Algorithms

Greed (pt 2)

- Huffman trees
- Stable matching

Recap

- HW: due Monday (dyn. pro.)
- Readings: Thurs. + Sun.
  (over graphs)
- Office hours:
  Friday: after class
  & 3pm
- New Zoom: sending after today
  (password is required)

Friday: use new Zoom link!

Greed:

- First, find a strategy.
  (Not always clear!)
- Often involves playing w/ possible counterexamples.

Then, try to prove you're right:

- Assume optimal is different than greedy
- Find the "first" place they differ.
- Argue that we can exchange the two without making optimal worse.

⟹ there is no "first place" where they must differ, so greedy in fact is an optimal solution.

Example: Huffman trees

Many of you saw this in data structures.

Why?

- cool use of trees
- non-trivial use of other data structure
- useful! data storage & prefix codes are used all over.

Really - it's greedy!

& it's non-linear:

less clear how to be greedy

Goal: Minimize cost

↳ here, minimize total length of encoded message:

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

Compute: T a binary tree, with characters in leaves

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

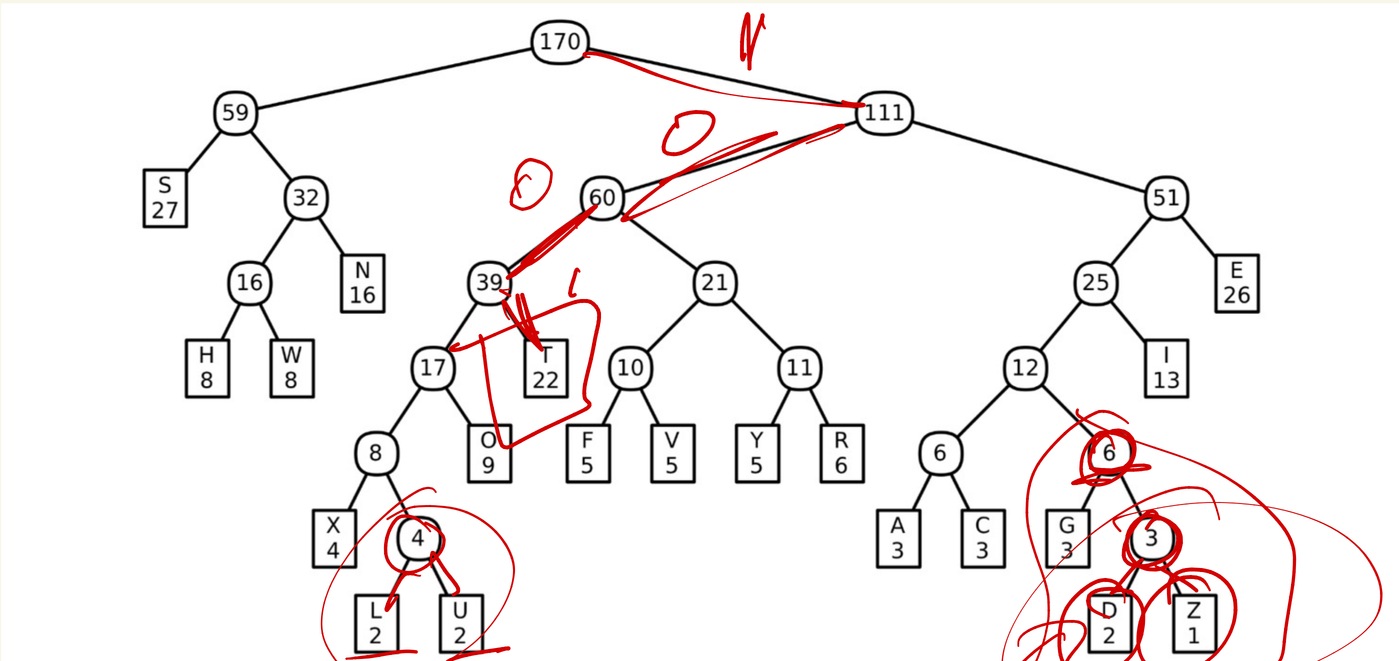
if $f[i] = f[j]$ in T

Strategy:

- Pick 2 least common letters & make them leaves
- "Merge them": remove letters, & add a new letter with sum of their frequencies
- Recurse!

