CSCI 270 Lecture 11: Interval Scheduling

We are given n tasks, each with a start time s_i, and finish time f_i. Find the largest subset of tasks such that none of them overlap.

- Propose a greedy criteria. How should we decide which interval to include next?
- Can you find counter-examples to any of these criteria?
- Once you've narrowed down your list of candidate algorithms to one, what should you do?

Greedy algorithms are easy to design, easy to write, and very efficient. The tradeoff is that they also tend to be unclear. **Why** does the algorithm work? How can you convince someone that the algorithm is correct?

Greedy algorithms often require a proof of correctness. It is not clear at all that our proposed algorithm for Interval Scheduling is actually correct.

Proving the correctness of an entire algorithm is rather overwhelming, so it is useful to break it down. Just as Greedy Algorithms tackle each bite-size decision sequentially, we will prove the correctness of each of these decisions sequentially, via an inductive proof.

Inductive Hypothesis: There is an optimal solution which includes the first k intervals from our solution.

The base case is almost always trivial. Specifically, our base case is $k = 0$.

Now we need to show there is an optimal solution which includes the first $k + 1$ intervals from our solution.

What we know:

- There is some interval i which is the $k + 1$st choice in our solution.
- There is at least one (possibly multiple) optimal solutions which include the first k choices we did.
- We need to show at least one of those optimal solutions also include i.

Take an arbitrary optimal solution OPT which includes the first k intervals from our solution. Presumably OPT does not include interval i (otherwise we're done). However, of all the intervals it does include (other than the first k), there must be one with smallest finish time. We'll call this interval j.

- Who do f_i and f_j compare?
- How should I transform OPT?
- What is the runtime of our algorithm?

Proof template:

- There is an optimal solution which includes the first 0 intervals from our solution (trivial base case).
- Assume there is an optimal solution which includes the first k intervals from our solution, and call it OPT.
- Exchange some element of OPT with your algorithm's $(k+1)$st choice, to produce a new solution OPT'. Figuring out which element to exchange is a large part of the challenge.
- Prove that OPT' is still valid; that is, our exchange did not somehow break the rules.
- Prove that OPT' is still optimal; that is, the value of its solution is just as good as OPT.
- You have found an optimal solution OPT' which includes the first $(k+1)$ intervals from our solution!
- Therefore there is an optimal solution identical to ours! Proved by Induction.

Minimum Spanning Trees

We want to prove that Kruskal's Algorithm is correct, which adds edges from smallest to largest unless adding an edge creates a cycle. We will call the solution produced by Kruskal's Algorithm 'US'.

Base Case: There is an optimal solution which uses the first 0 edges added by US.

Inductive Hypothesis: There is an optimal solution OPT which uses the first k edges added by US.

Suppose US adds edge e_{k+1} next, but OPT does not include this edge.

- Suppose we add edge e_{k+1} to OPT. What happens?

- There is some edge e_f in the cycle C which is not in US (otherwise we could not have added e_{k+1}).
- Exchange e_f with e_{k+1} in OPT to create OPT'.
- Prove that OPT' is valid, that is, it is a spanning tree.
- Prove that OPT' is optimal, that is $c(e_f) \geq c(e_{k+1})$.
  - Assume (by way of contradiction) that $c(e_{k+1}) > c(e_f)$.
  - Consider the subgraph of OPT consisting of all edges with cost $\leq c(e_{k+1})$, called OPT_C
  - $\{e_1, e_2, ..., e_k\} \cup e_f \subseteq OPT_C$.
  - $\{e_1, e_2, ..., e_k\} \cup e_f$ does not contain any cycles, so US would have added e_f before e_{k+1}.
  - Contradiction: $c(e_f) \geq c(e_{k+1})$. Therefore OPT' is optimal.