Experiment No. 6
Fraction Knapsack
Name : Vaidehi D. Gadag
Branch/Div.: Comps-1 (C47)
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<stdlib.h>
#include<conio.h>

// Create a structure for profit and weight (also can have 2D array)
struct Item
{
  float profit;
  float weight;
};

// Implementing the function to print the table
void printTable(struct Item arr[], int n)
{
  printf("Item\tProfit\tWeight\tRatio\n");
  for (int i = 0; i < n; i++)
  {
   float ratio = arr[i].profit / arr[i].weight;
   printf("%d\t%.2f\t%.2f\t%.2f\n", i + 1, arr[i].profit, arr[i].weight, ratio);
  }
}

// Implementing the Knapsack Function
float knapsack(struct Item arr[], int n, int W, float x[])
{
 // Sorting items based on profit/weight ratio
 for(int i = 0; i < n - 1; i++)
 {
  for (int j = 0; j < n; j++)
  {
   float ratio1 = arr[j].profit / arr[j].weight;
   float ratio2 = arr[j + 1].profit / arr[j + 1].weight;
   if(ratio1 < ratio2)
   {
    // Swapping the entire structure array
    struct Item temp = arr[j];
    arr[j] = arr[j + 1];
    arr[j + 1] = temp;
   }
```

```
  }
 }

 // Print table after sorting
 printf("\n The Sorted Table is :\n");
 printTable(arr, n);

 float totalWeight = 0;
 float totalProfit = 0;

 for (i = 0; i < n; i++)
 {
  // Check the main condition
  if (totalWeight + arr[i].weight <= W)
  {
   x[i] = 1; // Taking the whole item
   totalWeight += arr[i].weight;
   totalProfit += arr[i].profit;
  }
  else
  {
   x[i] = (W - totalWeight) / arr[i].weight; // Taking fraction of item
   totalWeight += arr[i].weight * x[i];      // Update total weight
   totalProfit += arr[i].profit * x[i];
   break; // Knapsack is full
  }
 }
 return totalProfit;
}

int main()
{
 // Input the number of items
 int n;
 clrscr();
 printf("Enter the number of items: ");
 scanf("%d", &n);

 // create array of structure and fraction
 struct Item arr[100];
 float x[100];
```

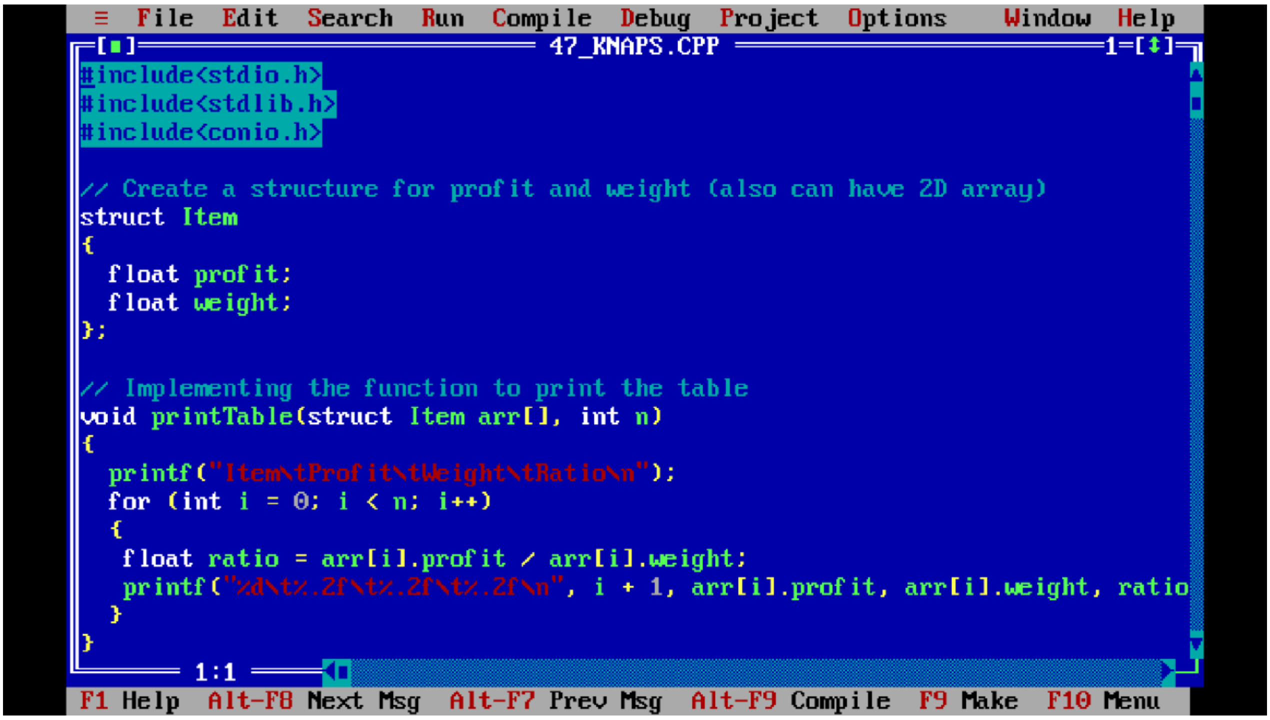
```
// Input the profits & weights of the items
printf("Enter the profit and weights of the items respectively: \n");
for (int i = 0; i < n; i++)
{
 printf("Item %d: ", i + 1);
 scanf("%f %f", &arr[i].profit, &arr[i].weight);
}

// Input the Knapsack Capacity
int W;
printf("Enter the capacity of Knapsack: ");
scanf("%d", &W);

// Print sorted table
printf("\nThe Table is :\n");
printTable(arr, n);

