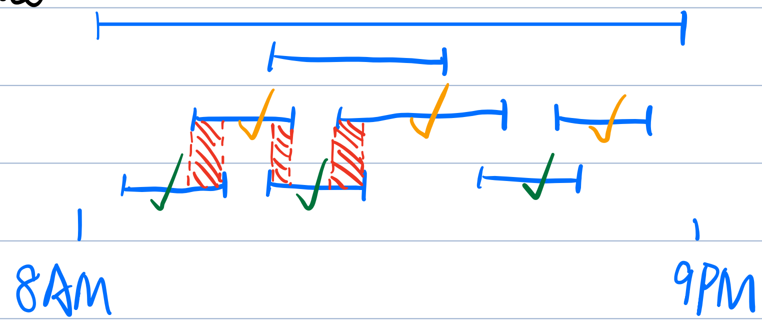
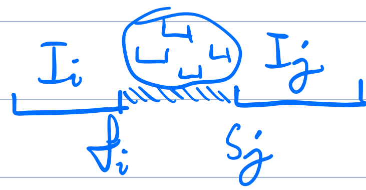
1. Activity Selection Problem / Interval Scheduling

Courant is running low on # classrooms. So we want to maximize the usage of each classroom. Say, for a given classroom, here's a list of courses we can schedule:



Input: List of Intervals $S = \{I_1, \dots, I_n\}$
where $I_i = [s_i, f_i)$

Goal: Find $S' \subseteq S$ of non-intersecting intervals of largest size $|S'|$.



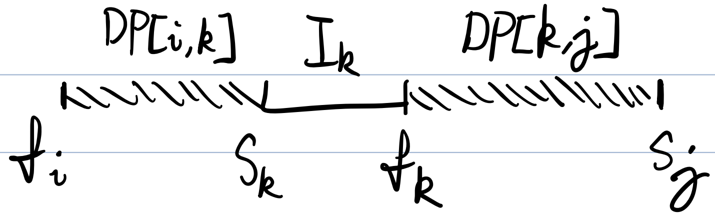
DP:

① **Subproblems**: $DP[i, j]$ is the optimal solution for intervals starting after f_i and ending before s_j. Imagine $f_0 = -\infty$ and $s_{n+1} = \infty$.

② **Guess**: which activity is selected

③ Recurrence: We can only select I_k if $[S_k, f_k) \subseteq [f_i, s_j)$, and if I_k is selected, the solution will be

$$DP[i,k] + DP[k,j] + 1$$



We want to take the max of all possible k's. So the recurrence is

$$DP[i,j] = \max_{\substack{k \in \{1,\dots,n\} \\ \text{s.t. } [S_k, f_k) \subseteq [f_i, S_j)}} \left(DP[i,k] + DP[k,j] + 1 \right)$$

④ Runtime (memo/bottom-up):

\# subproblems $= O(n^2)$

time/subproblem $= O(n)$

$\Rightarrow O(n^3)$ runtime

⑤ ☺

Improved DP: Sort activities by finish time, and then decide whether to include the last interval

① <u>Subproblem</u>: DP[i] = The optimal solution for intervals $\{I_1, \dots, I_i\}$

② <u>Guess</u>: whether we take interval I_i

③ <u>Recurrence</u>:

$$DP[i] = \max\left(DP[i-1],\ 1 + DP[k_i]\right)$$

k_i is the last interval that finishes before S_i

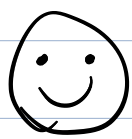
④ Memo/bottom-up

\# subproblems $= O(n)$

time/subproblem $= O(\log n)$ (need to compute k_i)

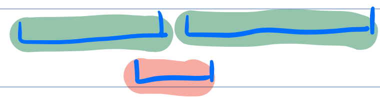
$\Rightarrow$ DP Runtime $= O(n \log n)$

(<u>Plus sorting at the beginning!</u>)

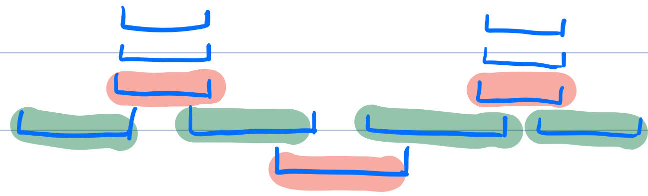
⑤ 

Greedy?

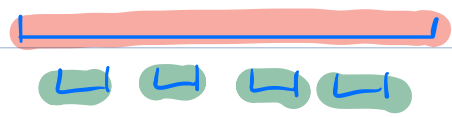
- Pick shortest interval? X



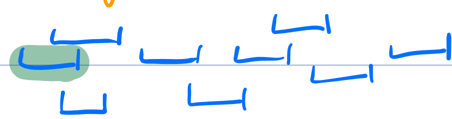
- Pick activity that intersects the least? X
  (kinda, if intersects with 0, then sure)



- Pick activity w/ earliest starting time? X



- Pick activity w/ earliest finish time!!!

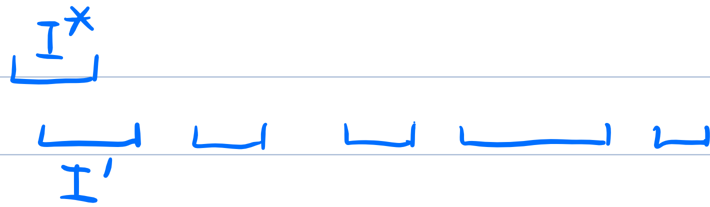


<u>Thm</u>: For any list of intervals S, exists an optimal solution that includes the activity that ends first, I^*.

Proof: Take <u>any</u> optimal solution for S.

① if it includes I^*, we're done.
② if it does not contain I^*, let I' be the first interval in the solution.

Solution:



Notice that I^* ends before I', so I^* won't intersect with other intervals in the solution. So if we replace I' with I^*, then we get another valid solution that is optimal, <u>AND it includes I^*</u>.

\#

Pseudo-Code (Assume input sorted by finish time)

```
curr_fin = -∞
For i=1 to n:
    If  S_i ≥ curr_fin:
        Print i
        curr_fin = f_i
```

Runtime $= O(n)$
$(O(n\log n)$ if we count in sorting.)