IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 7.39 ⁜ Vol. 11, Issue 1, January 2022

DOI: 10.17148/IJARCCE.2022.11104

Analysis of Dynamic Programming Approach for Optimal Substructure

Barkha Gupta[1]

Programme Assistant, Department of Computer, Agriculture University, Jodhpur, Rajasthan, India[1]

Abstract: Algorithm in general terms is set of rules to be followed for doing a calculation. In computer science, an algorithm is a finite set of steps of well-defined instructions to perform a particular task or to solve a well-defined problem. Various type of algorithm is developed and used in the field of networking. Some may be divide and conquer algorithm other may follows the standards of dynamic programming and some may follow the greedy approach.

Dynamic programming is an algorithmic approach that solves a given problem by dividing or breaking it into subproblems and further dividing that subproblems into smaller subproblems until one reach to the smallest subproblem. It stores that result of the subproblem such that if in near future that same problems encounters then that previously saved output can be reused instead of solving that problem again and again. Hence it saves sand increase the time efficiency and uses the optimal solution approach.

Keywords: Bottom-up Approach, Optimal Dynamic Programming approach, Standard dynamic programming, Fibonacci Sequence, Optimal substructure programming, Tower of Hanoi Problem, Word Break Problem, All Pair shortest Path Algorithm.

I. INTRODUCTION

Dynamic Programming is a computer programming method. This method is developed in 1950 by the Richard Bellman. Dynamic programming is a process of simplifying the process by dividing the complex and big problems into smaller subproblems in a recursive manner. Recursive here implies repeated calls for same input. The simple idea behind this is to store the result of previously solved subproblems and reuse that solution instead of solving the same type of problem again and again. Smaller subproblems are solved independently. Solution is achieved by optimum solution of smaller subproblem. By memorization these solutions are remembered and used for similar subproblems.

II. CATEGORIES OF DYNAMIC PROGRAMMING

(i) **Optimization Problem –** A problem which is in search of finding the best solution from all the feasible solution is known as optimization problem.

(ii) **Combinatorial Problem -** Combinatorial problems is a finite collection of objects and a set of constraints It is the solution of finding an object of the collection that satisfies all constraints and that also optimizes some objective function. It expects you to figure out the number of ways to do something, or the probability of some event happening.

III. APPROACHES OF SOLVING DYNAMIC PROGRAMMING

(i) **Top-down Approach –** Top-down approach begins with the general features and moves towards the specification. It seeks to identify the big picture and al its components. Initially it is formulated without going into details then it divides the problem into different parts and defining it into more details and specification. It flows from the upward to the downward direction.

(ii) **Bottom-up Approach-** Reverse of top-down approach is the bottom-up approach. It begins with the specification or more detailed and moves towards the general. In this approach, individual parts of the system are specified in detail in the beginning. These parts are linked to form larger components, which are in turn linked until a complete system is formed. It flows from the downward to the upward direction.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 7.39 ⁜ Vol. 11, Issue 1, January 2022

