CS 3100: Algorithms

Greedy Algorithms

Announcements

- Starting ~~Ch~~ Lec. Notes 7 today
- Don't forget those problem session worksheets!
- Oral HW on Friday

Dynamic Programming vs Greedy

Dyn. pro: try all possibilities
↳ but intelligently!

In greedy algorithms, we avoid building all possibilities.

How?
- Some part of the problem's structure lets us pick a local "best" and have it lead to a global best.

But - be careful!

Most common mistake:

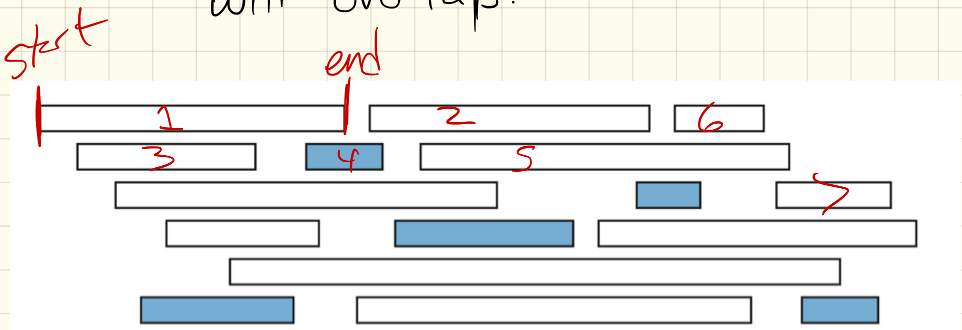
Students often design a greedy strategy, but don't check that it yields the best global one.

Example:

HW question 3 (a+b)

Problem: Interval Scheduling

Given a set of events (intervals with a start + end time), select as many as possible so that no 2 chosen will overlap.



A maximal conflict-free schedule for a set of classes.

More formally

Input: Two arrays $S[1..n]$ & $F[1..n]$

where interval i starts at $S[i]$ & ends at $F[i]$

Output: a set $I = \{i_1, i_2, \ldots, i_k\}$

with $F[i] \leq S[i+1]$

$\forall i \in I$

maximizing k

How would we formalize a dynamic programming approach?

Recursive structure:

Consider interval 1:

in or out

↓ remove any overlapping intervals + then recurse

↓ recurse on intervals 2..n

Remove Any j s.t.

$S[i] \le S[j] \le F[i]$

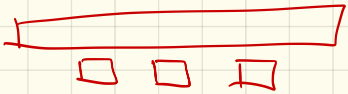
or $S[i] \le F[j] \le F[i]$

Intuition for greedy:

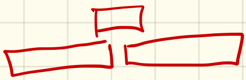
Consider what might be a good first one to choose.

Ideas?

X - earliest start time

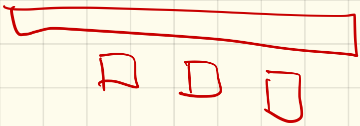


X - shortest interval



- latest end time

X



- take smallest end time

Key intuition:

If it finishes as early as possible, we can fit more things in!

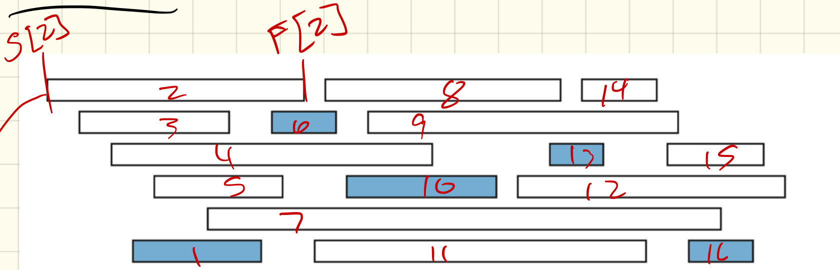
So - strategy:

Sort by finish time.

