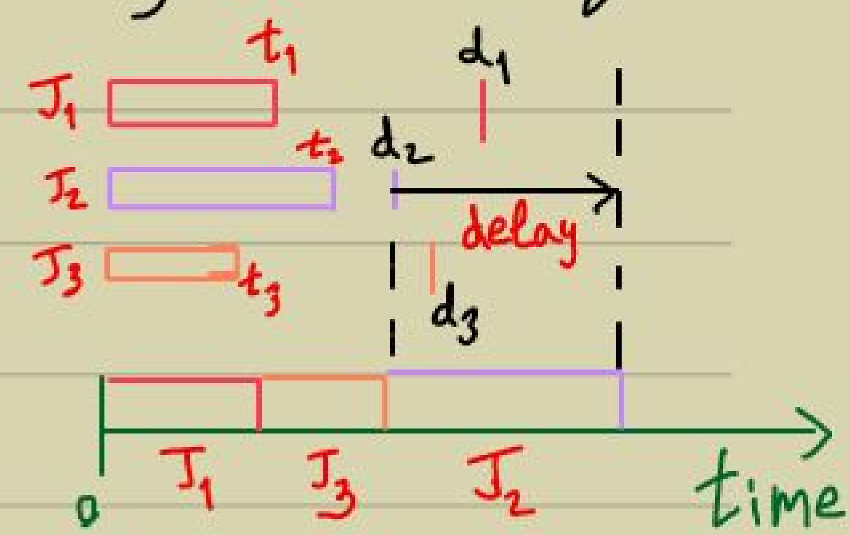
Job Scheduling

Input:
- There are n jobs $\{J_1, \ldots, J_n\}$.
- Job J_i takes t_i time to complete.
- Job J_i has deadline d_i.

Output: Schedule them on a single server such that the maximum delay is minimized.

- **Qn:** Is this a hard problem?



- **Ideas:** 1) Schedule in the order of t_i's.
  2) " " " " d_i's.
- Shortest job vs Earliest deadline
- Counterexample for Idea-1:
  $\{J_2, J_3\}$ above. It is better if J_2 is scheduled first.

▷ For two jobs, Idea-2 always works.

Proof: • Let $\{J_1, J_2\}$ be the jobs.

• If J_1 is scheduled first, the delay is $t_1+t_2-d_2$.

$\max(t_1-d_1, t_1+t_2-d_2)$

$\wedge$

$\max(t_2-d_2, t_1+t_2-d_1)$

• If J_2 is scheduled first, then the delay is $t_1+t_2-d_1$.

$\Rightarrow$ delay is minimized if we schedule according to the earliest deadline. $\square$

- Qn: What to do with $n>2$ jobs?

- Consider a scheduling in the order $(J_1, J_2, \ldots, J_n)$. Focus on $\{J_i, J_{i+1}\}$.
  - If $d_i > d_{i+1}$ then swapping them gives us a "better" scheduling.

$\Rightarrow$ Thus, in an optimal schedule we have $d_1 \leq d_2 \leq \cdots \leq d_n$ (without loss of generality).

Theorem: Job Scheduling (min max delay) can be done in $O(n \lg n)$ time.

Greedy Paradigm

- In the last algorithm we used a local approach to get a global one.
  (From $n=2$ to $n>2$.)

- Given an optimization problem P, with instance A of size n:
  - Greedy Step identifies an instance A' of size $n'<n$.
  - In the Proof you are required to formally show that:

A lemma is needed → $OPT(A')$ follows from $OPT(A)$,
& $OPT(A)$ " " $OPT(A')$.

This is what gives the pseudocode.

- Greedy paradigm is a very powerful technique.
  In the end, you only need sorting.

Binary Coding of files

- Suppose a file F has m letters & there are n alphabets in the language.

- Qn: How large is the binary coding?

- Ans: At least $m \cdot \lg n$.

- Additional assumption: Suppose we know the frequency distribution of the alphabets in F.

Can we use the distribution to have a smaller coding of the file?

- Eg. in English texts 'e' appears 13%, 't' appears 9%, but 'j' 'q' 'x' 'z' appear $< \frac{1}{2}\%$!

