Northeastern University
CS5800 – Algorithms
Fall 2024, Prof. Ryan Rad

Homework #3

Homework 3 is based on the topics of week 5 and 6 (Greedy Programming & Graph Algorithms) and a bit of week 4 (Dynamic Programming).

Important Note: I would like to give you the luxury of designing your own challenges on Greedy Programming (See Question 4). So again, there are FOUR questions only, where problems $1-3$ each worth *eight* marks while problem 4 worth *ten* marks. Therefore, HW2 is out of 34 marks (plus one Bonus part that is worth 3%).

Submit a single PDF file on Canvas, with the answer to all the problems you have tackled in this homework. Make sure you label your HomeWork #3 submission appropriately - e.g. RyanRad-HW3.pdf. Make sure you start your answer to each question on a brand new page! If you are using a pen and paper, please neatly handwrite your solutions on standard A4 paper, with your name(s) and question # at the top of each solution.

Please note that all problems are to be completed **individually**. While a solution must be perfect to receive full marks, we will be generous in awarding partial marks for incomplete solutions that demonstrate progress.

So that there is no ambiguity, there are two non-negotiable rules. A violation of either rule constitutes plagiarism and will result in you receiving an F for this course.

(a) If you meet with a classmate to discuss one of the Individual Problems, the articulation of your thought process (i.e., what you submit), must be an individual activity, done in your own words, away from others. Please remember that the solution-writing process is where so much of your learning will occur in this course: much more than anything we do in class, and even more than the time you spend on solving the problems. Do not be surprised if it takes you 3 to 5 times as long to write up a solution than it takes you to actually solve the problem.

(b) This Problem Set has been designed to be challenging, because struggling through problems is how we learn best. Your educational experience is cheapened by going online and finding the solution to a problem; even using the Internet to look for a "small hint" is unacceptable. In return, your wonderful TAs will be readily available during the office hours (drop-in or by appointment), and upon request, we will happily support you in the process.

Problem #1 – Longest Subsequence Problem (Week 4: Dynamic Prog.)

Difficulty: ■■■■□

The *Longest Subsequence Problem* is a well-studied problem in Computer Science, where given a sequence of distinct positive integers, the goal is to output the longest subsequence whose elements appear from smallest to largest, or from largest to smallest.

For example, consider the sequence $S = [9, 7, 4, 10, 6, 8, 2, 1, 3, 5]$. The longest increasing subsequence of S has length three ($[4, 6, 8]$ or $[2, 3, 5]$), and the longest decreasing subsequence of S has length five ($[9, 7, 4, 2, 1]$ or $[9, 7, 6, 2, 1]$).

And if we have the sequence $S = [531, 339, 298, 247, 246, 195, 104, 73, 52, 31]$, then the length of the longest increasing subsequence is 1 and the length of the longest decreasing subsequence is 10.

(a) **(2 marks)** Find a sequence with nine distinct integers for which the length of the longest increasing subsequence is 3, and the length of the longest decreasing subsequence is 3. Briefly explain how you constructed your sequence.

(b) **(3 marks)** Let S be a sequence with ten distinct integers. Prove that there *must* exist an increasing subsequence of length 4 (or more) or a decreasing subsequence of length 4 (or more).

Hint: For each k in your sequence, define $(x(k), y(k))$, where $x(k)$ and $y(k)$ are lengths of longest increasing and decreasing subsequences starting with k, respectively. Note that each pair is unique. Explain why different numbers in your sequence cannot have identical $(x(k), y(k))$ pairs.

(c) **(3 marks)** In class, we unpacked the Longest Common Subsequence (LCS) problem, where we showed that if our two sequences have size n, then our Dynamic Programming algorithm runs in $O(n^2)$ time.

Let S be your 9-integer sequence from part (a), and let S^* be the same sequence where the 9 numbers are sorted from smallest to largest. Using the LCS algorithm from class, determine the length of the longest common subsequence of S and S^*. (Your answer will be 3. Do you see why?)

Let's look into the general case. Specifically, if S is a sequence of n distinct integers, show that the length of the longest **increasing** subsequence of S must equal the length of the longest **common** subsequence of S and S^*, where S^* is the sorted sequence of S.

BONUS **(1 mark - 3%) [You Can SKIP this]:** The results of part (c) immediately give us an $O(n^2)$ time algorithm to determine the length of the longest increasing subsequence of an input sequence S with n distinct integers. But this is (unsurprisingly) not the optimal algorithm. Your goal is to improve this result.

Given an input sequence S with n distinct integers, design a linearithmic algorithm (i.e., running in $O(n \log n)$ time) to output the length of the longest increasing subsequence of S. Clearly explain how your algorithm works, why it produces the correct output, and prove that the running time of your algorithm is $O(n \log n)$.

