Lecture 7

Greedy Algo Paradigm

There exists "an" optimal solution where

→ **Synchronising electric circuit problem:**

In the optimal solution, the delay along any path from u to any leaf node is $\max(D_L(u), D_R(u))$.
↳ max delay in its left subtree.

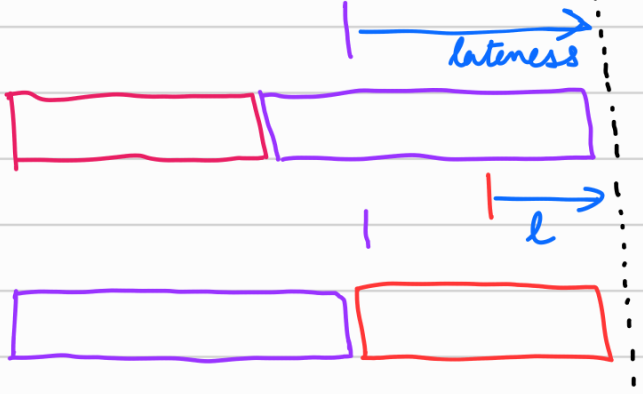
```
Sync (u) {                                   O(n)
  if u is a leaf node → return 0;
   D_L = D(left child) + wt(u → lt)
   D_R = D(right child) + wt(u → rt)
   D = max(D_L, D_R)
  if (D == D_L) {
     wt(u → rt) += D_L - D_R; return D_L;}
  else wt(u → lt) += D_R - D_L; ret D_R;}
```

Job Scheduling

- n jobs : time t_i, deadline d_i
  Lateness of a job : max (fin time − deadline, 0)

→ **Proof of correctness**

Schedule jobs in increasing order of d_i.



$l_1 = t_1 + t_2 - d_2$

$l_2 = t_1 + t_2 - d_1$

$l_2 < l_1$

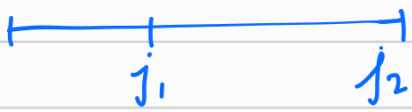
So if we have some permutation of jobs, we can choose any two jobs i and j arranged s.t. j comes after i, and $d_j < d_i$. Swapping them would either decrease the lateness, or keep it the same.

Formal Proof:

There exists an optimal solution in which the earliest deadline job would be done first.

Suppose jobs j_1 and j_2 s.t. $d_{j_1} > d_{j_2}$.

i) $j_1 \rightarrow j_2$



$L_{j_1} = \max(0, t_{j_1} - d_{j_1})$

$L_{j_2} = \max(0, t_{j_1} + t_{j_2} - d_{j_2})$

$d_{j_1} > d_{j_2} \Rightarrow -d_{j_1} < -d_{j_2} \Rightarrow t_{j_1} - d_{j_1} < t_{j_1} - d_{j_2}$

$\Rightarrow t_{j_1} - d_{j_1} < t_{j_1} + t_{j_2} - d_{j_2}$

$\Rightarrow Late_1 = L_{j_2} = \max(t_{j_1} + t_{j_2} - d_{j_2}, 0)$

ii) $j_2 \rightarrow j_1$

$L_{j_2} = \max(0, t_{j_2} - d_{j_2})$

$L_{j_1} = \max(0, t_{j_2} + t_{j_1} - d_{j_1})$

$d_{j_2} < d_{j_1} \Rightarrow -d_{j_2} > -d_{j_1} \Rightarrow t_{j_2} - d_{j_2} > t_{j_2} - d_{j_1}$

$\Rightarrow Late_2 = \max(t_{j_2} - d_{j_2}, t_{j_2} + t_{j_1} - d_{j_1})$

in both cases, max lateness in second is small.

- if second greater, $\because d_{j_2} < d_{j_1} \Rightarrow \Sigma t - d_{j_2} > \Sigma t - d_{j_1}$
- if first, then too $t_{j_1} > 0 \Rightarrow t_{j_1} - d_{j_1}$ is smaller

→ For any 2 consecutive jobs j_1 and j_2, if $d_{j_1} > d_{j_2}$, we can always swap them by lateness not increasing.

→ There exists an optimal solution in which the earliest deadline job would be done first. If not, I can swap it with j_1 & lateness ≤.

$$opt(A) = \min d_j + opt(A')$$

→ Generic Way to Prove Greedy

A: instance of size n of problem P.