- Idea: More frequent alphabets should map to shorter strings.

- Eg. Alphabet A: a b c d e

	a	b	c	d	e
Frequency f:	0.45	0.18	0.15	0.12	0.10
Coding γ:	0	10	110	101	111

- This gives an average bit length ABL of:

$$0.45 \times 1 + 0.18 \times 2 + (0.15 + 0.12 + 0.10) \times 3$$

$$= 1.92 = \sum_{a \in A} f(a) \cdot |\gamma(a)|$$

$\Rightarrow$ A file of size m has an encoding of size $\approx 1.92\,m$, which is smaller than $3m$.

- But, this coding has ambiguity.

01010111 is abbe
adae

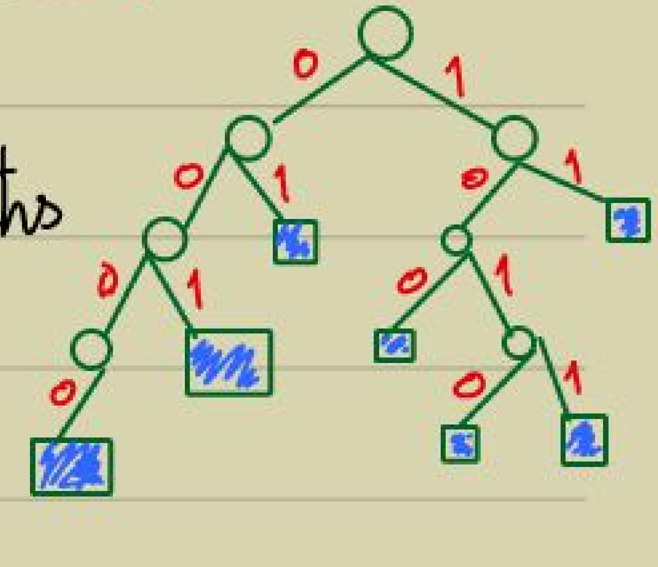
▷ 'b' is a prefix of 'd'.

- Prefix coding: If $\nexists\, x \neq y \in A$ s.t. $\gamma(x)$ is a prefix of $\gamma(y)$.

- Algorithmic problem:
  Given A of n alphabets with their frequencies, compute a prefix encoding γ s.t. $ABL(\gamma)$ is minimum.

- Brute-force: A naive algorithm would go over all the n-subsets of $[2n]$. all ($\leq \lg n$)-bit strings.
  $\Rightarrow 2^{\Omega(n)}$-time taken.

- Instead, we can model a prefix code as a labeled binary tree.



- Eg. the blue leaves' paths form a prefix code!

(Exercise)

Huffman Code – Optimal prefix code

- We make the following observations about the labelled binary tree T of the optimal prefix code γ.

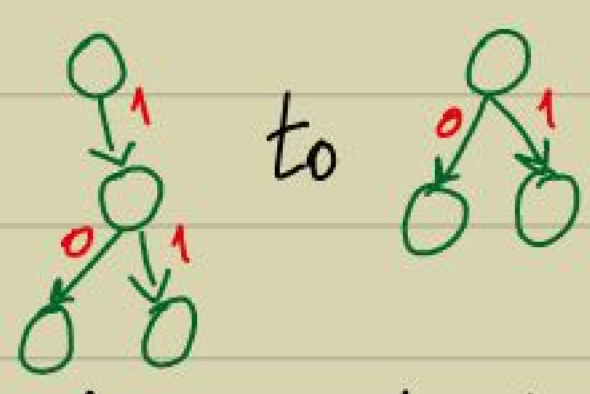
out-deg = 2 per internal node

Lemma 1: T must be a full binary tree.

Proof:

- If there is a node with out-degree ≤ 1 then we can shrink it.

e.g.



$\Rightarrow$ Every node has out-deg $= 2$.

$\Rightarrow$ T is full. $\square$

Lemma 2: More frequent alphabets are close to the root.

