**/\* Knapsack Problem: Given a set of items (n), each with a weight (w) and a profit (p), determine a subset of items to include in a collection so that the total weight is less than or equal to a given limit (m) and the total profit is as large as possible.**

**Consider the following instance of the Knapsack Problem:**

**n = 3, m = 20, (p1, p2, p3) = (25,24, 15), and (w1, w2, w3) = (18, 15, 10)**

\*/

#include<iostream.h>

#include<conio.h>

#define MAX 10

struct product

{

int product\_num;

int profit;

int weight;

float ratio;

float take\_quantity;

};

void main()

{

clrscr();

cout<<"\n KNAPSACK PROBLEM USING GREEDY METHOD";

product p[MAX],temp;

int i,j,total\_product,capacity,rounded;

float value=0;

cout<<"\nEnter the number of items:";

cin>>total\_product;

cout<<"\nEnter the capacity of sack:";

cin>>capacity;

cout<<"\n";

for(i=0;i<total\_product;i++)

{

p[i].product\_num=i+1;

cout<<"Enter profit and weight of product"<<i+1<<":";

cin>>p[i].profit>>p[i].weight;

p[i].ratio=(float)p[i].profit/p[i].weight;

p[i].take\_quantity=0;

}

for(i=0;i<total\_product;++i)

{

for(j=i+1;j<total\_product;++j)

{

if(p[i].ratio<p[j].ratio)

{

temp=p[i];

p[i]=p[j];

p[j]=temp;

}

}

for(i=0;i<total\_product;++i)

{

if(capacity==0)

break;

else if(p[i].weight<capacity)

{

p[i].take\_quantity=1;

capacity-=p[i].weight;

}

else if(p[i].weight>capacity)

{

p[i].take\_quantity=(float)capacity/p[i].weight;

capacity=0;

}

}

cout<<"\n Product to be taken:";

for(i=0;i<total\_product;++i)

{

cout<<"\n Take product"<<p[i].product\_num<<":"<<p[i].take\_quantity\*p[i].weight<<"Units";

value+=p[i].profit\*p[i].take\_quantity;

}

cout<<"\n The knapsack value is:"<<value;

getch();

}

}

**OUTPUT:**

