**Name: Akshita Pathak**

**Roll No: 102203796**

**Subgroup: 2CO18**

Lab Assignment 4

Write a program to implement the following using backtracking approach:

1. **n-queen problem**

/\*

1. n queen problem:

\*/

#include<iostream>

using namespace std;

//x- colm where queen is placed

//i->row

//j->colm

const int n=4; //ie 4 queens and 4x4 chessboard

//x[i] is a global array that stores column nos

int x[n];

//returns true/false:

bool place(int i,int j){

for(int k=1;k<i;k++){

//if not safe ie conflict occurs-> ie no same clm/diag

if(x[k]==j || abs(x[k]-j)==abs(k-i)){

return false;

}

}

//safe to place queen

return true;

}

void nqueen(int i,int n){

//where n is dimension of board ie no of queens

//i is row

//first we pass i=1 ie first row

for(int j=1;j<=n;j++){

if(place(i,j)){

//if it is safe to place queen

x[i]=j;

if(i==n){

//on reaching last row print all positions of queen on board

//print x[1..n];

for(int m=1;m<=n;m++){

cout<<x[m]<<" ";

}

cout<<endl;

}

else{

nqueen(i+1,n); //recursion for next row

}

}

}

}

int main()

{

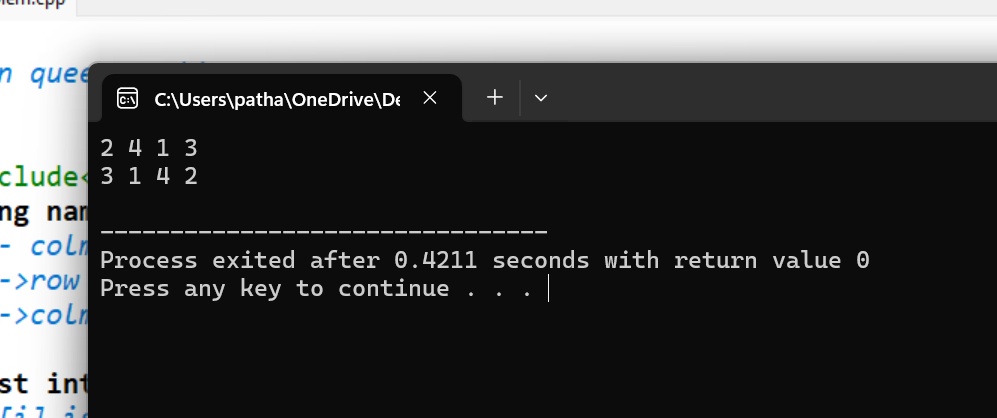
//n: dimension of chess board ie no of queens

nqueen(1,n);

return 0;

}

Output:



1. **Sum-of-Subsets**

/\*

2. Sum-of-Subsets

FIND A SUBSET OF WEIGHTS SUCH THAT THEIR SUM EQUALS SOME M

\*/

#include<iostream>

using namespace std;

//s:current sum of weights (w\*x)

//k:index of object under consideration

//r: remaining weight(not yet included objects - w)

//w: array to store weights of elements

//x:array whetehr element is included in subset

void sumofsubsets(int w[],int x[],int s,int k,int r,int M){

//if sum of current elements in subset=M then print it

if(s==M){

for(int i=0;i<=k;i++){

//include first element in subset

if(x[i]==1){

//this will print if element is included:1 or not:0

cout<<w[i]<<" ";

}

}

cout<<"are elements to be included in subset. \n";

cout<<endl;

}

//check if left child can be included n does not exceed M:

else if(s+w[k]<=M && s+r>=M){

x[k]=1;

//sumofsubsets(int w[],int x[],int s,int k,int r,int M)

sumofsubsets(w,x,s+w[k],k+1,r-w[k],M);

}

//exclude this element from subset list-

x[k]=0;

//check if RIGHT child can be included:

if(s+r-w[k]>=M && s+w[k+1]<=M){

//sumofsubsets(int w[],int x[],int s,int k,int r,int M)

sumofsubsets(w,x,s,k+1,r-w[k],M);

