Homework 2
Design and Analysis of Algorithms
Sections BCS-4C, BCS-6A, Spring 2021

Due Friday, April 31, 2021 at 1159pm

In this homework, you will be asked to code in C++ from time to time. **C++ codes are not to be submitted, but you will** provide their outputs on the given inputs. Wherever you're asked to provide pseudo-codes, you must provide them in your hard-copy submissions.

Problem 1 "Re-visiting the LIS: Longest Increasing Sub-sequence problem"

Recall the DP algorithm for the LIS problem. The algorithm used the following recurrence:

$$LIS[i] = \max_{\substack{j<i \\ A[j]<A[i]}} \{LIS[j]\} + 1$$

In addition to the LIS array, we also maintained an array called P (for previous) to keep track of the actual sequence corresponding to the LIS values. Refer to your notes to see how we did that.

(i) In class, we coded this recurrence using bottom-up DP. Here you have to provide pseudo-code for the top-down algorithm using memoization.

(ii) Convert your pseudo-code in (i) into C++ code. Run the code on the files "lis_input1.csv" and "lis_input2.csv", and report the last 10 numbers of the LIS in both cases, plus, the lengths of these LIS.

(The following problem was posed to you as a question on Google Classroom. Here is another chance to get it right.)

It is possible that the input has more than one LIS. Our current algorithm reports any one of them. However, we're interested in knowing how many unique LIS exist in the input.

(iii) Give a modified version of the bottom-up algorithm written in class that reports the number of unique LIS in the input. Note: it doesn't return all these LIS, simply, reports how many of them are there. It should take no more than O(n^2) time. Give pseudo-code.

(iv) To your answers of (ii), also add the number of unique LIS in the two input files. Again, no need to submit the code, simply report the outputs.

(v) Our requirements have changed a bit. We now wish to compute the longest-most-odd increasing subsequence (LmoIS). This is an LIS that has the greatest number of odd numbers among all the LIS in the input. For example, if an input had 1 3 4 8, and 2 3 7 9 both as LIS, our algorithm will report 2 3 7 9 since it has more odd numbers in it. Provide the pseudo-code for this algorithm. It should take no more than O(n^2) time.

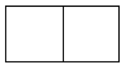
(vi) To your answers of (ii) and (iv) also add the last 10 numbers of the LmoIS (Note: there may exist more than one LmoIS, and you may report the last 10 numbers of any one of them).

Problem 2 "The Strange Room"

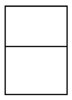
You have a strange room in your house and a strange set of mats to cover its floor with. The room has dimensions $2 \times k$ sq. feet, and each mat is a rectangle of exactly 2 sq. feet. Following pictures explain the situation:

A 2 sq. feet mat may be placed horizontally or vertically as shown below. Note: the colors in the following pictures are only there to show you the orientations of the mats. All mats are identical in reality, having the same color.

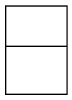
A mat placed horizontally



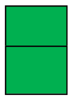
A mat placed vertically



a 2x1 floor

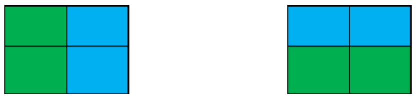


There is only one way to mat this floor, and that is:



a 2x2 floor

There are two ways to mat this floor, and they are:



Can you see that there are 3 ways to cover a 2x3 floor?

How many ways are there to cover a 2x4 floor?

Write a DP algorithm to count the number of ways there are to mat a 2 x n sq. feet floor. Give a recurrence. Write a bottom-up pseudo-code. Implement the code in C++ and report the count for: n = 10, 20 and 30.

Problem 3 "The Strange Highway"

The semester is over! You've rented a car and are ready to set out on a long drive on the Strange Highway. There are n tourist stores on the Strange Highway, numbered 1, 2, …, n and you would want to stop at these and buy some souvenirs (only one souvenir may be bought from a store and all souvenirs everywhere have the same price). You are a greedy shopper and are most interested in maximizing the total discount you get on your shoppings. You know that each store i offers a discount d_i on a souvenir. But it's a strange highway after all. It turns out that you can only buy a souvenir from a store i if you have not bought anything from the previous f_i stores. For example, if $f_6 = 3$ then you can only buy a souvenir from store 6, and get the discount d_6, if you haven't bought anything from stores 3, 4, and 5 (that is, the previous 3 stores from store 6). All the d_i and f_i are known to you in advance (part of input). You have recently learnt the DP technique in your algorithms course and wish to apply it here in order to maximize your total discount under the given constraints.

