KNAPSACK PROBLEM

The knapsack problem is a problem in combinatorial optimization: Given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible. It derives its name from the problem faced by someone who is constrained by a fixed-size knapsack and must fill it with the most valuable items. The problem often arises in resource allocation where the decision makers must choose from a set of non-divisible projects or tasks under a fixed budget or time constraint, respectively.

TYPES

- Fractional Knapsack Problem

- 0 / 1 Knapsack Problem

- FRACTIONAL KNAPSACK PROBLEM

The fractional knapsack problem means that we can divide the item. For example, we have an item of 3 kg then we can pick the item of 2 kg and leave the item of 1 kg. The fractional knapsack problem is solved by the Greedy approach.

Problem Statement:

Given a set of N items each having value V with weight W and the total capacity of a knapsack. The task is to find the maximal value of fractions of items that can fit into the knapsack.

Brute Force Approach

The most basic approach is to try all possible subsets and possible fractions of the given set and find the maximum value among all such fractions.

The time complexity will be exponential, as you need to find all possible combinations of the given set.

Efficient Approach (Greedy)

The Fractional Knapsack problem can be solved efficiently using the greedy algorithm, where you need to sort the items according to their value/weight ratio.

Algorithm

- → **Sort the given array of items according to weight / value (W /V) ratio in descending order.**
- → **Start adding the item with the maximum W/V ratio.**
- → **Add the whole item, if the current weight is less than the capacity, else, add a portion of the item to the knapsack.**
- → **Stop, when all the items have been considered and the total weight becomes equal to the weight of the given knapsack.**

- 0 / 1 KNAPSACK PROBLEM

The 0/1 knapsack problem means that the items are either completely or no items are filled in a knapsack. For example, we have two items having weights 2kg and 3kg, respectively. If we pick the 2kg item then we cannot pick 1kg item from the 2kg item (item is not divisible); we have to pick the 2kg item completely. This is a 0/1 knapsack problem in which either we pick the item completely or we will pick that item. The 0/1 knapsack problem is solved by the dynamic programming.

Dynamic Programming Approach

Given a knapsack problem with n items and knapsack weight of W.

We will first compute the maximum benefit, and then determine the subset.

To use dynamic programming, we solve smaller problems and use the optimal solutions of these problems to find the solution to larger ones.

Steps for Solving

Steps 1:

→ ❖Draw a table say 'T' with (n+1) number of rows and (w+1) number of columns.
→ ❖Fill all the boxes of 0th row and 0th column with zeroes as shown-

Steps 2:

→ Start filling the table row wise top to bottom from left to right.
→ Use the following formula-
→ Here, T(i , j) = maximum value of the selected items if we can take items 1 to i and have weight restrictions of j.
→ This step leads to completely filling the table.
→ Then, value of the last box represents the maximum possible value that can be put into the knapsack.

Steps 3:

→ To identify the items that must be put into the knapsack to obtain
→ that maximum profit, Consider the last column of the table.
→ Start scanning the entries from bottom to top.
→ On encountering an entry whose value is not same as the value stored in the entry immediately above it, mark the row label of that entry.
→ After all the entries are scanned, the marked labels represent the items that must be put into the knapsack.

Dynamic Programming Approach

-0/1 Knapsack Problem

➢Assume a subproblem in which the set of items is restricted to {1,..., i } where i □ n, and the weight of the knapsack is w, where 0□ w □ W.

➢Let B [i, w] denote the maximum benefit achieved for this problem.

➢Our goal is to compute the maximum benefit of the original problem B [n, W]

➢We solve the original problem by computing B[i, w] for i = 0, 1,..., n and for w = 0, 1,...,W.

➢We need to specify the solution to a larger problem in terms of a smaller one

Knapsack Problem by DP

Algorithm

```
Algorithm DPKnapsack(w[1..n], v[1..n], W)
var V[0..n,0..W], P[1..n,1..W]: int
for j := 0 to W do
        V[0,j] := 0
for i := 0 to n do Running time and space:
        V[i,0] := 0 O(nW)
for i := 1 to n do
        for j := 1 to W do
                if w[i]≤ j and v[i] + V[i-1,j-w[i]] > V[i-1,j] then
                    V[i,j] := v[i]+V[i-1,j-w[i]]; P[i,j] := j-w[i]
                Else
                    V[i,j] := V[i-1,j]; P[i,j] := j
return V[n,W] and the optimal subset by backtracing
```