Fundamental Algorithms, Assignment 8

Solutions

1. Set $W = \lfloor\sqrt{N}\rfloor$. We are *given* $PRICE[I]$, $1 \leq I \leq W$, the price of a rod of length I. Give a program that will output the optimal revenue for a rod of length N^2 and give the time, in Θ-land, of the algorithm. Use an auxilliary array $R[J]$, $0 \leq J \leq N^2$. You may **not** use the term `MAX` is your program Explain, in clear words, how your program is working. (You can use `MAX` in your explanations.) Use an auxilliary array $R[J]$, $0 \leq J \leq N^2$.
   `Solution:`The idea is that
   $$R[J] = \max[PRICE[I] + R[J - I]$$
   where, critically, I ranges over $1 \leq I \leq \min[I, W]$.
   $R[0] = 0; R[1] = PRICE[1]$ (* initialization *)
   FOR $J = 2$ to N^2 (*here we calculate $R[J]$ given previous values*)
   $S = J$; IF $W \leq S$ THEN $S = W$ (*so $S = \min(J, W)$)
   $TEMP = 0$ (*initializing to find max*)
   FOR $I = 1$ TO S; IF $PRICE[I] + R[J - I] \geq TEMP$ THEN $TEMP \leftarrow PRICE[I] + R[J - I]$; ENDFOR (* TEMP becomes the maximal value *)
   $R[J] \leftarrow TEMP$
   ENDFOR (* for J *)
   RETURN $R[N^2]$.

   The outer loops goes $1 \leq J \leq N^2$. So for the time we need to *add* the times for the inner loop over J. The time for the inner loop is basically $\min(J, W)$. There are two ranges. While $1 \leq J < W$ this is J steps and so it adds to $1+2+\ldots+(W-1) \sim W^2/2 = \Theta(N)$. While $W \leq J \leq N^2$ this is W steps so it adds to $W(N^2 - W) = \Theta(N^{2.5})$. The total is the sum which is dominated by the second range so the time is $\Theta(N^{2.5})$.

2. Suppose, in the Activity Selector problem, we instead select the last activity to start that is compatible with all previously selected activities. Describe how this approach works, write a program for it (psuedocode allowed) and prove that it yields an optimal algorithm.
   `Solution:`This approach is symmetric to the one presented in the textbook. It is a greedy solution in that at each point, we're selecting the last activity to start, and recursing down to the single subproblem of finding the optimal solution for all remaining activities compatible

with the ones already scheduled. We can give the recursive algorithm as follows:

RECURSIVE-ACTIVITY-SELECTOR(s, f, i)
$m \leftarrow i - 1$
while $m > 0$ and $s_i < f_m$
 do $m \leftarrow m - 1$
if $m > 0$
 then return $\{a_m\} \cup$ RECURSIVE-ACTIVITY-SELECTOR (s, f, m)
 else return $\emptyset$

We initially run this algorithm with $i = n$ where n is the number of tasks. This is a greedy algorithm, in that we're decomposing the problem recursively into a single optimal subproblem. We can rigorously prove that this gives us an optimal solution by induction on the number of activities. But it is easier to note that a dynamic programming solution to this problem would yield an optimal result, and, as we saw in the book, there are two key observations we can use on the general recurrence of this problem that show that the greedy solution is equivalent to the dynamic programming solution. The recurrence for this solution is given in the book. We note the following two observations (and prove them):

Consider any nonempty subproblem S_{ij}, and let a_m be the activity in S_{ij} with with the latest start time:

$$s_m = \max\{s_k : a_k \in S_{ij}\}$$

1 Activity a_m is used in some maximum-size subset of mutually compatible activities of S_{ij}

[Pf] Suppose that A_{ij} is a maximum-size subset of mutually compatible activities of S_{ij}. We also suppose that the activities in A_{ij} are ordered in monotonically increasing order of starting time. Let a_k be the last activity in A_{ij}. If $a_k = a_m$, we're done, since we've shown that a_m is used in constructing the schedule. Otherwise, we construct the subset $A\prime_{ij} = A_{ij} - \{a_k\} \cup \{a_m\}$. We know the activities in the subset are disjoint, since a_k is the last activity to start, and $s_m \geq s_k$. The number of activities in the subset are the same, so it is a maximum-size subset of activities that includes a_m.

2 The subproblem S_{mj} is empty, so that choosing a_m leaves the subproblem S_{im} as the only one that may be nonempty

[Pf] Suppose that S_{mj} is nonempty, so there is some activity a_k such that $f_m \leq s_k < f_k \leq s_j < f_j$. Then, a_k is also in S_{ij}, which has a later start time than a_m, contradicting our initial assumption.

3. Students (professors too!) often come up with very clever ideas for optimization programs. The problem (often!) is that they (sometimes, but that is enough) give the wrong answer. Here are three approaches and *your* problem, in each case, is to give an example where it yields the wrong answer.

   (a) Pick the activity of the shortest duration from amongst those which do not overlap previously selected activities.

   (b) Pick the activity which overlaps the fewest other remaining activities from amongst those which do not overlap previously selected activities.

   (c) Pick the activity with the earliest start time from amongst those which do not overlap previously selected activities.

   `Solution:` One example to show that the approach of selecting the activity of least duration does not yield an optimal solution is the set of tasks $a_1 = (5, 7)$, $a_2 = (1, 6)$, $a_3 = (6, 10)$. a_1 is selected first, but this locks out the other two, which clearly comprise the optimal solution.
   An example to show that the strategy of choosing the activities that overlap the fewest other remaining activities is the following set of tasks: $a_1 = (0, 1)$, $a_2 = (1, 3)$, $a_3 = (3, 5)$, $a_4 = (5, 6)$, $a_5 = (0, 2)$, $a_6 = (0, 2)$, $a_7 = (2, 4)$, $a_8 = (4, 6)$, $a_9 = (4, 6)$. a_1, a_4, and a_6 have two overlapping activities, while the others have three. This means that a_1, a_4, and a_6 will be the activities selected, but the optimal solution is a_1, a_2, a_3, a_4,
   An example to show that the strategy of choosing the compatible remaining activities with the earliest start time is $a_1 = (1, 4)$, $a_2 = (4, 5)$, $a_3 = (2, 3)$, $a_4 = (3, 4)$. a_1 will be selected first, followed by a_2, when clearly, the optimal solution consists of a_3, a_4, and a_2.