**Experiment No:** 10  **Date:** 29/04/2021

**Aim:** Implementation of 0/1 Knapsack Problem

(Dynamic Programming) and estimate its step count

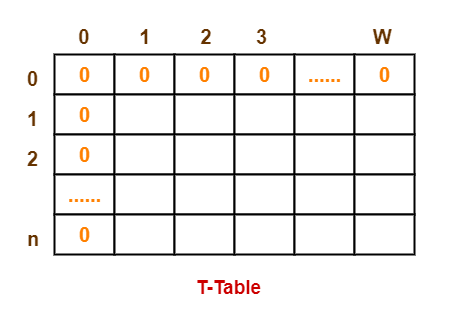
**Theory:**

**0/1 Knapsack Problem**

* In 0/1 Knapsack Problem items are indivisible here.
* We cannot take the fraction of any item.
* We have to either take an item completely or leave it completely.
* It is solved using dynamic programming approach.

**0/1 Knapsack Problem Using Dynamic Programming**

* Let us Consider:
  + Knapsack weight capacity = w
  + Number of items each having some weight and value = n
* 0/1 knapsack problem is solved using dynamic programming in the following steps-
  + **STEP 01:**
* Draw a table say ‘T’ with (n+1) number of rows and (w+1) number of columns.
* Fill all the boxes of 0th row and 0th column with zeroes as shown-



* **STEP 02:**
  + - Start filling the table row wise top to bottom from left to right.
    - Use the following formula-

***T (i , j) = max { T ( i-1 , j ) , valuei + T( i-1 , j – weighti ) }***

* Here, T(i , j) = maximum value of the selected items if we can take items 1 to i and have weight restrictions of j.
* This step leads to completely filling the table.
* Then, value of the last box represents the maximum possible value that can be put into the knapsack.
* **STEP 03:**
* To identify the items that must be put into the knapsack to obtain that maximum profit.
* Consider the last column of the table.
* Start scanning the entries from bottom to top.
* On encountering an entry whose value is not same as the value stored in the entry immediately above it, mark the row label of that entry.
* After all the entries are scanned, the marked labels represent the items that must be put into the knapsack.

**Time Complexity**

* Each entry of the table requires constant time θ(1) for its computation.
* It takes θ(nw) time to fill (n+1)(w+1) table entries.
* It takes θ(n) time for tracing the solution since tracing process traces the n rows.
* Thus, overall θ(nw) time is taken to solve 0/1 knapsack problem using dynamic programming.

**Algorithm**

PW=record{ float p;float w; }

Algorithm Dknap (p,w,x,n,m)

{

//pair[] is an array of PW’s.

b[0]:=1;pair[1].p=pair[1].w:=0.0; //S0

t:=1; h:=1; //Start and end of S0

b[1]:=next:=2; //Next free spot in pair[]

for i:=1 to n-1 do

{//Generate Si.

k:=t;

u:=Largest(pair,w,t,h,i,m);

for j:=t to u do

{//Generate S1(i-1) and merge.

pp:=pair[j].p+p[i]; ww:=pair[j].w+w[i];

// (pp,ww) is the next element in S1(i-1).

while((k≤h) and (pair[k].w≤ww)) do

{

pair[next].p:=pair[k].p;

pair[next].w:=pair[k].w;

next:=next+1; k:=k+1;

}

if ((k≤h) and (pair[k].w=ww)) then

{

if pp<pair[k].p then pp:=pair[k].p;

k:=k+1;

}

if pp>pair[next-1].p then

{

pair[next].p:=pp; pair[next].w=ww;

next:=next+1;

}

while ((k≤h) and (pair[k].p≤pair[next-1].p))

do k:=k+1;

}

//Merge in remaining terms from Si-1.

while(k≤h) do

{

pair[next].p:=pair[k].p; pair[next].w:=pair[k].w;

next:=next+1; k:=k+1;

