ARYAN DEEPAK PAWAR
COMPBEB1326
72017912C

Page No. Date

GROUP A : ASSIGNMENT No. 3

Title: Fractional Knapsack Problem

Objective: To solve a Fractional Knapsack Problem using greedy strategy.

Problem Statement:

Write a program to solve a Fractional Knapsack problem using a greedy method.

Software & Hardware Requirement:

1. Desktop / Laptop
2. Any Operating System
3. Python
4. IDE or Code Editor

Theory:

Knapsack Problem:

Given a set of items, each with a weight & a value, determine a subset of items to include in a collection so that the total weight is less than or equal to a given limit & the total value is as large as possible.

The knapsack problem is in combinatorial problem optimization problem. It appears as a subproblem in many, more complex mathematical models of real-world problems. One general approach to difficult problems is to identify the most restrictive constraint, ignore the others, solve a knapsack problem, & somehow adjust the solution to satisfy the ignored constraints.

Applications:

In many cases of resource allocation along with some constraint, the problem can be derived in a similar way of knapsack problem. Following is a set of examples:

1> Finding the least wasteful way to cut raw materials.

2> Portfolio optimization

3> Cutting stock problems

Problem Scenario:

A thief is robbing a store & can carry a maximal weight of W into his knapsack. There are n items available in the store & weight of i^{th} item is w_i & its profit is p_i. What items should the thief take?

In this context, the items should be selected in such a way that the theif will carry those items for which he will gain maximum profit. Hence, the objective of theif is to maximize the profit.

Fractional Knapsack:

In this, items can be broken into smaller pieces, hence the theief can select fraction of items.

According to problem statement,

There are n items in the store

Weight of i^{th} item $w_i > 0$

Profit for i^{th} item $p_i > 0$ &

Capacity of the knapsack is W

So optimal solution will be,

$$\boxed{\text{maximize} \sum_{i=1}^{n} (x_i p_i)} \quad \text{where} \sum_{i=1}^{n} (x_i w_i) \leq W$$

Algorithm:

```
Greedy-Fractional-Knapsack (w[1..n], P[1..n], w)
  for i=1 to n
      do x[i] = 0
  weight = 0
  for i=1 to n
      if weight + w[i] ≤ W then
          x[i] = 1
          weight = weight + w[i]
      else
          x[i] = (W - weight) / w[i]
          weight = W
          break
  return x
```

Analysis:

If the provided items are already sorted into a descending order of $\frac{P_i}{w_i}$, then the whileloop takes a time in O(n),

Therefore, the total time including the sort is in O(n log n)

Testcase:

Input:

Item	A	B	C	D
Profit	280	100	120	120
Weight	40	10	20	24
Ratio	7	10	6	5

Expected Output = 440

Actual Output: 440

i.e. (100 + 280 + 120*(10/20)) = 380 + 60 = 440

So as expected as actual output are same, testcase is passed.

Conclusion:

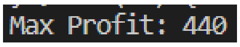
Successfully implemented fractional knapsack problem using greedy method.

28/9/22

Code:

```
#include <bits/stdc++.h>

using namespace std;
struct Item
{
    int value, weight;

    Item(int value, int weight): value(value), weight(weight)
    {
    }
};

bool cmp(struct Item a, struct Item b)
{
    double r1 = (double)a.value / a.weight;
    double r2 = (double)b.value / b.weight;
    return r1 > r2;
}

double fractionalKnapsack(struct Item arr[],int N, int size)
{
    sort(arr, arr + size, cmp);

    int curWeight = 0;

    double finalvalue = 0.0;

    for (int i = 0; i < size; i++)
    {

        if (curWeight + arr[i].weight <= N)
        {
            curWeight += arr[i].weight;
            finalvalue += arr[i].value;
        }
        else
        {
            int remain = N - curWeight;
            finalvalue += arr[i].value * ((double)remain / arr[i].weight);

            break;
        }
    }
    return finalvalue;
}

int main()
{
    int N = 60;

    Item arr[] = {{100, 10},
                  {280, 40},
                 {120, 20},
                 {120, 24}};

    int size = sizeof(arr) / sizeof(arr[0]);

    cout << "Max Profit: "<< fractionalKnapsack(arr, N, size);
    return 0;
}
```

Output:



```
Max Profit: 440
```