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)

JHANVI PAREKH

60009210033

CSE(Data Science)

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

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

CODE:
```
# include<stdio.h>
void knapsack(int n, float weight[], float profit[], float capacity) {
float x[20], tp = 0;
int i, j, u;
u = capacity;
for (i = 0; i < n; i++)
x[i] = 0.0;
for (i = 0; i < n; i++) {
if (weight[i] > u)
break;
else {
x[i] = 1.0;
tp = tp + profit[i];
u = u - weight[i];
}
}
if (i < n)
x[i] = u / weight[i];
tp = tp + (x[i] * profit[i]);
printf("\nThe result vector is:- ");
for (i = 0; i < n; i++)
printf("%f\t", x[i]);
printf("\nMaximum profit is:- %f", tp);
}
int main() {
float weight[20], profit[20], capacity;
int num, i, j;
float ratio[20], temp;
printf("60009210033 - Jhanvi Parekh");
printf("\nEnter the no. of objects:- ");
scanf("%d", &num);
printf("\nEnter the wts and profits of each object:- ");
```

```
for (i = 0; i < num; i++) {
scanf("%f %f", &weight[i], &profit[i]);
}
printf("\nEnter the capacity of knapsack:- ");
scanf("%f", &capacity);
for (i = 0; i < num; i++) {
ratio[i] = profit[i] / weight[i];
}
for (i = 0; i < num; i++) {
for (j = i + 1; j < num; j++) {
if (ratio[i] < ratio[j]) {

temp = ratio[j];
ratio[j] = ratio[i];
ratio[i] = temp;
temp = weight[j];
weight[j] = weight[i];
weight[i] = temp;
temp = profit[j];
profit[j] = profit[i];
profit[i] = temp;
}
}
}
knapsack(num, weight, profit, capacity);
return(0);
}
```

main.c

```c
# include<stdio.h>
void knapsack(int n, float weight[], float profit[], float capacity) {
float x[20], tp = 0;
int i, j, u;
u = capacity;
for (i = 0; i < n; i++)
x[i] = 0.0;
for (i = 0; i < n; i++) {
if (weight[i] > u)
break;
else {
x[i] = 1.0;
tp = tp + profit[i];
u = u - weight[i];
}
}
if (i < n)
x[i] = u / weight[i];
tp = tp + (x[i] * profit[i]);
printf("\nThe result vector is:- ");
for (i = 0; i < n; i++)
printf("%f\t", x[i]);
printf("\nMaximum profit is:- %f", tp);
}
int main() {
```

main.c

Run

```
}
int main() {
float weight[20], profit[20], capacity;
int num, i, j;
float ratio[20], temp;
printf("60009210033 - Jhanvi Parekh");
printf("\nEnter the no. of objects:- ");
scanf("%d", &num);
printf("\nEnter the wts and profits of each object:- ");
for (i = 0; i < num; i++) {
scanf("%f %f", &weight[i], &profit[i]);
}
printf("\nEnter the capacity of knapsack:- ");
scanf("%f", &capacity);
for (i = 0; i < num; i++) {
ratio[i] = profit[i] / weight[i];
}
for (i = 0; i < num; i++) {
for (j = i + 1; j < num; j++) {
if (ratio[i] < ratio[j]) {

temp = ratio[j];
ratio[j] = ratio[i];
ratio[i] = temp;
temp = weight[j];
weight[j] = weight[i];
```

Department of Computer Science and Engineering (Data Science)

main.c

```
scanf("%f %f", &weight[i], &profit[i]);
}
printf("\nEnter the capacity of knapsack:- ");
scanf("%f", &capacity);
for (i = 0; i < num; i++) {
ratio[i] = profit[i] / weight[i];
}
for (i = 0; i < num; i++) {
for (j = i + 1; j < num; j++) {
if (ratio[i] < ratio[j]) {

temp = ratio[j];
ratio[j] = ratio[i];
ratio[i] = temp;
temp = weight[j];
weight[j] = weight[i];
weight[i] = temp;
temp = profit[j];
profit[j] = profit[i];
profit[i] = temp;
}
}
}
knapsack(num, weight, profit, capacity);
return(0);
}
```

Output

```
/tmp/oOPgg7dC3E.o
60009210033 - Jhanvi Parekh
Enter the no. of objects:- 4
Enter the wts and profits of each object:- 40 280
10 100
20 120
24 120
Enter the capacity of knapsack:- 60
The result vector is:- 1.000000 1.000000    0.500000    0.000000
Maximum profit is:- 440.000000
```

Department of Computer Science and Engineering (Data Science)

60009210033
Jhanvi Parekh

Page No.:
Date: / /

Lab Work

Item	Profit (v_i)	weight (w_i)	$P_i = v_i / w_i$
A	280	40	7
B	100	10	10
C	120	20	6
D	120	24	5.

Sort the items according to profit in descending order

Item	v_i	w_i	P_i
A	100	10	10
B	280	40	7
C	120	20	6
D	120	24	5

start adding the items in the knapsack where the initial values are weight (w) = 0, Profit (P) = 0 selected item = { }, maximum capacity

$M = 60$

select item B.

$B \rightarrow 10$

$w = w + w_i$

$w = 0 + 10$

$w = 10$

$P = P + P_i$

$P = 0 + 100$

$P = 100$

selected item = {B}

Page No.:
Date: / /

selected item A

$A \rightarrow 4$

$w = w + w_i$

$w = \cancel{50+20}\ 10 + 40$

$w = 50$

$P = P + P_i$

$P = 100 + 280$

$P = 380$

selected item = {B, A}

select item C

$C \rightarrow 20$

$w = w + w_i$

$w = 50 + 20$

$w = 70$

The weight of Item C exceeds the capacity of bag breakdown the items using fractional knapsack technique

$$\text{fraction} = \frac{M - w}{w_i}$$

$$= \frac{60 - 50}{20}$$

$$= 0.5$$

∴ selected item = {B, A, 0.5×C}

$w = w + w_i \times 0.5$

$w = 50 + 20 \times 0.5$

$w = 60$

$P = P + P_i \times 0.5$

$P = 380 + (120 \times 0.5)$

$P = 440$ units

The capacity of knapsack is full therefore selected items are {B, A, 0.5×C} and gives a profit of 440 units.