Lab: Weighted Interval Scheduling

Recall Interval scheduling

Given a list of tasks, each with a time interval during which it would be completed. For instance, one task might run from 2:00 to 5:00 and another task might run from 6:00 to 8:00. The goal is to maximize the number of executed tasks without overlapping any of the tasks.

Thus a request corresponds to an interval of time. We say that a subset of requests is **compatible** if no two of them overlap in time and our goal is to accept as large a **compatible** subset as possible. A **compatible** set of maximum size is called optimal.

There are several solutions for this problem that are not optimal. For example, selecting the intervals that start earliest is not an optimal solution because if the earliest interval happens to be really long by accepting it we would have to reject many other shorter requests. Selecting the shortest intervals or selecting intervals with the fewest conflicts are also not optimal.

Greedy solution to Interval Scheduling:

Sort the requests by their finish time (earliest to latest)

Iterate through the sorted list of tasks until it is empty:

Choose the task with the earliest finishing time

Skip (in the sorted list) any tasks that start before that chosen tasks finish time

Weighted Interval Scheduling is a generalization of interval scheduling where a value is assigned to each task and the goal is to maximize the total value. Develop a Dynamic Programming algorithm to solve the Weighted Interval Scheduling problem.

Submit by 9:00 p.m. on Friday Nov 6 on PolyLearn. A pdf called *WIS analysis.pdf* that contains

1. The **recurrence relation** that describes the solution to the Weighted Interval Scheduling problem of a given size in terms of smaller problem instances. Give a definition of each component of the recurrence relation.
2. The **specification of the table** that you would use in a bottom up programmatic solution. That is, what do its entries represent.
3. The **specification of the algorithm** for constructing the table that you would use in a bottom up programmatic solution.
4. The derivation of the closed form solution to the recurrence relation using back substitution.

Submit a Java class as specified below by 9:00 p.m. on Friday Nov 6 using the handin command:

"handin gradertk cpe349WIS *WgtIntScheduler.java*"

The class WgtIntScheduler.java should contain a public method
int[] optSet **getOptSet** (int[] stime, int[] ftime, int[] weight)

- The input represents the start times, finish times, and weights of jobs 0 .. n-1.
- **getOptSet returns an array of** the jobs that make up the optimal set in order of their original indices..

Sample input might look like:

{4, 3, 2, 7, 4, 10} // start times

{7, 10, 6, 9, 8, 13} //finish times

{6, 6, 5, 8, 4, 2} // weights

Output: {