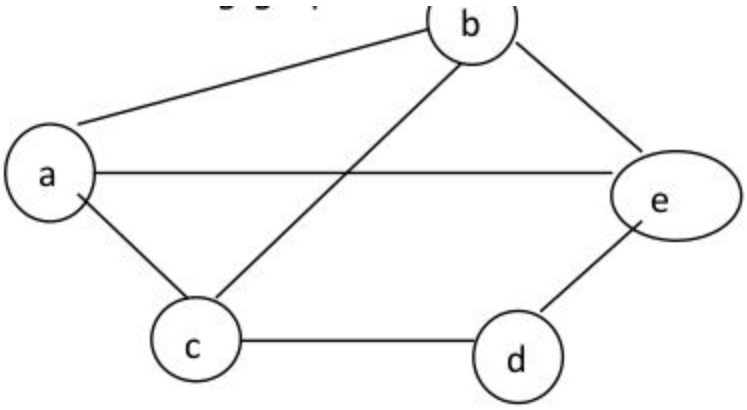
LAB 6: BACKTRACKING APPROACH

Date : 11.4.18

Submitted By : Abhineet Singh

1.***Write a program to implement a graph coloring problem with 3 colors using backtracking for the following graph.***

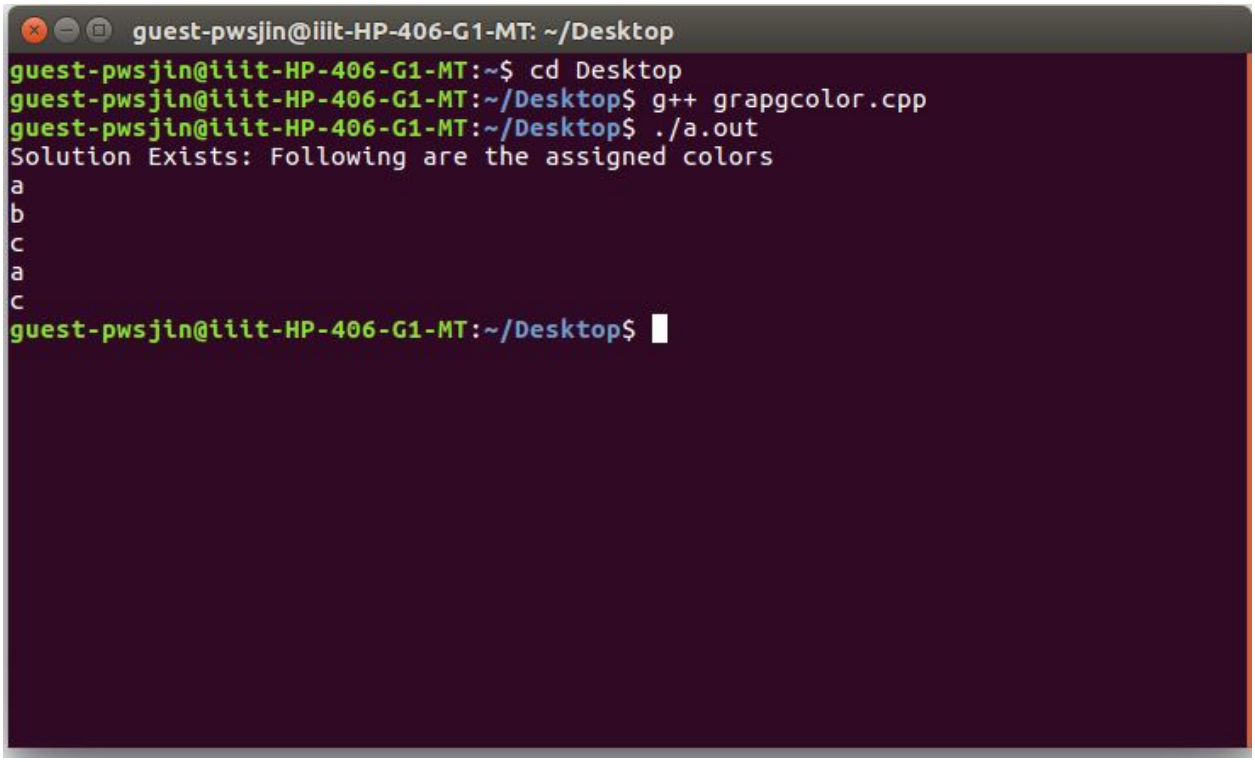



```
#include<iostream>
#include<cstdio>
#include <stdbool.h>
#include<vector>


// Number of vertices in the graph
#define V 5
using namespace std;
void printSolution(int color[]);

/* A utility function to check if the current color assignment
   is safe for vertex v */
bool isSafe (int v, bool graph[V][V], int color[], int c)
{
        for (int i = 0; i < V; i++)
        if (graph[v][i] && c == color[i])
        return false;
        return true;
}

/* A recursive utility function to solve m coloring problem */
bool graphColoringUtil(bool graph[V][V], int m, int color[], int v)
```