(i) I will help you by defining the optimal sub-structure. In fact, I will give you two possible definitions:

A. $D[i] =$ max total discount when store i is the last store where you buy a souvenir.
B. $D[i] =$ max total discount for the trip *till store* i whether buying at store i or not.

Provide recurrences for D[i] in both cases A and B. Do both these recurrences produce equally efficient algorithms?

(ii) Provide complete pseudo-code of the bottom-up DP algorithm based on one of the recurrences above. The algorithm should return the optimal total discount as well as the list of stores where to buy the souvenirs to get that discount.

Problem 4 "k heads in n tosses" (problem to be covered in the April 21 lecture)

You have n coins to toss. Each coin has a different probability of turning up heads or tails. Specifically, the probability that the i^{th} coin will turn up heads is p_i (so the probability that it will turn up tails is $1- p_i$. If all the coins were 'fair', then each p_i would be 1/2; however, the coins are not necessarily fair so p_i can be any real number in the interval [0, 1]. You need to compute the probability that when these n coins are tossed one after the other, any k of them will turn up heads.

(a) Give the following sub-structure definition, write down the recurrence to compute it. Also list all the base cases.

P[n, k] = probability that k out of n tosses turn up heads.

(b) Give pseudo-code for the iterative (bottom-up) algorithm to compute P[n, k]. Also show the structure of the matrix P, which should be an O(nk) sized matrix.

(c) Compute P[12, 7], for the following list of probabilities. Just give us the final answer.

p_1	0.57
p_2	0.51
p_3	0.29
p_4	0.39
p_5	0.31
p_6	0.51
p_7	0.55
p_8	0.89
p_9	0.99
p_{10}	0.09
p_{11}	0.01
p_{12}	0.12

(d) How can you change your code in (b) so that it now takes only O(k) space, instead of O(nk)?

Problem 5 “Another Highway”

You are going to Europe by road! Wow. And its one single road (wow!). You start on the road at distance d_1 =0. There are many stopovers on this roads and these are located at distances $d_1 < d_2 < \cdots < d_n$. You are driving the car yourself and will need to stop at some of these stopovers. In the end, you will definitely stop at the final stopover at distance d_n.

Your aim is to travel 500 kms every day (you make stopover at night), as this seems to be the ideal distance that balances the fatigue with the pleasure of travelling the most. However, it’s not always possible to do so because the stopovers are not always 500 kms apart from each other. (If they were, you could travel 500 Kms and make a stop, and so on). Being of a quantitative bent of mind, you have formulated that if you travel α miles in a day, the amount of ‘loss’ (in terms of more fatigue and less joy) for that day is $(500 - \alpha)^2$.

You want to make stops such that this total loss is minimized — that is, the sum, over all travel days, of the daily losses is minimized. Give an efficient DP algorithm that determines the optimal sequence of stopovers at which to stop so as to minimize loss.

(a) Define a subproblem
(b) Write a recurrence
(c) Write a bottom up algorithm to compute the subproblem solutions
(d) Write a method to retrieve the optimal stopover sequence that produces the optimal (minimum) value of loss.

Problem 6 “The War of Zopita”

You were born during the 5th great war between the planets Earth and Zopita. Growing up, you despised the atrocities of interplanetary-conflict and dreamt of universal peace. After a long, long struggle – years of warfare (and a betrayal in love by a Zopita sweetheart) – it all boiled down to you solving a computational problem to put the wretched war to an end! This was what you had to do: Given an unbroken stream of characters, transmitted from the Zopita Headquarters, you had to determine whether the stream was a string (a sentence) of valid English words, or not. For example, the stream “uuydhllhfis” does not give a string of valid words, but the stream, “unleashdeadlyturtleheadsonearthplease”, is just the sentence: “unleash deadly turtleheads on earth please”. To your aide you have a lexicon of concise 40th century English. Given a string w, this lexicon returns true if w is a valid English word and false otherwise in O(1). Given a stream of length n i.e. n characters, describe a DP algorithm that will detect whether or not the input string is a valid English sentence. Your algorithm should also print the valid string. Beware: the future of the Earth is in your hands!

- Define an optimal subproblem, E[k]
- Define a recurrence to compute subproblem E[k] using smaller subproblems.
- Define an array S[k] that can be used to retrieve a valid splitting of the war message into words, if one exists.