↑ well, a bit imprecise...

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



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

characters

size n

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

4

16

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

Obs: tree will have $2n-1$ nodes

Implementation: use priority queue

↳ $O(\log n)$

```
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
```

sets up leaves

in priority Q

Huffman!

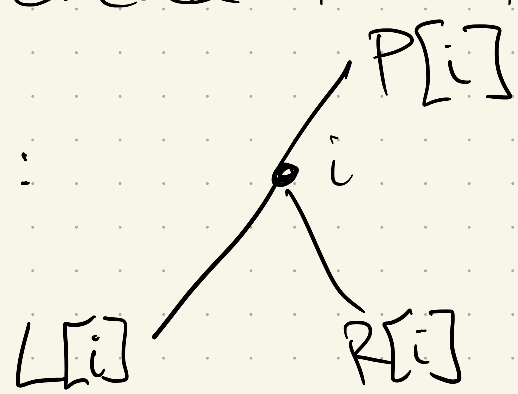
appending an internal node

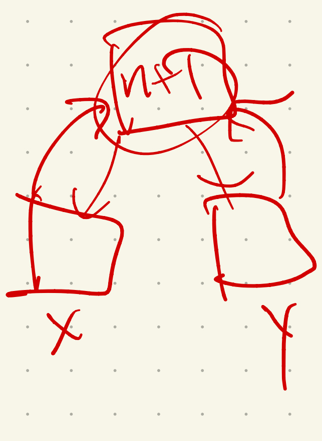
not using pointers!

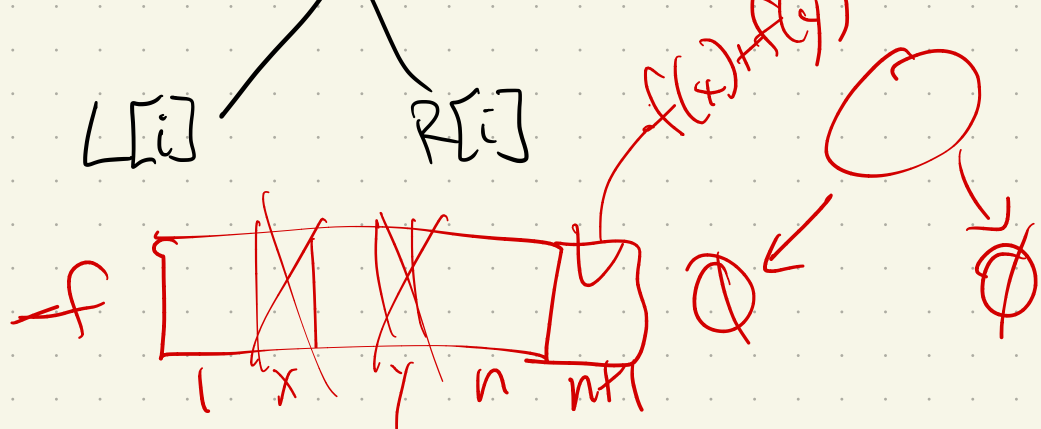
plus f

3 arrays: L, R, P to encode the tree

node i :







So:

BANANA

index:	1	2	3	4
letters:	B	A	N	EOM
freq: f:	1	3	2	1

n

PQ: 1, 3, 2, 1, 2, 4, 7

```
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
```