}

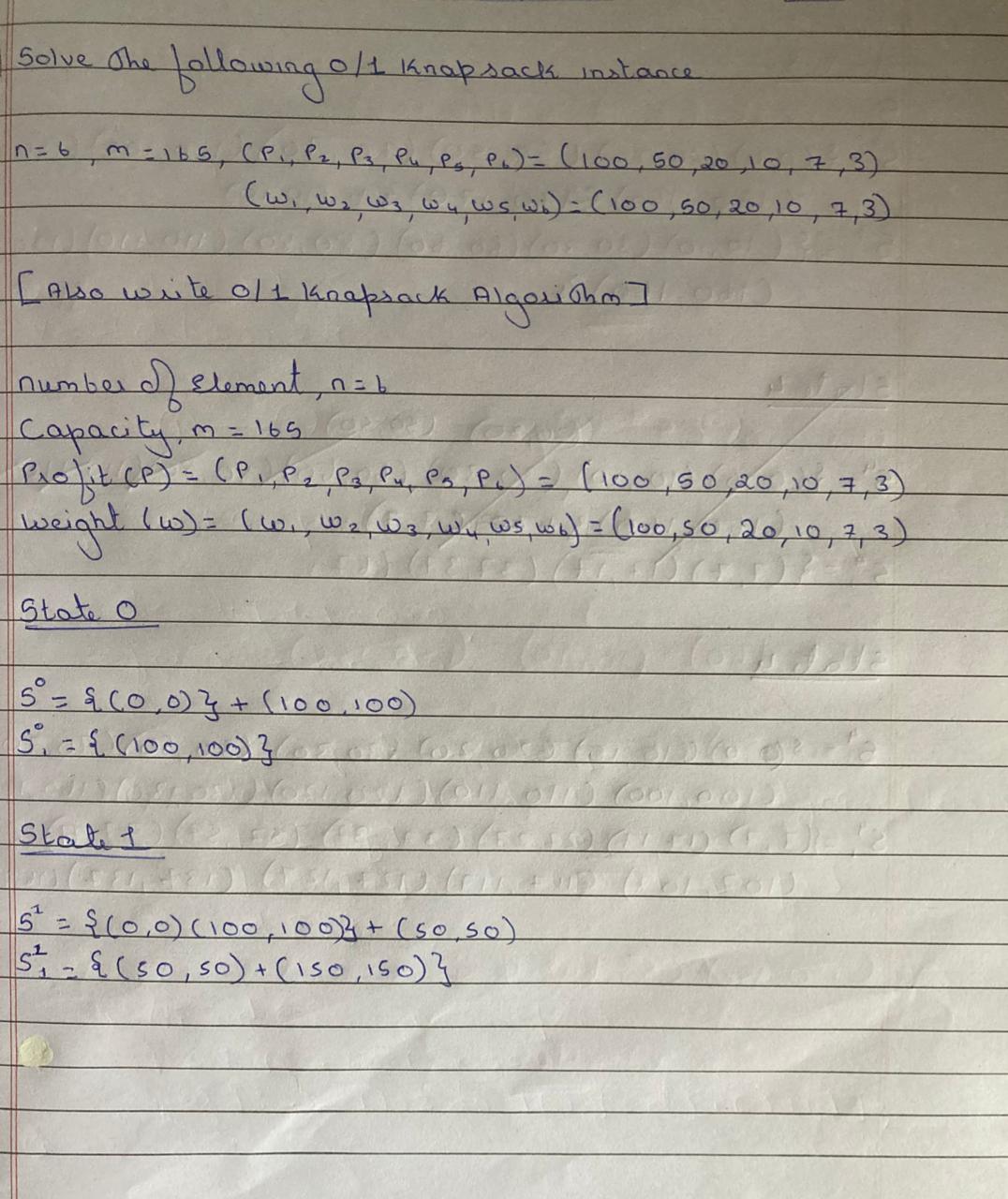
//Initialize for Si+1.

