Greedy Algorithms (Contd).

Obs:

- Elements added to the solution set are not touched again till the end of the algorithm.
- Not every greedy strategy is optimal.

This makes proof of correctness vital!

Covered so far:

- Shortest paths
- Minimum Spanning tree
- Huffman coding

today
- Interval Scheduling
- Fractional knapsack.

Fractional Knapsack:

A store in a promotional offer allows you to pick as much as you want* of items $I_1, I_2, \ldots, I_m$ (which have an availability of $w_1, w_2, \ldots, w_m$ kgs each with values $v_1, v_2, \ldots, v_m$.

* Provided they fit into a bag of total capacity W.

Goal: Maximize the value of contents in your bag.

$$\begin{cases} \max: \sum \hat{w}_i \cdot v_i \\ \text{subj} \quad \hat{w}_i \in [0, w_i] \quad \forall i \in [m] \\ \qquad \sum \hat{w}_i \leq W \end{cases}$$

Strategy: Sort by value

wt	50	~~20~~ 5	10
value	1000	5000	7000

↳ Sort it by per kg value.

$\boxed{50}$

7000 10
5000 20
1000 20

$$\frac{(7000 + 10000 + 2000)}{10}$$

Interval Scheduling

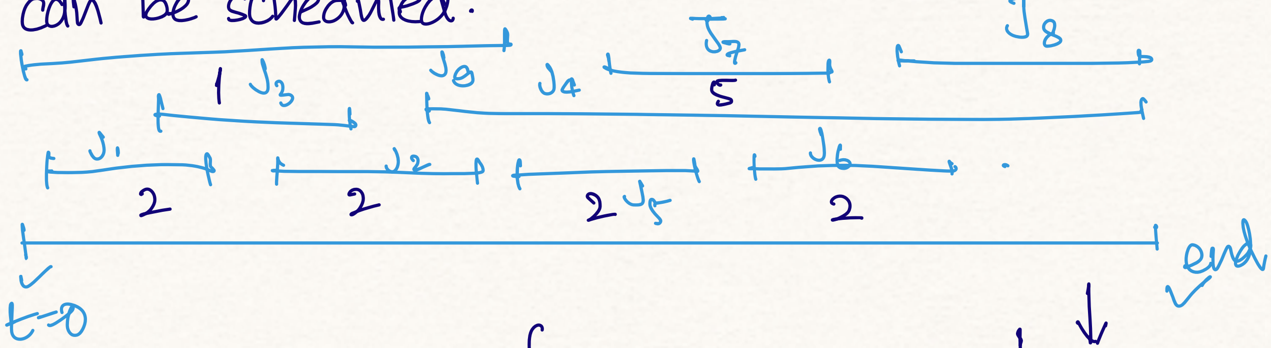
$J_1 \quad s(1) \quad f(1)$

$J_2 \quad s(2) \quad f(2)$

Single processor with a set of jobs $J_1, J_2, \ldots, J_m$ (given with start and finish times).

Goal is to find the size of a maximal subset of jobs that can be scheduled.

Subset of non-overlapping jobs.



→ Start by J_1, check for overlapping jobs

↳ Pick based on end time?

Shortest interval first does not work.

Start first also does not work

$J \leftarrow (J_1, \ldots, J_m)$

↳ While J is not empty and processor is free:

Pick a job from J with earliest finish time

Schedule that job and wait for the job to finish

Remove all overlapping jobs from J.

Question: Why is this optimal?

$A = \{I_1, \ldots, I_k\}$ ← Output of the algo

Suppose $\exists$ a subset $O = \{I_1', \ldots, I_\ell'\}$ s.t $\ell > k$.
(For the sake of contradiction).

$A = \{I_1, \ldots, I_k\}$
$O = \{I_1', \ldots, I_k', \ldots\}$

<u>Claim</u>: For all $r \in [1, k]$, $f(I_r) \leq f(I_r')$

finish time.

Claim $\Rightarrow$ Optimality: If I_k finishes before I_k' and algorithm terminates

- $I_{k+1}', \ldots, I_\ell'$ do not overlap with $I_1, \ldots, I_k$ thus can be added to A itself.
- Algo terminates only when there are no more non-overlapping jobs left to be scheduled.

<u>r=1</u>: $f(I_1) \leq f(I_1')$
because $f(I_1) \leq \{f(I_j) \mid \forall j\}$

For some $r = k-1$, by induction let us assume that the statement holds.

<u>W.T.P</u>: $f(I_k) \leq f(I_k')$

I.H: $f(I_{k-1}) \leq f(I_{k-1}')$.

$s(I_k') \geq f(I_{k-1}') \geq f(I_{k-1})$ ✓

<u>Obs</u>: I_k' is also non-overlapping with I_{k-1}

Algo picks I_k over I_k' only if $f(I_k) \leq f(I_k')$.

Algo "stays ahead" of O.

Qn: Scheduling theory $\rightarrow$ Maximum lateness

$J_1, J_2, \ldots, J_m$, $\rightarrow$ $P_1, \ldots, P_m$

Jobs $\qquad w_1 \qquad w_m$

Minimize

$$\sum_{i=1}^{m} w_{\sigma(i)} \left(\sum_{j=1}^{\sigma(i)} P_j \right)$$

{ Weighted Completion time

↳ Want a sequence that optimizes weighted completion time.

wt → 5 6
time taken → 10 12

1 2 → $5(10) + 6(10+12)$

2 1 → $6(12) + 5(10+12)$

$w_1 \quad w_2$
$P_1 \quad P_2$

1 2 → $w_1 P_1 + w_2 P_2 + w_2 P_1$

$w_2 P_2 + w_1 P_2 + w_1 P_1$

1,2 < 2,1

if $w_2 P_1 < w_1 P_2$ → $\frac{w_2}{P_2} < \frac{w_1}{P_1}$

$\underline{1 \quad 2 \quad 3}$
k

Strategy
Choose the job that maximizes $\frac{w}{P}$ ratio.