Dynamic programming: Weighted interval scheduling

Weighted interval scheduling is another classic DP problem. It is the more general version of a problem we'll see next time (activity selection, CLRS 16), and knowing this more general version is helpful. It is not in the textbook.

The problem: You are given a set of jobs: each job has a start time, an end time, and has a certain value or weight. The weight of a job measures its importance—higher weight if more important (if all jobs were "equal" they would have all equal weights). Furthermore, imagine we have a single classroom where we can schedule these jobs. We want to pick a set of jobs that we can assign to the classroom such that:

- First, this set of jobs has to be *compatible* (i.e. non-overlapping), so that they can be scheduled in the same classroom. Overlapping at an endpoint is considered acceptable.
- Second, we want to weigh in the weights: Of all possible choices, we want to find a subset of jobs (to schedule in the classroom) of maximum total weight.

Here is the more abstract version of this problem: Given a set of intervals $I_1, I_2, ..., I_n$ with start times $s[1..n]$, finish times $f[1..n]$, and values $v[1..n]$, find a subset of compatible (non-overlapping) intervals of maximal total value:

$$\text{find } S \subseteq I \text{ such that } S \text{ compatible and} \sum_{i \in S} v_i \text{ is maximized}$$

- **Optimal substructure:** How would you argue that the problem has optimal sub-structure?

Simplify: As always, for simplicity, we'll start by computing *only* the maximal value of the subset. Once we know how to do this, we'll extend the solution to compute not only the value, but the actual subset of jobs that achieve this value.

- **Recursive formulation: IN or OUT.** Consider a subset of activities S that represents an optimal solution. An interval I_k is either in S, or not. When we come to deciding whether an activity is part of the solution or not, we have precisely two choices. Now let's try to come up with a recursive formulation that uses this insight.

- **Notation I:** The trick is coming up with a recursive formulation whose parameter is an integer, not a set. We'll use the following notation:

  Let us assume that the jobs are ordered by non-decreasing finish time f_i:

  $$f_1 \leq f_2 \leq f_3 .... \leq f_n$$

  Define $p(j)$ to be the job with the largest index less than j which is compatible with job j. In other words, the largest index k such that $f_k \leq s_j$. If no such job exists then define $p(j) = 0$.

  Note that none of the intervals $I_1, I_2, ...I_{p(j)}$ intersects with I_j.

Draw a set of 8 activities, number them so that they are orderd by finish time, and show $p(j)$ for all activities.

- Show how to compute all $p(i)$, $i = 1..n$ in $O(n^2)$ time; what about in $O(n \lg n)$ time? Remember that we assume the activitoes are numbered in order of their finish times; you may also assume that you have a list of the activities sorted by their start times.

- **Notation II:** Let $S(i)$ denote the maximum weight of any set of compatible jobs, all of which finish by f_i.

  We want to compute $S(n)$.

  Assume we have computed $p(i)$, $i = 1..n$.

  Give the recursive formulation for $S(i)$, including the basecase:

- Write pseudocode to compute $S(i)$.

- **Analysis:** Let $T(n)$ be the worst-case running time of computing $S(n)$. Write a recurrence relation for $T(n)$.

- Argue that $T(n)$ is exponential.

- Describe a dynamic programming solution and analyze its running time.