|  |  |
| --- | --- |
| **Course Code:** CT2352 | **Course Name:** Lab - Design & Analysis of Algorithms |

|  |  |
| --- | --- |
| **Name of the Student:**  S Akshansh | **Sem/Sec:**  6th/A |
| **Branch:**  Computer Technology | **Roll no.:** 72  **Enrolment no.:** 19010927 |

**PRACTICAL NO. 5**

**AIM:** We are given n objects and a knapsack. For i = 1, 2, ..., n object i has a positive weight w, and a positive value v₁. The knapsack can carry weight not exceeding W. Our aim is to fill knapsack in a way that maximizes the value of the included objects, while respecting the capacity constraints. Here, the objects can be broken up into smaller pieces, so we may decide to carry only a fraction x; of the object i, where 0≤xi≤ 1. In symbol, the problem can be stated as follows:

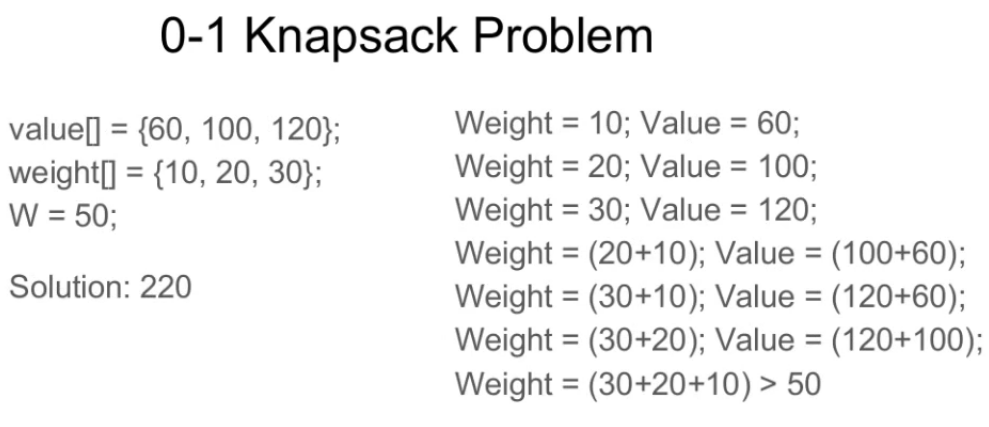
*maximizexi vi subject to xi wi≤W*

Insert an item into knapsack with maximum profit. Explore all the possible solutions to the given problem based on Greedy Algorithm.

**THEORY**:

The **Knapsack Problem** is a famous Dynamic Programming Problem that falls in the **optimization** category. It derives its name from a scenario where, given a set of items with specific weights and assigned values, the goal is to maximize the **value** in a knapsack while remaining within the weight constraint. Each item can only be selected once, as we don’t have multiple quantities of any item.

Given weights and values of n items, put these items in a knapsack of capacity W to get the maximum total value in the knapsack. In other words, given two integer arrays val[0..n-1] and wt[0..n-1] which represent values and weights associated with n items respectively. Also given an integer W which represents knapsack capacity, find out the maximum value subset of val[] such that sum of the weights of this subset is smaller than or equal to W. You cannot break an item, either pick the complete item or don’t pick it (0-1 property).



**Algorithm:**

**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

**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("\n\nMaximum profit is:- %f", tp);

}

int main() {

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

int num, i, j;

float ratio[20], temp;

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

scanf("%d", &num);

printf("\nEnter the weights 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;

}

}

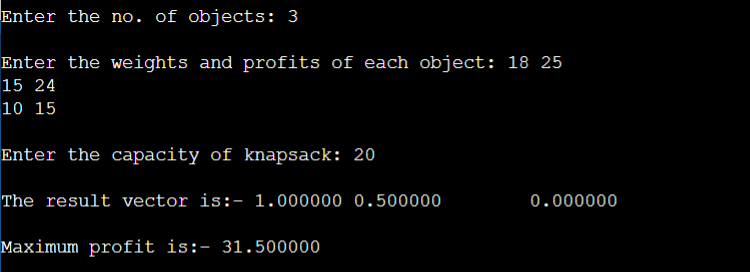
}

knapsack(num, weight, profit, capacity);

return(0);

}

**Output:**

****

**CONCLUSION:** In this practical I executed program based on knapsack problem.