Activity selection:

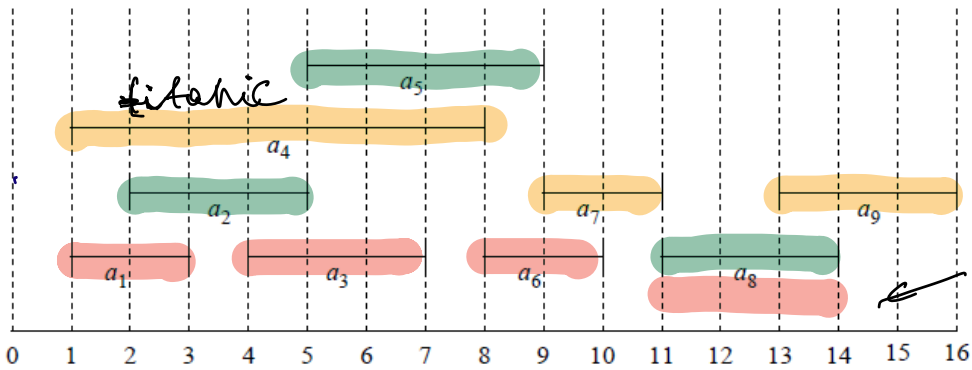
Input: n activities $a_i = [s_i, f_i)$

Goal: maximise # of non-overlapping activities.

Example

S sorted by finish time: *[Leave on board]*

i	1	2	3	4	5	6	7	8	9
s_i	1	2	4	1	5	8	9	11	13
f_i	3	5	7	8	9	10	11	14	16



GREEDY (=OPT)

$4 > 3 = 3$

is it OPT? Can we get 5? NO ☹

F_2(orange solution) = 11

F_2(green) = 9

F_2(red) = 7 ← indeed smallest

- is it unique? NO

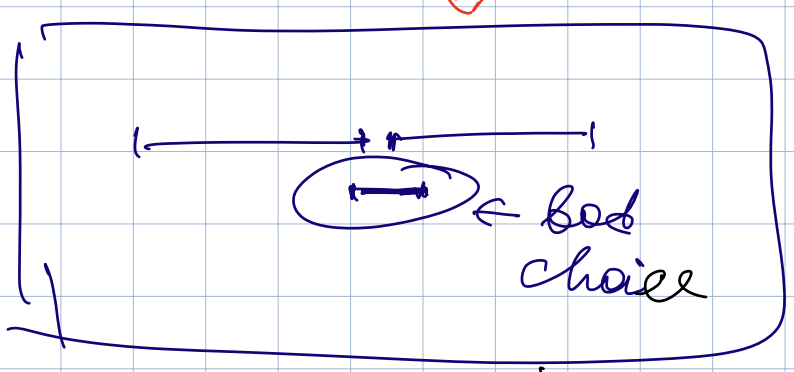
1) Dynamic Programming.

- YES, but too slow
- ∃ counter-example (GREEDY)

---

2) How to choose a "safe" movie to watch
- Can recurse.

(a) shortest movie
- NO



(b) a movie overlapping least movies
- NO (exercise)

Can try to prove by induction. ← what induction?
- in this case, you'll be stuck

(c) Start with the movie that ends earliest.

2 techniques:

(1) Greedy always stays ahead (GASA)

(2) Local Swap (LS)

GASA

- Define a correct "measure of progress" formalising that for every step i, greedy solution G is "better" than any other valid

solution 2.

- Intuition: explains why we chose this GREEDY.
- 10% rule: give it a name & define formally.

$\forall\ 1 \le i \le (\le OPT)$, $\forall$ valid solution Z,

$F_i(Z)$ = end time of the i^{th} movie (chronologically) selected by Z

$(= \infty$ if $|Z| < i)$

- 89% rule: argue by induction that $\forall\ i$, $\forall$ valid Z.

$$F_i(G) \le F_i(Z)$$

- 1%: conclude G produces an optimal solution.

Take $i = |OPT|$

$$F_i(G) \overset{89\%}{\leq} F_i(OPT) < \infty$$

$$\Rightarrow |G| \geq i = |OPT|.$$

proof of 89%:

Base case $i = 1$: (actually easy)

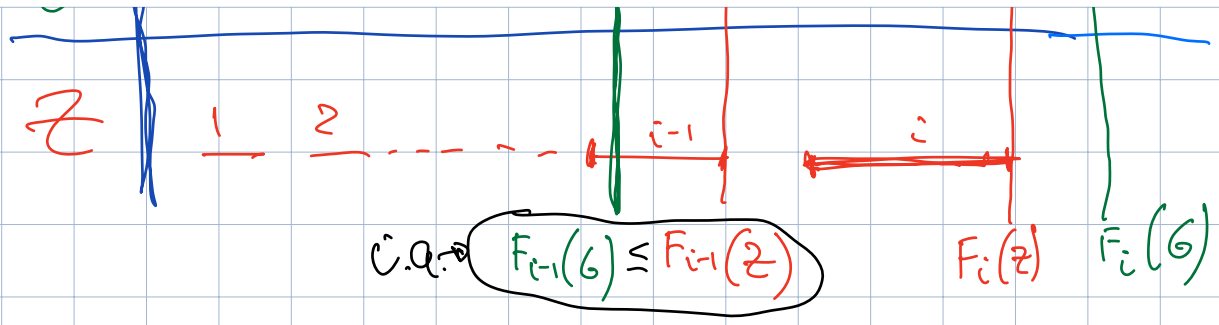
$F_1(G)$ = smallest end time of any movie $\leq F_1(Z)$

Inductive step:

Assume $F_{i-1}(G) \leq F_{i-1}(Z)$. i.e.