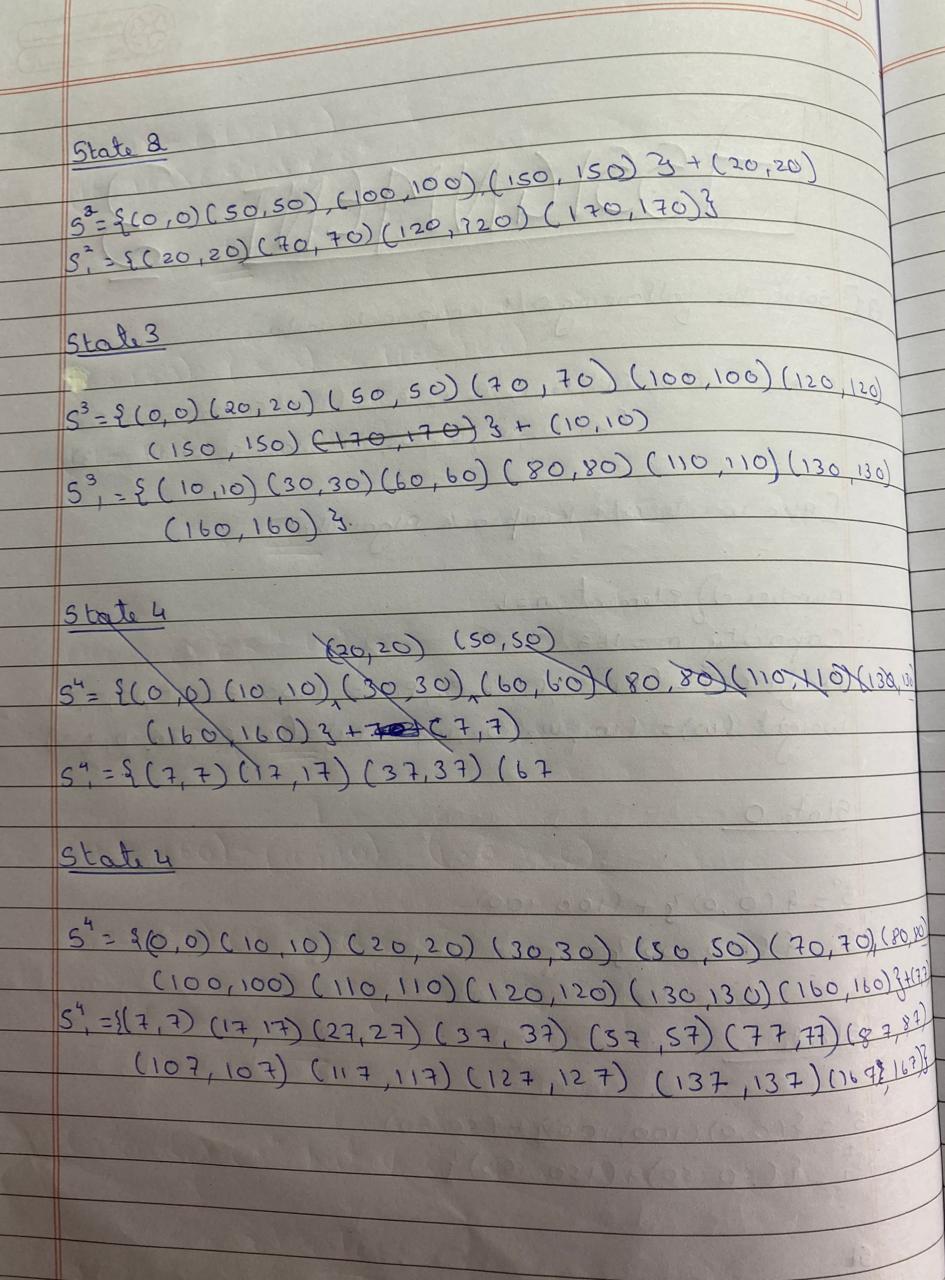
t:=h+1; h:=next-1; b[i+1]:=next;

