CSCI 570: Homework 7

Graded Problems

1. 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 return 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. (20 pts)

   Solution:

   a. Define (in plain English) subproblems to be solved. (4 pts)

      Let f [1], . . . , f [n] be the array where f [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 f [i]: if the subarray [a[i]] achieves the largest sum, then we have f [i] = a[i]; otherwise, the optimal subarray ending at index i should consist of the optimal subarray achieving f [i − 1] at index i − 1, therefore, f [i] = a[i] + f [i − 1] in this case due to the contiguous requirement. Combining these two cases together, we have the following equality for all i ≥ 2 :

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

   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)

      Input: an integer array a and the length of this array n.
      Initialize: set f [1] = a[1] and Ans = f [1].
      for i = 2, . . . , n do
          Compute f [i] as
              f [i] = max {f [i − 1], 0} + a[i]

if f [i] > Ans then
Ans = f [i]
Return: the largest sum stored in Ans.

d. Make sure you specify
   1. base cases and their values (2 pts)

      f [1] = a[1]

   2. where the final answer can be found (1 pt)

      max(f)

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.

2. 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 ith day, find the maximum profit you can achieve through various buy/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. (20 pt)

Solution:

a. Define (in plain English) subproblems to be solved. (4 pts)

   Consider buy(i) to be the maximum profit you can make if you start at day i, assuming you currently do not own a unit of stock.
   Consider sell(i) to be the maximum profit you can make starting at day i, assuming you already own a unit of the stock.

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

   We will apply the transaction fee during the sale of a stock. The recurrence relation is as follows:

buy(i) = max(sell(i + 1) − prices(i), buy(i + 1))
sell(i) = max(buy(i + 1) + prices(i) − fee, sell(i + 1))

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

Algorithm:

Input: an arrays: *prices*, of length N for N days containing the stock price and fee for each day. A number fees denoting the fee per transaction
Initialize:
Let buy[0, ..., n + 1] be a new array, with values initialized to 0
Let sell[0, ..., n + 1] be a new array, with values initialized to 0

```
for i = n to 0 do
        buy[i] = max(sell[i + 1] − prices[i], buy[i + 1])
        sell[i] = max(buy[i + 1] + prices[i] − fee, sell[i + 1])
end for
```

Return: return buy[0]

d. Make sure you specify

1. base cases and their values (2 pts)

   buy[0, ..., n + 1] initiallized to 0
   sell[0, ..., n + 1] initiallized to 0

2. where the final answer can be found (1 pt)

   buy[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.

3. You are given an array of positive numbers $a[1], \ldots, a[n]$. For a sub-sequence $a[i_1], a[i_2], \ldots, a[i_t]$ of array a (that is, $i_1 < i_2 <, \ldots, i_t$): 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 subse-quence [22, 44, 55] has happiness score (1) × (22) + (2) × (44) + (3) × (55) = 275; the increasing subsequence [22, 33, 55] has happiness score (1) × (22) + (2) × (33) + (3) × (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 (20 pts)

a. Define (in plain English) subproblems to be solved. (4 pts)

Let $L(i, k)$ 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)

$L(i, k) = max_{j<i,\, a[j]<a[i]} L(j, k-1) + k \times a[i]$,

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

Algorithm:

Input: an integer array a and the length of this array n.
Initialize: set L(i, j) = $-\infty$ for all (i, j) that i, j $\in$ 1, . . . n, and Ans = $-\infty$.

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

Ans = L(i, k)

Return: the highest happiness score stored in Ans.

d. Make sure you specify

i. base cases and their values (2 pts)

L(i, j) = $-\infty$ for all (i, j) that i, j $\in$ 1, . . ., n

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

$max(L)$

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. You are given an $m \times n$ binary matrix $g \in \{0,\ 1\}^{m \times n}$. Each cell either contains a "0" or a "1". Give an efficient algorithm that takes the binary matrix g, and returns the largest side length of a square that only contains 1's. You are **not** required to give the optimal solution. (20 pts)

a. Define (in plain English) subproblems to be solved. (4 pts)

We define the $m \times n$ dynamic programming matrix f with $f(i,\ j)$ being the side length of the maximum square matrix of 1's whose right bottom cell is $(i,\ j)$. That is, the sub matrix whose left top cell is $(i\ -\ f(i,\ j)\ +\ 1,\ j\ -\ f(i,\ j)\ +\ 1)$ is the largest square matrix that is filled with 1's and ending at cell $(i,\ j)$, and either $f(i,\ j) = min\ \{i,\ j\}$ or the matrix $(i\ -\ f(i,\ j),\ j\ -\ f(i,\ j))$ to $(i,\ j)$ contains some 0's.

