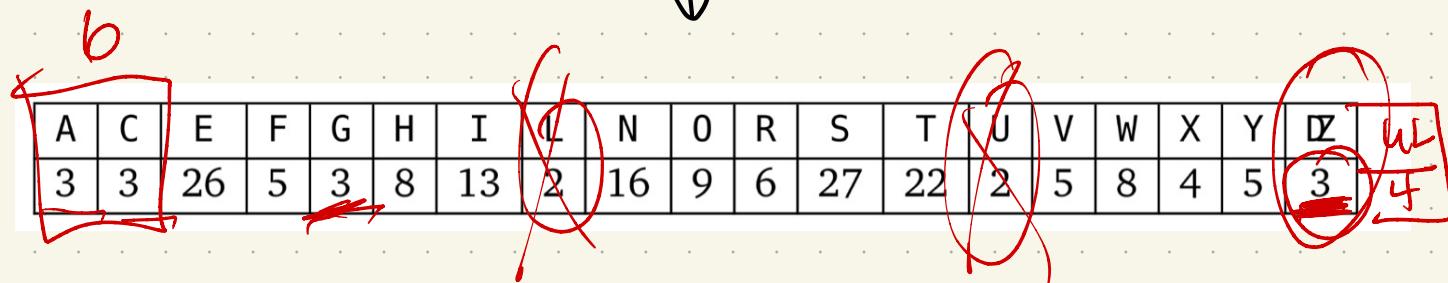
Huffman's alg

Take the two least frequent characters.

Merge them into one letter, which becomes a new "leaf":

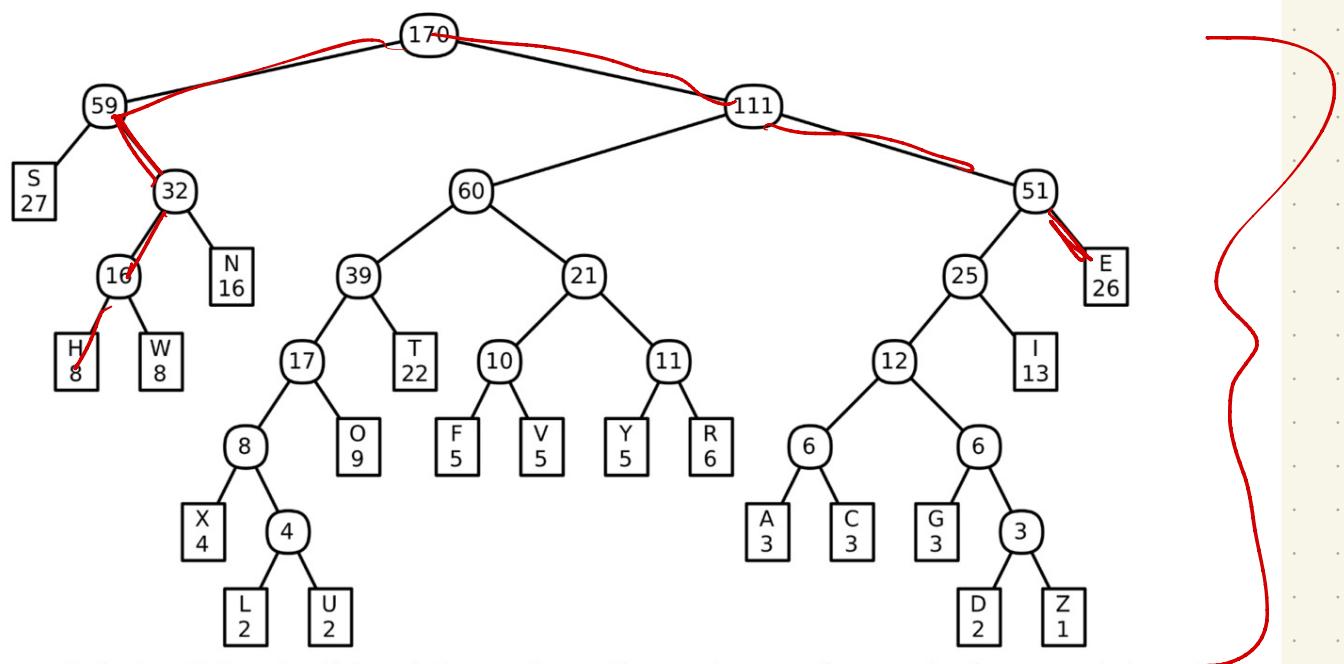


A	C	D	E	F	G	H	I	L	N	O	R	S	T	U	V	W	X	Y	Z
3	3	2	26	5	3	8	13	2	16	9	6	27	22	2	5	8	4	5	1



In the end, get a tree with letters at the leaves:

A	C	D	E	F	G	H	I	L	N	O	R	S	T	U	V	W	X	Y	Z
3	3	2	26	5	3	8	13	2	16	9	6	27	22	2	5	8	4	5	1



A Huffman code for Lee Sallows' self-descriptive sentence; the numbers are frequencies for merged characters

If we use this code, the encoded message starts like this:

1001 0100 1101 00 00 111 011 1001 111 011 110001 111 110001 10001 011 1001 110000 ...
 T H I S S E N T E N C E C O N T A ...

