DAA Experiment-5
(Batch-A/A1)

Name	Ansari Mohammed Shanouf Valijan
UID Number	2021300004
Class	SY B.Tech Computer Engineering(Div-A)
Experiment Number	5
Date of Performance	30-03-23
Date of Submission	06-04-23

Aim:
To solve Fractional Knapsack Problem using Greedy approach.

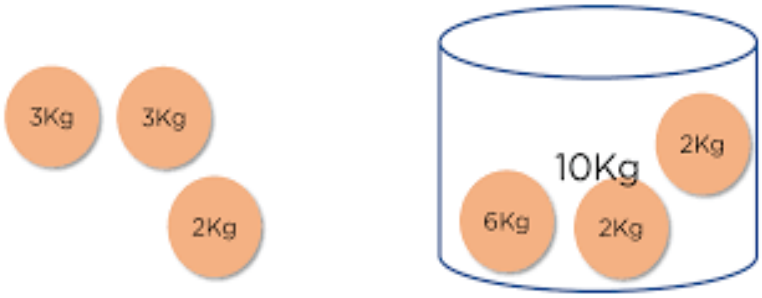
Theory:
The fractional knapsack problem is a classic optimization problem in computer science and operations research. It involves selecting a subset of items to place into a knapsack, subject to the constraint that the total weight of the selected items cannot exceed the capacity of the knapsack. Each item has a weight and a value, and the goal is to maximize the total value of the selected items while having a greedy approach towards both- weight and profit.

The key difference between the fractional knapsack problem and the classical knapsack problem is that in the fractional version, items can be selected partially, i.e., fractions of an item can be placed into the knapsack. In contrast, in the classical version, items must be either included entirely or excluded entirely.

The fractional knapsack problem can be solved using a greedy algorithm. The basic idea is to sort the items by their value-to-weight ratio and then add items to the knapsack in descending order of this ratio until the knapsack is full. If an item cannot fit entirely into the knapsack, a fraction of it is included based on the remaining capacity of the knapsack.

The time complexity of the greedy algorithm for the fractional knapsack problem is O(nlogn), where n is the number of items, due to the need to sort the items by value-to-weight ratio.

Given below is a conceptual view of knapsack problem-



In the above image, orange circles represent the items that have to be selected for putting in the knapsack which is represented by the cylinder. 10Kg is the capacity of knapsack in this case.

Algorithm:

[A] For solving the Knapsack Problem-

I. Start.
II. Take in the input structure array of items from the user.
III. Sort the array in descending order of the profit by weight ratio.
IV. Check if weight of the item is less than or equal to the capacity of knapsack.
V. If less than or equal to, include the complete profit of that item in the overall profit and update the capacity.
VI. If not, take the required fraction of the item and update the capacity to 0.
VII. Repeat steps 4 to 6 till the capacity of knapsack becomes zero.
VIII. End.

Program:

```c
//header files
#include<stdio.h>
#include<stdlib.h>

//structure for an item in the knapsack problem
struct item{
    int itemId;
    double weight;
    double profitVal;
    double profitByWeightRatio;
};

//function to solve the knapsack problem
int solveKnapsack(struct item items[], int numOfItems, double capacity){

    //sorting the array
```

```
    struct item tempItem;
    for(int i=0; i<numOfItems-1; i++){
        if(items[i].profitByWeightRatio<items[i+1].profitByWeightRatio){
            tempItem=items[i];
            items[i]=items[i+1];
            items[i+1]=tempItem;
            i=-1;
        }
    }
    printf("\nThe rearranged items based on their profit to weight ratio are as follows-
\n\n");
    printf("Item\tProfit\tWeight\tProfit to Weight Ratio\n");
    for(int i=0; i<numOfItems; i++){
        printf("I%d\t%.0lf\t%.0lf\t%.2lf\n",items[i].itemId,items[i].profitVal,items[i].we
ight,items[i].profitByWeightRatio);
    }

    //selecting the required items as per greedy approach
    int index=0;
    double maxProfitVal=0;
    while(index<numOfItems){
        if(items[index].weight<=capacity){
            maxProfitVal+=items[index].profitVal;
            capacity-=items[index].weight;
            index++;
        }
        else if(capacity>0){
            maxProfitVal+=(items[index].profitVal)*(capacity/items[index].weight);
            break;
        }
    }
    printf("\nThe max profit value as obtained(considering greedy approach towards both
weight and profit) -----> %.2lf\n",maxProfitVal);

    return index;

}

//main function
void main(){

    //taking user inputs
    int numOfItems;
    double capacity;
    printf("\nEnter the number of items to be considered for knapsack -----> ");
    scanf("%d",&numOfItems);
    printf("Enter the capacity of knapsack -----> ");
    scanf("%lf",&capacity);

    //dynamically allocating the memory for array of items
    struct item* items=malloc(numOfItems*sizeof(struct item*));

    //taking inputs regarding all the items
    printf("\nEnter the Weight and Profit value of all the items-\n");
```

```
    for(int i=0; i<numOfItems; i++){
        printf("Item-%d -----> ",i+1);
        items[i].itemId=i+1;
        scanf("%lf",&items[i].weight);
        scanf("%lf",&items[i].profitVal);
        items[i].profitByWeightRatio=(items[i].profitVal)/(items[i].weight);
    }

    //solving the knapsack problem
    int index=solveKnapsack(items,numOfItems,capacity);
    double weightSum=0;
    printf("Set of items to be included in the knapsack -----> {");
    for(int i=0; i<index; i++){
        printf("I%d, ",items[i].itemId);
        weightSum+=items[i].weight;
    }
    if(weightSum<capacity){
        printf("(%.0lf/%.0lf) of I%d}\n\n",capacity-
weightSum,items[index].weight,items[index].itemId);
    }

    //deallocating the used memory
    free(items);

}
```