Problem #2 – Fractional Knapsack Problem

Difficulty: ■■□□□

The knapsack problem is a classic optimization problem in computer science and mathematics. While we have explored the 0-1 knapsack problem using dynamic programming and brute force approaches, in this question, we will focus on the fractional knapsack problem and its relation to greedy algorithms. Recall that in the fractional knapsack problem, items can be broken into smaller pieces, allowing you to take fractions of items to maximize the total value while staying within the weight limit.

(a) **(3 marks)** Provide a clear pseudocode of a greedy algorithm for the fractional knapsack problem. Your implementation should:

- Take as input:
  - A list of items, where each item is represented by its weight and value
  - The maximum weight capacity of the knapsack
- Return:
  - The maximum value that can be achieved
  - The list of items (and fractions thereof) selected to achieve this maximum value

(b) **(2 marks)** Analyze the time and space complexity of your algorithm. Briefly Justify your answer.

(c) **(3 marks)** Compare and contrast the fractional knapsack problem with the 0-1 knapsack problem. Provide a brief proof of why a greedy approach works for the fractional knapsack but not for the 0-1 knapsack. Your explanation should include:

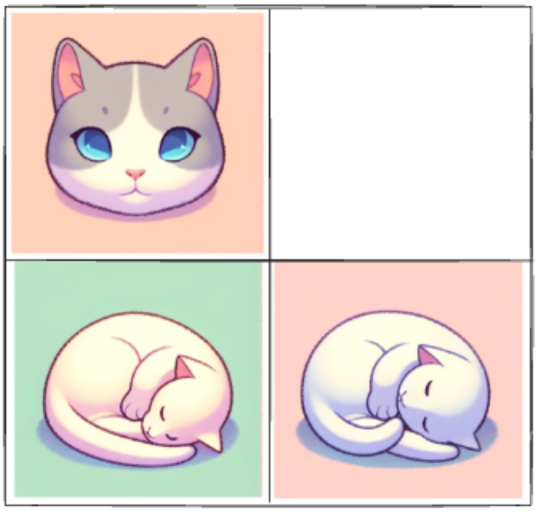
- A brief discussion of the key differences between the two problems
- A counterexample showing why the greedy approach doesn't work for the 0-1 knapsack
- A brief proof on why greedy approach is optimal for the fractional knapsack problem

Problem #3 – Sleepy Cat (Week 6: Graph Algo.)

Difficulty: ■■□□□

For an $m \times n$ matrix, each cell could be either empty, or placed with a lovely cat. While some cats are awake, others are still in dreams. Every minute the sleeping cats next to an awake cat will be woken up (by next to it means the up/down/left/right positions).

In the following example, there're 1 awake cat and 2 sleeping cats. After the first minute, the sleeping cat on the bottom left corner will be woken up. Then after the second minute, the sleeping cat on the bottom right corner will be woken up too. Now all cats are awake for a total of 2 minutes.



(a) **(3 marks)** For such a matrix and a sleeping cat in one cell, how long does it take for it to get woken up? Assume the matrix is $m \times n$ and the cell is located at (x, y), cell values are either 0 (for empty), 1 (for awake cats) or -1 (for sleeping cats). Briefly describe your algorithm and write a clear pseudocode.

(b) **(3 marks)** How long does it take for all cats to be awake? Briefly describe your algorithm and write a clear pseudocode and provide a brief analysis of its running time.

(c) **(2 marks)** Continue with the case in part a, can you track down how the sleeping cat gets woken up from the beginning? There could be multiple ways, but you only need to figure out one of them. In the above example, the path would be $[(0,0),(1,0),(1,1)]$. Can you provide a solution for this? Briefly describe your algorithm and write a clear pseudocode.

Problem #4 – Build a Portfolio on Greedy Programming (Week 5)

Difficulty: □□□□□ (flexible)

LeetCode (www.leetcode.com) is a popular website for Northeastern MSCS students, especially when preparing for job interviews.

`https://leetcode.com/problemset/algorithms/`

There are over a thousand "coding challenges" from which students can practice and improve their skills in Algorithm Analysis and Design, and the website supports numerous programming languages, including C, Java, and Python.

In this Programming Project, you will create a **portfolio** consisting of TWO LeetCode problems on Greedy Programming you will solve over the next two weeks. You can choose ANY set of TWO problems from the following site (click on "Greedy Programming" to filter the results), but see the following note first.

