Design and Analysis of Algorithms
Assignment 3
Submission Date: April 20, 2024

Problem 1

Recall the dynamic programming solution of computing the edit distance of two strings s and t having length n and m respectively.

a. Give pseudo code to reconstruct the optimal sequence of operation given the s, t, and table ED in O(m+n) time
b. Show how to compute the Edit distance of two string in O(min(m,n)) space instead of O(mn) space
c. While finding the edit distance between strings $s = s_1 \dots s_n$ and $t = t_1 \dots t_m$ multiple optimal solutions are possible. In the following example, $s =$ TREE and $t =$ TOR, and four optimal solutions are possible, each having cost 3. These are shown below:

T R E E	T R E E	T R E E	T _ R E E
T O R _	T O _ R	T _ O R	T O R _ _
Replace(R,O),	Replace(R,O),	Delete(R),	Insert(O),
Replace(E,R),	Delete(E)	Replace(E,O),	Delete(E),
Delete(E)	Replace(E,R),	Replace(E,R)	Delete(E)

Give a modified version of the bottom-up algorithm written in class that counts the number of unique optimal solutions. Note: it doesn't return all operation sequences, simply, reports how many of them are there. It should take no more than O(nm) time.
First give the recurrence relation, and then give the bottom up solution of the problem.

Problem 2

A sequence of numbers is called a *zig-zag sequence* if the differences between successive numbers strictly alternate between positive and negative. The first difference (if one exists) may be either positive or negative. A sequence with two or fewer than two elements is trivially a zig-zag sequence.

For example, 1,7,4,9,2,5 is a zig-zag sequence because the differences (6,-3,5,-7,3) are alternately positive and negative. In contrast, 1,4,7,2,5 and 1,7,4,5,5 are *not* zig-zag sequences, the first because its first two differences are positive and the second because its last difference is zero.

Give a dynamic programming solution to the problem: Given a sequence of n integers, compute the longest zig-zag subsequence.

First give the recurrence relation, using optimal substructure property and then give the bottom up solution of the problem. Also compute its time complexity.

Problem 3

Consider the following game. A dealer produces a sequence $s_1, \dots, s_n$ of .cards, face up, where each card s_i has a value v_i. Then two players take turns picking a card from the sequence, but can only pick the first or the last card of the (remaining) sequence. The goal is to collect cards of largest total value. (For example, you can think of the cards as bills of different denominations.) Assume n is even.

(a) Show a sequence of cards such that it is not optimal for the first player to start by picking up the available card of larger value. That is, the natural *greedy* strategy is suboptimal.
(b) Give a polynomial time algorithm to compute an optimal strategy for the first player.

Problem 4

Suppose we have a single machine that can make photocopies, and we have a set of requests {1, 2, . . . , *n*}. We are only able to use this machine for the period between time 0 and time *T*, for some number *T*. Each request corresponds to a job that requires time *ti* to make photocopies. If our goal is to process jobs so as to keep the machine as busy as possible up to the "cut-off" *T*, which jobs should we choose. More specifically we want to select a subset of jobs S such that total processing time of all the jobs must not exceed T ($\sum_{i \in S} t_i \leq T$)and we want to maximize the total processing time i.e. $max \sum_{i \in S} t_i$

Devise an efficient solution for this problem. You can assume that all t_i's are integers. First give the recurrence relation, using optimal substructure property and then give the bottom up solution of the problem. Also compute its time complexity.

Problem 5

You are given a rectangular piece of cloth with dimensions X x Y, where X and Y are positive integers, and a list of n products that can be made using the cloth. For each product i you know that a rectangle of cloth of dimensions a_i x b_i is needed and that the final selling price of the product is c_i. Assume the a_i, b_i, and c_i are all positive integers. You have a machine that can cut any rectangular piece of cloth into two pieces either horizontally or vertically. Design an algorithm that determines the best return on the X xY piece of cloth, that is, a strategy for cutting the cloth so that the products made from the resulting pieces give the maximum sum of selling prices. You are free to make as many copies of a given product as you wish, or none if desired.

First give the recurrence relation, using optimal substructure property and then give the bottom up solution of the problem. Also compute its time complexity.

Problem 6

Suppose you're managing a consulting team of expert computer hackers, and each week you have to choose a job for them to undertake. Now, as you can well imagine, the set of possible jobs is divided into those that are *low-stress* (e.g., setting up a Web site for a class at the local elementary school) and those that are *high-stress* (e.g., protecting the nation's most valuable secrets). The basic question, each week, is whether to take on a low-stress job or a high-stress job.

If you select a low-stress job for your team in week *i*, then you get a revenue of *li* > 0 dollars; if you select a high-stress job, you get a revenue of *hi* > 0 dollars. The catch, however, is that in order for the team to take on a high-stress job in week *i*, it's required that they do no job (of either type) in week *i* − 1; they need a full week of prep time to get ready for the crushing stress level. On the other hand, it's okay for them to take a lowstress job in week *i* even if they have done a job (of either type) in week *i* − 1.

So, given a sequence of *n* weeks, a *plan* is specified by a choice of "low-stress," "high-stress," or "none" for each of the *n* weeks, with the property that if "high-stress" is chosen for week *i* > 1, then "none" has to be chosen for week *i* − 1. (It's okay to choose a high-stress job in week 1.) The *value* of the plan is determined in the natural way: for each *i*, you add *li* to the value if you choose "low-stress" in week *i*, and you add *hi* to the value if you choose "high-stress" in week *i*. (You add 0 if you choose "none" in week *i*.)

Devise an efficient algorithm for given sets of values l1, *l*2, . . . , *ln* and *h*1, *h*2, . . . , *hn*, find a plan of maximum value. First give the recurrence relation, using optimal substructure property and then give the bottom up solution of the problem. Also compute its time complexity.

What to Submit: *Give C++ files for the bottom up solutions of all the problems and word/pdf document to answer recurrence and time complexity part. Name it with your roll number and submit it to google classroom.*

Note: Plagiarism of any sort is not acceptable and will be severely punished.