Another:

01001111000010100001010001
 [-] E →
 H E

How many bits?

char.	A	C	D	E	F	G	H	I	L	N	O	R	S	T	U	V	W	X	Y	Z
freq.	3	3	2	26	5	3	8	13	2	16	9	6	27	22	2	5	8	4	5	1
depth	6	6	7	3	5	6	4	4	7	3	4	4	2	4	7	5	4	6	5	7
total	18	18	14	78	25	18	32	52	14	48	36	24	54	88	14	25	32	24	25	7

Total is $\sum f[i] \cdot \text{depth}(i)$

$$= 646 \text{ bits here}$$

~~(+ tree)~~

How would ASCII do on these
170 letters

8 bits per letter

$$\hookrightarrow 170 \times 8 = 1350$$

bits

Implementation: use priority queue

BUILDHUFFMAN($f[1..n]$):

for $i \leftarrow 1$ to n
 $\rightarrow L[i] \leftarrow 0; R[i] \leftarrow 0$
INSERT($i, f[i]$)

for $i \leftarrow n$ to $2n - 1$
 $\checkmark x \leftarrow \text{EXTRACTMIN}()$
 $\checkmark y \leftarrow \text{EXTRACTMIN}()$
 $\rightarrow f[i] \leftarrow f[x] + f[y]$
 $L[i] \leftarrow x; R[i] \leftarrow y$
 $P[x] \leftarrow i; P[y] \leftarrow i$
INSERT($i, f[i]$)

$P[2n - 1] \leftarrow 0$

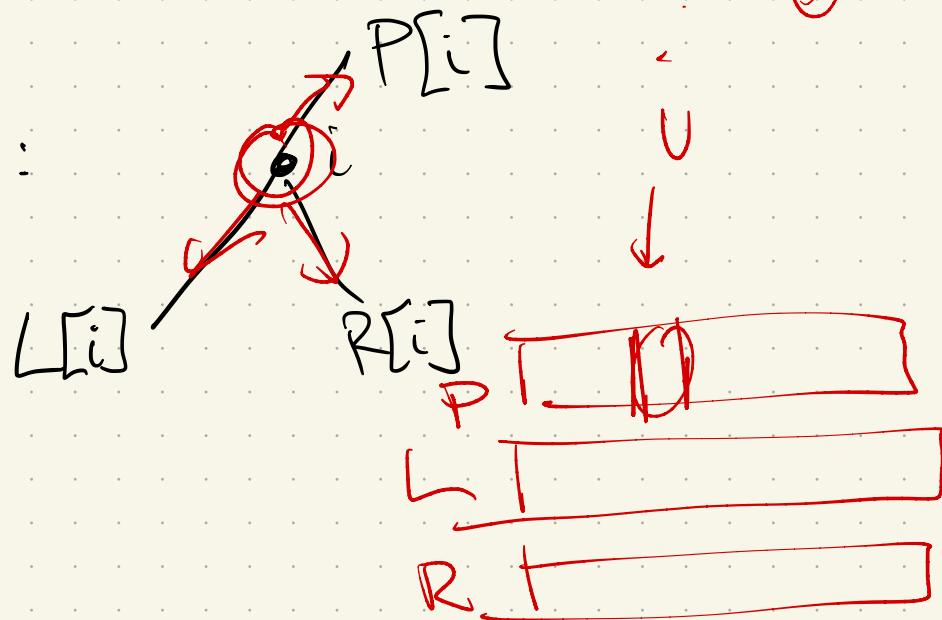
heap
 $O(\log n)$
per add/
delete



3 arrays: L , R , P

to encode the tree

node i :



So:

BANANA (EoM)

index:

1

2

3

4

EoM

letters:

Freq: f:

B

A

N

EoM

1

3

2

1

2

4

4

6

4

$$x = \frac{1}{4}$$

$$y = \frac{1}{4}$$

n leaves

n-1 internes

BUILDHUFFMAN($f[1..n]$):

```

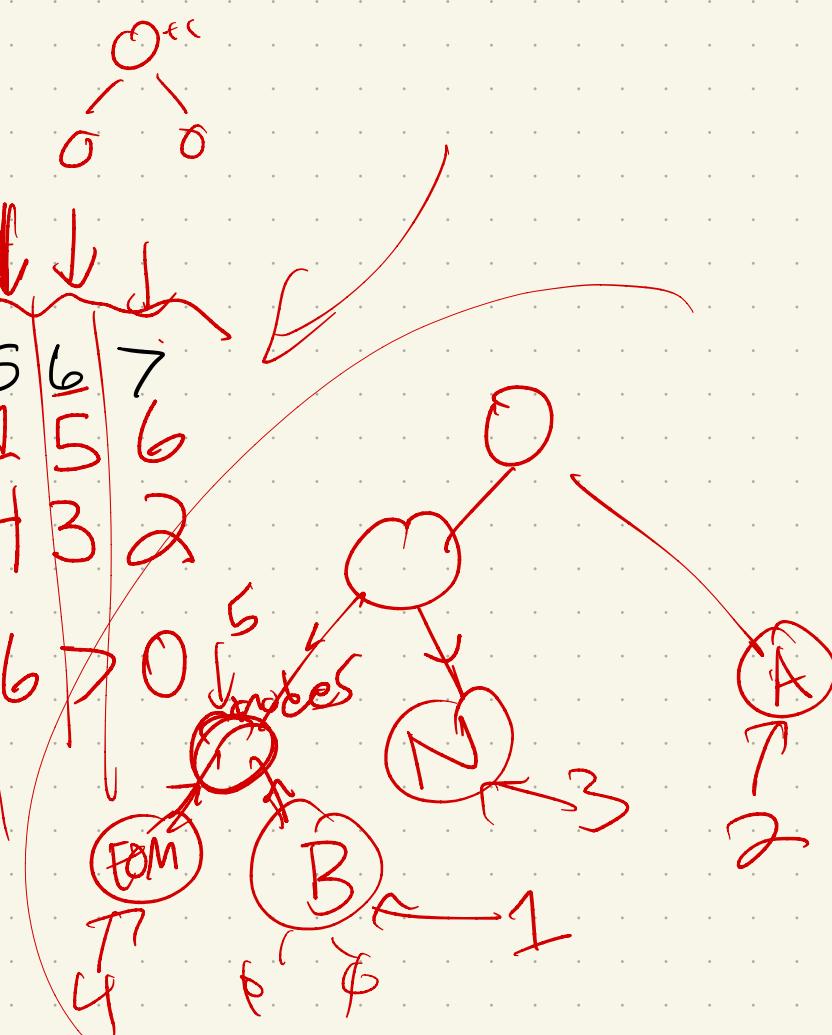
for i ← 1 to n
    L[i] ← 0; R[i] ← 0
    INSERT(i, f[i])
for i ← n to 2n - 1
    x ← EXTRACTMIN()
    y ← EXTRACTMIN()
    f[i] ← f[x] + f[y]
    L[i] ← x; R[i] ← y
    P[x] ← i; P[y] ← i
    INSERT(i, f[i])
P[2n - 1] ← 0

