Experiment. No. 04

Title : write a program to solve a 0-1 knapsack problem using dynamic programming or branch and bound strategy.

objective : To understand and solve 0-1 knapsack problem using dynamic programming

Theory :

what is Dynamic Programming :

Dynamic Programming is also used in optimization problems. Like divide and conquer method, Dynamic programming algorithm solves each sub-problems just once by combining the solutions of sub-problems.

Dynamic Programming algorithm solves each sub-problem just once and then uses its answers in a stable, thereby avoiding the work of re-computing the answer everytime.

Two main properties of a problem suggest that the given problem can be solved using Dynamic Programming. These properties are overlapping sub-problems & optimal substructure.

For example, Binary search does not have overlapping sub-problem, whereas recursive problems of Fibonacci numbers have many overlapping sub-problems.

steps of Dynamic Programming Approach

It is designed using the following f

1. characterize the structure of an optima
2. Recursively define the value of an opti solution.
3. Compute the value of the optimal soluti typically in a bottom-up fashion.
4. construct an optimal solution from the computed information.

Application of Dynamic Programming

- matrix chain multiplication
- Longest common subsequence
- Travelling salesman Problem

knapsack problem:

- A knapsack with limited weight capac
- few items each having some weigh value

knapsack problem has the following tw variants.

1. Fractional knapsack problem
2. 0/1 knapsack problem

0/1 Knapsack problem using Greedy method

consider
Knapsack weight capacity = w
No. of items each having some weight & value=n

step 1 : Draw a table say 'T' with (n+1) no. of rows & (w+1) no. of columns
Fill all the boxes of 0^{th} row & 0^{th} column with zeros.

	0	1	2	3	.....	w
0	0	0	0	0	.....	0
1	0					
2	0					
	....					
n	0					

T-table

step 2 - start filling the table row wise top to bottom from left to right.

Use the following formula -

$$T(i,j) = \text{mean}\{T(i-1,j), \text{value} : +T(i-1,j-\text{weight} :)\}$$

This step leads to complexity filling the table
Then, value of the last box represents the maximum possible value that can be put into the

knapsack

step 3 - To identify the items that must be put into the knapsack to obtain the profit.

- Consider the last column of the table
- start scanning the entries from bottom to
- on encountring an entry where value is as the value stored in the entry immedia above it, make the row label of that entry
- After all the entries are scanned, the labels represent the items that must be into the knapsack.

Time complexity :

It takes $\theta(n w)$ time to fill $(n+1)(n+1)$ ta
It takes $o(n)$ time for tabing the solu
taking process traces the n rows
Thus overall $o(nw)$ time is taken to so

conclusion -

In this way, we have studied the concept of 0/1 knapsack using dynamic appr