CS314 - Greedy Algorithms

- Oral grading - done later today, posted back Monday
- Next HW - up later today
Dynamic Programming versus Greedy

- Try all possibilities, but intelligently.

- With greedy algorithms, we can avoid trying all possibilities.

How?

- Some part of structure lets us pick a local “best” and have it lead to global “best”.

Sound suspicious?
Greedy is dangerous!

- Often students design a “greedy” strategy, but it doesn’t yield the best solution.

Example: HW2, Q3

The key: Proof of correctness
**Interval Scheduling**

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

**Goal:** Select as many intervals as possible so that no two overlap.
Notation:

Input: Two arrays \( S[1..n] \) and \( F[1..n] \)
- Start times
- Finish times

Output: Largest subset \( X \subseteq \{1..n\} \)
- \( F[i] < S[j] \) or \( S[i] > F[j] \)
Dynamic Programming

Recursive Structure?

- i in X
- i not in X

remove all overlapping
+ recurse

recurse without i

Run time: exponential?
Intuition for greedy strategy
Consider first class we'd pick.
What would be a good choice?
- earliest end time
- latest start time
- smallest interval first
Idea: If it finishes as early as possible, that's good!

So strategy:

- Sort based on end times
Picture:
(same schedule, but sorted by F[i])
**Pseudocode**

```pseudocode
GREEDYSCHEDULE(\(S[1..n], F[1..n]\)):
    sort \(F\) and permute \(S\) to match
    \(count \leftarrow 1\)
    \(X[count] \leftarrow 1\)
    for \(i \leftarrow 2\) to \(n\)
        if \(S[i] > F[X[count]]\)
            \(count \leftarrow count + 1\)
            \(X[count] \leftarrow i\)
    return \(X[1..count]\)
```

**Run time?**

\(O(n \log n)\)
Correctness:

But why does this work?
(Note: No longer trying all possibilities!)

So we need to be very careful on our proofs for greedy strategies.

Our intuition from before is the start of our proof...
Lemma: Can assume the schedule contains the class that finishes first.

Proof: Suppose not: \[<o_1, o_2, o_3, \ldots, o_k>\] (sorted in order)
\[<g_1, g_2, \ldots, g_k>\]

\[F[g_1] \leq F[o_1]\] since \(g_1\) was first thing to finish
also, \[S[o_2] > F[o_1]\]

\[\Rightarrow F[g_1] < S[o_2]\]

Since \(g_1\) finishes before 2nd element in \textit{optimal}, \[<g_1, o_2, \ldots, o_k>\] is also valid.
Thm: The greedy schedule is optimal.

Proof: Suppose not. So there's an optimal schedule with more intervals in it.

Look at first time they are different:

Greedy schedule: \(<g_1, g_2, \ldots, g_e>\)

Optimal schedule: \(<g_1, g_2, \ldots, g_i, o_{i+1}, \ldots, o_k>\)

Know: \(F[o_{i+1}] \geq F[g_{i+1}]\)

Also: \(S[o_{i+2}] > F[o_{i+1}]\) since it is a valid schedule
pf: (cont) So replace $o_{ir1}$ with $g_{ir1}$

OPT: $<g_{i1}, g_{i2}, \ldots, g_{ir1}, g_{i1}, g_{i2}, \ldots, o_k>$

This works for any $i$!
Strategy for greedy proofs:

1. Assume optimal solution is different from the greedy solution.
2. Find the "first" place they differ.
3. Argue that we can exchange the two without making optimal any worse

⇒ there is no "first place" they differ, so greedy is optimal.