↓ Greedy step: Lemma about opt(A)

A': instance of size $n' < n$ of prob P.

i) Establish a relation b/w opt(A) and opt(A')
ii) • derive a soln of A from opt(A') (from construction)
• " " " of A' " opt(A) (using Lemma).

Huffman Coding

alphabet set A: $\{a_1, a_2, \ldots, a_n\}$
m alphabets ⟶ $m \lceil \log_2 n \rceil$ bits

Binary coding: γ: A → binary strings

Fixed length coding: each alphabet gets a unique binary string of length $\lceil \log_2 n \rceil$.
So to decode, we will partition the bits into blocks of equal size = $\lceil \log_2 n \rceil$.

More frequent alphabets : coding with shorter bit string.

ABL : Average bit length per symbol

$$ABL(\gamma) = \sum_{x \in A} f(x) \cdot |\gamma(x)|$$

Prefix Coding

A coding $\gamma(A)$ is called a prefix coding if there does not exist $x, y \in A$ st $\gamma(x)$ is a prefix of $\gamma(y)$.

Prob: γ is a prefix coding on set A of n alphabets, and $ABL(\gamma)$ is minimum.

(Labelled Binary Tree)

nodes ⟶ alphabets

then code of an alphabet = Label of path from root

Theorem:

For each prefix code of a set A of n alphabets, there exists a binary tree T on n leaves st

i) bijective mapping b/w alphabets ⟷ leaf nodes

ii) Label of a path from root to a leaf corresponds to the prefix code of that alphabet.

$$ABL = \sum_{x \in A} f(x) |\gamma(x)| = \sum_{x \in A} f(x) \cdot |depth_T(x)|$$

Lemma: The binary tree corresponding to an optimal prefix coding must be a full binary tree (Every internal node degree = 2)

Should be arranged in "non-decreasing"

order of frequencies. $(f(a_i) \le f(a_{i+1}))$

more freq elements ⇒ closer to root.

Theorem: There exists an optimal prefix coding in which a_1 and a_2 appear as siblings.

$a_1, a_2, \ldots\ldots a_n$ ⟶

Non Decreasing order of freq

If a_1 is not at the deepest leaf, we can surely swap it with the node at the deepest level. Doing so cannot increase the ABL, because $f(a_1) \le f(a_K)$.
Now depth (f_{a_K}) has decreased.

$a_1 \longrightarrow d_1 \qquad a_K \longrightarrow d_K. \qquad d_K > d_1, \; f(a_1) \le f(a_K)$

$ABL_1 = f(a_1) \times d_1 + f(a_K)\, d_K$

Swap $ABL_2 = f(a_1) \times d_K + f(a_K)\, d_1$

$$ABL_2 - ABL_1 = (f(a_1))(d_K - d_1) + (f(a_K))(d_1 - d_K)$$
$$= (d_K - d_1)(f(a_1) - f(a_K))$$
$$ABL_2 = ABL_1 + \underbrace{(d_K - d_1)}_{0+} \underbrace{(f(a_1) - f(a_K))}_{-}$$

Now, for a_2, since it is a full binary tree, we must have a_2 as the sibling of a_1

Now, if we merge a_1 and a_2 to form a_K, with depth one less,

$$(f(a_1)\, d_1 + f(a_2) d_1) + K = \overbrace{(f(a_1) + f(a_2))(d_1 - 1)}^{\text{soln for } n-1 \text{ nodes}} + K + f(a_1) + f(a_2)$$

($f(a')$, depth d')

$$obt(A) = obt(A') + f(a_1) + f(a_2)$$

opt(A) = opt(A') + f(a1) + f(a2)

- **opt(A) from opt(A') using construction**

$a' \rightarrow a_1 + a_2$ split

$opt(A) \leq opt(A') + f(a_1) + f(a_2)$

- **opt(A') from opt(A) using lemma:**

$a \rightarrow$ remove a_1 & a_2 and add a'

$opt(A') \leq opt(A) - f(a_1) - f(a_2)$

$\Rightarrow opt(A) \geq opt(A') + f(a_1) + f(a_2)$

Pseudo code: $O(n^2)$

```
opt (A) {
  if |A| = 2 return {a1: 0, a2: 1};
  let a1, a2 → symbols with least freq.
  remove a1 and a2 from A.
  Create a new symbol a'.
    f(a') = f(a1) + f(a2).
  Insert a' into A;
  T ← opt (A)
  replace a' by {a1: 0, a2: 1}
    a1 and a2 depth = 1 + depth of a'.
  return T;
}
```

$O(n \log n)$ → use min heap for frequencies