Prove this $\Rightarrow F_i(G) \leq F_i(Z)$

(ignore trivial ∞)



Key question: would greedy consider i ?

YES!

Why?

Z is valid

Short time at red i $\geq F_{i-1}(Z)$ induc $\geq F_{i-1}(G)$

Hence, $F_i(G) \le$ event-time at red i $= F_i(Z)$

Side benefit of CASA: "explains" in "deeper" way Greedy is better that other soluti

local swap

Let Cost(Z) = cost of solution Z.

Let ALL = {all solutions Z}.

ALL_1 = {all solutions Z which "agree" with G in 1st "step"} $\subseteq$ ALL

Step 1: (84%)

$$\max_{Z \in ALL} Cost(Z) = \max_{Z \in ALL_1} Cost(Z)$$

- usually involves "local swap"
- shows "safe" to assume first step of OPT agrees w/ GREEDY.

(15%)

Step 2: finding optimum $Z_1 \in ALL_1$

"reduces" to instance I_1 of original problem I, where $|I_1| < |I|$.

- usually easy.

(1%) Step 3: first step of Greedy on I_1 = second step of Greedy on I

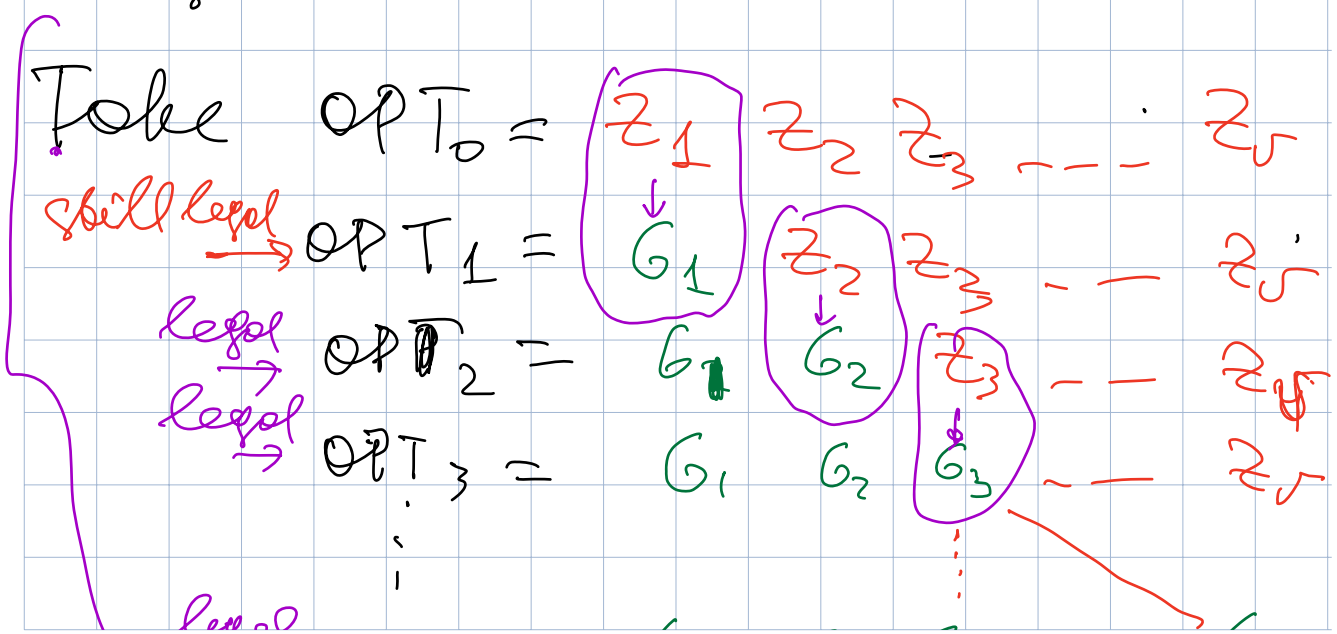
- offer redundant, as can define GREEDY(I) as:

1) "take first greedy step on $I_0 = I$

2) compute smaller problem I_1.

