# Muhammad Ahmed 18K-0256

**Abdul Basit 18K-0274**

Design And Analysis of Algorithms Project Report

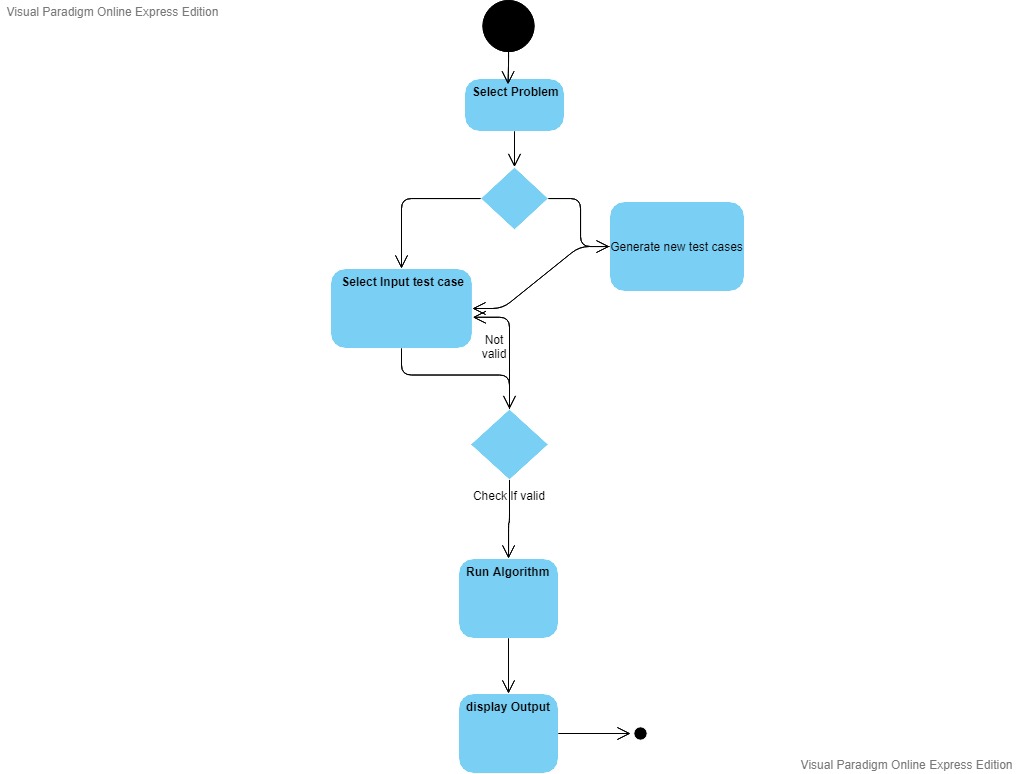
# Abstract

The huge increase in computational power over the recent few years has enhanced a lot our ability to solve complex problems with their performance evaluations in various areas of science. With the recent developments in the field of optimizations, these methods are now become lucrative to make decisions. Dynamic Programming is one of the elegant algorithm design technique and is powerful tool which yields classic algorithms for a variety of combinatorial optimization problems. In this project, different problems are solved with the dynamic programming approach on the randomly generated input data set. Dynamic programming involves two approaches: top down approach and the bottom up approach. We used the bottom up approach to solve problems in polynomial time. It is a technique to save intermediate results in a calculation in any comfortable format so that the result can be used for further computation instead of calculating it all over again at the arrival of an additional input.

# Introduction

The project aims to implement dynamic programming solutions to 10 different problems. The project provides a nice user interface where user can select a desired algorithm with different input test cases. The problems addressed are: LCS, SCS, LIS, Word Break, Levenshtein Distance, Matrix Chain Multiplication, 0-1-knapsack-problem, Partition-problem, Rod Cutting Problem, Coin change-making-problem. Since the problems have overlapping sub-problems, they can be solved in a much more efficient way rather than brute force, saving sub problems results in a table and using it to solve the next larger sub problem. The inputs datasets are generated using the given guidelines in a random fashion.

Proposed System:



# Experimental Setup:

The inputs dataset is generated with the random sequence of alphabets or numbers. We used the random function to generate inputs usually 30 to 100 characters long and stored them in a text file.

Problem 1,2,3:

Alphabets of both group members are taken in random order. For example, “thtiaAaaaiAtmAtAaashedAththhBBadhi” where Member1 = ‘Ahmed’ and Member2 = ‘Basit’

Problem 4,5,7:

A sequence of n random numbers from 0 to 100 (n varies from 30 to 100).

