**Experiment No: 13**

**AIM:** Implementation of graph coloring problem

**THEORY:**

Given a graph G having *n* nodes, we want to discover whether the nodes of G can be colored in such a way that no two adjacent nodes have the same color and yet only *m* nodes are used, where *m* is a positive integer. This is termed as *m-colorability decision making problem*. Notably, if d is the degree of the graph then the graph can be colored with d+1 colors.

The problem is efficiently solved in programming using a concept called Backtracking.

Backtracking is an algorithmic technique for solving problems recursively by trying to build a solution incrementally, one piece at a time, removing those solutions that fail to satisfy the constraints of the problem at any point of time.

**ALGORITHM:**

**Algorithm** mColoring(k)

// This algorithm was formed using the recursive backtracking

// schema. The graph is represented by its Boolean adjacency

// matrix G[1:n, 1:n]. All assignments of 1,2,..., m to the

// vertices of the graph such that adjacent vertices are

// assigned distinct integers are printed, k is the index

// of the next vertex to color.

{

Repeat

{

//Generate all legal assignments for x[k].

NextValue(k); // Assign to x[k] a legal color.

if (x[k]= 0) then return;// No new color possible

if (k = n) then // At most m colors have been used to color the n vertices.

write (x[1:n]);

else mColoring(k+1);

}until(false);

Algorithm NextValue(k)

// x[1],..., x[k-1] have been assigned integer values in

// the range [1,m] such that adjacent vertices have distinct

// integers. A value for x[k] is determined in the range

// [0,m]. x[k] is assigned the next highest numbered color

// while maintaining distinctness from the adjacent vertices

// of vertex k. If no such color exists, then x[k] is 0.

{

Repeat

{

x[k] :=(x[k]+1) mod(m+ 1);// Next highest color.

if (x[k]=0) then return;// All colors have been used.

for j:=1 to n do

{

// Check if this color is distinct from adjacent colors.

if ((G[k,i]≠0)and (x[k]=x[j)) // If (k,j) is an edge and if adj. vertices have the same color.

then **break**;

}

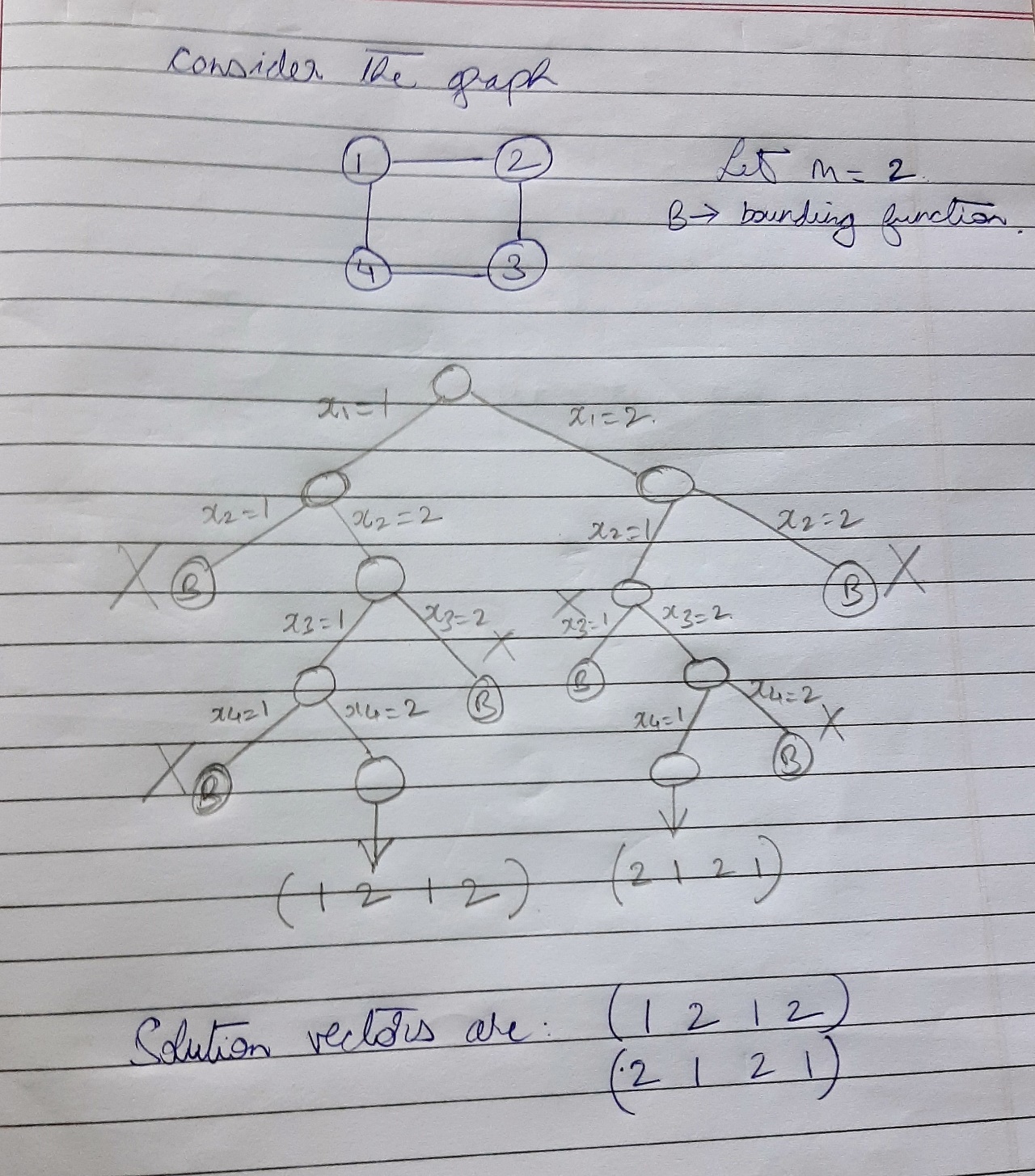
if (j = n + 1) then return; // New color found