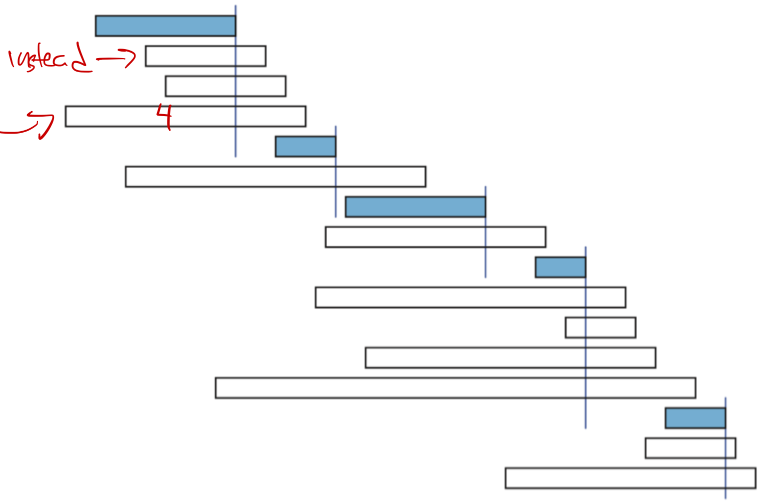
Select the first interval.

Remove any that overlap. / + continue

Picture:



A maximal conflict-free schedule for a set of classes.



The same classes sorted by finish times and the greedy schedule.

Pseudocode

```
GREEDYSCHEDULE(S[1..n], F[1..n]):
    sort F and permute S to match
    count ← 1
    X[count] ← 1
    for i ← 2 to n        //go in order of finish time
        if S[i] > F[X[count]]
            count ← count + 1
            X[count] ← i
    return X[1..count]
```

$O(n \log n)$ — sort F

$O(n)$ — loop

Runtime:

~~$O(n)$~~

$O(n \log n)$

Correctnes:

Why does this work?

Note: No longer trying all possibilities or relying on optimal substructure!

So we need to be very careful on our proofs

(Clearly, intuition can be wrong!)

<u>Lemma</u>: We may assume the optimal schedule includes the class that finishes first.

<u>pf</u>: by contradiction

then opt is some intervals:

$\langle O_1, O_2, O_3 \cdots, O_k \rangle$

(sort so O_1 finishes before O_2 starts, + so on)

$\Rightarrow F[O_i] < S[O_{i+1}]$

Consider g, the interval that finished first:

$F[g] < F[O_1]$

this means $F[g] < S[O_i]$ $\forall i \geq 2$

so also optimal is:

$\langle g, O_2, \ldots, O_k \rangle$

∎

Thm: The greedy schedule is an optimal.

Pf: Suppose not.

Then ~~∃~~ all ~~an~~ optimal schedule that has more intervals than the greedy one. say $k > \ell$

Consider first time they differ:

greedy:

$\langle g_1, g_2, g_3, \dots, g_\ell \rangle$

optimal:

$\langle g_1, g_2, \dots, g_i, o_{i+1}, \dots, o_k \rangle$

(same up to i, & then not)

(i exists & is ≥ 1, ~~by~~ by lemma)

Know: $F[o_{i+1}] > F[g_{i+1}]$ since greedy

also, $S[o_{i+2}] > F[o_{i+1}]$ since O is opt schedule.

pf cont

SO: we can replace o_{i+1} with g_{i+1} & still be valid:

OPT was: $\langle g_1, \dots, g_i, \underline{o_{i+1}}, \dots, o_k \rangle$

$\hookrightarrow \langle g_1, g_2, \dots, g_{i+1}, o_{i+2}, \dots, o_k \rangle$ is still valid.

contradiction ↯

∎

Overall greedy strategy:

- Assume optimal is different than greedy
- Find the "first" place they differ.
- Argue that we can exchange the two without making optimal worse.

⟹ there is no "first place" where they must differ, so greedy in fact is an optimal solution.

Another example in notes: storing the most files on a tape

Intuition: (check notes)