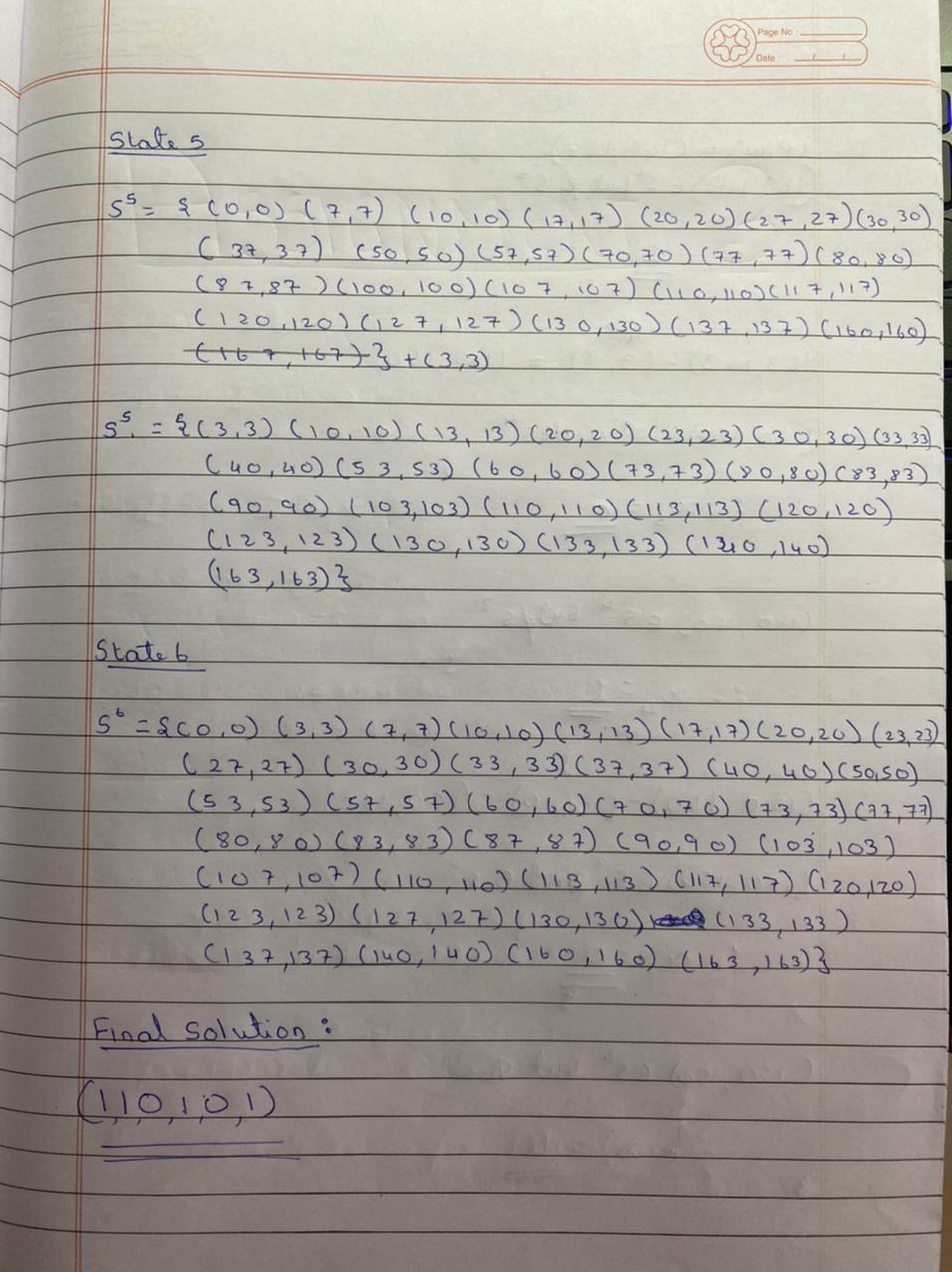
}

