|  |
| --- |
| **Subject: DAA Class: S.E.(Comp)**    **Practical No.: 2 Date:** |

**AIM:** **Implement Greedy method by knapsack problem.**

**Title:** Write algorithm and program for Greedy method by using knapsack problem.

**Theory:**

In this tutorial, earlier we have discussed Fractional Knapsack problem using Greedy approach. We have shown that Greedy approach gives an optimal solution for Fractional Knapsack. However, this chapter will cover 0-1 Knapsack problem and its analysis.

In 0-1 Knapsack, items cannot be broken which means the thief should take the item as a whole or should leave it. This is reason behind calling it as 0-1 Knapsack.

Hence, in case of 0-1 Knapsack, the value of ***xi*** can be either ***0*** or ***1***, where other constraints remain the same.

0-1 Knapsack cannot be solved by Greedy approach. Greedy approach does not ensure an optimal solution. In many instances, Greedy approach may give an optimal solution.

The following examples will establish our statement.

### Example-1

Let us consider that the capacity of the knapsack is W = 25 and the items are as shown in the following table.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Item** | **A** | **B** | **C** | **D** |
| Profit | 24 | 18 | 18 | 10 |
| Weight | 24 | 10 | 10 | 7 |

Without considering the profit per unit weight (***pi/wi***), if we apply Greedy approach to solve this problem, first item ***A*** will be selected as it will contribute maximum profit among all the elements.

After selecting item ***A***, no more item will be selected. Hence, for this given set of items total profit is ***24***. Whereas, the optimal solution can be achieved by selecting items, ***B*** and C, where the total profit is 18 + 18 = 36.

**Source Code:**

#include<stdio.h>

#include<conio.h>

#include<alloc.h>

void Knapsack(int m,int n,int p[],int w[])

{

float \*x,maxprofit;

int \*ino,i,j,U;

x=(float \*)malloc(sizeof(float)\*n);

ino=(int \*)malloc(sizeof(int)\*n);

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

ino[i]=i+1;

for(i=0;i<n-1;i++)

for(j=0;j<n-1-i;j++)

{

if((float)p[j]/w[j]<(float)p[j+1]/w[j+1])

{

int t;

t=p[j];

p[j]=p[j+1];

p[j+1]=t;

t=w[j];

w[j]=w[j+1];

w[j+1]=t;

t=ino[j];

ino[j]=ino[j+1];

ino[j+1]=t;

}

}

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

x[i]=0;

maxprofit=0;

U=m;

printf("\n Selected object and their fractions:");

printf("\nItem no\tprofit\tweight\tU\tXi\tTotal profit");

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

{

if(w[i]>U)

break;

x[i]=1;

U=U-w[i];

maxprofit+=p[i];

printf("\n%d\t%d\t%d\t%d\t%.2f\t%.2f",ino[i],p[i],w[i],U,x[i],maxprofit);

}

if(i<=n-1)

{

x[i]=(float)U/w[i];

U=0;

maxprofit=maxprofit+x[i]\*p[i];

printf("\n%d\t%d\t%d\t%d\t%.2f\t%.2f",ino[i],p[i],w[i],U,x[i],maxprofit);

}

printf("\n Max profit: %.2f",maxprofit);

}

void main()

{

int i,n,m,\*p,\*w;

clrscr();

printf("\nKnapsack Problem implementation using Greedy Technique");

printf("\n Enter no of items: ");

scanf("%d",&n);

w=(int \*)malloc(sizeof(int)\*n);

p=(int \*)malloc(sizeof(int)\*n);

printf("\n Enter capacity of Knapsack: ");

scanf("%d",&m);

printf("\n Enter profit and weight of items: ");

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

{

printf("\n Item %d(profit and weight):",i+1);

scanf("%d %d",&p[i],&w[i]);

}

Knapsack(m,n,p,w);

getch();

}

**Output:**

