CSCI 570 Fall 2025
Homework 7

Problem 1

Jack has gotten himself involved in a very dangerous game called the octopus game where he needs to pass a bridge which has some unreliable sections. The bridge consists of 3n tiles as shown below. Some tiles are strong and can withstand Jack's weight, but some tiles are weak and will break if Jack lands on them, leading to a fatal fall. We have been given this information in an array called BadTile(3,n) where **BadTile (j, i) = 1 if the tile is weak and 0 if the tile is strong**. At any step Jack can move either to the tile exactly in front of him (i.e. from tile (j, i) to $(j, i + 1)$), or diagonally to the left or right (if they exist). (No sideways or backward moves are allowed and one cannot go from tile $(1, i)$ to $(3, i + 1)$ or from $(3, i)$ to $(1, i + 1)$). Jack is allowed to start in any (strong) tile $(j, 1)$ in the beginning and end at any (strong) tile (j, n) at the end.

Use dynamic programming to find out how many ways (if any) there are for Jack to pass this deadly bridge. In Fig. 1, we show an example of bad tiles in gray and one of the possible ways for Jack to safely cross the bridge alive.

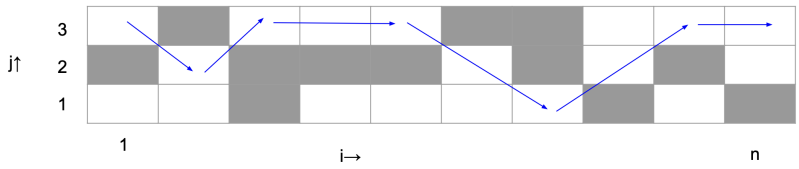

Figure 1: Example of octopus game.

1. Define (in plain English) subproblems to be solved. (2 points).
2. Write a recurrence relation for the subproblems. (3 points)
3. Using the recurrence formula in part b, write pseudocode using iteration to compute the total number of ways to safely cross the bridge. (4 points). Make sure you specify:
   - Base cases and their values.
   - Where the final answer can be found (e.g. opt(n), or opt(0,n), etc.).

Problem 2

Given n balloons, indexed from 0 to $n - 1$. Each balloon is painted with a number on it represented by array $nums$. You are asked to burst all the balloons. If the you burst balloon i you will get $nums[i - 1] \times nums[i] \times nums[i + 1]$ coins. Here left and right are adjacent indices of i. After the burst, the left and right then becomes adjacent. You may assume $nums[-1] = nums[n] = 1$ and they are not real therefore you cannot burst them. Design a dynamic programming algorithm to find the maximum coins you can collect by bursting the balloons wisely. Analyze the running time of your algorithm.

Example. If you have the $nums = [3, 1, 5, 8]$. The optimal solution would be 167, where you burst balloons in the order of 1, 5, 3 and 8. The array $nums$ after each step is:

$$[3, 1, 5, 8] \rightarrow [3, 5, 8] \rightarrow [3, 8] \rightarrow [8] \rightarrow [] \tag{1}$$

And the number of coins you get is $3 \times 1 \times 5 + 3 \times 5 \times 8 + 1 \times 3 \times 8 + 1 \times 8 \times 1 = 167$.

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

Problem 3

We are given an array A of size n and an array B of size m where $n \geq m$. Our goal is to remove $n - m$ elements from A, in such a way that that removing them will transform A to B. Notice that the relative order stays the same for the remaining elements.
As an example, let $A = [3, 1, 1, 4, 4, 1]$ and $B = [3, 1, 4, 1]$, then there are 4 ways to remove the elements:

- $[3, 1, 1, 4, 4, 1]$,
- $[3, 1, 1, 4, 4, 1]$,
- $[3, 1, 1, 4, 4, 1]$,
- $[3, 1, 1, 4, 4, 1]$.

Design a dynamic programming algorithm to find out the number of different ways to remove elements from A that will transform A to B. To qualify full credit, your algorithm needs to run in $O(mn)$ time, sub-optimal algorithm will receive at most 10 pts.

1. Define (in plain English) sub-problems to be solved. (3 pts)
2. Write a recurrence relation for the sub-problems and specify the base cases. (5 pts)
3. Analyze its runtime complexity and explain why. (2 pts)

Problem 4

You are given an integer array A of size N, where each element in the array satisfies $1 \leq A[i] \leq N$. The array may contain duplicate numbers. Your task is to determine the length of the longest **consecutive subsequence** that can be formed from A.

A **subsequence** is a sequence derived from A by removing some elements (possibly none) while keeping the order of the remaining elements unchanged. A **consecutive subsequence** is a special type of subsequence where the numbers appear in strictly increasing and consecutive order. In other words, it should be of the form $[x, x + 1, x + 2, \dots]$ for some integer x, meaning that every number in the sequence must be exactly one more than the previous number.

For example, consider the array $A = [1, 2, 6, 4, 3, 2]$. The sequence $[2, 6, 3]$ is a valid subsequence but **not** a consecutive subsequence because 6 does not equal to $2 + 1$. On the other hand, $[1, 2]$ is a valid consecutive subsequence because all the numbers appear in strictly increasing consecutive order.

Your goal is to design an efficient **dynamic programming algorithm** to compute the **maximum length** of any consecutive subsequence in A. Given that N can be as large as 10^5, your solution should be optimized to run efficiently within these constraints.

1. Define (in plain English) sub-problems to be solved. (2 points)
2. Write a recurrence relation for the sub-problems. (3 points)
3. Using the recurrence formula in part b, write an iterative pseudo-code to find the solution. Make sure you specify base cases. (2 points)
4. What is the complexity of your solution? (1 points)

Ungraded Problems

You are given $n + 1$ rooms, labeled from Room 0 to Room n. There is a heater in Room 0, which warms all rooms from Room 0 to Room r, where r is an unknown integer satisfying $1 \leq r \leq n$. This means that any room with an index less than or equal to r is warm, while any room with an index greater than r is cold. To determine the exact value of r, you have k thermometers. Each time you bring a thermometer into a room, it provides feedback based on the room's temperature. If the room is warm, the thermometer remains functional and can be used again in another test. However, if the room is cold, the thermometer breaks immediately and can no longer be used. Your objective is to find the exact value of r while minimizing the total number of times you enter a room. Since thermometers are limited and can break, an efficient strategy is necessary to optimize the testing process. The challenge is to develop a dynamic programming algorithm that determines the minimal number of room entries required to accurately find r.

1. Write down the recursive formula and the meaning of each part.
2. What is the time complexity of the algorithm?

Assume a truck with capacity W is loading. There are n packages with different weights, i.e. $(w_1, w_2, \ldots w_n)$, and all the weights are integers. The company's rule requires that the truck needs to take packages with exactly weight W to maximize profit, but the workers like to save their energies for after work activities and want to load as few packages as possible. Assuming that there are combinations of packages that add up to weight W, design an algorithm to find out the minimum number of packages the workers need to load.

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 using iteration to compute the minimum number of packages to meet the objective.
4. Make sure you specify base cases and their values; where the final answer can be found. (2 points)
5. What is the worst-case runtime complexity? Explain your answer.