$2n-1 = 7$

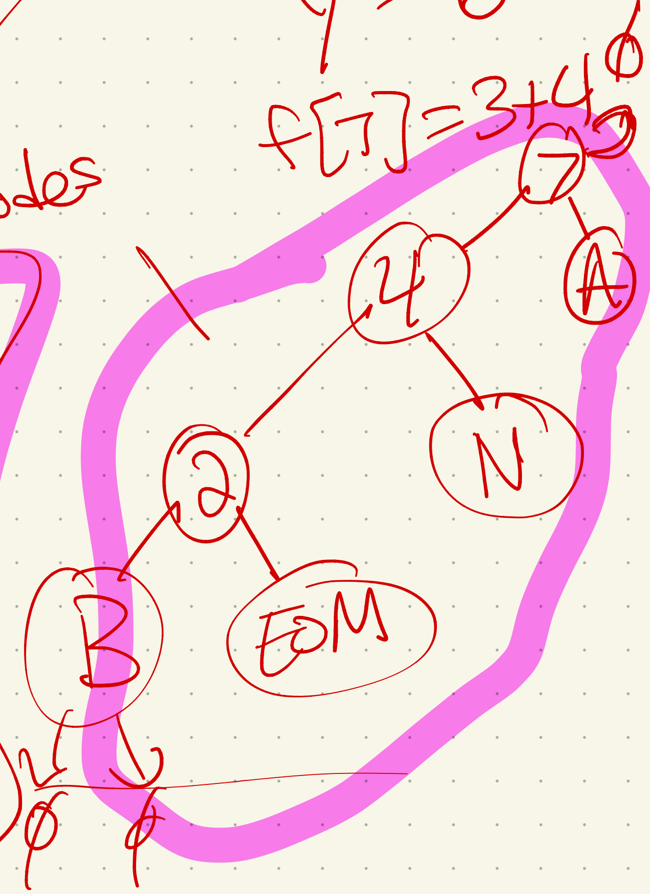
i = 5
x: 1 for B
y: 4 for EOM

i = 6
x = 3
y = 5

i = 7
x = 2
y = 6
f[7] = 3 + 4

input | internal nodes

	1	2	3	4	5	6	7
L:	0	0	0	0	1	3	2
R:	0	0	0	0	4	5	6
P:	5	7	6	5	6	7	0
F:	1	3	2	1	2	4	7
	B	A	N	EOM			



Runtime?

$n \log n$ n $\log n$ $O(1)$

BuildHuffman($f[1..n]$):
- for $i \leftarrow 1$ to n
  - $L[i] \leftarrow 0$; $R[i] \leftarrow 0$
  - Insert($i, f[i]$)
- for $i \leftarrow n$ to $2n-1$
  - $x \leftarrow$ ExtractMin()
  - $y \leftarrow$ ExtractMin()
  - $f[i] \leftarrow f[x] + f[y]$
  - $L[i] \leftarrow x$; $R[i] \leftarrow y$
  - $P[x] \leftarrow i$; $P[y] \leftarrow i$
  - Insert($i, f[i]$)
- $P[2n-1] \leftarrow 0$