```

L: 1 | 2 3 4 | 5 6 7
 0 0 0 0 1 5 6
 0 0 0 0 4 3 2

R: 0 0 0 0 4 3 2

P: 5 7 6 5 6 7 0

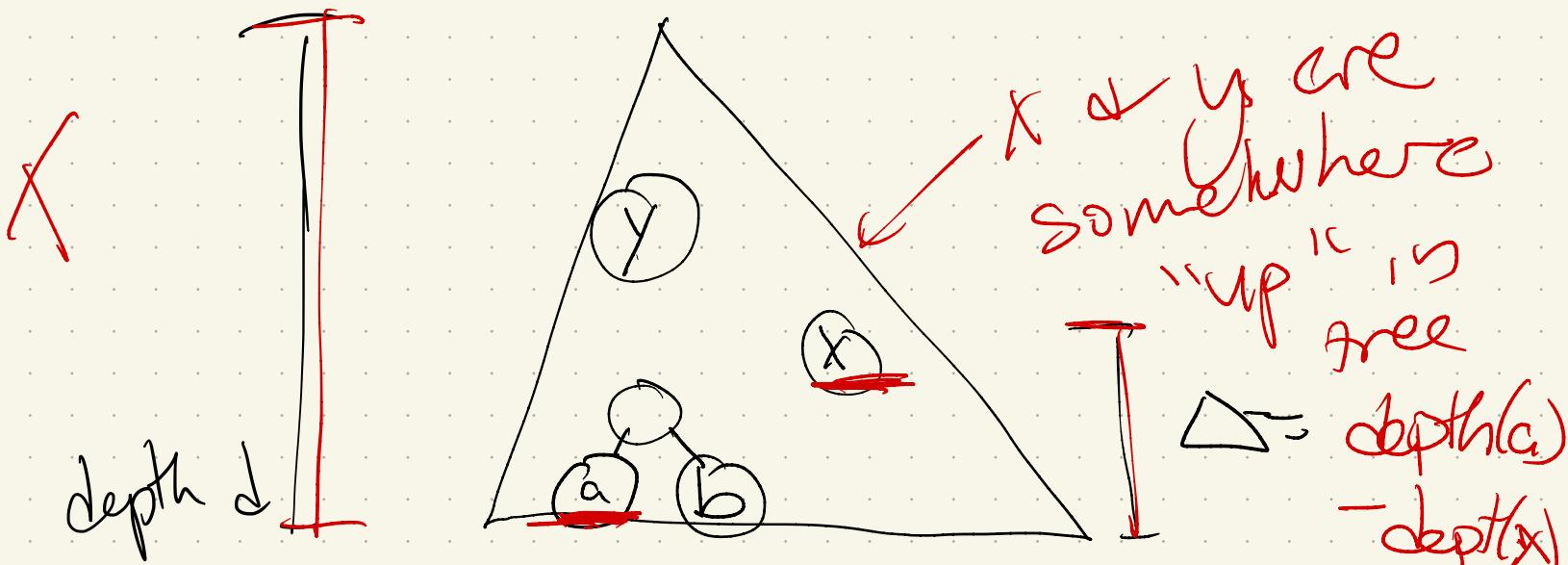


Correctness :

1st Lemma : There is an optimal prefix tree where the two least common letters are siblings at the largest depth.

Pf: Sups not. Then

optimal tree T has some depth d , but at least 2 common letters $x + y$ are not at that depth.



Note some other letters $a + b$ are deepest

Pf cont:

least frequent

Know $f[x] \leq f[a]$, $f[x] - f[a] \leq 0$

but $\underline{\text{depth}(a)} = \underline{\text{depth}(x)} + \Delta$

+ recall that:

$$\text{cost}(T) = \sum_{i=1}^n f[i] \cdot \underline{\text{depth}(i)}$$

Build T' :

Swap a and x in tree
(All other nodes stay same.)

$$\text{Cost}(T') = \sum_{i=1}^n f[i] \cdot \text{depth}(T')$$

$$= \text{Cost}(T) + \Delta \cdot f[x]$$

$$-\Delta \cdot f[a]$$

$$= \text{Cost}(T) + \Delta(f[x] - f[a])$$

and $f[a] \geq f[x]$, so $\Delta \leq 0$ \square

Thm: Huffman trees are optimal.

Pf:

Use induction (+ swap).