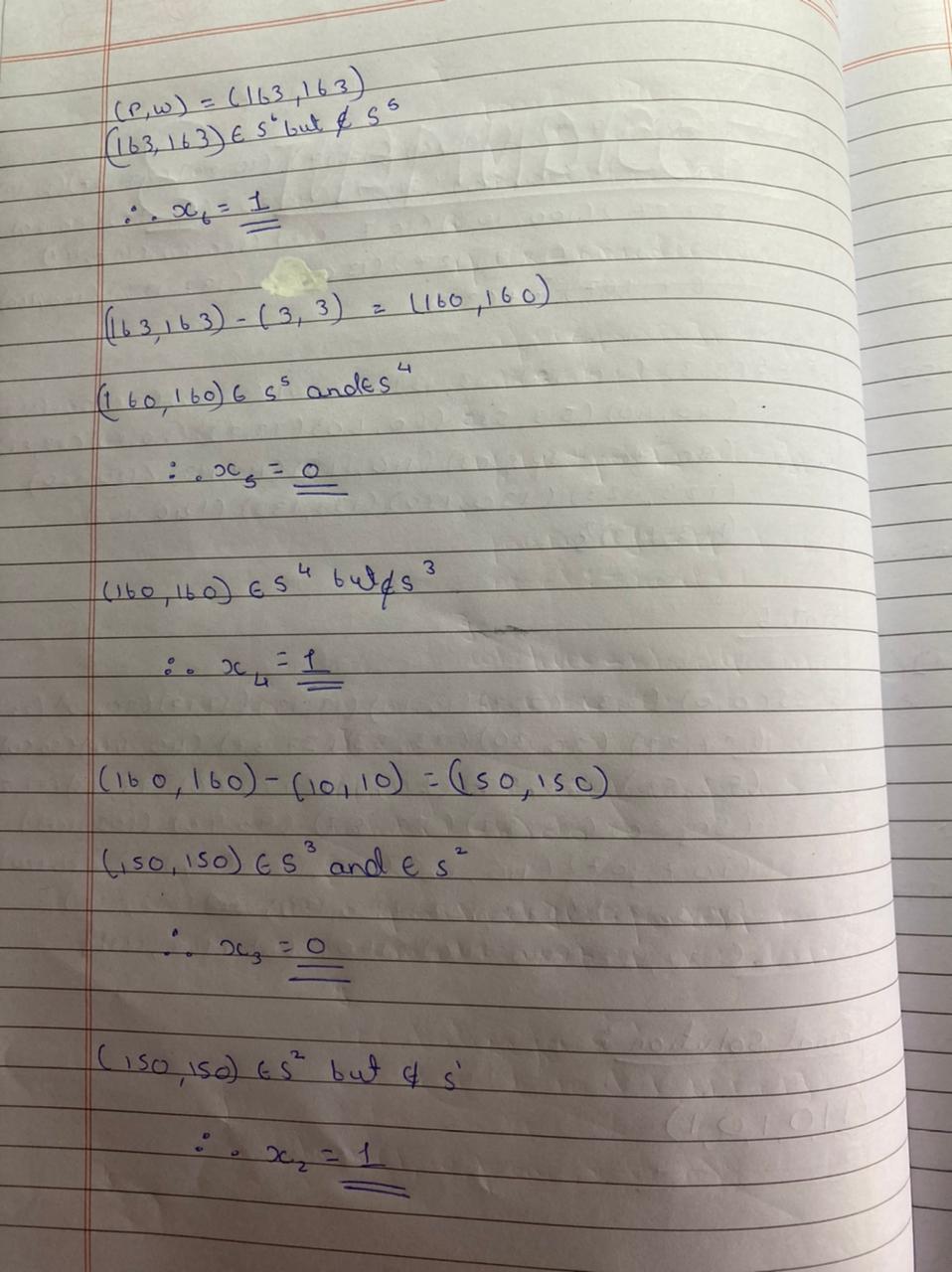
TraceBack(p,w,pair,x,m,n);