DOI: 10.17148/IJARCCE.2022.11104

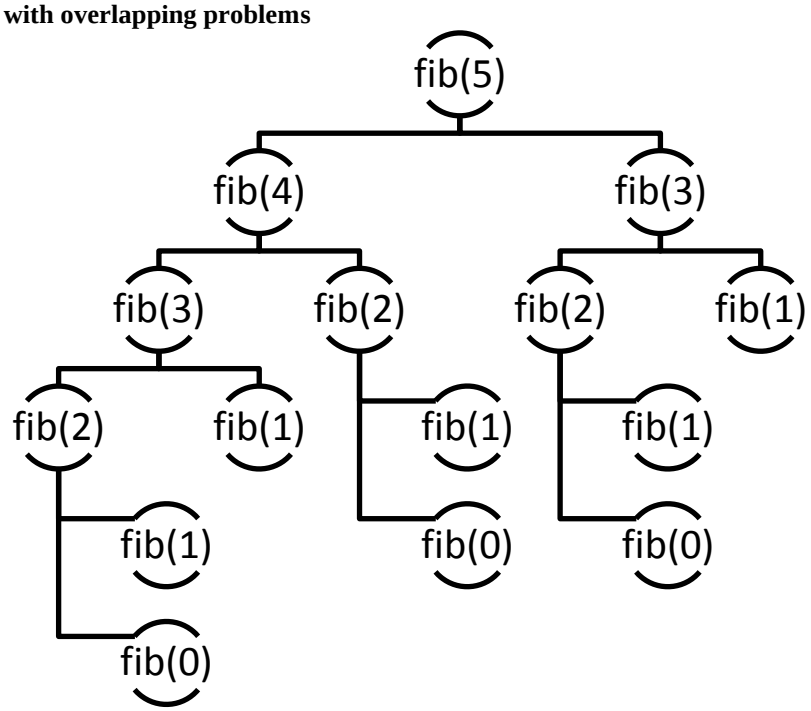
IV. FIBONACCI SEQUENCE

When Fibonacci numbers formed a sequence, it is termed as Fibonacci sequence. Fibonacci series is the process of generating the next number by adding the two preceding ones. Fibonacci series starts from two numbers that are 0 and 1 respectively. Here, Dynamic Programming combines solutions to sub-problems. Dynamic Programming is used when solutions of the same subproblems are reused or needed again and again. In dynamic programming, computed solutions to subproblems are stored in a table so that the same type of problems are not recomputed or resolved again and again. So Dynamic Programming is not at all useful when there are no common or overlapping subproblems because there is no sense of storing the solutions if they are not needed again in the near future.

Fibonacci series first element = 0
Fibonacci series second element = 1
Fibonacci series first element + second element = 1 and so on
Fibonacci Series will be 0, 1, 1, 2, 3, 5, 8, 13, 21……n

So, while solving the Fibonacci series problem by the dynamic programming, we will store the previously solved subproblem into the list say fib []. To solve the problem, we will execute the function let say solve () to check if that subproblems is already solved or not. If that problem is previously solved then we will not solve it again and we will store the solution in the list fib [].

Recursive tree with overlapping problems



As here in the above mentioned diagram fib (3) is called twice, fib (2) called thrice. So here we will make sure that the overlapping problem is solved only one time by the mechanism of dynamic programming.

V. KNAPSACK PROBLEM

Knapsack basically means a bag or understand a luggage bag. Knapsack problem is given with two array value [1…100] and weight [1…100]. Value implies value of the item and weight is associated with its value. Maximum weightage of the bag (W) is also given. This is a standard greedy algorithm. The objective of knapsack algorithm is to find out the maximum value subset of value [ ] such that sum of the weights of this subset is smaller than or equal to Weightage of bag (W).

Let's understand knapsack problem with an example
Maximum capacity of the sack is 8 and find out the maximum value within the given weight. Value and its associated weight are given below.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 7.39 ⁜ Vol. 11, Issue 1, January 2022

DOI: 10.17148/IJARCCE.2022.11104

Value (V)	10	18	5	25	15
Weight (W)	5	2	6	4	1

Arrange them in the ascending order of their weight

Value (V)	15	18	25	10	5
Weight (W)	1	2	4	5	6

Weight Limit	0	1	2	3	4	5	6	7	8
V = 15, W=1	0	15	15	15	15	15	15	15	15
V=18, W=2	0	15	18	33	33	33	33	33	33
V=25, W=4	0	15	18	33	25	40	43	58	58
V= 10, W=5	0	15	18	33	25	10	25	28	43
V=5, W= 6	0	15	18	33	25	10	5	20	23

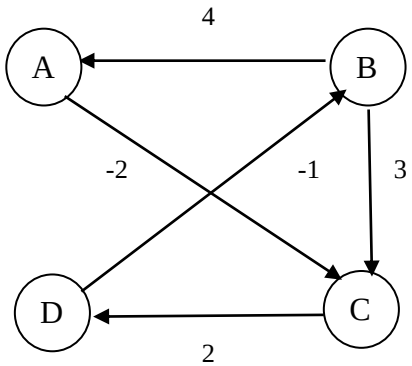
So, the maximum weight of the sack is 58

To, solve the knapsack problem using the dynamic programming, we need to do followings: -

- Divide the problem into smaller subproblems until you reach the smallest subproblem.
- Find the solutions of the smallest subproblems.
- Create a table that stores the solutions of subproblems.
- Reuse the solved subproblems, to solve further problems and store its result into the table.
- Repeat this until you find the solution of the original problem.