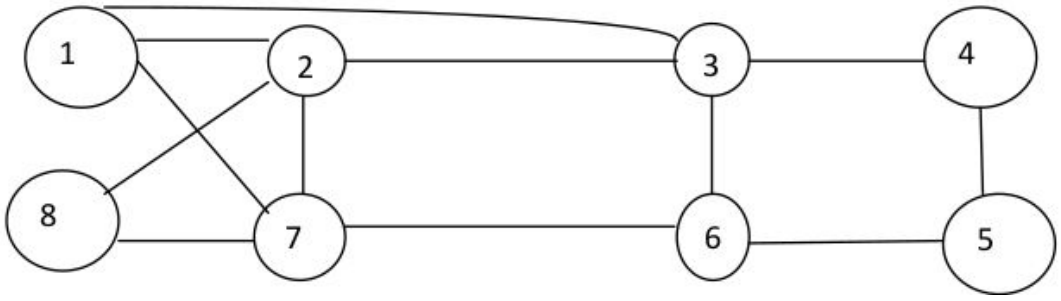
```
{
        /* base case: If all vertices are assigned a color then
        return true */
        if (v == V)
        return true;

        /* Consider this vertex v and try different colors */
        for (int c = 1; c <= m; c++)
        {
        /* Check if assignment of color c to v is fine*/
        if (isSafe(v, graph, color, c))
        {
        color[v] = c;

        /* recur to assign colors to rest of the vertices */
        if (graphColoringUtil (graph, m, color, v+1) == true)
        return true;

        /* If assigning color c doesn't lead to a solution
        then remove it */
        color[v] = 0;
        }
        }

        /* If no color can be assigned to this vertex then return false */
        return false;
}

/* This function solves the m Coloring problem using Backtracking.
  It mainly uses graphColoringUtil() to solve the problem. It returns
  false if the m colors cannot be assigned, otherwise return true and
  prints assignments of colors to all vertices. Please note that there
  may be more than one solutions, this function prints one of the
  feasible solutions.*/
bool graphColoring(bool graph[V][V], int m)
{
        // Initialize all color values as 0. This initialization is needed
        // correct functioning of isSafe()
        int *color = new int[V];
        for (int i = 0; i < V; i++)
        color[i] = 0;

        // Call graphColoringUtil() for vertex 0
```

```
        if (graphColoringUtil(graph, m, color, 0) == false)
        {
        printf("Solution does not exist");
        return false;
        }

        // Print the solution
        printSolution(color);
        return true;
}

/* A utility function to print solution */
void printSolution(int color[])
{
        printf("Solution Exists:"
        " Following are the assigned colors \n");\
        vector <int> v;
        vector <char> t;

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

        v.push_back( color[i]);

   for(int i=0;i<v.size();i++)
   {
         if(v[i]==1)
                t.push_back('a');
         else if(v[i]==2)
                t.push_back('b');
         else if(v[i]==3)
                t.push_back('c');
        }
   for(int i=0;i<t.size();i++)
   cout<<t[i]<<" "<<endl;

}

// driver program to test above function
int main()
{

        bool graph[V][V] = {{0, 1, 1, 0,1},
                {1, 0, 1, 0,1},
```

```
                {1, 1, 0, 1,0},
                {0, 0, 1, 0,1},
                 {1, 1, 0, 1,0},
        };
        int m = 3; // Number of colors
        graphColoring (graph, m);
        return 0;
}
```


