Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Computer Science and Engineering (Data Science)

NAME: ALISTAIR SALDANHA SAPID:60009200024

BATCH: K1

Experiment 4

(Greedy Algorithm)

Aim: Implementation of fractional Knapsack using greedy algorithm.

Theory:

Given a set of items, each with a weight and 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 and the total value is as large as possible.

The knapsack problem is in combinatorial 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, and 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 example.

- Finding the least wasteful way to cut raw materials
- portfolio optimization
- Cutting stock problems

In this case, items can be broken into smaller pieces, hence the thief can select fractions of items.

According to the problem statement,

- There are n items in the store
- Weight of i^{th} item $w_i>0$
- Profit for i^{th} item $p_i>0$ and
- Capacity of the Knapsack is W
-

Pseudocode:

```
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
```

Complexity:

Time Complexity: O(n logn).

Example:

Problem: Consider the following instances of the fractional knapsack problem: n = 3, M = 20, V = (24, 25, 15) and W = (18, 15, 20) find the feasible solutions.

Solution:

Arrange items by decreasing order of profit density. Assume that items are labeled as X = (I1, I2, I3), have profit V = {24, 25, 15} and weight W = {18, 15, 20}.

Item (x_i)	**Value (v_i)**	**Weight (w_i)**	$p_i = v_i / w_i$
I_2	25	15	1.67
I_1	24	18	1.33
I_3	15	20	0.75

Initialize, Weight of selected items, SW = 0,

Profit of selected items, SP = 0,

Set of selected items, S = { },

Here, Knapsack capacity M = 20.

Iteration 1 : SW= (SW + w_2) = 0 + 15 = 15

SW ≤ M, so select I_2

S = { I_2 }, SW = 15, SP = 0 + 25 = 25

Iteration 2 : SW + w_1 > M, so break down item I_1.

The remaining capacity of the knapsack is 5 unit, so select only 5 units of item I_1.

frac = (M – SW) / W[i] = (20 – 15) / 18 = 5 / 18

Department of Computer Science and Engineering (Data Science)

$S = \{ I_2, I_1 * 5/18 \}$

$SP = SP + v_1 * frac = 25 + (24 * (5/18)) = 25 + 6.67 = 31.67$

$SW = SW + w_1 * frac = 15 + (18 * (5/18)) = 15 + 5 = 20$

The knapsack is full. Fractional Greedy algorithm selects items $\{I_2, I_1 * 5/18 \}$, and it gives a profit of **31.67 units**.

Lab Assignment to Complete:

The capacity of the knapsack W = 60 and the list of provided items are shown in the following table –

Item	A	B	C	D
Profit	280	100	120	120
Weight	40	10	20	24

Solution:

Experiment - 04 Knapsack using Greedy Algorithm

Example

Item	Profit	Weight
I_1	280	40
I_2	100	10
I_3	120	20
I_4	120	24

Solution

Step-1 Compute Profit \ Weight for each item.

$P_1 = \frac{280}{40} = 7$, $P_2 = \frac{100}{10} = 10$, $P_3 = \frac{120}{20} = 6$

$P_4 = \frac{120}{24} = 5$

Step-2 Sort the items according to the ratio

Item	Profit	Weight	P/w
B I_2	100	10	10
A I_1	280	40	7
C I_3	120	20	6
D I_4	120	24	5

Step-3 Initially Weight = 0 $S = \{ \}$

Profit = 0 M = 60 units

Step-4 Start Adding

Add I_2, $W = 0 + 10 = 10 \le M$ ✓

$P = 0 + 100 = 100$ units

$S = \{ I_2 \}$, $M = 60 - 10 = 50$

Add I_1, $W = 10 + 40 = 50 \le M$

$P = 100 + 280 = 380$ units

$S = \{ I_2, I_1 \}$ $M = 50 - 40 = 10$

Add I_3, $W = 50 + 20 = 70 > M$

Breakdown the item using Fractional Knapsack Techniq

FOR EDUCATIONAL USE

Department of Computer Science and Engineering (Data Science)