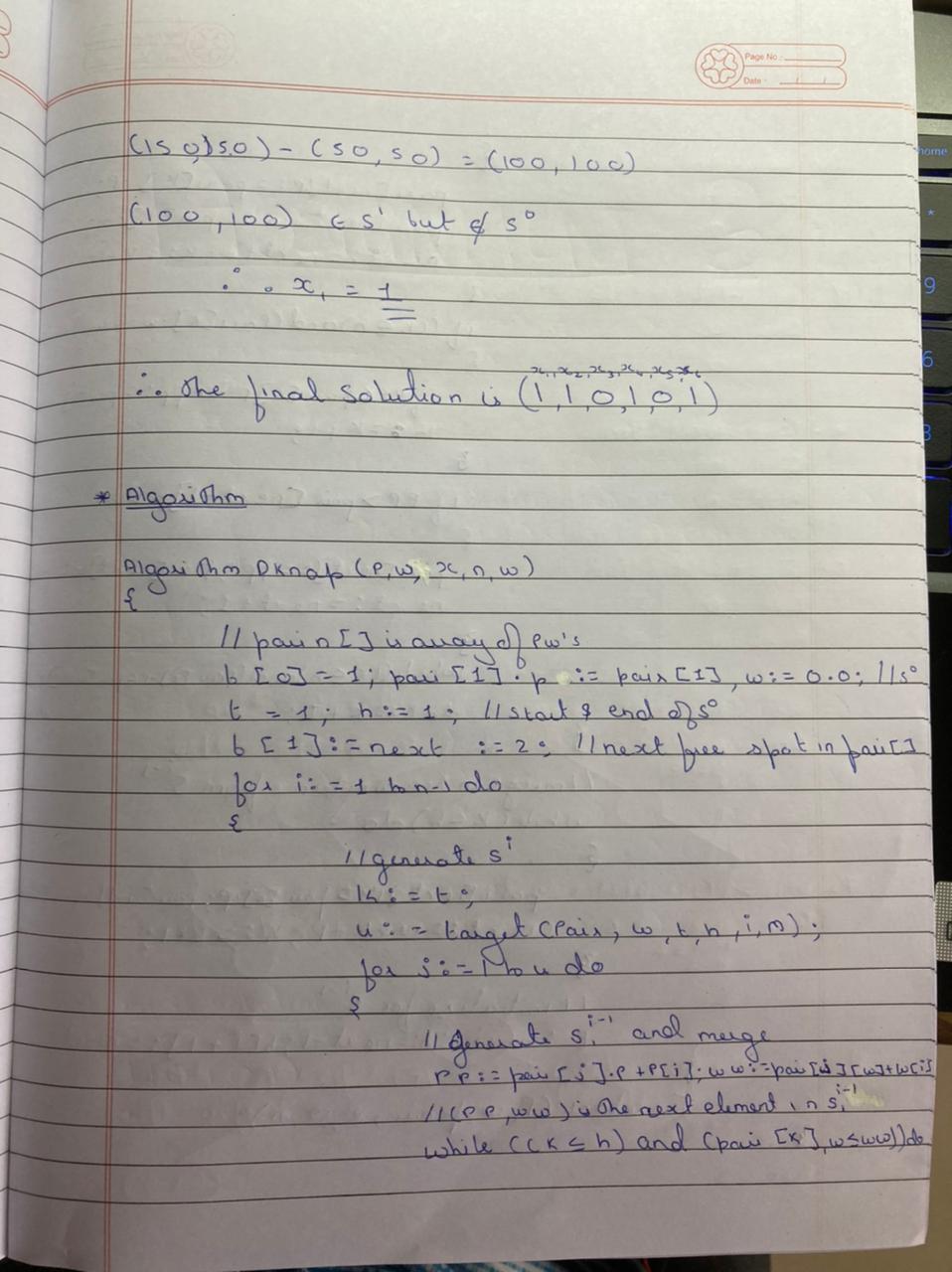
}

**Tracing With Example**









**Program**

#include<iostream>

using namespace std;

struct PW

{

float p=0.0;

float w=0.0;