3) Go to 1) with $I = I_1$

Visualisation (not needed in proof)



$\Rightarrow OPT_J = g_1\ g_2\ g_3\ \dots\ g_J$

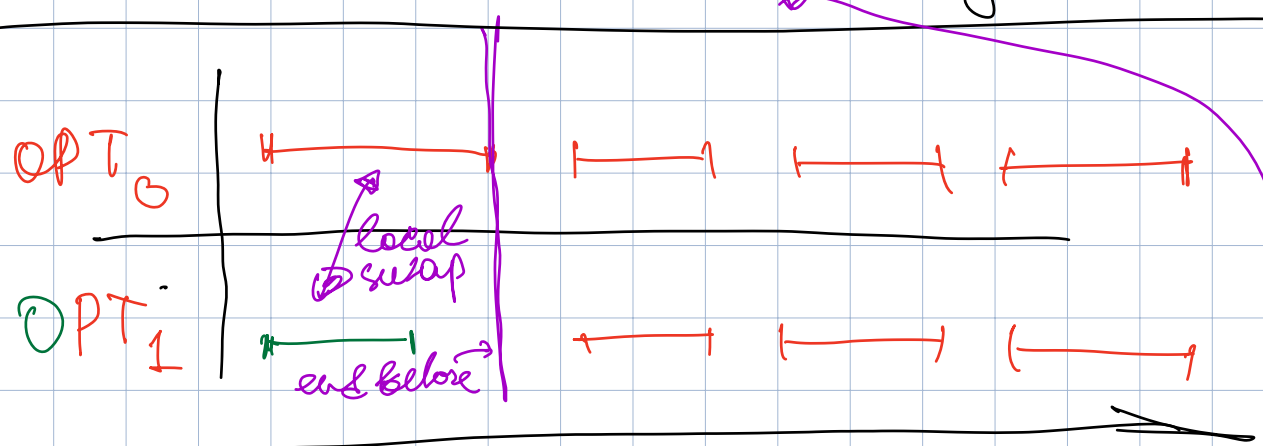
Back to LS for Activity selection.

Step 2:

$$\max_{Z \in ALL} Cost(Z) = \max_{Z \in ALL_1} Cost(Z)$$

ALL – all valid schedules

ALL_1 – all valid schedules which start w/ movie q_1^* ending earliest



Step 2: finding OPT for ALL_1 (=> finding OPT for all

$I_1 = \{a \mid a$ starts after q_1^* ends$\}$

– indeed smaller than $|I|$, (doesn't contain q_1^*)

Greedy Algorithms

1 Activities Selection

Given: $a = [s_1, f_1), [s_2, f_2), \ldots, [s_n, f_n)$

Assumption: $f_1 \leq f_2 \leq \ldots \leq f_n$

1. Select the largest set of non-overlapping activities.
2. Observation: OK to start with activity ending first.
3. Then, take the activity whose starting time is closer to the finishing time on first activity.

Recursive (already sorted with finishing time)

REC-AS(s, f, i) ◁ i is initially 0

```
1 m ← i + 1
2 while m ≤ n ∧ s_m < f_i do
3 |   m ← m + 1
4 if m ≤ n then
5 |   return a_m ∪ REC-AS(s, f, m)
6 else
7 |   return ∅
```

← eliminate conflicts with $a_m - a_{i+1}$

recursive

Iterative

TOP-DOWN-AS(s, f) ← even iterative top-down, as only 1 choice.

```
1 i ← 1.  A = {a_1}
2 for m = 2 to n do
3 |   if s_m ≥ f_i then ← skip conflicts
4 |   |   A = A ∪ a_m
5 |   |   i = m
6 return A
```

- □ Greedy is often Dynamic Programming with **1** known subproblem
- □ Iterative version of Greedy Algorithm is top-down

time = $O(n)$[1], not counting sorting

Huffman Codes:

not sorted.

	a	b	c	d	e	f	Cost
input → %	45	13	12	16	9	5	
fixed	000	001	010	011	100	101	3
opt	0	101	100	111	1101	1100	2.24
$\mathcal{C}_0$	0	00	01	1	10	11	

what's wrong?

Want assign binary representation to letters

$\mathcal{C}_0$: ambiguous?

00 → aa
→ b
unclear.

Codes:

$1 \times 0.45 + 3 \times 0.13 +$
$+ 3 \times 0.12 + 3 \times 0.16$
$+ 4 \times 0.09 + 4 \cdot 0.05$
$= 2.24$

- prefix-free (sufficient)

↓

nice, can decode right away.

Goal: find PFE with smallest average cost!