repeats $n-1$ times

$O(1)$

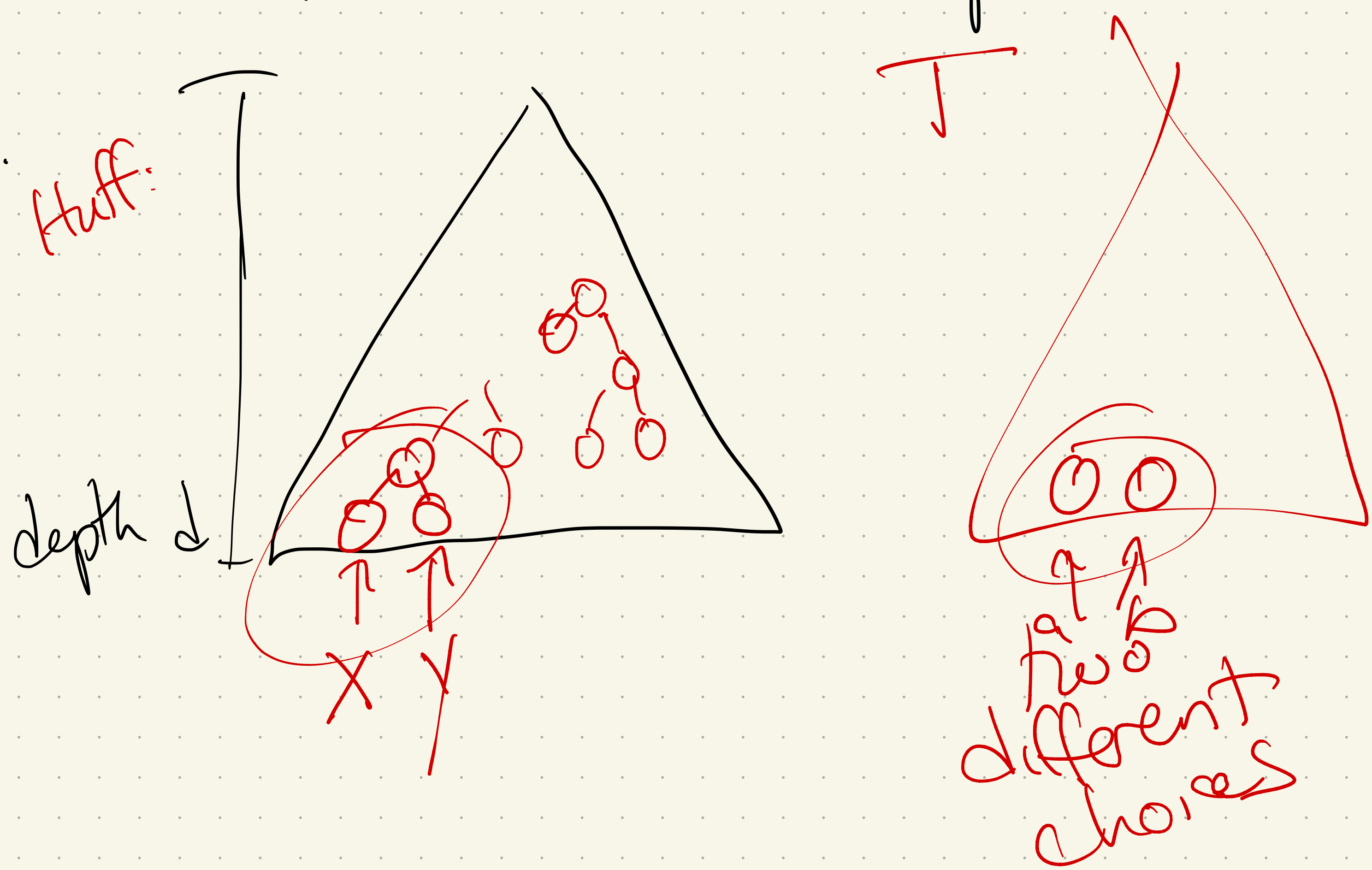
$\log n$

$$\Rightarrow O(n \log n)$$

Correctness:

1st Lemma: There is an optimal prefix tree where the 2 least common letters are siblings + have largest depth.

pf: Spps not. Then optimal tree T has some depth d, but 2 least common letters are not at that depth.

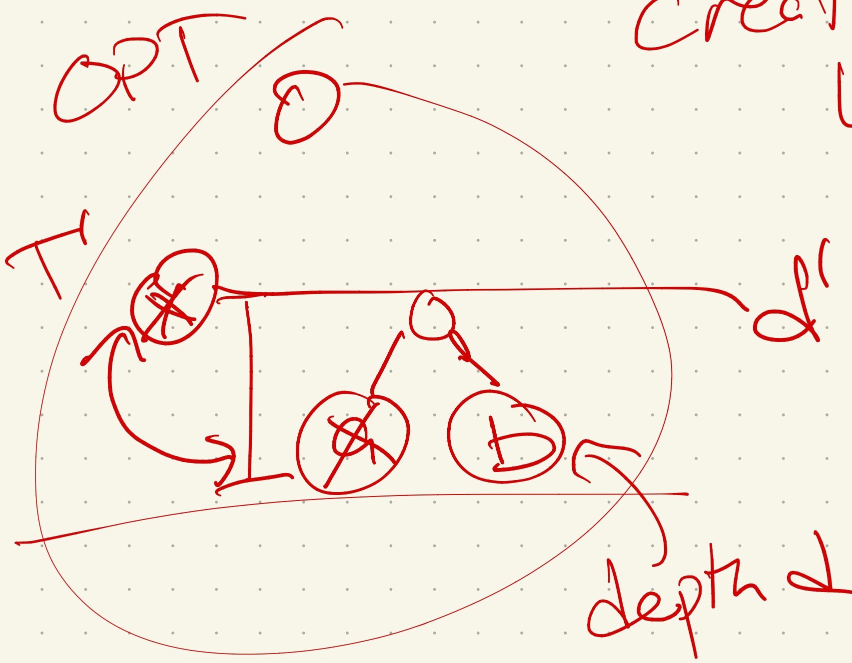


$$\text{cost}(T) = \sum_i f[i] \cdot \text{depth}[i]$$

$\underline{x}$ + y are higher
a + b lower

swap: move x to a's spot

create T': T with x + a swapped



$$\text{cost}(T)' = \text{cost}(T) \boxed{\pm \text{change}(a+x)}$$

$d' < d$

$\underline{\text{freq}[a] \geq \text{freq}[x]}$

$$\Delta = \left[+ \text{freq}[x] \cdot d - \text{freq}[a] \cdot d' \right]$$

Thm: Huffman trees are optimal.