Remaining Capacity = 10

$$\text{Fractional} = \frac{(M - W)}{w_i} = \frac{10}{20} = 0.5$$

$$\therefore S = \{I_2, I_1, I_3 * 0.5\}$$

$$W = 50 + 20 * 0.5 = 60 \text{ units}$$

$$P = 380 + 120 * 0.5 = 440 \text{ units.}$$

Code:

```
// Knapsack Problem (Greedy Algorithm)
# include <stdio.h>
# define MAX 10
// This structure stores Item_no, Item_wgt, Item_value, Item_ratio.
typedef struct item{
    int num, weight;
    float value, ratio;
}item;
// This structure stores the list of items in array.
typedef struct Item_list{
    item data[MAX];
    int n;
}Item_list;
Item_list L1;
// m_c=>Max Capacity of Knapsack, Items[]=>stores the selected items,
x=>indexing for Items[]
int max_capacity, items[MAX], x=0, n;
// Fraction=>Answer of Fractional Knapsack formula, p=sum of all profits
float fraction=0.0, p=0.0;
// Taking the Input
void get_input()
{
    int i, weight[MAX], count=0;
    float value[MAX], ratio[MAX];
    item temp;
    printf(" Enter the number of items: ");
    scanf("%d", &n);
    printf(" Enter the maximum capacity: ");
    scanf("%d", &max_capacity);
    L1.n=0;
    for(i=0;i<n;i++)
    {
        L1.data[i].num = i+1;
```

Department of Computer Science and Engineering (Data Science)

```
        printf(" Enter the value and weight for Item-%d: ", i+1);
        scanf("%f %d", &value[i], &weight[i]);
        L1.data[L1.n].value = value[i];
        L1.data[L1.n].weight = weight[i];
        ratio[i] = (float)value[i] / (float)weight[i];
        L1.data[L1.n].ratio = ratio[i];
        L1.n++;
    }
}
// Sorting data according to the ratio in desc. order
void sort() {
    int i,j;
    item temp;
    for(i=1; i<L1.n; i++)
    {
        for(j=0; j<L1.n-1; j++)
        {
            if(L1.data[j].ratio<L1.data[j+1].ratio)
            {
                temp = L1.data[j];
                L1.data[j] = L1.data[j+1];
                L1.data[j+1] = temp;
            }
        }
    }
}
// To display the table
void display()
{
    int i;
    printf("\n");
    printf("Items\tValue\tWeight\tV / W");
    printf("\n");
    for(i=0;i<L1.n;i++)
    {
        printf("%d\t", L1.data[i].num);
        printf("%.2f\t", L1.data[i].value);
        printf("%d\t", L1.data[i].weight);
        printf("%.2f\t", L1.data[i].ratio);
        printf("\n");
    }
}
// Knapsack Function
void knapsack()
{
    int i=0, M, w=0;
    printf("Adding Items..");
    M = max_capacity;
    x = 0;
//  While current weight < Max Capacity
```

```
    while(L1.data[i].weight <= M)
    {
//      If adding the next weight leads to excess => break
        if((w + L1.data[i].weight) > M) break;
//      else keep adding weight, profit, decrease max_capacity and add item_num
to selected item_list (items[])
        else
        {
            w += L1.data[i].weight;
            p += L1.data[i].value;
            M -= L1.data[i].weight;
            items[x] = L1.data[i].num;
            x++;
        }
        i++;
    }
//  Finally, when excess condition is met, use fractional formula
    fraction = (float)M / (float)(L1.data[i].weight);
    w += L1.data[i].weight * fraction;
    p += L1.data[i].value * fraction;
    items[x] = L1.data[i].num;
}
// Final Output (Selected Items & Max Profit)
void final_output()
{
    int i;
    printf("\n\nSelected Items: ");
    for(i=0;i<x;i++)
        printf("I%d ", items[i]);
    printf("I%d*%.2f", items[x], fraction);

    printf("\n\nMaximum Profit = %.2f", p);
}
int main()
{
//  INPUT
    get_input();
//  DISPLAY
    printf("\nBefore sorting");
    display();
//  SORTING WITH RESPECT TO RATIO
    sort();
    printf("\nAfter sorting");
    display();
//  KNAPSACK
    knapsack();
//  FINAL OUTPUT
    final_output();
}
```

Output:

```
 Enter the number of items: 4
 Enter the maximum capacity: 60
 Enter the value and weight for Item-1: 280 40
 Enter the value and weight for Item-2: 100 10
 Enter the value and weight for Item-3: 120 20
 Enter the value and weight for Item-4: 120 24

Before sorting
Items   Value   Weight  V / W
1       280.00  40      7.00
2       100.00  10      10.00
3       120.00  20      6.00
4       120.00  24      5.00

After sorting
Items   Value   Weight  V / W
2       100.00  10      10.00
1       280.00  40      7.00
3       120.00  20      6.00
4       120.00  24      5.00
Adding Items..

Selected Items: I2 I1 I3*0.50

Maximum Profit = 440.00
--------------------------------
Process exited after 19.23 seconds with return value 25
Press any key to continue . . .
```