BC: For $n=1, 2$, or 3 , Huffman works

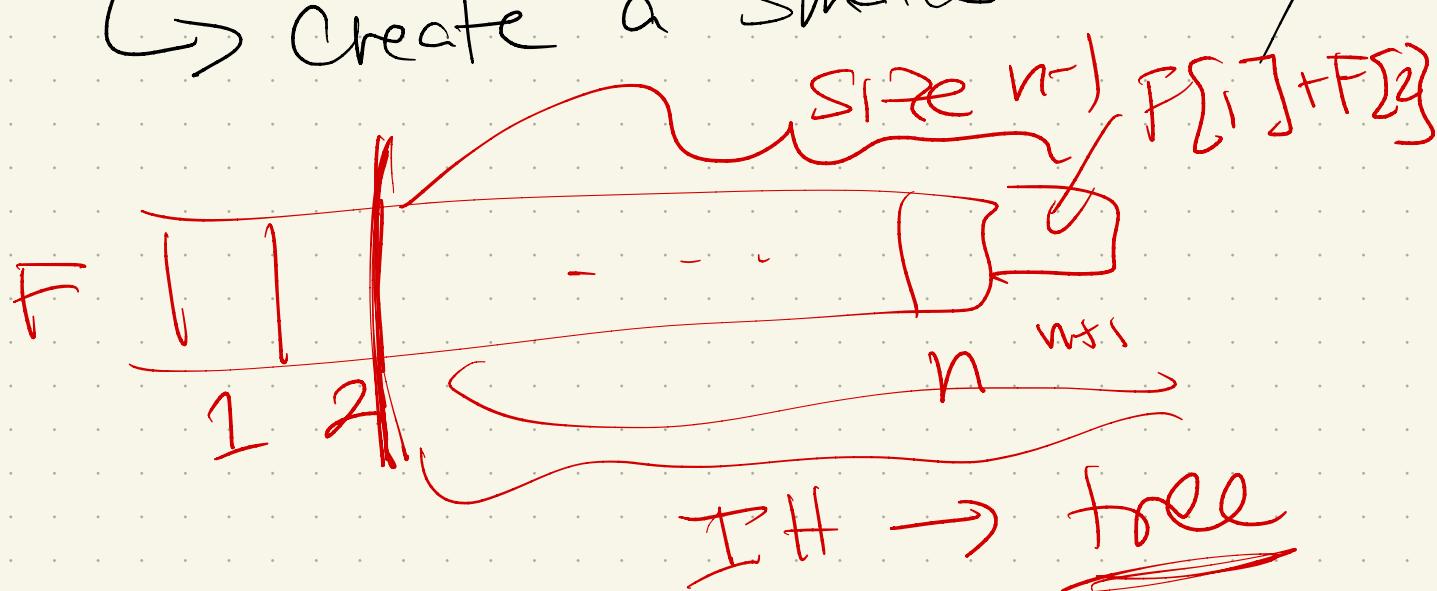
Why?



IH: Assume Huffman works on $\leq n-1$ characters

IS: Input $F[1..n]$, + spp's
 $F[1] + F[2]$ are min freq.

↳ create a smaller array



IS : optimal tree T' of

$F[3..n+1]$:

size $n-1$
array T'

Note : $n+1$
is in tree.

→ leaf somewhere



Build a tree T for $F[1..n]$:

→ take leaf $n+1$, convert
to internal node, & add

Claim: T is optimal.

1 & 2 as
children

Why?

Why is T optimal??
(we know T' is $\rightarrow IH!$)

Cost(T) =

$$\left(\sum_{i=1}^n F[i] \cdot \text{depth}[i] \right)$$

$$= \text{cost}(T') + \underbrace{\text{changes we made}}$$

WT M1
↓ (odd 1
↓ 2)

→ subtract M1's cost
and add 1 & 2's

Spos not:

then M1 was in
wrong place & by
lemma, we know it
wasn't.

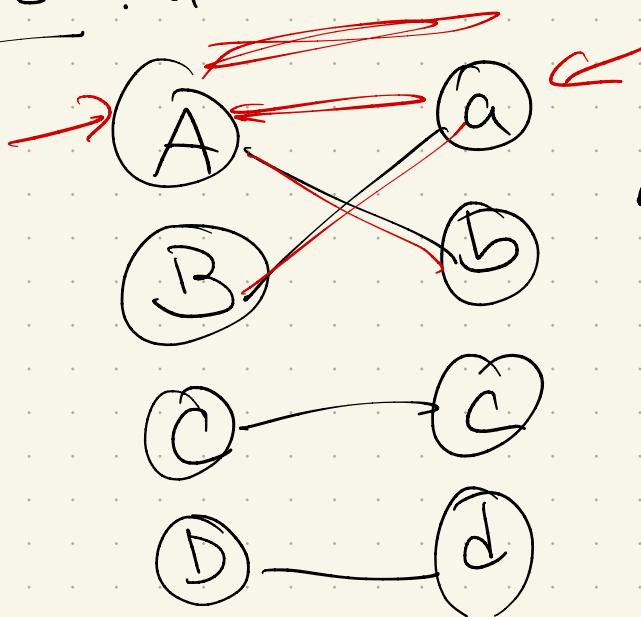
Stable matching

Really useful! Many variants:

- ties
- incomplete preference lists
- one side picks many from the other
- "egalitarian" matchings
- minimizing "regret"

Really a lot of choices to be made.

First: "unstable":



- (A, a) is unstable
 - If A prefers a to current match
 - and a prefers A to current match

In a sense: if put
together & realizing they
both prefer each other,
would (A, a) leave current
matches?

↳ unstable!

History: used to be "stable
marriage"

(long history of strange
papers & variants.)

Algorithm: (wikipedia)

Algorithm [edit]

```
algorithm stable_matching is
    Initialize all  $m \in M$  and  $w \in W$  to free
    while  $\exists$  free man  $m$  who still has a woman  $w$  to propose to do
         $w :=$  first woman on  $m$ 's list to whom  $m$  has not yet proposed
        if  $w$  is free then
             $(m, w)$  become engaged
        else some pair  $(m', w)$  already exists
            if  $w$  prefers  $m$  to  $m'$  then
                 $m'$  becomes free
                 $(m, w)$  become engaged
            else
                 $(m', w)$  remain engaged
            end if
        end if
    repeat
```

In book, data structures matter
for runtime:

		Hospital	1	2	\dots	n
Doctor			○	○	①	—○
1						
2						
:						
n						

Not obvious why it works.
(or even how to be greedy!)

Good example of why the
proof matters.

Nice example of fairness:

This algorithm sucks for
one side.

(Not all solutions are equal!)