part of input

$$\text{Cost}(\mathcal{C}) = \sum f(\ell) \cdot \text{length}_{\mathcal{C}}(\ell),$$

letters ℓ

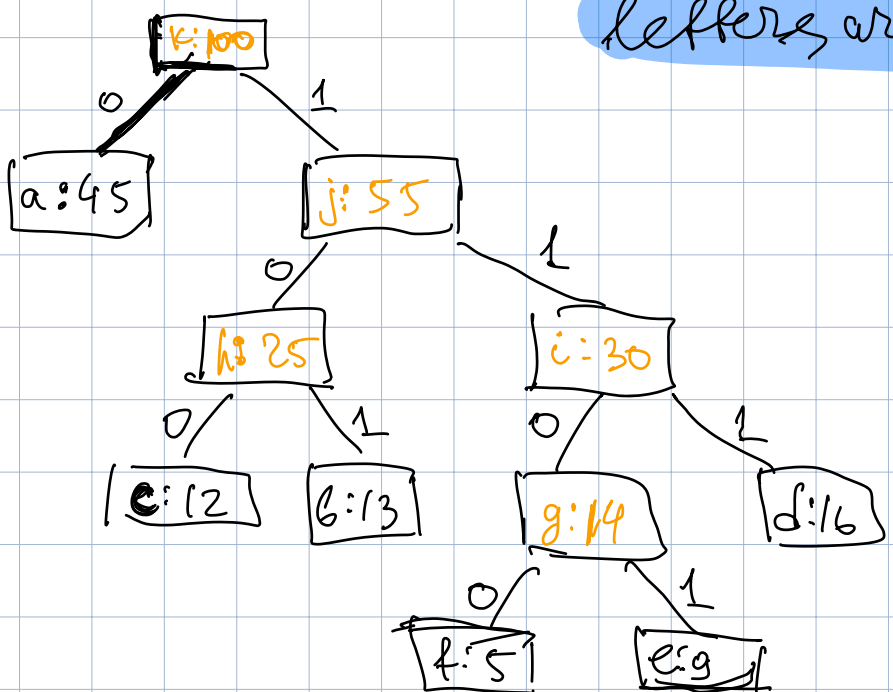
length of PFE encoding of ℓ in Z.

local swap

tree representation for PFE

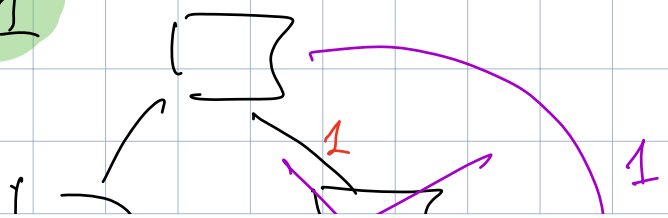
internal nodes don't have input letters.

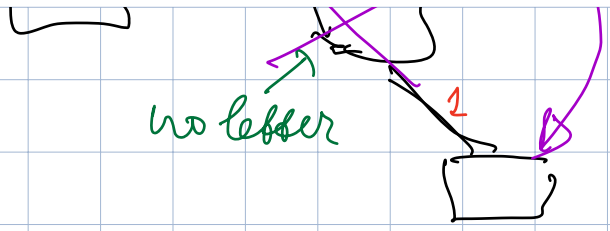
letters are leaves



start with $Z = OPT$.

