National University of Computer and Emerging Sciences, Lahore Campus

	Course:	**Design and Analysis of Algorithms**	**Course Code:**	**CS-2009**
	Program:	**BS (CS, SE)**	**Semester:**	**Spring 2024**
	Due Date:	**March 31, 2024**	**Total Marks:**	**50**
	Section:			
	Exam:	**Assignment#3 (Part-A)**		
	Time:			

Instruction/Notes:

Instructions: Plagiarism in any form will not be tolerated.

You need to submit both a soft copy and hard copy of your handwritten assignment.

Algorithm Design Problems

Question#1: Minimum Sum Partition Problem [Marks: 10]

Given a set of positive integers, the task is to divide this set into two sets S1 and S2 such that the absolute difference between their sums is minimum.

If there are "N" elements in the original input set "S", then if we assume Subset1 has "M" elements, Subset2 must have N-M elements and the value of abs(sum(Subset1) – sum(Subset2)) should be minimum. There may exists multiple subsets providing the optimal solution.

Example: S[] = {10,20,15,5,25} Consider the following partition for an optimal solution. S1[] = {10,20,5} => s1_sum = 35 S2[] = {15,25} => s2_sum = 40 Abs(s1_sum – s2_sum) = 5	Let's consider another partition for an optimal solution. S1[] = {10,25} => s1_sum = 35 S2[] = {20,15,5} => s2_sum = 40 Abs(s1_sum – s2_sum) = 5 //Both the partitions are providing the same answer.

//Attempt any one part of Q#2 i.e. either (a) or (b) for assignment.

Question#2 (a): Longest Increasing Sub sequence [Marks: 10]

You are given an unsorted array of integers of size "N". Your task is to design an algorithm using dynamic programming to determine the length of longest increasing subsequence of the array. You are not allowed to sort the input array.

Subsequence, Increasing subsequence and longest increasing subsequence?

A subsequence of an array is a list of elements of the array where some elements are deleted (or not deleted at all) and they should be in the same order in the subsequence as in the original array. For example, for the array: [4,2,7], the subsequences will be [{4},{2},{7},{4,2},{4,7},{2,7},{4,2,7}} but {7,4} is not a subsequence because its elements are not in the same order as the original array. The subsequences {4,7} and {2,7} are increasing subsequences so the length of longest increasing subsequence will be 2.

Sample Input Array: [3, 10, 2, 1, 20]

Output: 3 => (3,10,20)

Question#2(b): Longest Palindromic Subsequence

You are given an input string "str" of length "N". Your task is to determine the length of longest subsequence in a string that is a palindrome. **Remember, we are talking about subsequence not sub-array**.

Sample Input Array: ABBDCACB

Output: length = 5, longest palindromic subsequence (BCACB)

Sample Input Array: BBABCBCAB

Output: length = 7, longest palindromic subsequence (BABCBAB)

Question#3: Palindrome partitioning:

Given a string **str,** a partitioning of the string is palindrome partitioning if every sub-string of partition is a palindrome, the task is to **find minimum number of cuts needed for palindrome partitioning of the given string**.

Example: str[] = "madamadaseestopspot"
palindromic substrings of given string are => madam| ada | sees | topspot
So minimum 3 cuts are required for palindrome partitioning of the given string.

Question#4: Target Sum Problem [Marks: 10]

You are given an integer array **nums[1...N]** and an integer target "**K**". The task is to build an expression from the given array where we can place a '+' or '-' sign before of an integer. We want to place a sign before every integer of the array and get our required target. We need to **count the number of ways in which we can achieve our required target**.

Example: nums[] = {1,2,3,1}, K = 3
+1-2+3 +1 = 3 = K
-1+2+3-1 = 3 = K
So there are two possible ways to achieve our target.
// You can think of 0/1 knapsack variant where we need to return all possible solutions to the problem in such a way that there are no remaining items left whose weight is less than the remaining capacity of the knapsack.
This problem is pretty much similar to this variant with one key difference. Instead of deciding whether to include an item or not, we now must decide **whether to add or subtract an item**.