// Get the maximum profit
float maxProfit = knapsack(arr, n, W, x);
printf("\nMaximum profit: %.2f\n", maxProfit);

getch();
return 0;
}
```



```
≡ File Edit Search Run Compile Debug Project Options Window Help
[■] 47_KNAPS.CPP 1=[↕]
// Implementing the Knapsack Function
float knapsack(struct Item arr[], int n, int W, float x[])
{
 // Sorting items based on profit/weight ratio
 for(int i = 0; i < n - 1; i++)
 {
  for (int j = 0; j < n; j++)
  {
   float ratio1 = arr[j].profit / arr[j].weight;
   float ratio2 = arr[j + 1].profit / arr[j + 1].weight;
   if(ratio1 < ratio2)
   {
    // Swapping the entire structure array
    struct Item temp = arr[j];
    arr[j] = arr[j + 1];
    arr[j + 1] = temp;
   }
  }
 }

 // Print table after sorting
43:1
F1 Help  Alt-F8 Next Msg  Alt-F7 Prev Msg  Alt-F9 Compile  F9 Make  F10 Menu
```

```
≡ File Edit Search Run Compile Debug Project Options Window Help
[■] 47_KNAPS.CPP 1=[↕]
 // Print table after sorting
 printf("\n The Sorted Table is :\n");
 printTable(arr, n);

 float totalWeight = 0;
 float totalProfit = 0;

 for (i = 0; i < n; i++)
 {
  // Check the main condition
  if (totalWeight + arr[i].weight <= W)
  {
   x[i] = 1; // Taking the whole item
   totalWeight += arr[i].weight;
   totalProfit += arr[i].profit;
  }
  else
  {
   x[i] = (W - totalWeight) / arr[i].weight; // Taking fraction of item
   totalWeight += arr[i].weight * x[i];      // Update total weight
   totalProfit += arr[i].profit * x[i];
63:1
F1 Help  Alt-F8 Next Msg  Alt-F7 Prev Msg  Alt-F9 Compile  F9 Make  F10 Menu
```

```
File  Edit  Search  Run  Compile  Debug  Project  Options   Window  Help
[■]                    47_KNAPS.CPP                          1=[↕]
    break; // Knapsack is full
   }
  }
  return totalProfit;
}

int main()
{
 // Input the number of items
 int n;
 clrscr();
 printf("Enter the number of items: ");
 scanf("%d", &n);

 // create array of structure and fraction
 struct Item arr[100];
 float x[100];

 // Input the profits & weights of the items
 printf("Enter the profit and weights of the items respectively: \n");
 for (int i = 0; i < n; i++)
      84:1
F1 Help  Alt-F8 Next Msg  Alt-F7 Prev Msg  Alt-F9 Compile  F9 Make  F10 Menu
```

```
File  Edit  Search  Run  Compile  Debug  Project  Options   Window  Help
[■]                    47_KNAPS.CPP                          1=[↕]
 {
  printf("Item %d: ", i + 1);
  scanf("%f %f", &arr[i].profit, &arr[i].weight);
 }

 // Input the Knapsack Capacity
 int W;
 printf("Enter the capacity of Knapsack: ");
 scanf("%d", &W);

 // Print sorted table
 printf("\nThe Table is :\n");
 printTable(arr, n);

 // Get the maximum profit
 float maxProfit = knapsack(arr, n, W, x);
 printf("\nMaximum profit: %.2f\n", maxProfit);

 getch();
 return 0;
}
     105:1
F1 Help  Alt-F8 Next Msg  Alt-F7 Prev Msg  Alt-F9 Compile  F9 Make  F10 Menu
```

Output:

Enter the number of items: 5

Enter the profit and weight of the items respectively:

Item 1: 25 40

Item 2: 30 41

Item 3: 17 32

Item 4: 55 50

Item 5: 20 44

Enter the capacity of Knapsack: 70

The Table is :

Item	Profit	Weight	Ratio
1	25.00	40.00	0.62
2	30.00	41.00	0.73
3	17.00	32.00	0.53
4	55.00	50.00	1.10
5	20.00	44.00	0.45

The Sorted Table is :

Item	Profit	Weight	Ratio
1	55.00	50.00	1.10
2	0.00	0.00	-NAN
3	30.00	41.00	0.73
4	25.00	40.00	0.62
5	17.00	32.00	0.53

Maximum profit: 69.63

```
Enter the number of items: 5
Enter the profit and weights of the items respectively:
Item 1: 25 40
Item 2: 30 41
Item 3: 17 32
Item 4: 55 50
Item 5: 20 44
Enter the capacity of Knapsack: 70_
```

```
Item 1: 25 40
Item 2: 30 41
Item 3: 17 32
Item 4: 55 50
Item 5: 20 44
Enter the capacity of Knapsack: 70

The Table is :
Item    Profit  Weight  Ratio
1       25.00   40.00   0.62
2       30.00   41.00   0.73
3       17.00   32.00   0.53
4       55.00   50.00   1.10
5       20.00   44.00   0.45

 The Sorted Table is :
Item    Profit  Weight  Ratio
1       55.00   50.00   1.10
2       0.00    0.00    -NAN
3       30.00   41.00   0.73
4       25.00   40.00   0.62
5       17.00   32.00   0.53

Maximum profit: 69.63
```

Conclusion:

In conclusion, the knapsack problem is a classic optimization problem that has wide-ranging applications in various fields. It is about maximizing the total profit of a set of objects that can be placed in a knapsack with a limited capacity. The greedy method can be used to solve the fractional knapsack problem, where objects can be broken into pieces. The objective is to maximize the profit, and this is achieved by selecting objects in decreasing order of their profit-to-weight ratio until the knapsack is full. The time complexity of this greedy algorithm is O(n log n), where n is the number of items.