CSOR 4231 Midterm Exam 2
Solutions November 21, 2019, 10:10 AM

Problem 1 [**20 POINTS**] Based on your passion for greedy algorithms, you decide to set up a franchise of stores where people can come and bring their greedy algorithms to be solved. As a start you will locate them along Interstate 80, a road that travels the length of the United States. For this problem, assume that Interstate 80 is n miles long, and you are given, at each mile marker i, a profit p_i associated with building a store at that mile marker. You want to make sure that you don't build too many stores, because then they would compete with each other, so you decide that each one must be at least k miles away from any other stores.

a) [**10 POINTS**] Give an algorithm that determines the set of stores you can build to obtain the maximum profit.

Solution: This is not really a greedy algorithm. Let $A(i)$ be the most profit you can get from building some subset of stores at mile markers 1 through i when one is built at i. Then, Let

$$B(i) = \max_{j \leq i-k} A(j) + p_i,$$

and

$$A(i) = B(i) + p_i$$

where we consider $A(j) = -\infty$ for $j \leq 0$.

So the algorithm loops over i from 1 to n and computes $A(i)$ be the recurrence above. For the answer, you need to take the max over the A array, since $A(n)$ may not be the largest. You can really just take the max of the last k, since something must be built in the last k.

This is $O(n^2)$ time.

A more efficient solution would observe that $B(i)$ can actually be computed as $\max(B(i-1), A(i-k))$.

You could also give a recurrence where you don't require that the i's store be opened.

b) [**3 POINTS**] Analyze the running time of your algorithm

Solution: The straightforward answer above is $O(n^2)$. You can observe that you would never go more than $2k$ without building a store, so you could actually get $O(nk)$.

The more efficient solution is $O(n)$.

c) [**7 POINTS**] Prove that your algorithm is correct.

Solution: For the correctness we will prove optimal substructure.

Claim: If the optimal solution for 1 through i with a store at i has its next to last store at j, then we have the optimal solution for 1 through j plus store 1.

Proof. Suppose not. Let x be the value of the optimal solution for 1 through j with a store at j. Let x' be the value of your proposed optimal solution. Since it uses i, it must have value $y + p_i$ for some y. But x is the optimal solution for 1 through j, so $x + p_i > y + p_i$ and the new solution is still feasible, so we have a contradiction.

Problem 2. [**20 POINTS**] You are given a sequence of n envelopes, numbered with even numbers from 2 through $2n$. Each envelope i has two cards in it,

numbered i and $i+1$. Each card j has a pair of numbers (v_j, e_j) where v_j a value and e_j is the number of another envelope to take. You are promised that $e_j > j$, for $j < 2n$, and that e_j is even. When you open an envelope i , you choose to keep exactly one of the cards $j \in \{i, i+1\}$ in the envelope, keep that card, and then you must open the envelope e_j next. When you open envelope $2n$, you stop. (Note that the rules guarantee that you will eventually open envelope $2n$). You begin by opening envelope 2 and your payoff is the sum of the v_j values for the cards you kept.

This game sounds like a game of guessing, but suppose that you can cheat and obtain all the data about every card in advance.

Give a dynamic programming algorithm to optimally play the game. That is, you should give an algorithm that chooses the set of cards with the largest total value. You should prove optimal substructure, give a recurrence and explain what the running time will be and why. You do not need to give pseudocode. Be sure to explain in words what the variables in your recurrence stand for.

Solution:

First this can be seen as the longest path in a dag. It is a dag because the cards are increasing, and we want to find the path of maximum value. Dags were not on the syllabus for the midterm, so there is no expectation that this was the solution you would get.

There are several ways to view this problem, working either forwards or backwards. Here is a solution working backwards for the recurrence, and then forward through the envelopes.

Let $A(x)$ be the optimal value you can get after opening envelope x. Let $P(x)$ be the set of envelopes that have a card with $e_j = x$, and for any such envelope $y \in P(x)$, let $v(y)$ be the associated value on the card. Then $A(x) = \max_{y \in P(x)}\{A(y) + v(y)\}$.

We need to go over the envelopes in the topological order given by the dependencies among the envelopes. Thus we can get $O(n)$ time. If you don't realize this, there are several ways to get $O(n^2)$ time.

Optimal substructure: The optimal substructure is similar to subpaths of shortest paths are shortest paths. Except that for this problem, you have longest paths. But because the graph is a dag, you still have optimal substructure.

For the solution given above, the optimal substructure is that if the optimal "path" P of value $A(x)$ to envelope x goes has envelope y immediately before x, then P' which is P with x removed is an optimal "path" to y. Suppose not, and that P'' is actually the optimal "path" to y. Then, the path consisting of P'' followed by x, would actually be a path to x, and since the value of P'' is greater than the value of P', we would have a better way to get to x.

Problem 3. [**20 POINTS**] You are given an array A with n positive numbers, and the array is initially in an arbitrary order.

You are given a sequence of calls SELECTATION(i), where SELECTATION(i) returns the ith smallest number. As you know, one call to SELECTATION can be implemented in $O(n)$ time, using the linear time SELECT algorithm studied in class.

a) [**5 POINTS**] Suppose that you keep an array $B[1 \ldots n]$ with each $B[i] = -1$ initially. When you get a call to SELECTATION(i), you check if $B[i] = -1$. If it does, you run linear time SELECT(i) on array A and store the result in $B[i]$. If $B[i] \neq -1$, then you return $B[i]$. What is the worst case running time of this operation?

Solution: $O(n)$

b) [**5 POINTS**] Suppose that you have a total of n^2 calls to SELECTATION. What is the amortized running time of the algorithm described in part a?

Solution: You have at most n calls that take $\Theta(n)$ time, the first for each i. The remaining calls take $O(1)$ time. So the largest possible time is basically $n(n) + (n^2 - n)1 = O(n^2)$. This is amortized $O(1)$ per operation.

c) [**5 POINTS**] Suppose that you have a total of n calls to SELECTATION. What is the amortize running time of the algorithm described in part a?

Solution: It is possible that all calls are $\Theta(n)$, so the total is $O(n^2)$ for amortized $O(n)$ per operation.

d) [**5 POINTS**] Can you implement SELECTATION in a way that the n calls have a lower amortized running time than they do in your answer to part c? If so, please give the implementation and analyze the running time.

Solution: Yes. For the first call you sort. Then all subsequent calls take $O(1)$ time. The total is now $O(n \log n + n(1)) = O(n \log n)$. The amortized time per operations is $O(\log n)$.