CSCI 270 - Spring 2023 - HW 7

Due: March 8, 2023

1. [20 points] Solve Kleinberg and Tardos, Chapter 6, Exercise 5.

   (a) Define (in plain English) subproblems to be solved. (4 pts)
   Let $Y_{i,k}$ denote the substring $y_i y_{i+1} \dots y_k$. Let $OPT(k)$ denote the quality of an optimal segmentation of the substring $Y_{1,k}$.

   (b) Write a recurrence relation for the subproblems (6 pts)
   An optimal segmentation of this substring $Y_{1,k}$ will have quality equalling the quality last word (say $y_{i+1} \dots y_k$) in the segmentation plus the quality of an optimal solution to the substring $Y_{1,i}$. Otherwise we could use an optimal solution to $Y_{1,i}$ to improve $OPT(k)$ which would lead to a contradiction.

$$OPT(k) = \max_{0 \le i < k} OPT(i) + quality(Y_{i+1,k})$$

   (c) Using the recurrence formula in part b, write pseudocode to find the maximum total quality among all segmentation possibilities. (5 pts)
   See Algorithm 1.

Algorithm 1: Problem 1

Input: a string y, the length n and a blackbox function *quality*.
Output: $OPT(n)$.
Initialize: set $OPT(0) = 0$.
for *k = 1 to n* **do**
  for *i = 0 to k-1* **do**
    $OPT(k) = \max(OPT(k), OPT(i) + quality(Y_{i+1,k}))$
  end
end

   (d) Make sure you specify

i. base cases and their values (2 pts)
$OPT(0) = 0$

ii. where the final answer can be found (1 pt)
$OPT(n)$

(e) What is the complexity of your solution? (2 pts)
The algorithm runs in $O(n^2)$ since there are at most $O(n)$ entries and each entry takes $O(n)$ time to compute.

2. [20 points] You are given an integer array $a[1], \ldots, a[n]$, find the contiguous subarray (containing at least one number) which has the largest sum and only returns its sum. The optimal subarray is not required to return or compute. Taking $a = [5, 4, -1, 7, 8]$ as an example: the subarray [5] is considered as a valid subarray with sum 5, though it only has one single element; the subarray $[5, 4, -1, 7, 8]$ achieves the largest sum 23; on the other hand, $[5, 4, 7, 8]$ is not a valid subarray as the numbers 4 and 7 are not contiguous.

(a) Define (in plain English) subproblems to be solved. (4 pts)
Let $OPT[1], \ldots, OPT[n]$ be the array where $OPT[i]$: is the largest sum of all contiguous subarrays ending at index i.

(b) Write a recurrence relation for the subproblems. (6 pts)
Considering the recurrence relation of $OPT[i]$: if the subarray $[a[i]]$ achieves the largest sum, then we have $OPT[i] = a[i]$; otherwise, the optimal subarray ending at index i should consist of the optimal subarray achieving $OPT[i-1]$ at index $i-1$, therefore, $OPT[i] = a[i] + OPT[i-1]$ in this case due to the contiguous requirement. Combining these two cases together, we have the following equality for all $i \geq 2$:

$$OPT[i] = \max\{OPT[i-1], 0\} + a[i]$$

(c) Using the recurrence formula in part b, write pseudocode to find the subarray (containing at least one number) which has the largest sum. (5 pts)
See Algorithm 2.

(d) Make sure you specify

i. base cases and their values (2 pts)
OPT[1] = a[1]

ii. where the final answer can be found (1 pt)
$\max\{OPT\}$

Algorithm 2: Problem 2

Input: an integer array a and the length of this array n.
Output: the largest sum stored in Ans.
Initialize: set $OPT[1] = a[1]$ and $Ans = OPT[1]$.
for $i = 2, \ldots, n$ **do**
  Compute $OPT[i]$ as $OPT[i] = \max\{OPT[i-1], 0\} + a[i]$
  if $OPT[i] > Ans$ **then**
    $Ans = OPT[i]$
  end
end

(e) What is the complexity of your solution? (2 pts)
The time complexity of this proposed algorithm is $O(n)$ as there are n subproblems and each subproblem costs $O(1)$ to compute.

3. [20 points] You are given an array of positive numbers $a[1], \ldots, a[n]$. For a subsequence $a[i_1], a[i_2], \ldots, a[i_t]$ of array a (that is, $i_1 < i_2 < \cdots < i_t$): if it is an increasing sequence of numbers, that is, $a[i_1], a[i_2], \ldots, a[i_t]$, its happiness score is given by

$$\sum_{k=1}^{t} k \times a[i_k]$$

Otherwise, the happiness score of this array is zero.
For example, for the input $a = [22, 44, 33, 66, 55]$, the increasing subsequence $[22, 44, 55]$ has happiness score $(1) \times (22) + (2) \times (44) + (3) \times (55) = 275$; the increasing subsequence $[22, 33, 55]$ has happiness score $(1) \times (22) + (2) \times (33) + (3) \times (55) = 253$; the subsequence $[33, 66, 55]$ has happiness score 0 as this sequence is not increasing. Please design an efficient algorithm to **only** return the highest happiness score over all the subsequences.

