Assignment 4

Rajdhani
DATE / /

Title :-

Knapsack problem using Dynamic programming OR Branch - N - Bound

Aim :-

Write a program to solve a 0-1 Knapsack problem using Dynamic programming or Branch-n-Bound strategy.

Objectives :-

1. To understand dynamic programming.
2. To understand Branch - n - Bound.
3. To solve Knapsack problem using dynamic programming

Pre-requisites :-

Basic knowledge of Data structures and Algorithms.

Theory :-

0-1 Knapsack problem:

It is an optimization challenge where you have a set of items with different weights and values and you want to find the combination of items to maximize their total values within a Knapsack's weight and limit.

You may either take an item or leave it and must not exceed knapsack's capacity.
The problem is used in resource allocation, finance and Logistics.

✱ Branch-n-Bound strategy:-

The branch-n-bound strategy is an optimization technique that breaks a problem into smaller sub-problems estimates their potential and solves them & backtracks when necessary.

Approach to solve 0-1 knapsack problem using Branch-n-Bound strategy:

Step 1: Sort all items based on their value/weight (v/w) ratio.

Step 2: Insert a dummy node into the priority queue

Step 3: Repeat the following steps until the priority queue is empty.

a. Extract the peak element form the priority queue and assign it to the current node.
b. If the upper bound of the current node is less than minLB, the minimum lower bound of all the nodes

explored, then there is no point of exploration. So, continue with the next element. The reason for not considering the nodes whose upper bound is greater than minLB is that the upper bound stores the best value that can be achieved.

c. Update the path array.

d. If the current nodes level is N, then check whether the lower bound of the current node is less than final LB, minLB of all paths that reached the final level. If it is true, update the final path and finalLB, otherwise continue with next element.

e. Calculate the lower bound and upper bounds of the right child of the current node.

f. If the current item can be inserted, then calculate its lower and upper bound of the left child of current node.

g. Update the minLB and insert the children if the upper bound is less than minLB

Example -

Inputs - N=4, C=15, V[ ] = {10,10,12,18}

W[ ] = {2,4,6,9}

Solution -

Output: 1101

Maximum profit = 8

Explanation:

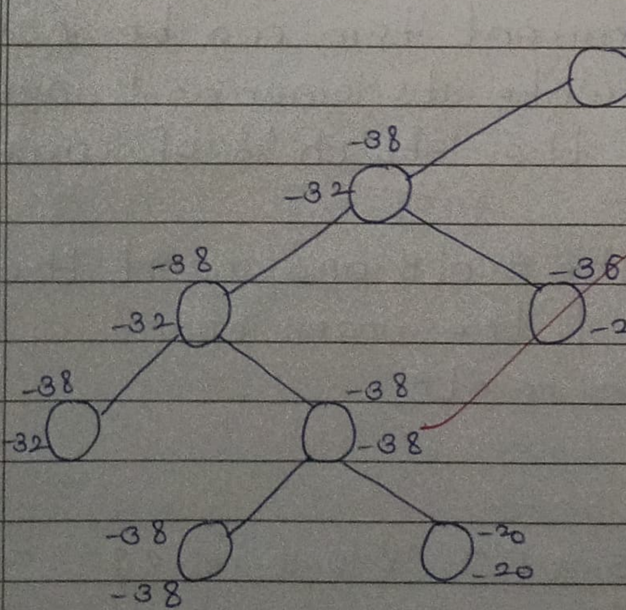
1 in the output indicates that the item is included in the knapsack.
While 0 indicates that the item is excluded.

(1 1 0 1) → cost = 2 + 4 + 9 = 15
Profit = 10 + 10 + 18 = 38

(0 0 1 1) → cost = 6 + 9 = 15
Profit = 12 + 18 = 30

(1 1 1 0) → cost = 2 + 4 + 6 = 12
Profit = 10 + 10 + 12 = 32

Hence, the maximum profit possible is 38.



Left and right branch at ith level stores the max. obtained profit including & excluding the ith item.

Rajdhani
DATE / /

Conclusion:-

Thus, we implemented the Branch-n-Bound method to solve the 0-1 Knapsack problem