Pf: Suppose not!

Use induction (& ~~contradiction~~ swap).

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

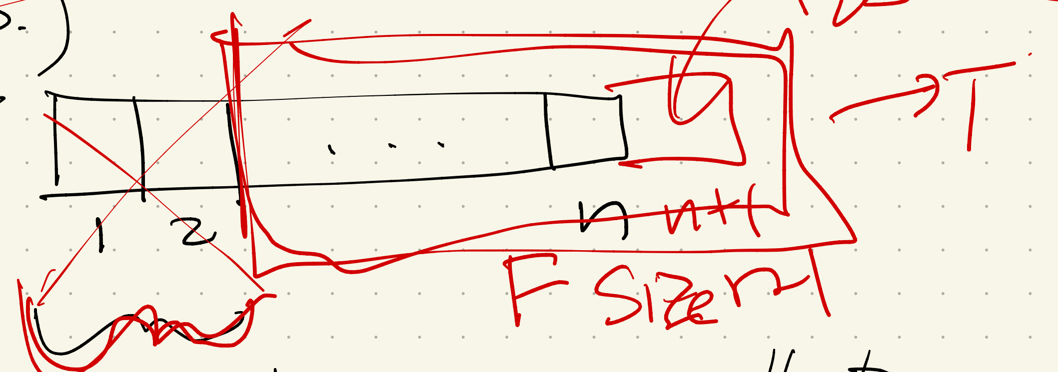
Why? brute force

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

IS: Input $F[1..n]$, & suppose Huff. fails.

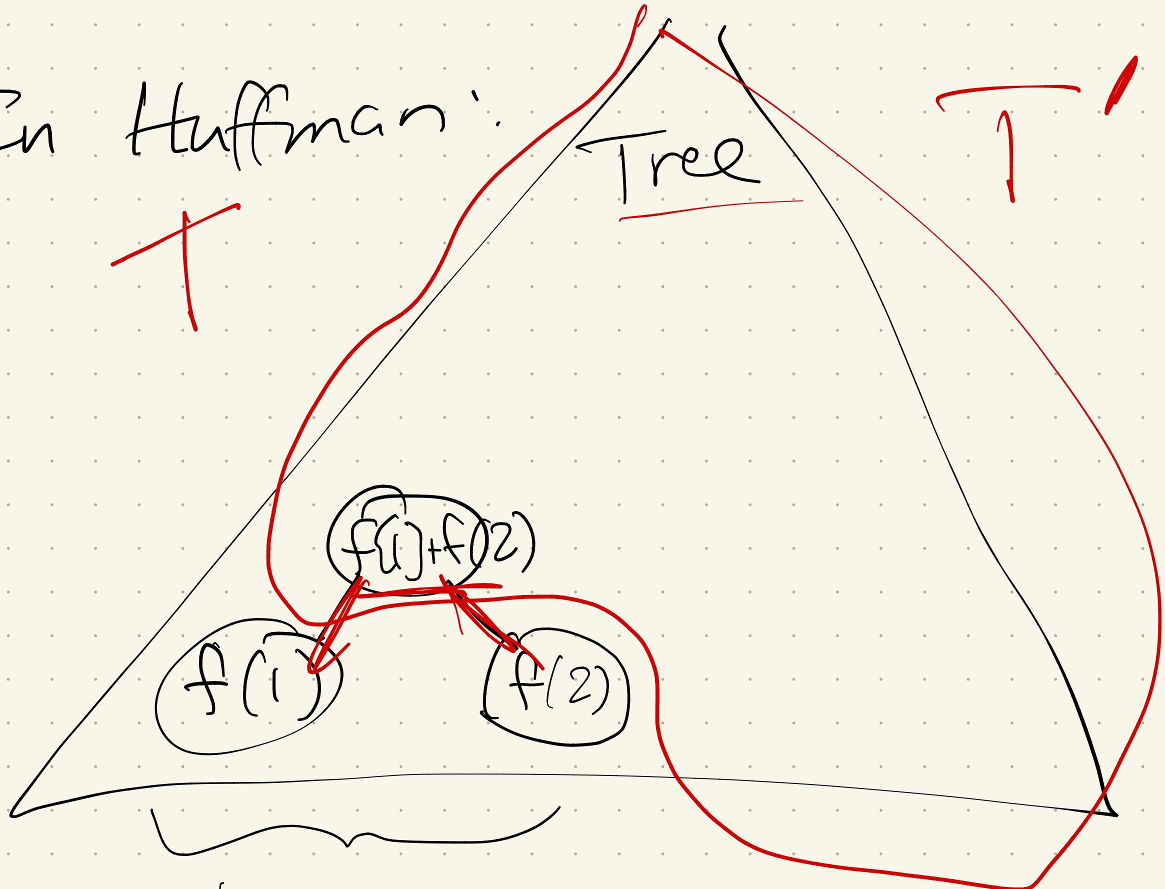
(~~So some other optimal tree exists.~~)