}

}

int main()

{

//M- target sum to be reached

int n;

int M;

cout<<"Enter no of elements: \n";

cin>>n;

cout<<"Enter target sum to be reached: \n";

cin>>M;

int sum=0;

int w[n];

int x[n];

//initially intitialise x array with zeroes

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

x[i]=0; //ie all elements are NOT currently included

}

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

cout<<"Enter weight of element "<<i+1<<" = ";

cin>>w[i];

sum+=w[i];

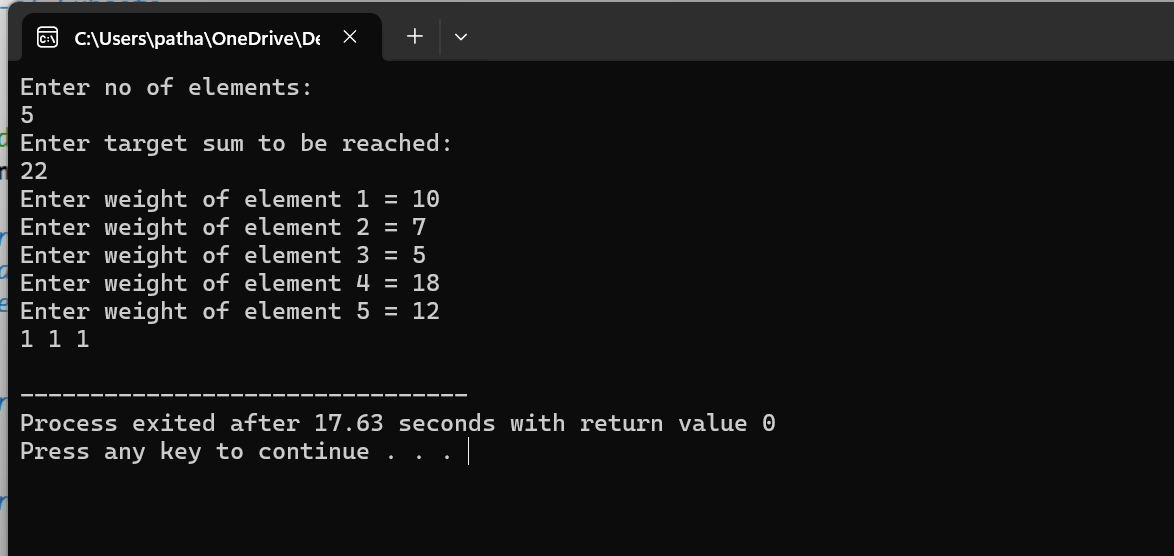
}

sumofsubsets(w,x,0,0,sum,M);

return 0;

}

Output:



1. **Graph Coloring**

/\*

3.Graph Coloring

\*/

#include<iostream>

using namespace std;

//vertex k

//color c

//no adjacent vertices have same color

//n- no of vertices

const int n=4;

//m-> no of colors

const int m=3;

//adjacency matrix-

//int G[n][n];

//array for storing color of each vertex

int x[n];

//is it safe to color vertex k with color c

bool isSafe(int k,int c,int G[][n],int x[]){

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

if(G[k][i]==1 && c==x[i]){

return false;

}

}

return true;

}

void graphcolor(int k,int G[][n],int x[]){

//thru all colors loop

for(int c=1;c<=m;c++){

if(isSafe(k,c,G,x)){

x[k]=c;

if(k+1<n){

graphcolor(k+1,G,x);

}

else{

cout<<"vertex colors are: \n";

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

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

}

cout<<endl;

}

x[k]=0; //for backtracking n try different path

}

}

}

int main()

{

//n-global-no of vertices

//graph G adjacency matrix

int G[n][n]={

{0,1,1,1},

{1,0,1,0},

{1,1,0,1},

{1,0,1,0}

};

// Initialize all vertices as uncolored

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

x[i]=0;

}

// Start coloring from vertex 0

graphcolor(0,G,x);

return 0;

}

Output:

