CSCI 570 - Fall 2021 - HW 6 - Solution

Graded Problems

For the grade problems, you must provide the solution in the form of pseudo-code and also give an analysis on the running time complexity.

Problem 1

[10 points] Suppose you have a rod of length N, and you want to cut up the rod and sell the pieces in a way that maximizes the total amount of money you get. A piece of length i is worth p_i dollars. Devise a **Dynamic Programming** algorithm to determine the maximum amount of money you can get by cutting the rod strategically and selling the cut pieces.

Solution: At each remaining length of the rod, we can choose to cut the rod at any point, and obtain points for one of the cut pieces and recursively compute the maximum points we can get for the other piece. It is also possible to recursively attempt to obtain the maximum value for both pieces after the cut, but that requires adding an extra check to see if the value obtained by selling a rod of this length is more than recursively cutting it up.

The bottom-up pseudo-code to obtain the maximum amount of money is:

Algorithm 1 Bottom-Up-Cut-Rod(p, n)

Let $r[0, ..., n]$ be a new array
$r[0] = 0$
for $j = 1$ to n **do**
  $q = -\infty$
  for $i = 1$ to j **do**
    $q = max(q, p[i] + r[j - i])$
  end for
end for
return r[n]

The time complexity of this algorithm is $\theta(n^2)$ because of the double nested for loop.

Rubric:

- 5 points for a correct dynamic programming solution

- 3 points if the solution runs in $\theta(n^2)$
- 2 points for providing analysis of runtime complexity

Problem 2

[10 points] Tommy and Bruiny are playing a turn-based game together. This game involves N marbles placed in a row. The marbles are numbered 1 to N from the left to the right. Marble i has a positive value m_i. On each player's turn, they can remove either the leftmost marble or the rightmost marble from the row and receive points equal to the sum of the remaining marbles' values in the row. The winner is the one with the higher score when there are no marbles left to remove.

Tommy always goes first in this game. Both players wish to maximize their score by the end of the game.

Assuming that both players play optimally, devise a **Dynamic Programming** algorithm to return the difference in Tommy and Bruiny's score once the game has been played for any given input.

Your algorithm **must** run in $O(N^2)$ time.

Solution:
We first calculate a prefix sum for the marbles array. This enables us to find the sum of a continuous range of values in $O(1)$ time.
If we have an array like this: $[5, 3, 1, 4, 2]$, then our prefix sum array would be $[0, 5, 8, 9, 13, 15]$.
Once we've done that, we can define OPT(i, j) as the maximum difference in score achievable by the player whose turn it is to play, given that the marbles from index i to j (inclusive) remain.

The pseudo-code for this algorithm, assuming 0-indexed arrays, is:

Algorithm 2 Max-Difference-Scores(*marbles*)

Let n be the length of the marbles array
Let $prefix\_sum$ be the calculated prefix sum array for the array marbles [takes $\theta(n)$ time]
Let $OPT[[0, ..., 0], ..., [0, ..., 0]]$ be a new $n * n$ array, with values initialized to 0

for $i = n - 2$ to 0 **do**
  for $j = i + 1$ to $n - 1$ **do**
    $score\_if\_take\_i = prefix\_sum_{j+1} - prefix\_sum_{i+1} - OPT_{i+1,j}$
    $score\_if\_take\_j = prefix\_sum_{j} - prefix\_sum_{i} - OPT_{i,j-1}$
    $OPT_{i,j} = max(score\_if\_take\_i, score\_if\_take\_j)$
  end for
end for
return $OPT_{0,n-1}$

The time complexity of this algorithm is $\theta(n^2)$ if a prefix sum array is calculated initially due to there being $n * n$ subproblems to calculate.

Rubric:

- 8 points for a correct dynamic programming solution
- 2 points for providing analysis of runtime complexity

Problem 3

[10 points] The Trojan Band consisting of n band members hurries to lined up in a straight line to start a march. But since band members are not positioned by height the line is looking very messy. The band leader wants to pull out the minimum number of band members that will cause the line to be in a *formation* (the remaining band members will stay in the line in the same order as they were before). The formation refers to an ordering of band members such that their heights satisfy $r_1 < r_2 < ... < r_i > ... > r_n$, where $1 \leq i \leq n$.

For example, if the heights (in inches) are given as

$$R = (67, 65, 72, 75, 73, 70, 70, 68)$$

the minimum number of band members to pull out to make a formation will be 2, resulting in the following formation:

$$(67, 72, 75, 73, 70, 68)$$

Give an algorithm to find the minimum number of band members to pull out of the line.

Note: you do not need to find the actual formation. You only need to find the minimum number of band members to pull out of the line, but you need to find this minimum number in $O(n^2)$ time.

For this question, you must write your algorithm using pseudo-code.

Solution: This problem performs a $Longest - Increasing - Subsequence$ operation twice. Once from left to right and once again, but from right to left. Once we have the values for the longest subsequence we can make after we "pull out" a few band members, we can iterate over the array and determine what minimum number of pull-outs will cause our line of band members to satisfy the height order requirements.

Let $OPT_{left}(i)$ be maximum length of the line to the left of band member i (including i) which can be put in order of increasing height (by pulling out some members) Note: the problem is symmetric, so we can flip the array R and find the same values from the other direction. Let's call those values $OPT_{right}(i)$.