How to even define "fair"?

minimizing
"regret"

Graphs

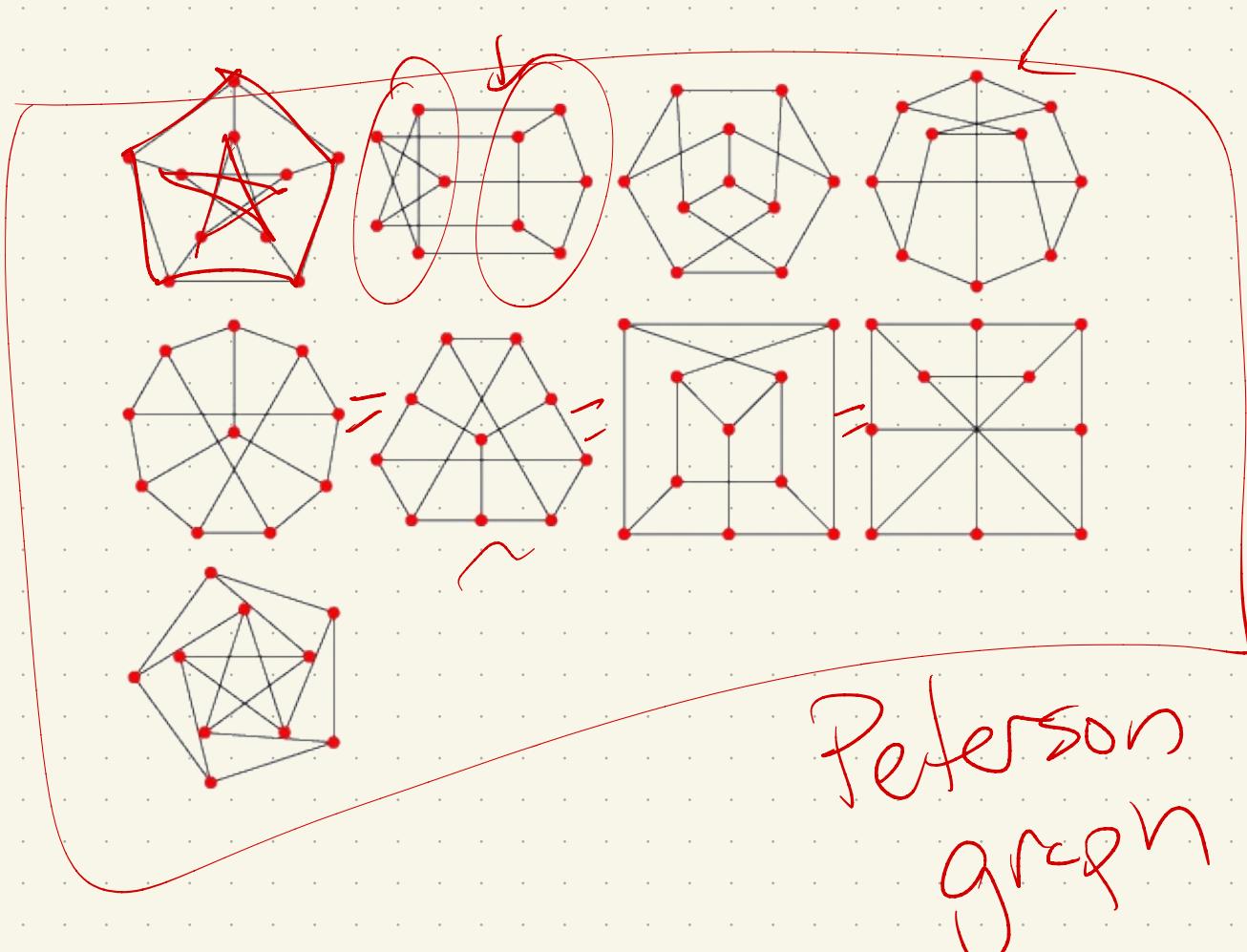
A ^{undirected} graph

$G = (V, E)$ is an ordered pair of 2 sets:

$V = \text{vertices} = \{v_1, \dots, v_n\}$

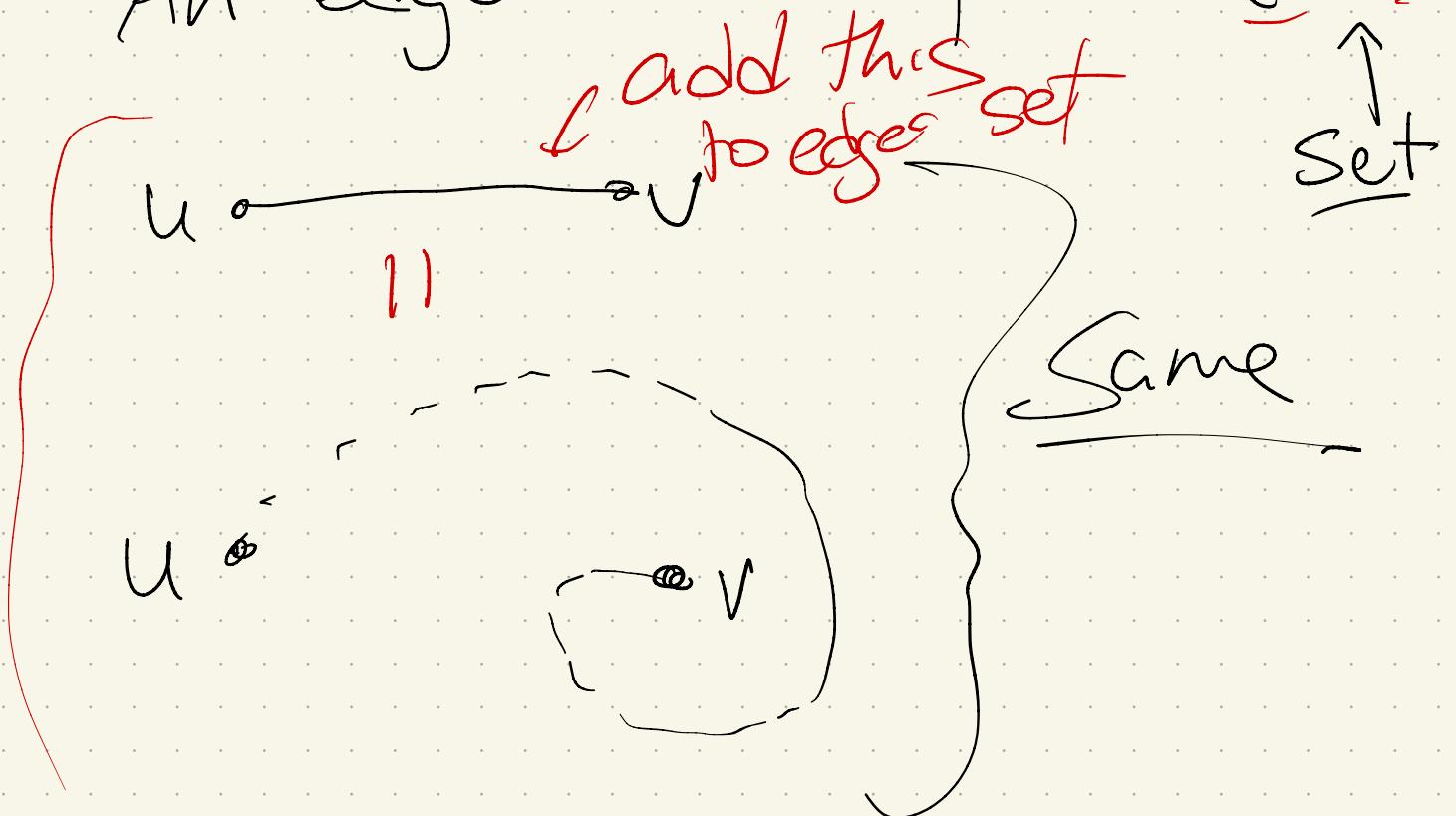
$E = \text{edges} = \{ \{v_i, v_j\}, \{v_1, v_2\}, \dots \}$
set of pairs

We often draw them, but they do not come with coordinates.

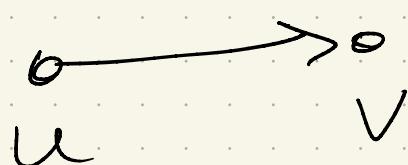


"Edges" \rightarrow not straight!

An edge is a pair $\{u, v\}$:



Directed edges $e = \underline{\{u, v\}}$

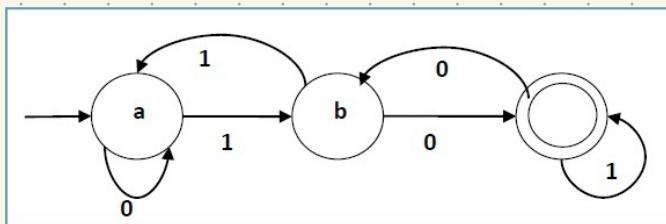


↑
order
matters

$$(u, v) \neq (v, u)$$

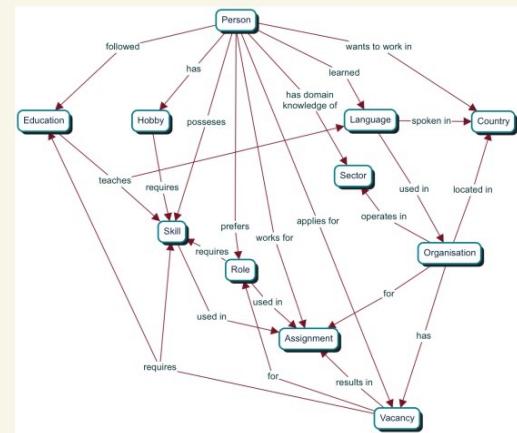
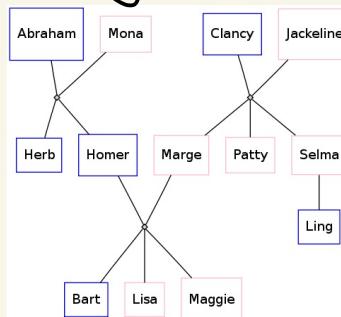
Why Study them?

DFA:



Concept map

lineages:



road network:



Why so much history??

→ context

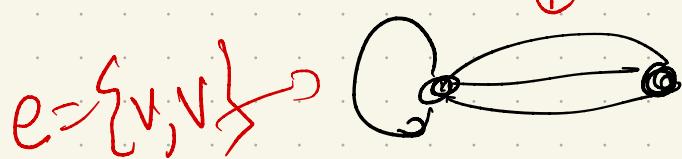
→ problem modeling

Definitions: See book!

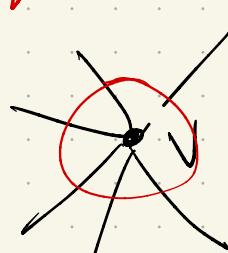
$$|V|=n$$

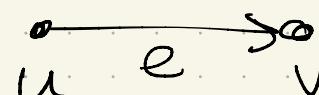
- Vertices (nodes), V
- Edges, E $|E|=m$ 
- endpoints of an edge $e \in E$
- head & tail 
 $\rightarrow u \rightarrow v$ or (u, v) .