Proof:

- If $f(a_1) < f(a_2)$ & a_2 is deeper in T than a_1, then swapping them reduces $\sum_{a \in A} f(a) \cdot |\gamma(a)|$. $\square$

Lemma 3: Let $A = \{a_1, \dots, a_n\}$ & $f(a_1) \le \dots \le f(a_n)$.
There is an optimal T where a_1 & a_2 are siblings in the deepest level.

Proof:

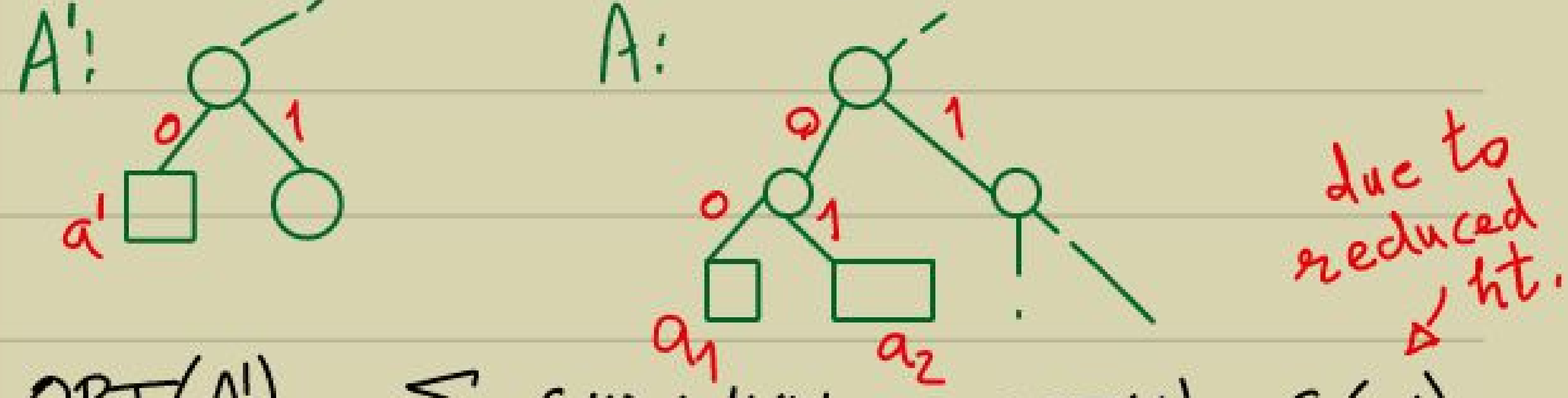
- Suppose b is a sibling of a_1 & $f(a_1) \le f(a_2) \le f(b)$.

$\Rightarrow$ Wlog we can swap b & a_2, getting a_2 in the deepest level with a_1. $\triangleright$

— By Lemma 3 we can modify the instance $A = \{a_1, a_2, \dots, a_n\}$ to $A' := \{a_3, a_4, \dots, a_n\} \cup \{a'\}$ by merging the two alphabets a_1 & a_2 to a'.

$\triangleright$ If we set $f(a') := f(a_1) + f(a_2)$ then $OPT(A')$ will also give us $OPT(A)$.

Pf:

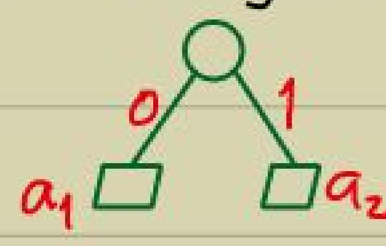


$$OPT(A') = \sum_{b \in A'} f(b) \cdot |\gamma'(b)| = OPT(A) - f(a')$$

minimizing $ABL(\gamma)$. □

- Developing the algorithm to find T:

Store the frequencies in an AVL Tree

  - First, compute T' for alphabet A'.
  - Where is a'? A leaf in T'!
  - Replace it by 

.
  - Return this tree T.

- Time: It takes $O(n \lg n)$ time as we reduce the alphabet size by one.

$T(n) = T(n-1) + O(\lg n)$

Theorem (Huffman '52): Optimal prefix code can be found in $O(n \lg n)$ time.

Note: AVL tree & labelled binary tree have size $O(n \lg n)$.

→ This technique is the mother of all data compression schemes!