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

Starting from cell $(1,\ 1)$, we compute the dynamic programming matrix f at cell $(i,\ j)$ as

$$f(i,j) = \begin{cases} min\ \{f(i\ -\ 1,\ j),\ f(i\ -\ 1,\ j\ -\ 1),\ f(i,\ j\ -\ 1)\}\ +\ 1 \\ , \ i \geq 2\ \&\ j \geq 2\ \&\ g(i,\ j)\ =\ 1, \\ \\ g(i,\ j), \text{Otherwise.} \end{cases}$$

EXPLANATION (NOT GRADED):

The recurrence equation can be verified directly. Suppose the largest 1's square ended at (i, j) has the length of k, then it is ensured that min {f(i − 1, j), f(i − 1, j − 1), f(i, j − 1)} ≥ k − 1 as these three sub-matrices of size k − 1 (that is, the matrices ended at (i − 1, j), (i, j − 1) and (i − 1, j − 1)) are all filled with 1's. On the other hand, suppose that min {f(i − 1, j), f(i − 1, j − 1), f(i, j − 1)} = k and g(i, j) = 1 hold, then we have f(i, j) ≥ k + 1 as the square matrix of size k+1 ending at (i, j) only contains 1's. Combining these two inequalities ensure the correctness of the aforementioned recurrence equation. We also remember the largest side length of a square found so far. In this way, we traverse the dynamic programming matrix once and find out the required maximum size. This gives the side length of the square (say maxsqlen).

c. Using the recurrence formula in part b, write pseudocode using iteration to compute the the largest side length of a square that only contains 1's to meet the objective. (5 pts)

Algorithm:

Input: a binary matrix g, the number of rows m and the number of columns n.
Initialize: set $f(i,\ j)\ =\ 0$ for all cells $(i,\ j)$ that $i \in 1, \ldots, m,\ j \in 1, \ldots, n$. Set $Ans\ =\ 0$ for answer.

for $i\ =\ 1, \ldots, m$ do
    for $j\ =\ 1, \ldots, n$ do
        if $g(i,\ j)\ =\ 0\ or\ i\ =\ 1\ or\ j\ =\ 1$ then ▷ boundary condition
            $f(i,\ j)\ =\ g(i,\ j)$
        else
            $f(i,\ j) = min\{f(i\ -\ 1,\ j),\ f(i,\ j\ -\ 1),\ f(i\ -\ 1,\ j\ -\ 1)\}$
        if $f(i,\ j)\ >\ Ans$ then ▷ update answer
            $Ans\ =\ f(i,\ j)$

Return: the largest side length stored in Ans.

d. Make sure you specify

i. base cases and their values (2 pts)

$f(i,\ j)\ =\ 0\ for\ all\ (i,\ j)\ that\ i \in 1, \ldots, m,\ j \in 1, \ldots, n$

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

$max(f)$

e. What is the complexity of your solution? (2 pts)

The algorithm runs in O(mn) since there are at most O(mn) entries and each entry takes O(1) time to compute.

Practice Problems

1. Given n sand piles of weights $w[1], \ldots, w[n]$ on a line, we want to merge all the sand piles together. At each step we can only merge adjacent sand piles, with cost equal to the weight of the merged sand pile. In particular, merging sand piles i and $i + 1$ has cost $w_i + w_{i+1}$. Furthermore, one gets a single sand pile of weight $w_i + w_{i+1}$ in its place which is now adjacent to sand piles $i - 1$ and $i + 2$. Note that different merging orders will result in different final costs, please find the minimum cost to merge all the sand piles. The optimal merging order is **not** required. (20 pts)

- Sample Input: $n = 4$, $w[1, \ldots, n] = [100, 1, 1, 100]$.
- Sample Output: the minimum cost is 1+1+2+100+102+100 = 306 by first merging (1, 1), then (100, 2) and (102, 100).

a. Define (in plain English) subproblems to be solved. (4 pts)

We define a subproblem $f(i, j)$ to be the minimum possible cost of merging all the piles from i to j.

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

We compute the dynamic programming matrix f at cell (i, j) as

$$f(i, j) = min_{i \le k < j}\{f(i, k) + f(k + 1, j)\} + \sum_{k=i}^{j} w_k \text{, for all } i < j$$

EXPLANATION (NOT GRADED):
For any solution that merges the sand piles from i to j must have a final merge which merges piles from i to k and from k + 1 to j. Then those sub-intervals of sand piles should be merged at minimum cost which is defined recursively. The recurrence equation chooses the minimum cost solution of this form. Since there are at most n^2 subproblems, the space complexity is $O(n^2)$. The runtime is $O(n^3)$ since the computation of subproblem takes the minimum over O(n) terms with at most $O(n^2)$ subproblems.

c. Using the recurrence formula in part b, write pseudocode using iteration to compute the the minimum cost to merge all the sand piles to meet the objective. (5 pts)

Algorithm:

Input: the number of sand piles n, and the weights $w[1], \ldots, w[n]$.
Initialize: set entries of f as

$$f(i,j) = \begin{cases} +\infty, & 1 \le i < j \le n \\ w[i], & i = j \end{cases}$$

for $s = 2, \ldots, n$ do ▷ Iterate the number of sand piles
  for $i = 1, \ldots, n - s + 1$ do ▷ Iterate all pairs (i, j) that j − i + 1 = s
    Compute $j = i + s - 1$ and $w(i, j) = \sum_{k=i}^{j} w_k$
    for $k = i, \ldots, j - 1$ do
      $f(i, j) = min\{f(i, k) + f(k + 1, j) + w(i, j), f(i, j)\}$

Return: $f(1, n)$ as the minimum cost.

d. Make sure you specify

i. base cases and their values (2 pts)

$$f(i,j) = \begin{cases} +\infty, & 1 \le i < j \le n \\ w[i], & i = j \end{cases}$$

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

$f(1, n)$

e. What is the complexity of your solution? (2 pts)

The runtime is $O(n^3)$ since the computation of subproblem takes the minimum over O(n) terms with at most $O(n^2)$ subproblems.