- Simple: no parallel edges
~~or 1-edge loops~~

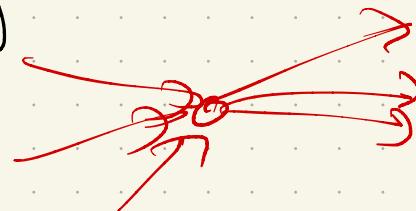


two vertices sharing an edge

- adjacent
- $\text{degree}(v)$ -  # edges for v
- predecessor + successor



Indegree + outdegree



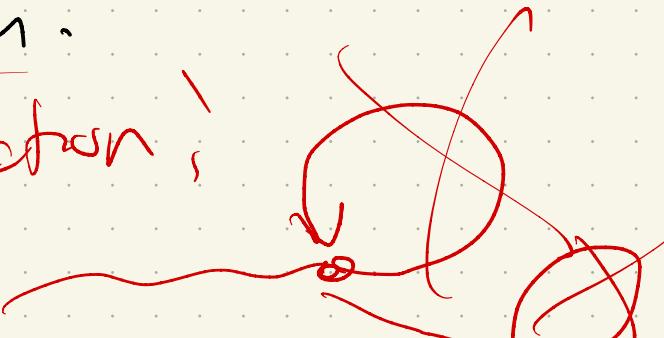
More!

- Subgraph : $\begin{matrix} \text{Subset of } V \\ \text{+ Subset of } E \text{ whose} \end{matrix}$
- Walk - v_1, v_2, e_2, \dots we only use vertex
- Path \leftarrow walk w/ no repeats

Note! If you have a walk $v \rightarrow v$, can make a path.

How?

Induction!



• Connected

• Closed

• Cycle

• Tree

First: some "easy" bounds.

Lemma: $E \leq \frac{v(v-1)}{2}$

Pf:

Lemma: $\sum_v d(v) = 2E$

Pf:

First question

Computers don't do well
with images! So pictures
won't help them.

We need to store this
info (somehow).

Ideas from data structures:

Adjacency (or vertex) lists :

V_1 :

V_2 :

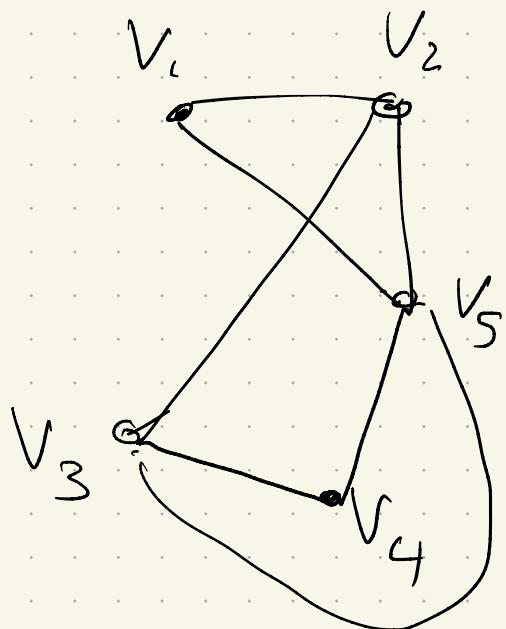
V_3 :

V_4 :

V_5 :

Size:

lookup: time to check if
 $u \& v$ are neighbors:



Implementation:

More buried data structures!
Could use:

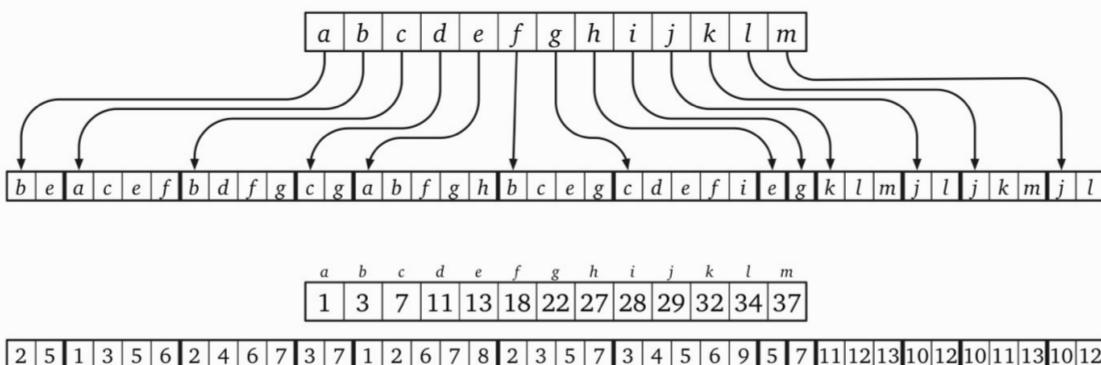


Figure 5.10. An abstract adjacency array for our example graph, and its actual implementation as a pair of integer arrays.

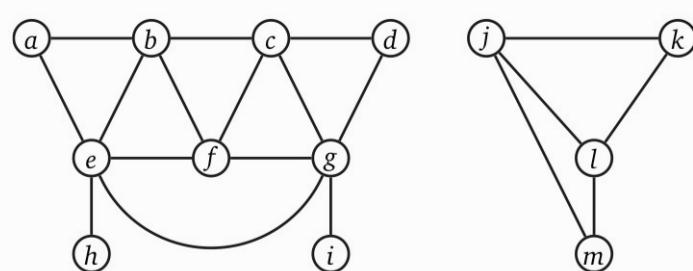
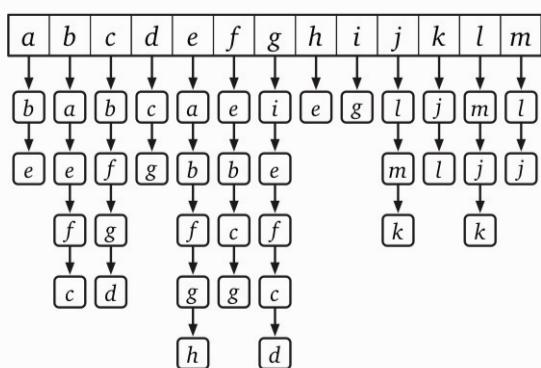
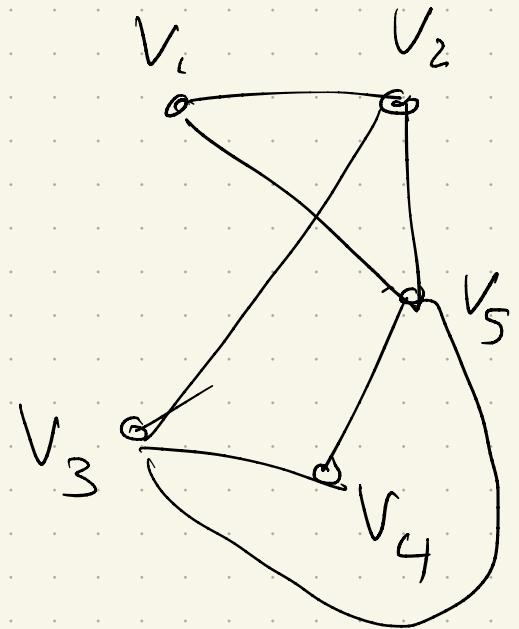