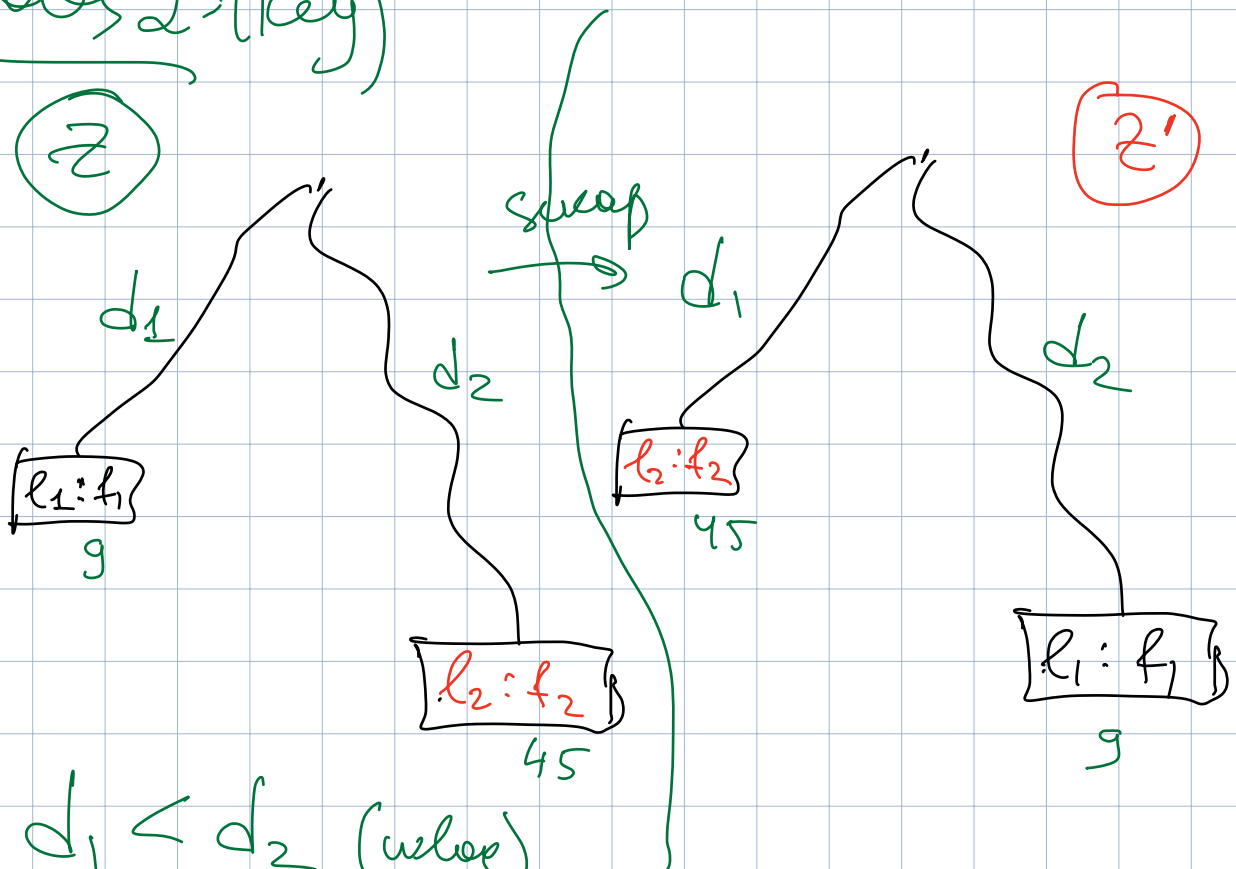
Obs 1: no internal node has degree = 1





Second 1 is redundant

Obs 2: (key)



$d_1 < d_2$ (wlog)

When should we swap?

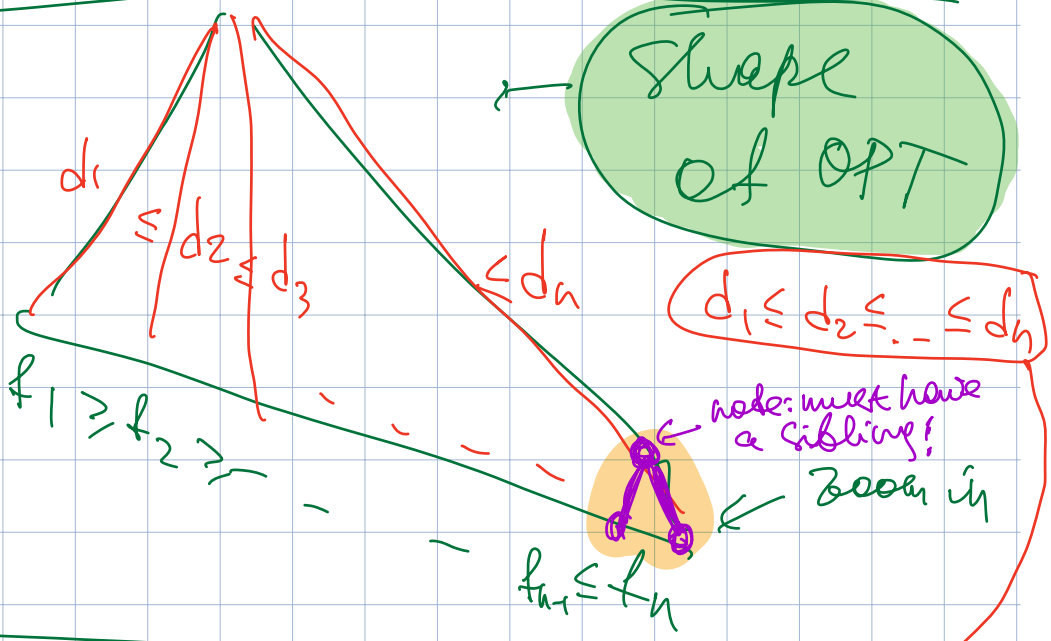
- if and if

$f < l$

1 2 exercise

$$d_1 \cdot f_2 + d_2 \cdot f_1 \leq d_1 f_1 + d_2 f_2$$

Obser 3: ← Shape of OPT

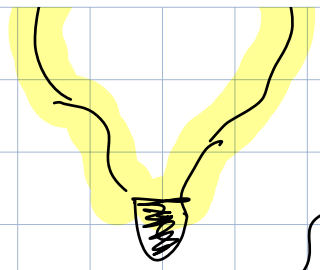


$f_1 \geq f_2 \geq f_3 \ldots$

OPT Cost $= f_1 \cdot d_1 + f_2 \cdot d_2 + \ldots + f_n d_n$

Step 1: Define $ALL_1 =$

{all trees T where two lowest frequency letters l_{n-1} & l_n are siblings at the deepest