Question#5: Attempt any one part: [Marks: 10]

We have studied various dynamic programming algorithms to find optimal solutions like (rod cutting, longest common subsequence, coin change etc.) The optimal solution may not be unique. For example, there can be multiple longest common subsequences.

a) Write an efficient dynamic programming algorithm that **counts the number of different optimal solutions for the rod cutting problem** given the rod length "N" and an array of prices "Pr[1…N]" having prices of various length of rod.
b) Consider a modification of the rod-cutting problem in which, in addition to a price p[i] for each rod, each cut incurs a fixed cost of c. The revenue associated with a solution is now the **sum of the prices of the pieces minus the costs of making the cuts**. Give a dynamic-programming algorithm to solve this modified problem.

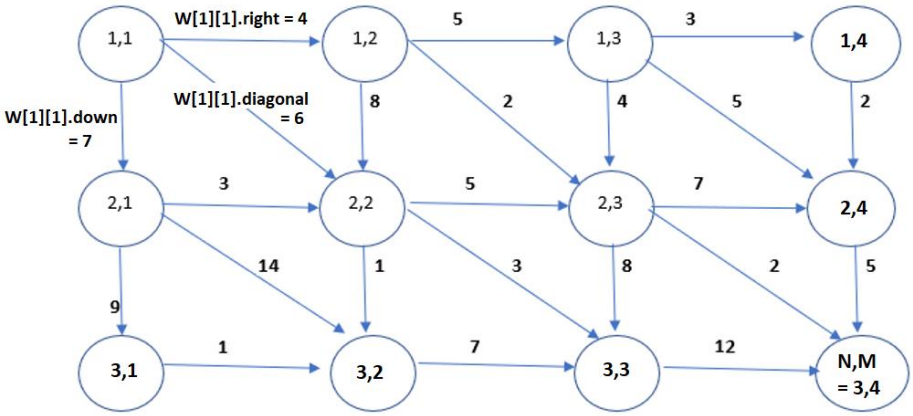
Practice Questions

Pacman Problem [Marks: 10]

Assume that there is a moving object of Pacman in a weighted grid of (N rows and M columns). Pacman starts from the cell (1,1) and can move a maximum distance of 1 unit cell at a time either rightwards, downwards, or diagonally, as shown in the figure below. Note that the object can only move forward and there is no way back. At each move Pacman can score some points. These points for each move are stored in the array W[1…N][1…M].

If Pacman is at cell (1, 1), it can move towards rightwards to cell (1, 2) and earn 4 points which is stored in W[1][1].right, or it can move downwards to cell (2, 1) and earn 7 points which is stored in W[1][1].down or it can move diagonally to cell (2, 2) and earn 6 points which is stored in W[1][1].diagonal. Pacman wants to earn maximum points while going from cell (1,1) to cell (N,M).

Design bottom-up dynamic programming algorithm that calculates the maximum score Pacman can achieve starting from the source cell (1,1) to destination cell (N,M). Weights of edges are given in a 2-D array W[1..N][1…M]. Each entry of this array holds three values i.e. W[i][j].right, W[i][j].down and W[i][j].diagonal.



Q#2:

In a certain course there is an MCQ exam with the help sheet. This help sheet contains **L** lines. There are **N** topics included in the exam. For each topic it is known that there will be **Q[]** questions in the exam and marks for these questions will be guaranteed if the information about the topic is included in the help sheet. For any topic there are **H[]** lines available and the student can either write all of these **H[]** lines in their sheet or do not write anything about that topic. The task at hand is to select the topic whose lines you want to include in the help sheet such that the marks guaranteed is maximized.

For this question you are required to write the pseudo code that maximize the marks guaranteed for above scenario. For each topic, the number of questions and the number of lines for the help sheet are available at the index of array Q and H respectively. Also note that each question carry equal marks.

Q#3:

Consider the following sequence X= {x_1, x_2, x_3, …, x_N} of N numbers such that

$x_1=x_2=1$ and $x_n = y_N - 1 + x_N - 2$

whereas $y_1=y_2=1$ and $y_N = x_N - 1 + y_N - 2$.

a. Write a recursive function to compute x_N

b. What is the time complexity of your recursive function

c. Design a more efficient iterative function to compute x_N