VI. ALL PAIR SHORTEST PATH ALGORITHM

The all-pair shortest path problem is used to find the minimum distance from any node to all the other node given in a graph. It works on the weighted graph. The all-pair shortest path algorithm sometimes also known as Floyd-Warshall algorithm. Let understand this by an example



The output adjacency matrix for the all-pair shortest path is

	A	**B**	**C**	**D**
A	0	-1	-2	0
B	4	0	2	4
C	5	1	0	2
D	3	-1	1	0

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 7.39 ⁜ Vol. 11, Issue 1, January 2022

DOI: 10.17148/IJARCCE.2022.11104

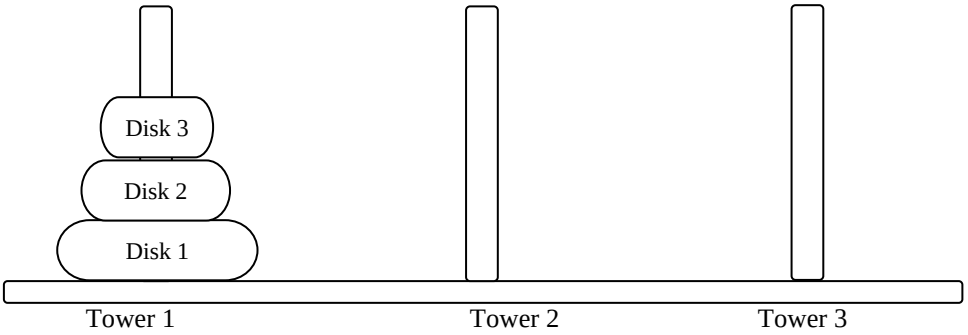
vertex A to vertex B is [0 —> 2 —> 3 —> 1]
vertex A to vertex C is [0 —> 2]
vertex A to vertex D is [0 —> 2 —> 3]
vertex B to vertex A is [1 —> 0]
vertex B to vertex C is [1 —> 0 —> 2]
vertex B to vertex D is [1 —> 0 —> 2 —> 3]
vertex C to vertex A is [2 —> 3 —> 1 —> 0]
vertex C to vertex B is [2 —> 3 —> 1]
vertex C to vertex D is [2 —> 3]
vertex D to vertex A is [3 —> 1 —> 0]
vertex D to vertex B is [3 —> 1]
vertex D to vertex C is [3 —> 1 —> 0 —> 2]

VII. TOWER OF HANOI PROBLEM

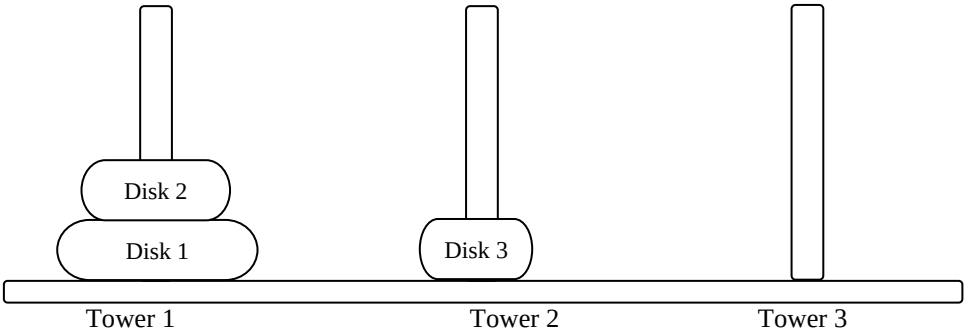
Tower of Hanoi is a mathematical puzzle in which there are towers also known as pegs and more than one ring will be there on the pegs. These rings are of different size and stacked over the peg in ascending order which implies biggest size ring will be at bottom and smallest size ring will be at top. The result or the output what we want to attain is that to moves all the rings from one tower to another by following follows rules: -

(i) Only one disk can be moved at any point of time.
(ii) Only the top most disk can be moved. No disk from in-between can be moved.
(iii) No large disk can be placed over smaller disk.

Let's understand Tower of Hanoi problem with example and target is to move all the disk to tower 2.