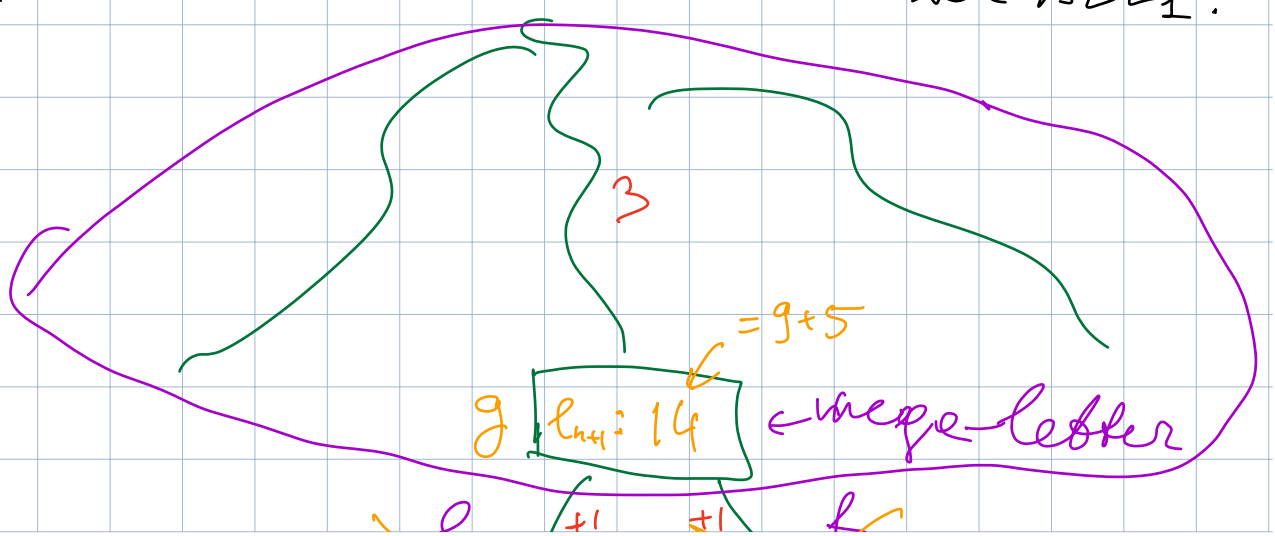
level of T 3.

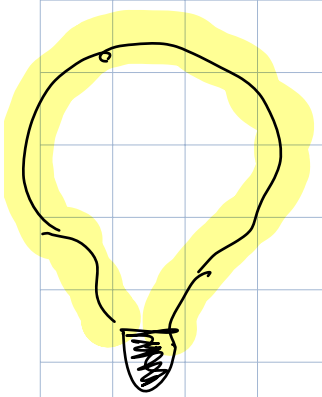
Indeed

$$\min_{T \in ALL} Cost(T, \{f_3\}) = \min_{T \in ALL_1} Cost(T, \{f_3\})$$

Wlog two lowest frequency letters l_{n-1} & l_n are siblings at the deepest level of optimal T.

Step 2: Define smaller instance for ALL_1.





Replace e and f by
1 mega-letter g where

$$\text{freq}(g) = \text{freq}(e) + \text{freq}(f)$$

Claim:

OPT free $(a\ b\ c\ d\ \underbrace{e\ f}_{9\ \ 5})$ =

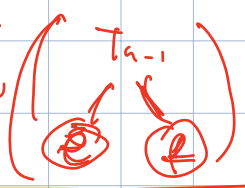
= OPT free $(a\ b\ c\ d\ \underbrace{g}_{14})$ +

+ 2 little children

Huffman: iteratively merge
2 least frequent letters into
a "mega-letter" of frequency =

sum of frequencies, and iterate, until only 1 mega-letter (= root) is left. Then "unmerge" backwards.

Let C_0 = cost ($\underbrace{OPT}_{T_{n-1}}$ on $n-1$ letters)

C_1 = cost()

$$C_1 = C_0 + \text{freq}(e) + \text{freq}(f)$$

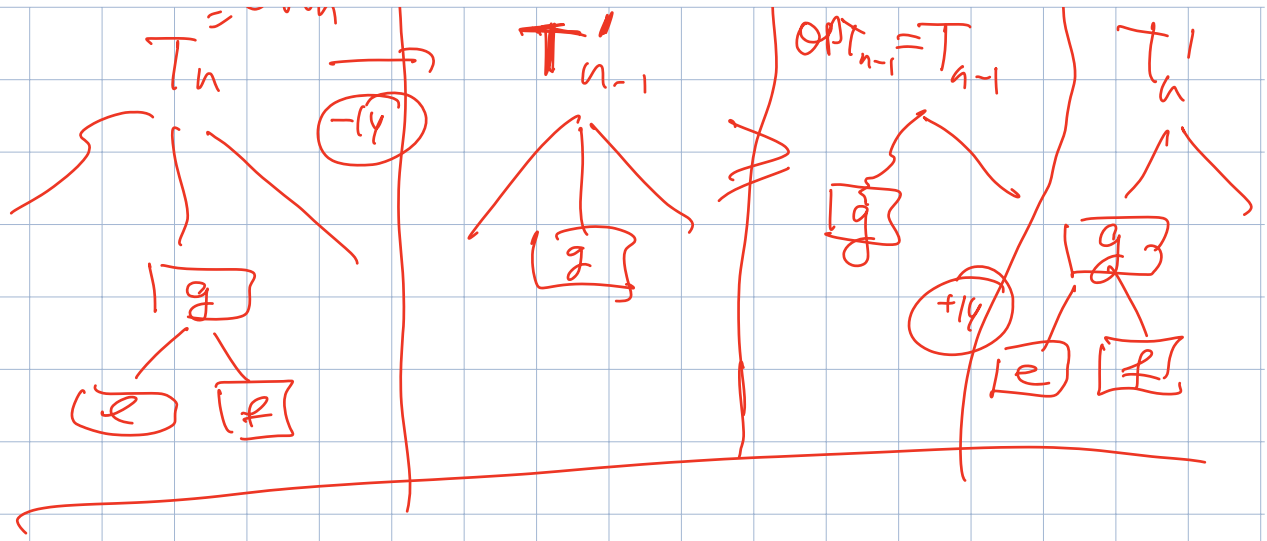
T_n = opt for n letters, where $e+f$ together

