COMP3121 Activity Selection Problem

Ian Park

Problem. Given a list of activities, $a_i, 1 \leq i \leq n$ with starting times s_i and finishing times f_i, find a subset of non-overlapping activities of maximal total duration.

Solution.

1. **Setup/Preprocessing.**
   First, sort the activities by their finishing time into a non-decreasing sequence; so, we will assume that $f_1 \leq f_2 \leq \cdots \leq f_n$.

2. **Subproblem.**
   For every $1 \leq i \leq n$, we define the subproblem $P(i)$ to be "find a subsequence σ_i of first i activities such that (1) σ_i consists of non-overlapping activities, (2) σ_i ends with activity a_i, and (3) σ_i is of maximal total duration among all such subsequences."

   Let $T(i)$ be the total duration of the optimal solution to the subproblem $P(i)$.

3. **Build-up order.**
   Solve the subproblems in the order $P(1), \ldots, P(n)$.

4. **Recursion.**
   For all $i > 1$, we have

$$T(i) = \max\{T(j) + f_i - s_i : j < i \text{ and } f_j < s_i\}$$

5. **Base case.**

$$T(1) = f_1 - s_1$$

6. **Final solution.**

$$T_{opt} = \max\{T(i) : 1 \leq i \leq n\}$$

7. **Time complexity.**
   We sort the activities by their finishing times in time $O(n \log n)$ using merge sort. Then, for each $P(i)$, we look at the activity a_i and check its compatibility with each of its preceding activities, which can be done in $O(n)$ time. There are n subproblems hence the overall time complexity of the algorithm is $O(n \log n) + n \times O(n) = O(n^2)$.

Notes:

1. Setup/Preprocessing step is where you might define some additional notations (e.g., "let $S_i = \langle a_1, \ldots, a_i \rangle$") for convenience, or perform some preprocessing tasks (e.g., sorting), but this step may not be necessary for some questions.

2. The order of base cases and recursive cases does not really matter.

3. Build-up order can also come after the cases if it is difficult to figure out before writing down your recursion (although, you really should have the recursion written down already in your sketch working out)

4. As for the "build-up order" part, you should not just state "use a bottom-up approach" or "use a top-down approach" - this is not what we are asking for. Specifically state in which order the subproblems may be solved such that the solution to each subproblem depends only on the solutions to the preceding subproblems, and not on those that come after in the order specified.

5. In some cases, the final solution may be require no further work, for instance, in the knapsack problem, it is given by $opt(C)$ (using lecture notes notations). In such a case, still explicitly state the final solution, e.g., "The final solution is given by $opt(C)$"

Basically the same answer, but with some minor modifications
First, sort the activities by their finishing time into a non-decreasing sequence; so, we will assume that $f_1 \leq f_2 \leq \cdots \leq f_n$. For every $1 \leq i \leq n$, let subproblem $P(i)$ be "find a subsequence σ_i of first i activities such that (1) σ_i consists of non-overlapping activities, (2) σ_i ends with activity a_i, and (3) σ_i is of maximal total duration among all such subsequences."

Now, solve the subproblems in the order $P(1), \ldots, P(n)$. If we let $T(i)$ be the total duration of the optimal solution to the subproblem $P(i)$, then we have

$$\begin{aligned} T(1) &= f_1 - s_1 \\ T(i) &= \max\{T(j) + f_i - s_i : j < i \text{ and } f_j < s_i\} \qquad \text{for } i > 1 \end{aligned}$$

The final solution is given by: $T_{opt} = \max\{T(i) : 1 \leq i \leq n\}$

Time complexity. We sort the activities by their finishing times in time $O(n \log n)$. Then, for each $P(i)$, we look at the activity a_i and check its compatibility with each of its preceding activities, which can be done in $O(n)$ time. There are n subproblems and thus the overall time complexity of the algorithm is $O(n \log n) + n \times O(n) = O(n^2)$.