Implementation:

The outputs for the corresponding set of inputs are as follows-

```
Enter the number of items to be considered for knapsack -----> 7
Enter the capacity of knapsack -----> 28

Enter the Weight and Profit value of all the items-
Item-1 -----> 2 9
Item-2 -----> 5 5
Item-3 -----> 6 2
Item-4 -----> 11 7
Item-5 -----> 1 6
Item-6 -----> 9 16
Item-7 -----> 1 3

The rearranged items based on their profit to weight ratio are as follows-

Item    Profit  Weight  Profit to Weight Ratio
I5      6       1       6.00
I1      9       2       4.50
I7      3       1       3.00
I6      16      9       1.78
I2      5       5       1.00
I4      7       11      0.64
I3      2       6       0.33

The max profit value as obtained(considering greedy approach towards both weight and profit) -----> 45.36
Set of items to be included in the knapsack -----> {I5, I1, I7, I6, I2, (10/11) of I4}
```

```
Enter the number of items to be considered for knapsack -----> 3
Enter the capacity of knapsack -----> 20

Enter the Weight and Profit value of all the items-
Item-1 -----> 18 24
Item-2 -----> 15 25
Item-3 -----> 20 15

The rearranged items based on their profit to weight ratio are as follows-

Item    Profit  Weight  Profit to Weight Ratio
I2      25      15      1.67
I1      24      18      1.33
I3      15      20      0.75

The max profit value as obtained(considering greedy approach towards both weight and profit) -----> 31.67
Set of items to be included in the knapsack -----> {I2, (5/18) of I1}
```

Inference:

On observing the output, I was able to understand the working of greedy approach in solving the Knapsack problem. While using the greedy approach, one could have been greedy about weight or profit exclusively. However, by solving examples in class, I was able to connect the dots as of why considering both weight and profit would be a better option in obtaining an optimal solution to the said problem.

Below are the solved examples that were used to verify the results obtained above-

Example-1:
Profit = {9, 5, 2, 7, 6, 16, 3}.
Weight = {2, 5, 6, 11, 1, 9, 1}.
Capacity = 28.

Example-2:
Profit = {24, 25, 15}.
Weight = {18, 15, 20}.
Capacity = 20.

06-04-23 SHANOUF ANSARI 2021800004

Solved Examples - Fractional Knapsack Problem

Example-1: Profit = {9, 5, 2, 7, 6, 16, 3}
Weight = {2, 5, 6, 11, 1, 9, 1}
Capacity = 28

Arranging items w.r.t profit/weight ratio

Item	Profit	Weight	Ratio
I5	6	1	6
I1	9	2	4.5
I7	3	1	3
I6	16	9	1.78
I2	5	5	1
I4	7	11	0.64
I3	2	6	0.33

First five items can be fully considered as capacity is not exceeded.

∴ Max profit = 6+9+3+16+5 = 39
Capacity = 28 − 18 = 10

Taking $\frac{10}{11}$ of I4 (Next item)

Max profit = $39 + \frac{10}{11}(7) = 45.36$

Hence maximum profit achieved can be 45.36

Items to be selected = $\{I5, I1, I7, I6, I2, \frac{10}{11} I4\}$

2021300004

Example-2: Profit = { 24, 25, 15 }
Weight = { 18, 15, 20 }
Capacity = 20

Arranging items w.r.t profit / weight ratio

Item	Profit	Weight	Ratio
I2	25	15	1.67
I1	24	18	1.33
I3	15	20	0.75

I2 can be fully selected,

∴ Max profit = 25
Capacity = 20 − 15 = 5

Taking 5/18 of I1, we get

Max profit = $25 + \frac{5}{18}(24) = 31.67$

Hence max profit that is achievable is 31.67

Items to be selected = { I_2, 5/18 of I_1 }

Conclusion:

By performing this experiment, I was able to implement the greedy approach for solving the Fractional Knapsack problem. I was also able to get familiar with greedy approach in general.