}until(false);// Otherwise try to find another color

*Time Complexity*

• The time complexity of this algorithm is O (m^n), where m is the number of colors that can be used.

*Problem Tracing*



PROGRAM IMPLEMENTATION:

#include<iostream>

using namespace std;

int n,m,graph[10][10],\*x,count=0;

void showset()

{

cout<<"( ";

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

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

cout<<")"<<endl;

count++;

}

void nextValue(int k)

{

int j;

do

{

x[k] = (x[k]+1)%(m+1);

if(x[k] == 0)

return;

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

if((graph[k][j] != 0) && (x[k] == x[j]))

break;

if(j==n)

return;

}while(true);

}

void mColoring(int k)

{

do

{

nextValue(k);

if(x[k] == 0)

return;

if(k == n-1)

showset();

else

mColoring(k+1);

}while(true);

}

int main()

{

cout<<"Enter number of vertices: ";

cin>>n;

x = new int[n];

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

x[i]=0;

char ans;

cout<<"Hit 'y' if there is an edge between the corresponding vertices, else hit 'n'\n\n";

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

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

{

if(graph[i][j]==1 || i>j)

continue;

graph[i][j]=0;

if(i!=j)

{

cout<<"Edge between "<<i<<" and "<<j<<" ? : ";

cin>>ans;

if(ans == 'y')

graph[i][j]=graph[j][i]=1;

}

}

int k=0;

cout<<"\nEnter number of colors: ";

cin>>m;

mColoring(k);

if(count == 0)

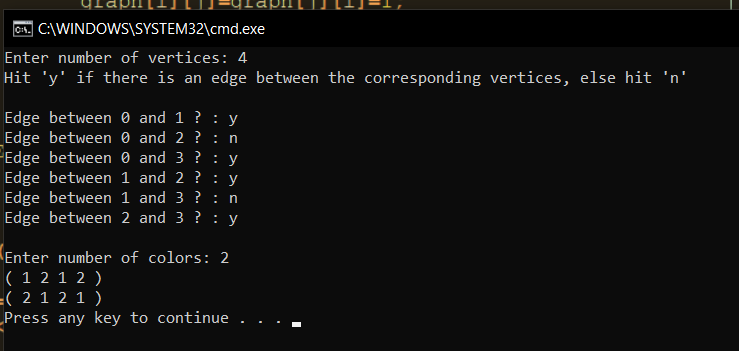
cout<<"No solutions."<<endl;

return 0;

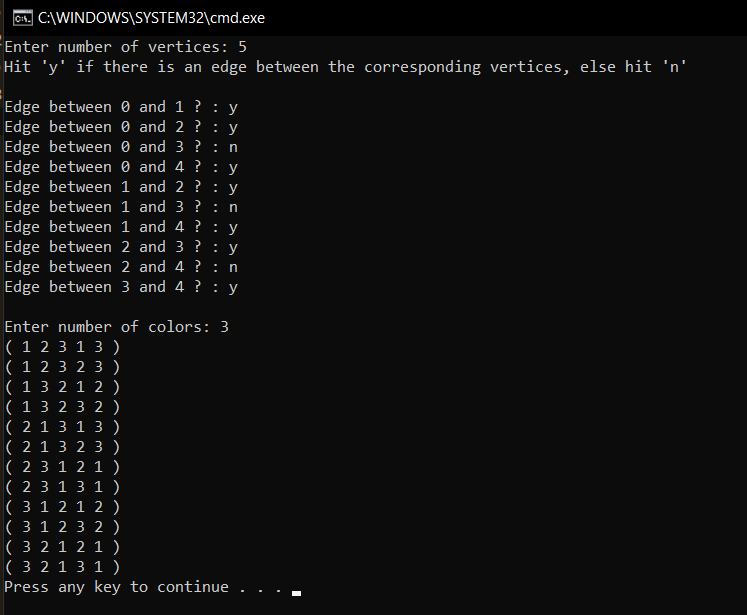
}

OUTPUTS:

1. When n=4



1. When n=5

****

**Conclusion**:

* **Time complexity of the algorithm is of the order of O(m^n), where m is the max number of colors that can be used and n is the number of nodes of the graph.**