CSCI 570 Fall 2025
Homework 6

Problem 1

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 ($1 \leq i \leq N$). 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. (Rod lengths are integers).

1. Define (in plain English) subproblems to be solved.
2. Write a recurrence relation for the subproblems.
3. Using the recurrence formula in part b, write pseudocode to find the solution. Make sure you specify
   (a) base cases and their values,
   (b) where the final answer can be found.
4. What is the complexity of your solution?

Solution

Let $r[0], \ldots, r[n]$ be the array where $r[i]$ is the maximum money we can get for a rod of length i. Considering the recurrence relation of $r[i]$: 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 compute the maximum points we can get for the other piece. $r[i] = \max(r[i], r[j] + p[i-j])$ for $(0 \leq j \leq i \leq n)$

Algorithm 1 Rod Cutting Maximum Value

```
Require: Array $p$ where $p[i]$ is the price for rod of length $i$, Integer $n$ for rod length
Ensure: Maximum value obtainable by cutting the rod
  Initialize array $r[0..n]$
  $r[0] \leftarrow 0$
  for $i \leftarrow 1$ to $n$ do
    $r[i] \leftarrow 0$
    for $j \leftarrow 0$ to $i$ do
      $r[i] \leftarrow \max(r[i], r[j] + p[i-j])$
    end for
  end for
  return $r[n]$
```

1. Base case is: $r[0] = 0$
2. Solution can be found at: $r[n]$

The time complexity of this proposed algorithm is $O(n^2)$ as there are n subproblems and each subproblem costs $O(n)$ to compute in the worst case.

Rubrics (10 points)

- Define (in plain English) subproblems to be solved (2 pts)
- Correct recurrence relation for the subproblems (3 pts)
- Correct pseudocode structure (2 pts)
  - Specify base cases and their values (1 pt)
  - Specify where the final answer can be found (1 pt)
- Correct Complexity of the solution (1 pt)

Problem 2

Alice and Bob are playing a turn-based game. It involves N stones placed in a row, numbered 0 to $N-1$ from the left to the right. Each stone i has a positive value s_i. On each player's turn, they can remove either the leftmost stone or the rightmost stone from the row and receive points equal to the sum of the remaining stones' values in the row. The winner is the one with the higher score when there are no stones remaining. Alice goes first in this game. Both players play to maximize their score by the end of the game. Devise a Dynamic Programming algorithm to return the maximum difference in score that Alice can achieve over Bob. Your algorithm must run in $O(N^2)$ time.

1. Define (in plain English) subproblems to be solved.
2. Write a recurrence relation for the subproblems.
3. Using the recurrence formula in part b, write pseudocode to find the solution. Make sure you specify
   (a) base cases and their values,
   (b) where the final answer can be found.
4. What is the complexity of your solution?

Solution

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.

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 $[0,5,8,9,13,15]$. Then, the recurrence relation becomes

$$\begin{aligned}
\text{ifTake}_i &= \text{prefixSum}_{j+1} - \text{prefixSum}_{i+1} - OPT_{i+1,j} \\
\text{ifTake}_j &= \text{prefixSum}_{j} - \text{prefixSum}_{i} - OPT_{i,j-1} \\
OPT_{i,j} &= \max(\text{ifTake}_i, \text{ifTake}_j)
\end{aligned}$$

for all i,j $(0 \le i \le N-1, i+1 \le j \le N-1)$. 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).

Algorithm 2 Maximum Difference in Score

```
1: Let $OPT[[0,\dots,0],\dots,[0,\dots,0]]$ be a new $n \times n$ array with values initialized to 0
2: for $i = n-2$ to 0 do
3:     for $j = i+1$ to $n-1$ do
4:         $\text{ifTake}[i] = \text{prefixSum}[j+1] - \text{prefixSum}[i+1] - OPT[i+1][j]$
5:         $\text{ifTake}[j] = \text{prefixSum}[j] - \text{prefixSum}[i] - OPT[i][j-1]$
6:         $OPT[i][j] = \max(\text{ifTake}[i], \text{ifTake}[j])$
7:     end for
8: end for
9: return $OPT[0][n-1]$
```

1. Base case is: $OPT_{i,j} = 0$ for all $i, j \in 0, \dots, n-1$
2. Solution can be found at: $OPT_{0,n-1}$

The time complexity of this proposed algorithm is $O(n^2)$ as there are n^2 subproblems and each subproblem costs $O(1)$ to compute.

Rubrics (10 points)

- Define (in plain English) subproblems to be solved (2 pts)
- Correct recurrence relation for the subproblems (3 pts)
- Correct pseudocode structure (2 pts)
  - Specify base cases and their values (1 pt)
  - Specify where the final answer can be found (1 pt)
- Correct Complexity of the solution (1 pt)

Problem 3

The Math Club consisting of n members hurries to line up in a straight line to take a group photo. Since the members are not positioned by height, the line is looking very messy. The club president decides to remove a few members so that the line follows a formation with the remaining members staying where they are. We say that k members remaining in the line with heights $r_1, r_2, \dots, r_k$ are in formation if $r_1 < r_2 < \dots < r_i > \dots > r_k$, for some $1 \le i \le k$. For example, if the initial sequence of heights in inches is

$$(58, 60, 62, 65, 63, 61, 59, 57)$$

then, removing member #5 and #7 gives us the formation:

$$(58, 60, 62, 65, 61, 57)$$

Give an algorithm that runs in $O(n^2)$ time to find the minimum number of members to remove from the line, so that we get a formation as described.

1. Define (in plain English) subproblems to be solved.
2. Write a recurrence relation for the subproblems.
3. Using the recurrence formula in part b, write pseudocode to find the solution. Make sure you specify
   (a) base cases and their values,
   (b) where the final answer can be found.
4. What is the complexity of your solution?

Solution

Let $OPT_{left}(i)$ be the maximum length of the line to the left of 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 $OPT_{right}(i)$.
The recurrence relations are:

$$OPT_{left}(i) = \max(OPT_{left}(i), OPT_{left}(j) + 1) \text{ such that } r_i > r_j \ \forall 1 \leq j < i$$
$$OPT_{right}(i) = \max(OPT_{right}(i), OPT_{right}(j) + 1) \text{ such that } r_i > r_j \ \forall 1 \leq j < i$$

Algorithm 3 Minimum Removals for Bitonic Array

```
Let $n$ be the length of the input array and $R$ be the input array
for $i = 1$ to $n$ do
    $OPT_{left}[i] = OPT_{right}[i] = 1$ {Base case}
end for
for $i = 2$ to $n$ do
    for $j = 1$ to $i - 1$ do
        if $r[i] > r[j]$ then
            $OPT_{left}[i] = \max(OPT_{left}[i], OPT_{left}[j] + 1)$
        end if
    end for
end for
for $i = 2$ to $n$ do
    for $j = 1$ to $i - 1$ do
        if $r[n - i + 1] > r[n - j + 1]$ then
            $OPT_{right}[i] = \max(OPT_{right}[i], OPT_{right}[j] + 1)$
        end if
    end for
end for
result = $\infty$
for $i = 1$ to $n$ do
    result = min(result, $n - (OPT_{left}[i] + OPT_{right}[n - i + 1] - 1)$)
end for
return result
```

1. Base case is: $OPT_{left}(i) = OPT_{right}(i) = 1$ for all $i, j \in 1, \ldots, n$
2. Solution can be found at: $\min(n - (OPT_{left}(i) + OPT_{right}(n - i + 1) - 1))$ for $i \in 1, \ldots, n$

The runtime of this algorithm is dominated by the nested for loops used to calculate the LIS both ways. This takes $O(n^2)$ each time. Therefore, the time complexity of the solution is $O(n^2)$.

Rubrics (10 points)

- Define (in plain English) subproblems to be solved (2 pts)
- Correct recurrence relation for the subproblems (3 pts)
- Correct pseudocode structure (2 pts)
  - Specify base cases and their values (1 pt)
  - Specify where the final answer can be found (1 pt)
- Correct Complexity of the solution (1 pt)

Problem 4

Consider the 0-1 knapsack problem where you have infinitely many coins of each type. Namely, there are n different types of coins. Coins of type i have a weight of w_i and a value of v_i ($1 \leq i \leq n$). There is an unlimited supply of coins of each type. Design a dynamic programming algorithm to compute the optimal value you can achieve with a knapsack that has a maximum weight capacity of W.

1. Define (in plain English) subproblems to be solved.
2. Write a recurrence relation for the subproblems.
3. Using the recurrence formula in part b, write pseudocode to find the solution. Make sure you specify
   (a) base cases and their values,
   (b) where the final answer can be found.
4. What is the complexity of your solution?

Solution

Similar to what is taught in the lecture, let $OPT(k, w)$ be the maximum value achievable using a knapsack of capacity $0 \leq w \leq W$ and with k types of items $1 \leq k \leq n$. We find the recurrence relation of $OPT(k, w)$ as follows. Since we have infinitely many items of each type, we choose between the following two cases:

1. We include another item of type k and solve the sub-problem $OPT(k, w - w_k)$.
2. We do not include any item of type k and move to consider the next type of item, thus solving the sub-problem $OPT(k - 1, w)$.

Therefore, we have

$$OPT(k, w) = \max\{OPT(k-1, w), OPT(k, w - w_k) + v_k\}$$

Let OPT be a $n \times W$ sized 2-D array.

Algorithm 4 Knapsack with Repetition

```
$OPT[0][0] \leftarrow 0$
**for** $k = 1$ **to** $n$ **do**
    **for** $w = 0$ **to** $W$ **do**
        **if** $w \geq w_k$ **then**
            $OPT[k][w] \leftarrow \max(OPT[k-1][w], OPT[k][w - w_k] + v_k)$
        **else**
            $OPT[k][w] \leftarrow OPT[k-1][w]$
        **end if**
    **end for**
**end for**
**return** $OPT[n][W]$
```

1. Base case is: $OPT(0, 0) = 0$
2. Solution can be found at: $OPT(n, W)$

The runtime of this algorithm is dominated by the nested for loops used to calculate the OPT. This takes $O(W)$ for $O(n)$ times. Therefore, the time complexity of the solution is $O(W \times n)$.

Rubrics (10 points)

- Define (in plain English) subproblems to be solved (2 pts)
- Correct recurrence relation for the subproblems (3 pts)
- Correct pseudocode structure (2 pts)
  - Specify base cases and their values (1 pt)
  - Specify where the final answer can be found (1 pt)
- Correct Complexity of the solution (1 pt)

Ugraded Problems

Solve Kleinberg and Tardos, Chapter 6, Exercise 5.

Solution

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 $Y_{1,k}$. An optimal segmentation of this substring $Y_{1,k}$ will have quality equalling the quality last word (say $y_i \dots y_k$) 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.

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

We can begin solving the above recurrence with the initial condition that $Opt(0) = 0$ and then go on to compute $Opt(k)$ for $k = 1, 2, \dots, n$ keeping track of where the segmentation is done in each case. The segmentation corresponding to $Opt(n)$ is the solution and can be computed in $\Theta(n^2)$ time.

Solve Kleinberg and Tardos, Chapter 6, Exercise 6.

Solution

Let $W = \{w_1, w_2, \dots, w_n\}$ be the set of ordered words which we wish to print. In the optimal solution, if the first line contains k words, then the rest of the lines constitute an optimal solution for the sub problem with the set $\{w_{k+1}, \dots, w_n\}$. Otherwise, by replacing with an optimal solution for the rest of the lines, we would get a solution that contradicts the optimality of the solution for the set $\{w_1, w_2, \dots, w_n\}$.

Let $Opt(i)$ denote the sum of squares of slacks for the optimal solution with the words $\{w_i, \dots, w_n\}$. Say we can put at most the first p words from w_i to w_n in a line, that is, $\sum_{t=i}^{p+i-1} c_t + p - 1 \leq L$ and $\sum_{t=1}^{p+i} w_t + p > L$. Suppose the first k words are put in the first line, then the number of extra space characters is

$$s(i,k) := L - k + 1 - \sum_{t=i}^{i+k-1} c_t$$

So we have the recurrence

$$Opt(i) = \begin{cases} 0 & \text{if } p \geq n - i + 1 \\ \min_{1 \leq k \leq p}\{(s(i,k))^2 + Opt(i+k)\} & \text{if } p < n - i + 1 \end{cases}$$

Trace back the value of k for which $Opt(i)$ is minimized to get the number of words to be printed on each line. We need to compute $Opt(i)$ for n different values of i. At each step p may be asymptotically as big as L. Thus the total running time is $O(nL)$.