Figure 5.9. An adjacency list for our example graph.

Adjacency Matrix

	v_1	v_2	v_3	v_4	v_5
v_1					
v_2					
v_3					
v_4					
v_5					



Space:
check nbr:

Implementation:
More data structures!

	a	b	c	d	e	f	g	h	i	j	k	l	m
a	0	1	0	0	1	0	0	0	0	0	0	0	0
b	1	0	1	0	1	1	0	0	0	0	0	0	0
c	0	1	0	1	0	1	1	0	0	0	0	0	0
d	0	0	1	0	0	0	1	0	0	0	0	0	0
e	1	1	0	0	0	1	1	1	0	0	0	0	0
f	0	1	1	0	1	0	1	0	0	0	0	0	0
g	0	0	1	1	1	1	0	0	1	0	0	0	0
h	0	0	0	0	1	0	0	0	0	0	0	0	0
i	0	0	0	0	0	0	1	0	0	0	0	0	0
j	0	0	0	0	0	0	0	0	0	1	1	1	1
k	0	0	0	0	0	0	0	0	0	1	0	1	0
l	0	0	0	0	0	0	0	0	1	1	0	1	1
m	0	0	0	0	0	0	0	0	0	1	0	1	0

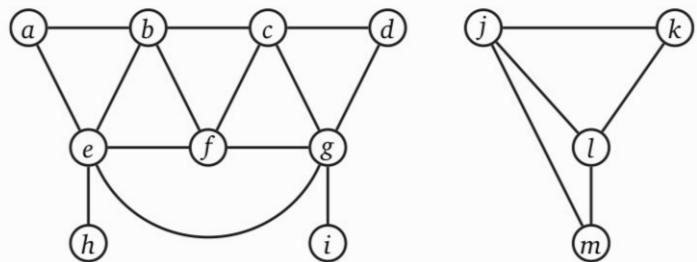


Figure 5.11. An adjacency matrix for our example graph.

Which is better?

Depends!

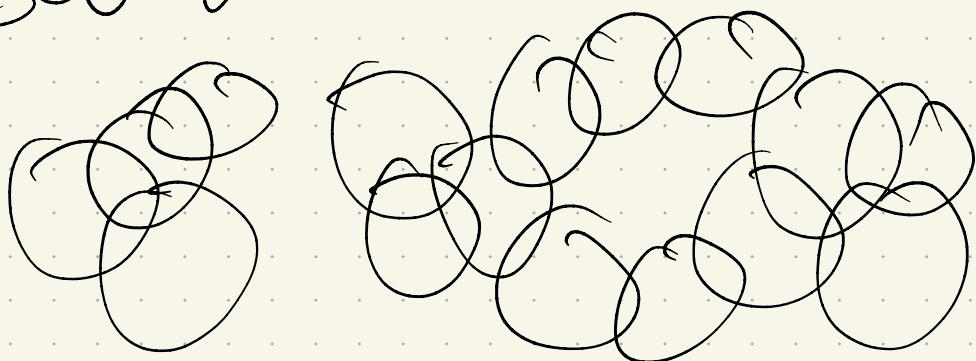
	Adjacency matrix	Standard adjacency list (linked lists)	Adjacency list (hash tables)
Space	$\Theta(V^2)$	$\Theta(V + E)$	$\Theta(V + E)$
Time to test if $uv \in E$	$O(1)$	$O(1 + \min\{\deg(u), \deg(v)\}) = O(V)$	$O(1)$
Time to test if $u \rightarrow v \in E$	$O(1)$	$O(1 + \deg(u)) = O(V)$	$O(1)$
Time to list the neighbors of v	$O(V)$	$O(1 + \deg(v))$	$O(1 + \deg(v))$
Time to list all edges	$\Theta(V^2)$	$\Theta(V + E)$	$\Theta(V + E)$
Time to add edge uv	$O(1)$	$O(1)$	$O(1)^*$
Time to delete edge uv	$O(1)$	$O(\deg(u) + \deg(v)) = O(V)$	$O(1)^*$

In the rest of this book, unless explicitly stated otherwise, all time bounds for graph algorithms assume that the input graph is represented by a standard adjacency list. Similarly, unless explicitly stated otherwise, when an exercise asks you to design and analyze a graph algorithm, you should assume that the input graph is represented in a standard adjacency list.

Really — might depend on
input!

- size of graph
- freq. of changes
- representation: usually,
some "word problem" is
handed to you! You'll
have to build the graph.

Ex: Given a set of overlapping
circles, find the largest
set where no 2 intersect?



Even more:

- Space available
- language used
- previous "legacy" code
- other developers.

o
o
o

To repeat — too keep it simple here.

In the rest of this book, unless explicitly stated otherwise, all time bounds for graph algorithms assume that the input graph is represented by a standard adjacency list. Similarly, unless explicitly stated otherwise, when an exercise asks you to design and analyze a graph algorithm, you should assume that the input graph is represented in a standard adjacency list.