9. Problem Statement: Fractional Knapsack

**Problem Analysis:**

The Greedy algorithm could be understood very well with a well-known problem referred to as Knapsack problem. Although the same problem could be solved by employing other algorithmic approaches, Greedy approach solves Fractional Knapsack problem reasonably in a good time.

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 **ith** item wi>0wi>0

• Profit for **ith** item pi>0pi>0 and

• Capacity of the Knapsack is **W**

In this version of Knapsack problem, items can be broken into smaller pieces. So, the thief may take only a fraction xi of ith item.

0⩽xi⩽10⩽xi⩽1

The **ith** item contributes the weight xi.wixi.wi to the total weight in the knapsack and profit xi.pixi.pi to the total profit.

Hence, the objective of this algorithm is to

maximize∑n=1n(xi.pi)maximize∑n=1n(xi.pi)

Thus, an optimal solution can be obtained by

∑n=1n(xi.wi)=W∑n=1n(xi.wi)=W

**Algorithm:**

GreedyKnapsack (m, n)

*// p[1:n] and w[1:n] contain the profits and weights respectively*

*// of the n objects ordered such that p[i] / w[i] ≥ p[i+1] / w[i+1].*

*// m is the knapsack size and x[1:n] is the solution vector.*

{

for i := 1 to n do x[i] := 0.0; *// Initialize x.*

U := m;

for i := 1 to n do

{

if ( w[i] > U ) then break;

x[i] := 1.0; U := U – w[i];

}

if ( i ≤ n ) then x[i] := U / w[i];

}

**Source 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("Maximum profit is: %.2f\n", tp);

}

int main()

{

float weight[20], profit[20], capacity;

int num, i, j;

float ratio[20], temp;

printf("Enter the no. of objects: \n");

scanf("%d", &num);

printf("Enter the weights of each object: \n");

for (i = 0; i < num; i++)

scanf("%f", &weight[i]);

printf("Enter the profits of each object: \n");

for (i = 0; i < num; i++)

scanf("%f", &profit[i]);

printf("Enter the capacity of knapsack: \n");

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);

}

**Sample Input:**

Enter the no. of objects:

3

Enter the weights of each object:

18 15 10

Enter the profits of each object:

25 24 15

Enter the capacity of knapsack:

20

**Sample Output:**

Maximum profit is: 31.50