T'_{n-1} = shrink (T_n by $e+f \to g$)

T_{n-1} = OPT for $(n-1)$ letters.

T'_n = expand (T_{n-1})

optim ?

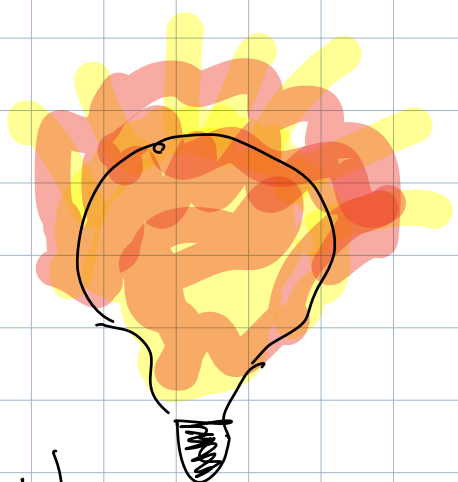


$$cost(T_n) = OPT \geq cost(T'_n)$$

$\geq$!

See book how to implement in time $O(n \log n)$ using

Min Heap

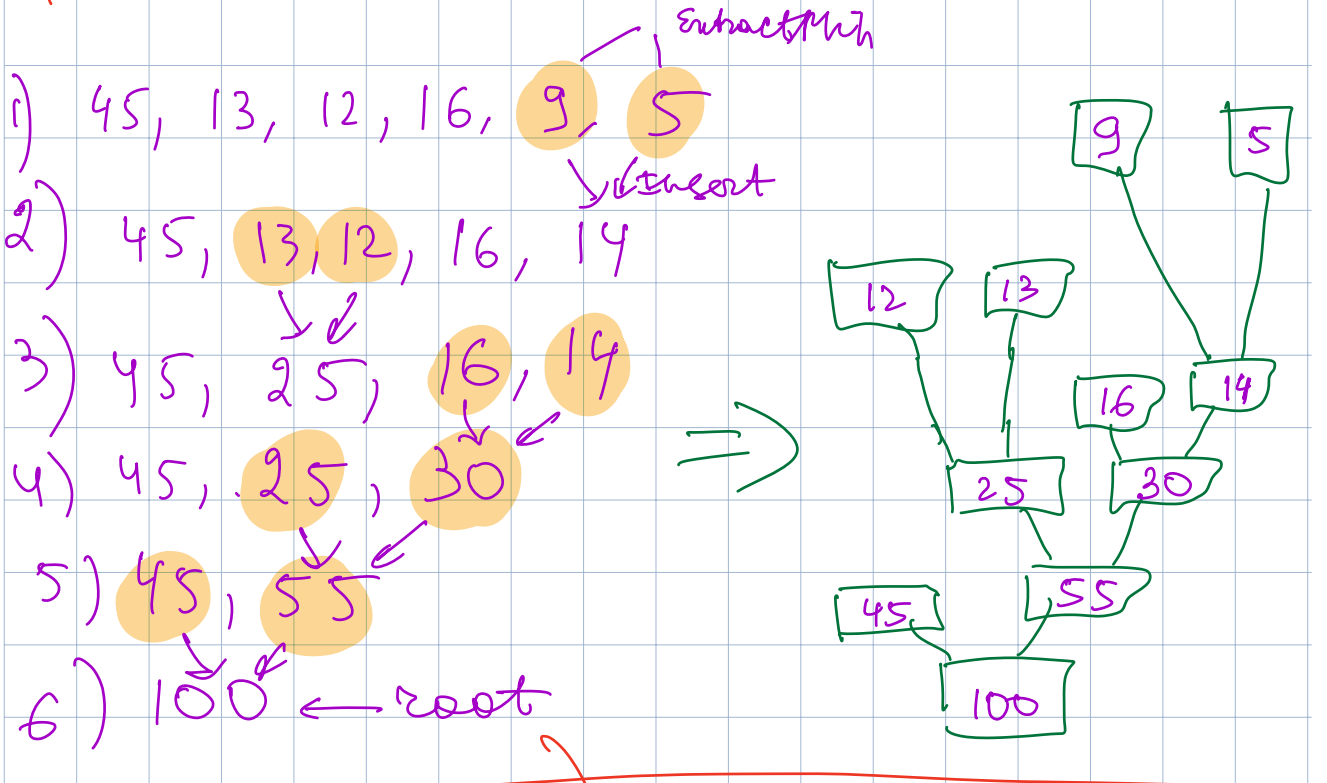


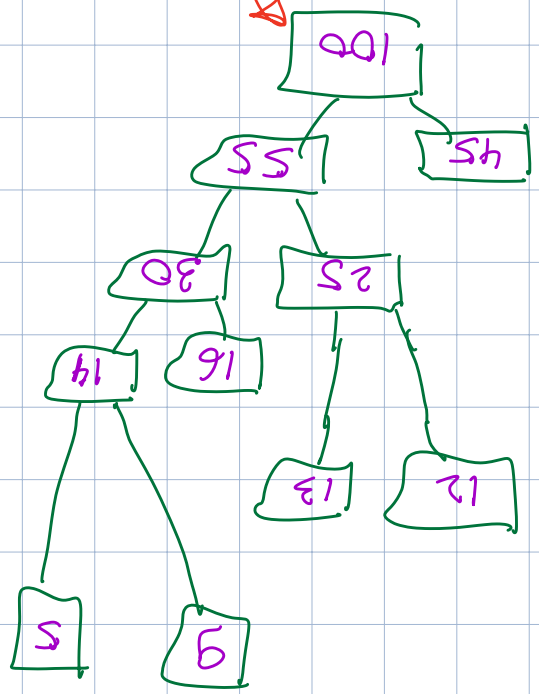
Idea: put all n frequencies in Min-Heap (priority queue)

n times:
- iteractively do 2 extract-mins $O(\log n)$
- insert new letter whose frequency is sum of 2 extracted letters $O(\log n)$

```
HUFFMAN(C)
1  n = |C|                          ← BuildPQ(C)
2  Q = C
3  for i = 1 to n − 1
4      allocate a new node z
5      z.left = x = EXTRACT-MIN(Q)
6      z.right = y = EXTRACT-MIN(Q)
7      z.freq = x.freq + y.freq
8      INSERT(Q, z)