};

int stepcount=0;

bool search(int l,int h,int pp,int ww,PW pair[]);

int largest(PW pair[],int w[],int t,int h,int i,int m);

void Traceback(int p[],int w[],PW pair[],int \*x,int b,int m,int n);

void DKnap(int p[],int w[],int \*x,int n,int m);

int main()

{

int m,n,profit=0,weight=0;

stepcount+=2;

cout<<"Enter the size of knapsack: ";stepcount++;

cin>>m;

stepcount++;

cout<<"Enter the number of objects: ";stepcount++;

cin>>n;

stepcount++;

int p[n+1],w[n+1],x[n+1];

for(int i=1;i<=n;i++)

{

stepcount++;

cout<<"\nObject "<<i<<endl;

stepcount++;

cout<<"Enter Profit: ";

stepcount++;

cin>>p[i];

stepcount++;

cout<<"Enter Weight: ";

stepcount++;

cin>>w[i];

stepcount++;

x[i]=0;

stepcount++;

}

stepcount++;

DKnap(p,w,x,m,n);

cout<<"\nSolution Vector = ( ";

stepcount++;

for(int i=1;i<=n;i++)

{

stepcount++;

cout<<"x"<<i<<",";

stepcount++;

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

stepcount++;

weight+=x[i]\*w[i];

stepcount++;

}

stepcount++;

cout<<"\b) = (";

for(int i=1;i<=n;i++)

{

stepcount++;

cout<<x[i]<<",";

}

stepcount++;

cout<<"\b)\n\nMaximum Profit = "<<profit<<endl;

cout<<"Weight= "<<weight<<endl;

cout<<"\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"Total Steps = "<<stepcount<<endl;

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

}

bool search(int l,int h,int pp,int ww,PW pair[])

{

int low=l,high=h;

stepcount+=2;

while(low<=high)

{

stepcount++;

int mid=(low+high)/2;

stepcount++;

if(pair[mid].p==pp && pair[mid].w==ww)

{

stepcount++;

return true;

}

else if(pair[mid].w<ww)

{

stepcount++;

low=mid+1;

}

else

{

high=mid+1;

stepcount++;

}

}

return false;

}

int largest(PW pair[],int w[],int t,int h,int i,int m)

{

int low=t,high=h,r;

stepcount+=2;

while(low<=high)

{

stepcount++;

int mid=(low+high)/2;

stepcount++;

if(pair[mid].w+w[i]<=m)

{

r=mid;stepcount++;

low=mid+1;

stepcount++;

}

else

{

high=mid-1;

stepcount++;

}

}

return r;

}

void Traceback(int p[],int w[],PW pair[],int \*x,int b[],int m,int n)

{

int end=b[n+1]-1,temp=n,pp=pair[end].p,ww=pair[end].w;

stepcount+=4;

while(pp>0 && ww>0)

{

stepcount++;

bool f=true;stepcount++;

for(int j=temp;j>=0;j--)

{

stepcount++;

f=search(b[j],b[j+1]-1,pp,ww,pair);

stepcount++;

if(!f)

{

stepcount++;

if(j!=n)

{

x[j+1]=1;stepcount++;

pp=pp-p[j+1];stepcount++;

ww=ww-w[j+1];stepcount++;

