CSCI 570 - Fall 2019 - HW 7 - Solutions

Question 1

Suppose you have a DAG with costs $c_e > 0$ on each edge and a distinguished vertex s. Give a dynamic programming algorithm to find the most expensive path in the graph that begins at s. Prove your algorithm's runtime and correctness. For full credit, your algorithm's runtime should be linear.

Answer 1 Let's consider the vertices in topological order. Suppose $LP(v)$ represents the value of the most expensive path from s to v. Base case:

$$\begin{aligned} LP(s) &= 0 \\ LP(v) &= -\infty \end{aligned} \tag{1}$$

Our recurrence becomes:

$$LP(v) = \max_{u:v\in adj[u]} LP(u) + c_{(u,v)} \tag{2}$$

We can fill in this table in increasing order of v, with the above defined base cases to ensure that any that aren't reachable from s aren't considered to be on the maximum path. We want the largest $LP(v)$ value; if we track the value of the maximum u, we can use that to backtrack and find the longest path. Our total running time is $O(m + n)$.

Question 2

Given a sequence $a1, a2, ..., an$ of n numbers, describe an $O(n^2)$ algorithm to find the longest monotonically increasing sub-sequence.

Answer 2 Let l_i denote the length of the longest monotonically increasing sub-sequence that ends with $a_i (l_1 = 1)$. Compute the sequences S_i, S_{ij} using the following recurrences.

- Initialize $S_1 = a_1$.
- For $1 \leq j < i$, if $a_j > a_i$ then $S_{ij} = a_i$. Otherwise, S_{ij} is set to S_j concatenated with a_i.
- S_i is set to the longest sequence among all the sequences S_{ij} , $1 \leq j < i$.

Claim: The length of $S_i, l(S_i) = l_i$.

Assume otherwise. Let k be the smallest index such that $l(S_k) < l_k$. Let O_k be a sequence of length l_i ending with a_k. Let a_j be the second last element of the sequence O_k. As $j < k, l_j = l(S_j)$

$$l(S_k) < l_k = l_j + 1 \rightarrow l(S_j) < l_j \tag{1}$$

This is a contradiction as $j < k$ and k is the smallest index such that $l(S_k) < l_k$. Thus our claim is true. Clearly the longest monotonically increasing sub-sequence is by definition the longest of the sequences $S_i, 1 \leq i \leq n$. The running time is $O(n2)$.

Question 3

Suppose you are in Casino with your friend, and you are interested in playing a game against your friend by alternating turns. The game contains a row of n coins of values v(i), where n is even. In each turn, a player selects either the first or last coin from the row, removes it from the row permanently, and receives the value of the coin. Determine the maximum possible amount of money you can definitely win if you move first.

Answer 3

Suppose OPT(i, j) represents the maximum value the user can collect from ith coin to jth coin. When we move first, we have two choices to make. Either we choose ith coin or jth coint.

Vi	Vi+1	Vj-1	Vj

Choice 1: The user chooses the ith coin with value Vi: The opponent either chooses (i+1)th coin or jth coin. The opponent intends to choose the coin which leaves the user with minimum value. The user can collect the value Vi + min(OPT (i+2, j), OPT (i+1, j-1))

Choice 2: The user chooses the jth coin with value Vj: The opponent either chooses ith coin or (j-1)th coin. The opponent intends to choose the coin which leaves the user with minimum value. The user can collect the value Vj + min(OPT (i+1, j-1), OPT (i, j-2))

We take the maximum of two choices. The recursive relation is as follows:

Base Cases

OPT (i, j) = Vi If j == i

OPT (i, j) = max(Vi, Vj) If j == i+1

Relation

$$\text{OPT}(i, j) = \text{Max}\left[\, V_i + \min\{\text{OPT}(i+2, j),\ \text{OPT}(i+1, j-1)\},\ \ V_j + \min\{\text{OPT}(i+1, j-1), \text{OPT}(i, j-2)\}\,\right]$$

The subproblem OPT (i+1, j-1) is calculated twice in above relation. Now we solve the problem via Dynammic Programming.

Question 4

Given a rod of length n inches and an array of prices that contains prices of all pieces of size smaller than n. Determine the maximum value obtainable by cutting up the rod and selling the pieces.

Answer 4

Let OPT(n) be the required (best possible price) value for a rod of lenght n.
Let prices[i] an array of prices that contains prices of all pieces of size smaller than n.

Optimal Substructure

We can get the best price by making a cut at different positions and comparing the values obtained after a cut. We can recursively call the same function for a piece obtained after a cut for all i in {0, 1 .. n-1}:

$$OPT\ (n) = \max(price[i] + OPT\ (n-i-1))$$

Overlapping Subproblems

In the above formulation, there are many subproblems which are solved again and again. Since same suproblems are called again, this problem has Overlapping Subprolems property. So the Rod Cutting problem has properties of a dynamic programming problem. Recomputations of same subproblems can be avoided by constructing a temporary array OPT[] in bottom up manner.

```
for (i = 1; i<=n; i++):
    max_val = INT_MIN;
    for (j = 0; j < i; j++):
        max_val = max(max_val, price[j] + OPT [i-j-1])
    OPT [i] = max_val
return OPT [n]
```

Time Complexity of the above implementation is O(n^2).