Important Note: You can pick anything **excluding** the Greedy/DP problems we will discuss in class during Week 5 – 6:

- Fibonacci or Climbing Stairs
  - Leetcode 70. Climbing Stairs
  - Leetcode 509. Fibonacci Number
- Rod Cutting
  - Leetcode 1547. Minimum Cost to Cut a Stick (similar problem)
- Longest Common (or Increasing) Sub-sequence
  - Leetcode 1143. Longest Common Subsequence
  - Leetcode 300. Longest Increasing Subsequence
- Activity-Selection
  - Leetcode 435. Non-overlapping Intervals (closest equivalent)
- Knapsack or Coin Change
  - Leetcode 322. Coin Change
  - Leetcode 416. Partition Equal Subset Sum (0/1 Knapsack variant)
- House Robber
  - Leetcode 198. House Robber

NOTE: To maximize your learning and problem-solving skills, we encourage you to explore these concepts through fresh challenges. The idea is for you to get more exposure to problem-solving, so for your own benefit, please stay away from these problems!

[**Read Carefully**] This HW offers a lot of flexibility but there are four important rules:

(I) If the problem took you less than $30-40$ minutes to solve, you may not include it in your portfolio. (Since then the question was an *exercise* for you, rather than a *problem* or a *challenge*.)

(II) You may only include problems you have solved after **Friday, Oct 04, 2024**.

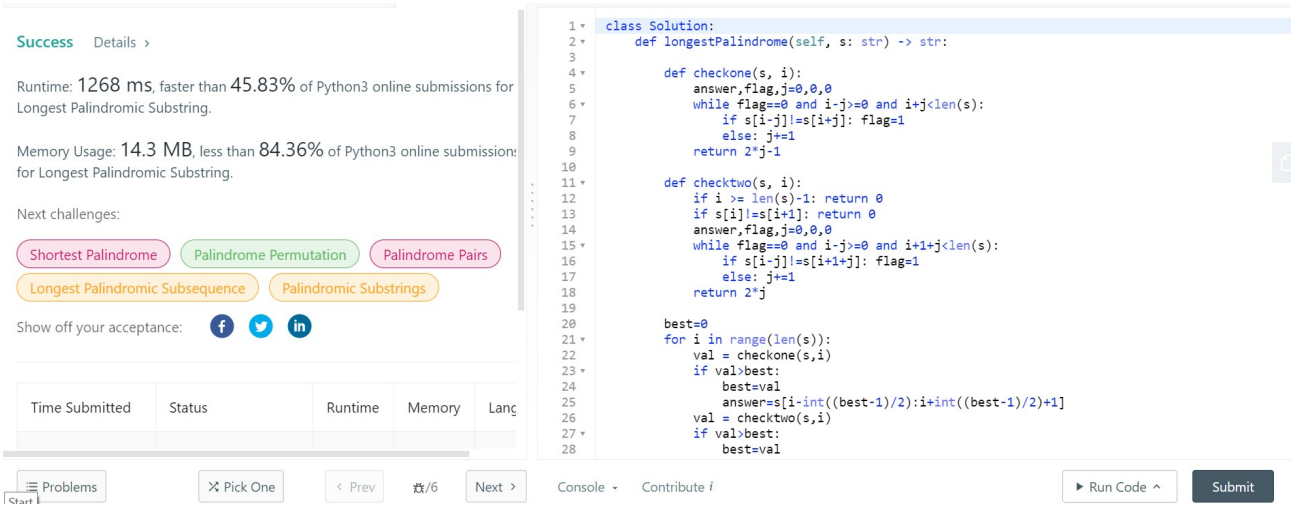
(III If you click on the LeetCode "Solution" or "Discuss" button *before* you solve the problem or look at any online resources for hints to solve a LeetCode problem, especially sites such as Chegg, GitHub, Stack Overflow, and Quora, you cannot include it in your portfolio. (You may look at these sites *after* you solve the problem.)

(IV) Under no circumstance may you use Co-Pilot, LLMs or any other AI-assisted programming tools.

Here is how your portfolio will be assessed.

(a) **(8 marks)** For each of the problems you are including in your portfolio (two problems), provide the problem number, problem title, difficulty level, and the screenshot (showing all the details, submission time, etc) of you getting your solution accepted by LeetCode. You will receive up to 4 marks for each problem you submit.

Here is an example from a sample portfolio: Longest Palindromic Substring (medium)



You will get full credit for *any* correct solution accepted by LeetCode, regardless of the difficulty of the problem, and regardless of how well your runtime and memory usage compares with other LeetCode participants.

(b) **(2 marks)** For **one** of the two problems you solved, explain the various ways you tried to solve this problem, telling us what worked and what did not work. Describe what insights you had as you eventually found a correct solution. Reflect on what you learned from struggling on this problem, and describe how the struggle itself was valuable for you. Reflect on what might be causing the obstacle (e.g. lack of familiarity with a particular data structure, lack of knowledge about a fast and efficient algorithm, etc.)

Note: For your reflections, write a *minimum of 250 words.*