guest-pwsjin@iiit-HP-406-G1-MT: ~/Desktop
guest-pwsjin@iiit-HP-406-G1-MT:~$ cd Desktop
guest-pwsjin@iiit-HP-406-G1-MT:~/Desktop$ g++ grapgcolor.cpp
guest-pwsjin@iiit-HP-406-G1-MT:~/Desktop$ ./a.out
Solution Exists: Following are the assigned colors
a
b
c
a
c
guest-pwsjin@iiit-HP-406-G1-MT:~/Desktop$

*2.**Write a program to find the Hamiltonian cycle in the given graphs.***

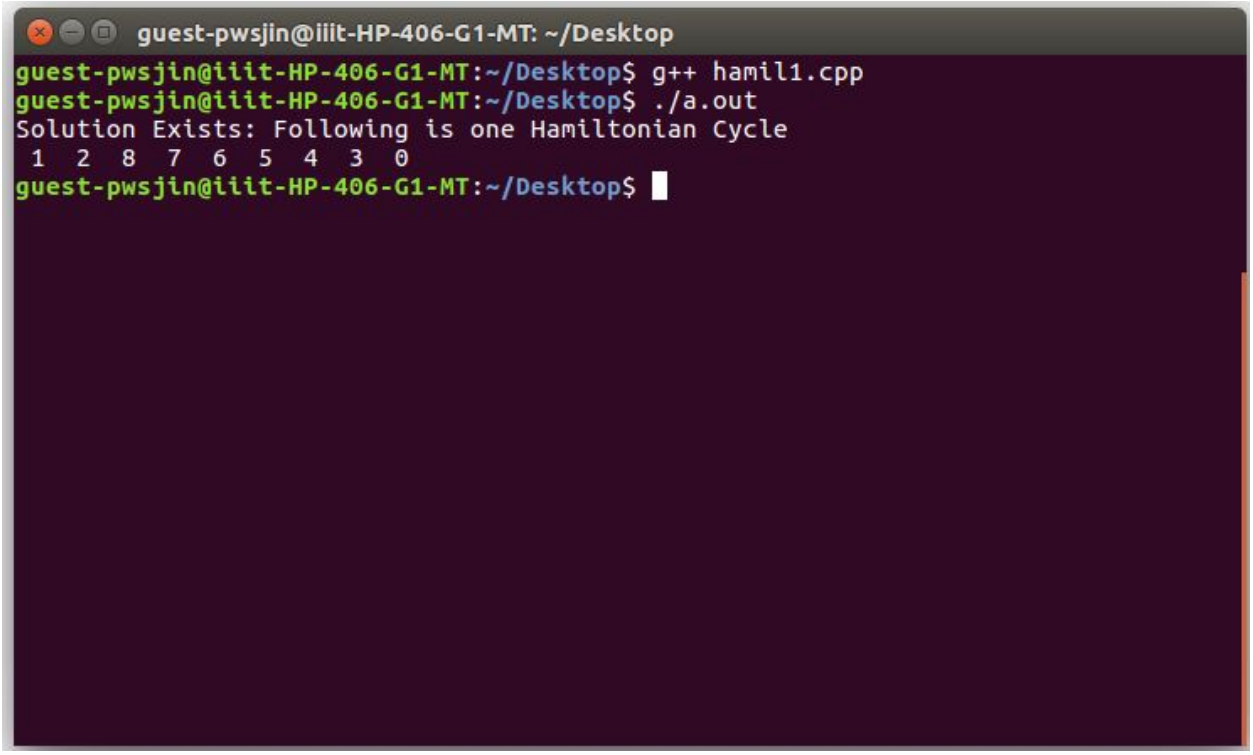



```
#include<iostream>
#include<cstdio>
#include <stdbool.h>

// Number of vertices in the graph
#define V 8

void printSolution(int path[]);

/* A utility function to check if the vertex v can be added at
   index 'pos' in the Hamiltonian Cycle constructed so far (stored
   in 'path[]') */
bool isSafe(int v, bool graph[V][V], int path[], int pos)
{
        /* Check if this vertex is an adjacent vertex of the previously
        added vertex. */
        if (graph [ path[pos-1] ][ v ] == 0)
        return false;

        /* Check if the vertex has already been included.
        This step can be optimized by creating an array of size V */
        for (int i = 0; i < pos; i++)
        if (path[i] == v)
        return false;

        return true;
}

/* A recursive utility function to solve hamiltonian cycle problem */
bool hamCycleUtil(bool graph[V][V], int path[], int pos)
{
        /* base case: If all vertices are included in Hamiltonian Cycle */
        if (pos == V)
```

```
        {
        // And if there is an edge from the last included vertex to the
        // first vertex
        if ( graph[ path[pos-1] ][ path[0] ] == 1 )
        return true;
        else
        return false;
        }

        // Try different vertices as a next candidate in Hamiltonian Cycle.
        // We don't try for 0 as we included 0 as starting point in in hamCycle()
        for (int v = 1; v < V; v++)
        {
        /* Check if this vertex can be added to Hamiltonian Cycle */
        if (isSafe(v, graph, path, pos))
        {
        path[pos] = v;

        /* recur to construct rest of the path */
        if (hamCycleUtil (graph, path, pos+1) == true)