9  return EXTRACT-MIN(Q)        // return the root of the tree
```

Example: BuildPQ (45, 13, 12, 16, 9, 5).



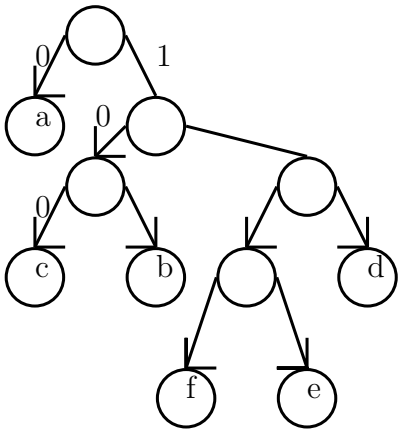


2 Huffman Codes

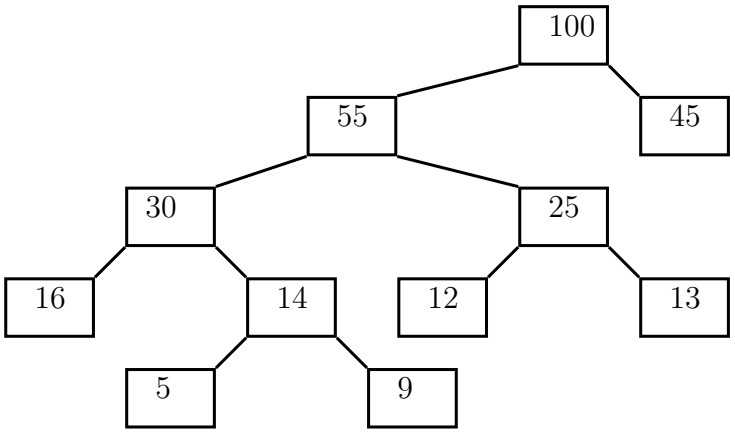
	a	b	c	d	e	f
%	45	13	12	16	9	5
fixed	000	001	010	011	100	101
opt	0	101	100	111	1101	1100

□ Prefix-Free Encoding:
$\forall c_1, c_2$, encoding of c_1 named $E(c_1)$ is not prefix of $E(c_2)$
$cost(E) = \Sigma f(c)|E(c)|$

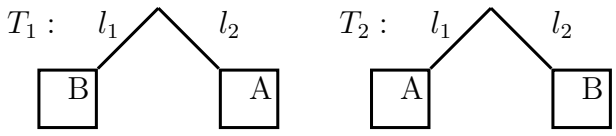
cost(fixed) = 3



cost OPT = 2.24



□ Correctness
Claim 1: OPT never has nodes of degree 1 (one child)
Claim 2: Let T_1, T_2 be solid trees which are identical except for 2 swapped frequences:



$l_1 \geq l_2 \wedge A \leq B$
$cost(T_2) \leq cost(T_1)$

Corollary: If we sort frequencies in ascending order, then optimal lengths will decrease.