Experiment No. 6
Fraction Knapsack
Date of Performance:07/03/2024
Date of Submission:14/03/2024

Experiment No. 6

Title: Fractional Knapsack

Aim: To study and implement Fractional Knapsack Algorithm

Objective: To introduce Greedy based algorithms

Theory:

Greedy method or technique is used to solve Optimization problems. A solution that can be maximized or minimized is called Optimal Solution.

The knapsack problem states that − given a set of items, holding weights and profit values, one must determine the subset of the items to be added in a knapsack such that, the total weight of the items must not exceed the limit of the knapsack and its total profit value is maximum.

The knapsack problem or rucksack problem is a problem in combinatorial optimization: Given a set of items, each with a mass 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 whois constrained by a fixed size knapsack and must fill it with the most valuable items. The mostcommon problem being solved is the 0-1 knapsack problem, which restricts the number xi of copies of each kind of item to zero or one.

In Knapsack problem we are given:

1. n objects
2. Knapsack with capacity m.
3. An object iis associated with profit Wi.
4. Object i is associated with profit Pi.
5. Object i is placed in knapsack we get profit Pi Xi .

Here objects can be broken into pieces (Xi Values) The Objective of Knapsack problem is to maximize the profit.

Example:

Find an optimal solution for fractional Knapsack problem.

Where,

Number of objects = 7

Capacity of Knapsack = 15

P1,P2,P3,P4,P5,P6,P7 = (10,5,15,7,6,18,3)

W1,W2,W3,W4,W5,W6,W7 = (2,3,5,7,1,4,1)

Solution:

Arrange the objects in decreasing order of Pi/Wi ratio.

Object	1	2	3	4	5	6	7
Pi	10	5	15	7	6	18	3
Wi	2	3	5	7	1	4	1
Pi/Wi	5	1.67	3	1	6	4.5	3

Select the objects with maximum Pi/Wi ratio:

Object	**Profit (Pi)**	**Weight (Wi)**	**Remaining Weight**
-	-	-	15
5	6	1	14
1	10	2	12
6	18	4	8
3	15	5	3
7	3	1	2
2	3.33	2	0
Total	**55.33**		

So, the maximum profit is 55.33 units.

Algorithm:

Fractional Knapsack Problem:

Here,

N- Total No. of Objects

M- Capacity of Knapsack

P- Initial profit. P=0

Pi- Profit of ith object

Wi- Weight of ith Object

Step 1:

For i=1 to N

Calculate Profit / Weight Ratio (i.e. Pi/Wi) } **O(n)**

Step 2: } **O(n.logn)**

Sort objects in decreasing order of Profit / Weight Ratio

Step 3: // Add all the profit by considering the weight capacity of fractional knapsack.

For i=1 to N

if M > 0 AND Wi <= M } **O(n)**

M = M –Wi

P = P + Pi

else

break

if M > 0 Then

P = P + Pi * (M/Wi)

Step 4:

Display Total Profit

Time Complexity = O(n) + O(n.logn) + O(n)
= Max(O(n),O(n.logn),O(n))
= O(n.logn)

Code:

```
#include <stdio.h>
#include <conio.h>
int fk(int v[][2], int W, int n){
   double ratios[100];
   int i, j, profit;
   int tempArray[2];
   for(i=0;i<n;i++){
         double ratio = (double)v[i][0] / v[i][1];
         ratios[i] = ratio;
   }
   for(i=0;i<n;i++){
         for(j=i+1;j<n;j++){
            if (ratios[i] < ratios[j]){
                  double temp = ratios[j];
                  ratios[j] = ratios[i];
                  ratios[i] = temp;

                  tempArray[0] = v[i][0];
                  tempArray[1] = v[i][1];
                  v[i][0] = v[j][0];
                  v[i][1] = v[j][1];
                  v[j][0] = tempArray[0];
                  v[j][1] = tempArray[1];
            }
         }
   }
   profit =0;
   i=0;
   while (W > 0 && i < n){
```

```
        if (W >= v[i][1]){
            W-=v[i][1];
            profit += v[i][0];
        }
        else{
            profit += (int)(((double)W / v[i][1]) * v[i][0]);
            W=0;
        }
        i++;
    }
    return profit;
}

int main() {
    // Write C code here
    int arr[100][2];
    int n,W,b,i;
    clrscr();
    printf("Enter the capacity: ");
    scanf("%d", &W);
    printf("Enter the number of elements: ");
    scanf("%d", &n);
    printf("Enter the elements: ");
    for(i=0;i<n;i++){
        printf("\n%d:\n ",i);
        printf("Profit: ");
        scanf("%d", &b);
        arr[i][0] = b;
        printf("Weight: ");
        scanf("%d", &b);
        arr[i][1] = b;
    }
```

```
    b = fk(arr, W, n);
    printf("The profit of the user is: ");
    printf("%d", b);
    getch();
    return 0;
}
```

Output:

Output

```
/tmp/ki5rOI4EWf.o
Enter the capacity: 15
Enter the number of elements: 7
Enter the elements:
0:
 Profit: 10
Weight: 2

1:
 Profit: 5
Weight: 3

2:
 Profit: 15
Weight: 5

3:
 Profit: 7
Weight: 7

4:
 Profit: 6
Weight: 1

5:
 Profit: 18
Weight: 4

6:
 Profit: 31
Weight: 1
The profit of the user is: 55

=== Code Execution Successful ===
```

Conclusion:

In conclusion, Fractional Knapsack is a greedy algorithm used to solve the knapsack problem where items can be broken down into fractions to maximize the total value of items selected without exceeding the capacity of the knapsack. It has a time complexity of O(n log n), where n is the number of items, due to sorting the items based on their value-to-weight ratio. This algorithm is efficient and suitable for situations where items can be divided into smaller portions, allowing for flexible optimization.