}

else

{

x[j]=1;stepcount++;

pp=pp-p[j];stepcount++;

ww=ww-w[j];stepcount++;

}

temp=j;

stepcount++;

}

}

}

}

void DKnap(int p[],int w[],int \*x,int m,int n)

{

int b[n+2];

PW pair[100];

pair[1].p=0;

stepcount++;

pair[0].w=0;

stepcount++;

int t=1,h=1,next;

stepcount+=2;

b[0]=1;

stepcount++;

next=b[1]=2;

stepcount++;

for(int i=1;i<=n;i++)

{

stepcount++;

int k=t;

stepcount++;

int u=largest(pair,w,t,h,i,m);

stepcount++;

for(int j=t;j<=u;j++)

{

stepcount++;

int pp=pair[j].p+p[i];

stepcount++;

int ww=pair[j].w+w[i];

stepcount++;

while(k<=h && pair[k].w<=ww)

{

stepcount++;

pair[next].p=pair[k].p;

stepcount++;

pair[next].w=pair[k].w;

stepcount++;

next++;

stepcount++;

k++;

stepcount++;

}

stepcount++;

if(k<=h && pair[k].w==ww)

{

stepcount++;

if(pp<pair[k].p)

{

pp=pair[k].p;

stepcount++;

}

k++;

stepcount++;

}

stepcount++;

if(pp>pair[next-1].p)

{

pair[next].p=pp;

stepcount++;

pair[next].w=ww;

stepcount++;

next++;

stepcount++;

}

while(k<=h && pair[k].p<=pair[next-1].p)

{

stepcount++;

k++;

}

stepcount++;

}

stepcount++;

while(k<=h)

{

stepcount++;

pair[next].p=pair[k].p;

stepcount++;

pair[next].w=pair[k].w;

stepcount++;

k++;

stepcount++;

next++;

stepcount++;

}

stepcount++;

t=h+1;

stepcount++;

h=next-1;

stepcount++;

b[i+1]=next;

stepcount++;

}

cout<<"\nSubsets are:\n";

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

{

cout<<"S"<<i<<" = {";

stepcount++;

for(int j=b[i];j<=b[i+1]-1;j++)

{

stepcount++;

cout<<"("<<pair[j].p<<","<<pair[j].w<<"), ";

}

cout<<"\b\b}"<<endl;

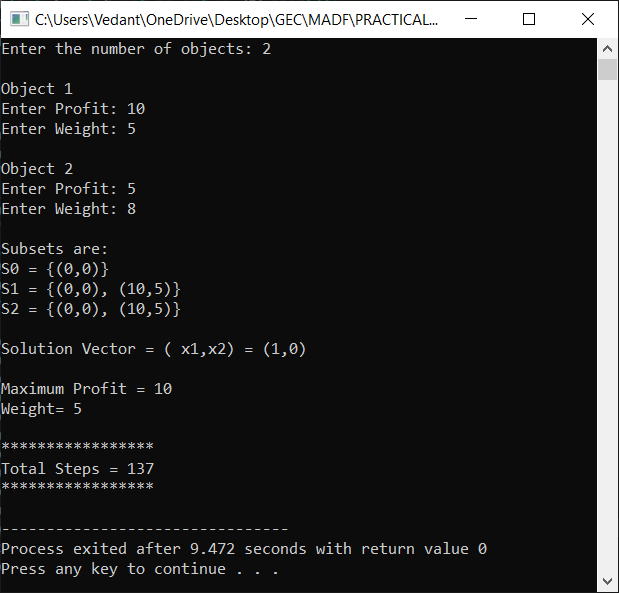
}

stepcount++;

Traceback(p,w,pair,x,b,m,n);

}

**Output**



**Conclusion**

* Detailed concept of 0/1 Knapsack Problem (Dynamic Programming) was studied successfully.
* Program using 0/1 Knapsack Algorithm was executed successfully.
* The step count for the 0/1 Knapsack Algorithm was obtained.