Step 1:



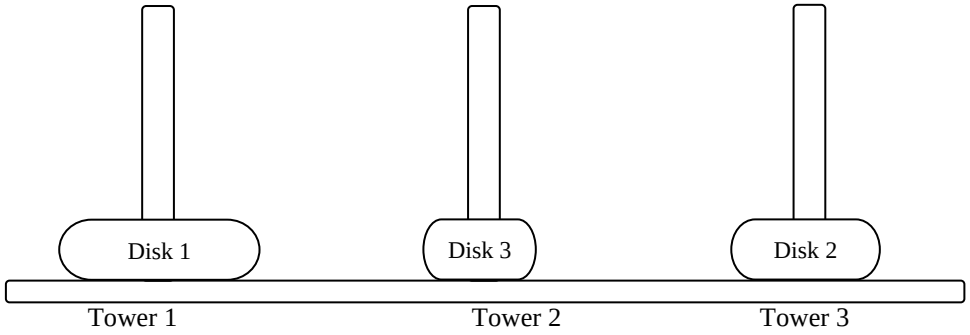
IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

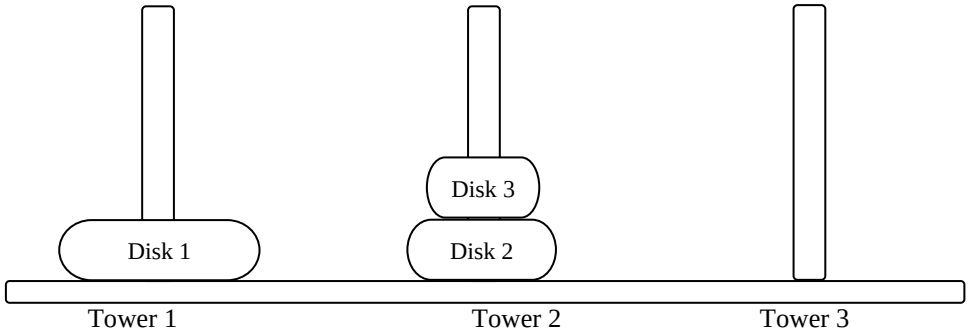
Impact Factor 7.39 ⁜ Vol. 11, Issue 1, January 2022

DOI: 10.17148/IJARCCE.2022.11104

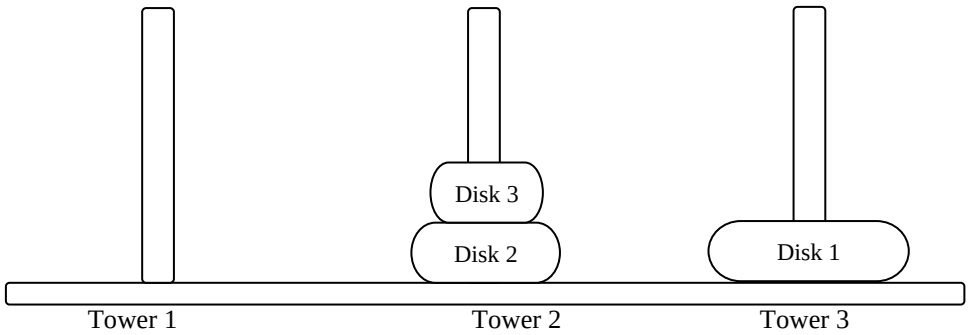
Step 2:



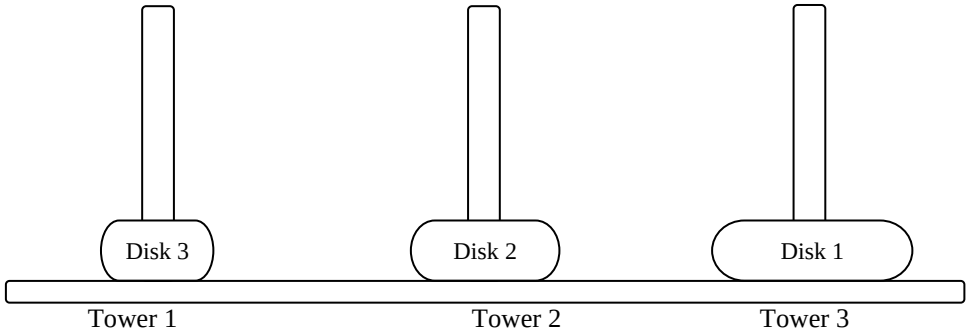
Step 3:



Step 4:



Step 5:



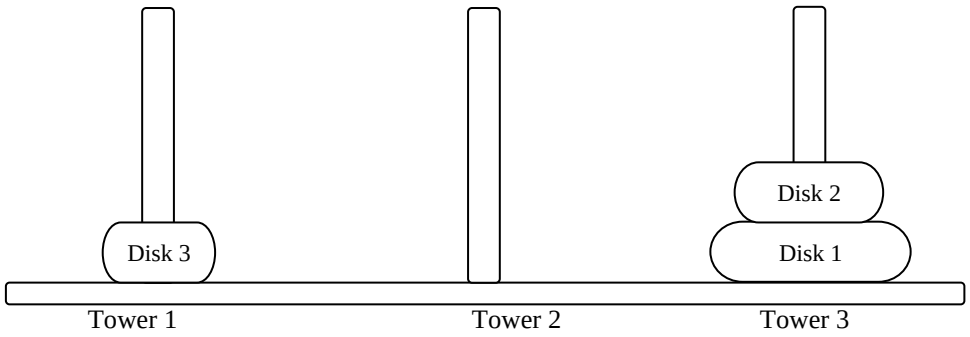
IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

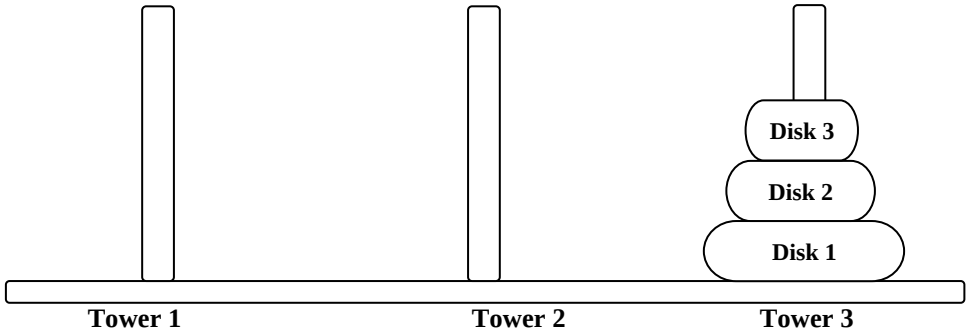
Impact Factor 7.39 ⁜ Vol. 11, Issue 1, January 2022

DOI: 10.17148/IJARCCE.2022.11104

Step 6:



Step 7:



The process of Tower of Hanoi with 3 towers and 3 disks are completed in 7 steps.

VIII. ROD CUTTING PROBLEM

Rod cutting algorithm is a process of cutting a rod into smaller pieces and earn the maximum profit. Given a rod of length n inches and an array of prices that includes prices of all pieces of size smaller than n. We have to determine the maximum value obtainable by cutting up the rod and selling the pieces.

Let's understand Rod cutting problem with an example

Length	1	2	3	4	5	6	7	8	9
Price	1	3	5	8	8	9	10	11	13

Rod length = 5

Then possible number of solutions are as follows: -

Solution No.	**No. of cuts**	**Cuts**	**Profit**	**Total Profit**
1	1	5	8	8
2	2	3,2	5+3	8
3	**2**	**4,1**	**8+1**	**9**
4	3	1,1,3	1+1+5	7
5	3	1,2,2	1+3+3	7
6	4	1,1,1,2	1+1+1+3	6
7	5	1,1,1,1,1	1+1+1+1+1	5

So, to determine maximum profit the number of cuts will be 2 and the maximum profit will be 9 with the solution no. 3 We can get the best price by making a cut at different positions and comparing the values obtained after a cut. We can recursively call the same function for a piece obtained after a cut and thus this can be implemented using dynamic programming as the task is going to repeat itself and hence, subproblem solution can be reused again and again.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 7.39 ⁜ Vol. 11, Issue 1, January 2022

DOI: 10.17148/IJARCCE.2022.11104

IX. LEVENSHTEIN DISTANCE PROBLEM

Levenshtein distance problem is a mechanism to find out that how the two strings are different from each other. Levenshtein distance problem is also known as edit distance problem. In which two different strings will be given and we have to find out minimum number of operations required to convert the one string into the another given string. The following operations are allowed on the strings that are as follows: -

(i) Insertion
(ii) Deletion
(iii) Substitution

Each above mentioned operation has a similar unit cost.

Let's understand Levenshtein distance problem with an example. We have to change string 1 into string 2

Input	**String 1**	**String 2**	**No. of cost associated**	**Operation Details**
1	Bus	Bass	2	(i) 'a' has to be substitute in place of 'u' (ii) 's' to be inserted at end
2	Chalk	Crook	3	(i) 'r' has to be substitute in place of 'h' (ii) 'o' has to be substitute in place of 'a' (iii) 'o' has to be substitute in place of 'l'
3	Car	Cab	1	(i) 'b' has to be substitute in place of 'r'

X. COIN CHANGE MAKING PROBLEM

Coin change making problem is a very common problem in which unlimited supply of coins are given with different denominations. In this case we have to find the minimum numbers of coins used while generating desired change. Here particular denominations can be used multiple or infinite number of times. This problem has the optimal substructure property as the problem can be solved using solutions to subproblems. This problem is also recursive in nature thus implements the overlapping subproblems.