For example: 49 0 78 5 43 10 45 91 100 6 17 98 60 84 44 71 65 17 40 40 82 47 14 12 99 91 92 99 69 43 15 29 86 73 27 45 67 42 95 60 96 95 96 27 75 23 3

Problem 6:

For example:

10 to 100 randomly generated profits and weights ranging from 1 – 100. W = 256.

values: 91 38 51 44 68 61 44 42 59 59 72 31 61 24 83 77 26 31 20 10 1 16 14 93 15 47 72 3 32 36 18 89 59 77 8 71 48 85 5 18 77 33 47 90 80 54 66 97 86 4 30 36 78 93 7 72 89 89 34 30 39 67 34 3

weights: 91 38 51 44 68 61 44 42 59 59 72 31 61 24 83 77 26 31 20 10 1 16 14 93 15 47 72 3 32 36 18 89 59 77 8 71 48 85 5 18 77 33 47 90 80 54 66 97 86 4 30 36 78 93 7 72 89 89 34 30 39 67 34 3

Problem 9:

For example: 49 0 78 5 43 10 45 91 100 6 17 98 60 84 44 71 65 17 40 40 82 47 14 12 99 91 92 99 69 43 15 29 86 73 27 45 67 42 95 60 96 95 96 27 75 23 3

Desired change: 256

**Results and Discussion**

**Longest Common Subsequence:-**

Time Complexity of the above implementation achieved is O(m\*n) which is much better than the worst-case time complexity of Naive Recursive implementation.

S1: reaaaedddSdhdaquSaMiieehhhuunqSShMairdirqaauauuuineS

S2: aMeeaqBdheinSuudMaSaBMMaddaiaiqieiuSSMiMuuMindueeaBaru

Output:

22

## Shortest Common Super sequence:-

## Time complexity of the above solution exponential O(2min(m, n)). Since there are [overlapping subproblems](https://www.geeksforgeeks.org/dynamic-programming-set-1/), we can efficiently solve this recursive problem using Dynamic Programming. Time complexity of the dp based solution is O(mn).

S1: qqSnqrBiSBBenaneaaaBaBaeqeeBnhuMeaenMhgannMBBdi

S2:hqnMSgaeihMMqMhhSeiiSgiheSeaBeegeiruneiddiMgMBnrMddgaaqiSeqggrdBeghueiuaiaiunSMSegiuugSq

Output: 112

## Levenshtein Distance:-

Output shows the levenshtein distance (edit-distance) from the two sequences provided in the input to the Algorithm.

## Longest Increasing Subsequence:-

Output shows the longest increasing subsequence from the sequence provided in the input to the Algorithm and also shows its length too.

Input: 80 1 36 46 72 84 73 77 21 68 1 34 10 20 69 22 90 25 55 72 34 23 3 77 71 43 54 73 34 80 50 32 36 73 63 19

Output: 10

## Matrix Chain Multiplication:-

The output shows the minimum/optimal number of multiplications needed to multiply the chain from the input provided to the Algorithm with a time complexity of O(n^3).

Input: 16 51 26 23 14 3 44 27 15 51 8 32 48 32 84 61 44 35 79 97 34 68 54 78 12 16 98 74 87 60 41 8 97 46 94 40 61 5.

Output: 267591

## knapsack Problem:-

The results obtained was achieved with a time complexity of O(n\*w). The pseudo-polynomial algorithm was implemented.

69 4 8 56 31 31 18 78 20 54 30 15 66 88 63 31 78 50 92 58 60 13

Output: 555

## Partition Problem:-

Output shows that either partition-problem is possible or not with respect to the provided input to the Algorithm.

Input: 66 74 10 65 99 28 20 3 63 17 88 6 52 78 5 15 92 64 20 18 2 54 91 68 59 20 63 7 2 38 22 45 72 47 65 25 92

Output: False

## Rod Cutting Problem:-

Output shows the number of pieces of the rod required with respect to the provided input to the Algorithm in order to get the required value

Input: 39 65 37 84 53 21 63 88 74 41 24 20 15 24 85 47 60 2 4 40 9 38 76 47 14 67 32 28 91 39 66 7 71 91 90 7 19 58 71 28 62 49 97 79 38 47 13 47 20 1 18 11 85 96 32 30 59 93 44 83 96 25 69 18 32 45 35 13 39 53 23 54 71 60 78 10 22 64 94 40 75 94 18 66 72 52 1 1 57 53 83 35 80 37 88 25 83 43 45 2 24 56 70 62 79 60 14 13 76 14 86 29 5 57

Output: 1860

## Coin Change Making Problem:-

Output shows the total number of combinations of the coins which can be arranged with respect to the provided input to the Algorithm in order to get the required value.

Input: 34 11 67 63 2 31 29 26 99 5 48 96 81 19 9 64 44 64 29 97 49 30 48 69 41 43 41 69 2 10 64 30 0 16 5 67 92

Output: 4

## Word Break Problem:-

Output shows that whether the given string can be segmented or not (Break) with respect to the provided input/word to the Algorithm.

# Conclusion:

Dynamic programming has many applications and gives very efficient solutions to many problems. Thus, dynamic programming results in solving many real lives use cases.

# References:

https://medium.com/free-code-camp/demystifying-dynamic-programming-3efafb8d4296