Input: F



Assume these are smallest

In Huffman:



by lemma:

Make T' : Huffman tree for
F[1]+F[2], F[3..n]

IH $\Longrightarrow$ this has minimum cost

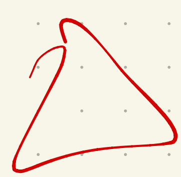
cost(T') $\leq$ any other tree

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

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

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

since T was built from T'



$$= \text{cost}(T') \begin{cases} + f[1] \cdot d \\ + f[2] \cdot d \\ - (\text{leaf } n+1 \text{ removed}) \end{cases}$$

depth $d-1$

freq $f(1)+f(2)$

$$+d(f[1]+f[2]) - (d-1)(f[1]+f[2]) = f[1]+f[2].$$

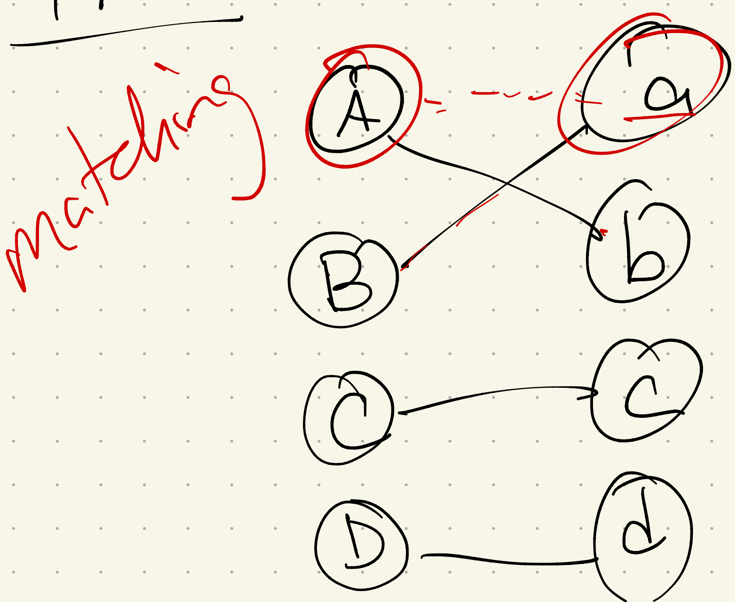
Stable matching

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 ← not paired

- 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!

<u>History</u>: used to be "stable marriage"

(long history of strange papers + variants.)

roomate ~~p~~problem

Algorithm: (wikipedia)

Algorithm [edit]

```
algorithm stable_matching is
    Initialize all m ∈ M and w ∈ W to free
    while ∃ 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

Doctor \ Hospital	1	2	...	n
1	ranks hospitals			
2	2 3 n 1 4			
⋮				
n				

O(1) access time

2 n×n arrays
another for hospitals

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"?

↳ EC reading