(a) Define (in plain English) subproblems to be solved. (4 pts)
Let $OPT(i, k)$ be the cumulative value of the happiest subsequence that ends at the first i items and using the i^{th} element for a subsequence of length k (similar to that of the longest increasing subsequence).

(b) Write a recurrence relation for the subproblems (6 pts)

$$OPT(i, k) = \begin{cases} \max\{OPT(j, k-1) + k \times a[i]\} \text{ if exists } j \text{ such that } j < i \text{ and } a[j] < a[i] \\ 0 \text{ otherwise} \end{cases}$$

(c) Using the recurrence formula in part b, write pseudocode to find the highest happiness score over all the subsequences. (5 pts)
See Algorithm 3.

Algorithm 3: Problem 3

Input: an integer array a and the length of this array n.
Output: the highest happiness score stored in Ans.
Initialize: set $OPT(i,j) = -\infty$ for all (i,j) that $i,j \in 1,\ldots,n$, and $Ans = -\infty$

```
for i = 1 to n do
    Set OPT(i,1) = a[i] and update Ans = max{Ans, L(i,1)}.
    for k = 2 to i do
        for j = 1 to i-1 do
            if a[j] < a[i] and k-1 ≤ j then
                OPT(i,k) = max{OPT(i,k), OPT(j,k-1) + k × a[i]}
            end
        end
        if OPT(i, k) > Ans then
            Ans = OPT(i,k)
        end
    end
end
```

(d) Make sure you specify

i. base cases and their values (2 pts)
$OPT(i,j) = -\infty$ for all (i,j) that $i,j \in 1,\ldots,n$

ii. where the final answer can be found (1 pt)
$\max(OPT)$ or Ans

(e) What is the complexity of your solution? (2 pts)
The algorithm runs in $O(n^3)$ since there are at most $O(n^2)$ entries and each entry takes $O(n)$ time to compute.

4. [20 points] You've started a hobby of retail investing into stocks using a mobile app, RogerGood. You magically gained the power to see N days into the future and you can see the prices of one particular stock. Given an array of prices of this particular stock, where $prices[i]$ is the price of a given stock on the i^{th} day, find the maximum profit you can achieve through various withoutStock/sell actions. RogerGood also has a fixed fee per transaction. You may complete as many transactions as you like,

but you need to pay the transaction fee for each transaction (only pay once per pair of buy and sell). Assume you can own at most one unit of stock.

(a) Define (in plain English) subproblems to be solved. (4 pts)
Consider $withoutStock(i)$ to be the maximum profit you can make if you start from day i and you currently do not own a unit of stock (so you can choose to buy a unit of stock at day i or not).
Consider $withStock(i)$ to be the maximum profit you can make starting from day i and you already own a unit of the stock (so you can choose to sell the stock at day i or not.)

(b) Write a recurrence relation for the subproblems (6 pts)
We will apply the transaction fee during the sale of the stock.

- There're two choices for us at day i if we don't own a stock - buy the stock or not. We need to compare the profits to be made under the two choices:

  $withoutStock(i) = \max(withStock(i+1) - prices(i), withoutStock(i+1))$

- There're two choices for us at day i if we already own a stock - sell the stock or not. We need to compare the profits to be made under the two choices:

  $withStock(i) = \max(withoutStock(i+1) + prices(i) - fee, withStock(i+1))$

(c) Using the recurrence formula in part b, write pseudocode to solve the problem. (5 pts)
See Algorithm 4.

Algorithm 4: Problem 4

Input: an array: $prices$, of length N for N days containing the stock price for each day. A number fee denoting the fee per transaction.
Output: $withoutStock[0]$.
Initialize: Let $withoutStock[0, ..., n+1]$ be a new array, with values initialized to 0.
Let $withStock[0, ..., n+1]$ be a new array, with values initialized to 0.
for $i = n$ *to* 0 **do**
  $withoutStock[i] = \max(withStock[i+1] - prices[i], withoutStock[i+1])$
  $withStock[i] = max(withoutStock[i+1] + prices[i] - fee, withStock[i+1])$
end

(d) Make sure you specify

i. base cases and their values (2 pts)
$withoutStock[0, ..., n+1]$ initialized to 0
$withStock[0, ..., n+1]$ initialized to 0

ii. where the final answer can be found (1 pt)
$withoutStock[0]$

(e) What is the complexity of your solution? (2 pts)
The time complexity of this solution is $O(n)$. We use a single for-loop to compute the maximum profits at each stage.