Let's understand coin change making problem with an example

Let take the supply of coins are {1, 3, 5, 7}

Desired Change	Coins required set 1	Coins required set 2	Coins required set 3	Coins required set 4	Coins required set 5	Minimum no. of coins required
12	5, 5, 1, 1	7, 3, 1, 1	3, 3, 3, 3	7, 5	5, 3, 3, 1	2
15	5, 5, 5	7, 7, 1	7, 3, 5	3, 3, 3, 1, 5	5, 5, 3, 1, 1	3
18	7, 7, 3, 1	5, 5, 5, 3	7, 5, 5, 1	5, 5, 5, 1, 1, 1	3, 3, 3, 3, 3, 3	4

XI. WORD BREAK PROBLEM

Word break problem is a problem in which a string is given and a word of dictionary is also given. In this word break problem, one has to determine that whether the string is separated or segmented with the space following the sequence of dictionary words. We are allowed to reuse same dictionary word multiple times. If the string can be separated, then that string will be accepted else it will be rejected. To do this we will consider all the prefixes of the current string one by one and check whether that prefix exists in the dictionary words or not. If that refix exists, we will add it to the output string and move with the next part of the remaining string. It will end when the string becomes empty and we will get the complete output as a result. This problem has a repetition or recursion to solve a problem hence recursive. It also follows the optimal substructure as the problem is divided into smaller subproblems which can be further divided into until the smallest problem is reached. This problem also exhibits overlapping subproblems, so we will end up solving the same subproblem again and again. If we draw the recursion tree, we can see that the same subproblems are getting computed repeatedly.

Let's understand word break problem with some examples

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 7.39 ⁜ Vol. 11, Issue 1, January 2022

DOI: 10.17148/IJARCCE.2022.11104

Example 1:
Input string: s = "welcome"
Word Dictionary = ["come","wel"]
Output: True/String Accepted
Explanation: Return true because "welcome" can be segmented as "wel come".

Example 2:
Input string: s = "applepineapple"
Word Dictionary = ["apple","pine"]
Output: True/String Accepted
Explanation: Return true because "applepineapple" can be segmented as "apple pine apple".

Example 3:
Input string: s = "catsandog"
Word Dictionary = ["cats","dog","sand","and","cat"]
Output: False/not accepted

XII. CONCLUSION

Dynamic Programming is the concept of breaking problems into smaller subproblems and save the result for the future use instead of solving the same problem again and again. Dynamic programming algorithm will examine the problems and its subproblems which has been solved previously and will combine its result for the further usage. It also makes sure that it gives the best possible solution within the less time frame by expanding and repeating the same solution again and again.
Therefore, dynamic programming algorithm is also used for optimization. Dynamic programming is heavily and commonly used in real life. It also used in computer network, find the common subsequence, routing, artificial intelligence, machine learning and many other fields.
Further the dynamic programming works on optimization of sub-problems which optimizes the overall solution also known as optimal substructure property.
There is no single approach or unified approach to solve and get the result of dynamic programming. There is multiple condition which may appear during the solving process. Initially after getting the problem, the foremost basic step is to rectify or recognize that the problem can be solved or comes under the category of dynamic programming and this is the most difficult part of problem solving.

XIII. ACKNOWLEDGEMENTS

Author is grateful to Agriculture University, Jodhpur for encouraging writing in my field.

XIV. REFERENCES

[1] Richard Bellman. Dynamic Programming. Publisher Dover Publications Inc, Reprint Edition.
[2] Moshe Sniedovich, Taylor and Francis. Dynamic Programming. Publisher Taylor and Francis.
[3] Meenakshi and Kamal Rawat. Dynamic Programming for Coding Interviews A bottom-up approach to problem solving. Publisher Notion Press Publisher.
[4] A. Lew. H. Mauch. Dynamic Programming: A Computational Tool. Publisher Springer-Verlag Berlin and Heidelberg GmbH and Co.
[5] Sudhir Kumar Pundir. Non-Linear and Dynamic Programming. Publisher CBS Publishers and Distributors.