                return true;

        /* If adding vertex v doesn't lead to a solution,
        then remove it */
        path[pos] = -1;
        }
        }

        /* If no vertex can be added to Hamiltonian Cycle constructed so far,
        then return false */
        return false;
}

/* This function solves the Hamiltonian Cycle problem using Backtracking.
  It mainly uses hamCycleUtil() to solve the problem. It returns false
  if there is no Hamiltonian Cycle possible, otherwise return true and
  prints the path. Please note that there may be more than one solutions,
  this function prints one of the feasible solutions. */
bool hamCycle(bool graph[V][V])
{
        int *path = new int[V];
        for (int i = 0; i < V; i++)
        path[i] = -1;
```

```
        /* Let us put vertex 0 as the first vertex in the path. If there is
        a Hamiltonian Cycle, then the path can be started from any point
        of the cycle as the graph is undirected */
        path[0] = 0;
        if ( hamCycleUtil(graph, path, 1) == false )
        {
        printf("\nSolution does not exist");
        return false;
        }

        printSolution(path);
        return true;
}

/* A utility function to print solution */
void printSolution(int path[])
{
        printf ("Solution Exists:"
        " Following is one Hamiltonian Cycle \n");
        for (int i = 0; i < V; i++)
        printf(" %d ", path[i]+1);

        // Let us print the first vertex again to show the complete cycle
        printf(" %d ", path[0]);
        printf("\n");
}

// driver program to test above function
int main()
{

   bool graph1[V][V] = {{0,1,1,0,0,0,1,0},
                        {1,0,1,0,0,0,1,1},
                        {1,1,0,1,0,1,0,0},
                        {0,0,1,0,1,0,0,0},
                        {0,0,0,1,0,1,0,0},
                        {0,0,1,0,1,0,1,0},
                        {1,1,0,0,0,1,0,1},
                        {0,1,0,0,0,0,1,0},
                        };
```

```
        hamCycle(graph1);

        return 0;
}
```


guest-pwsjin@iiit-HP-406-G1-MT: ~/Desktop
guest-pwsjin@iiit-HP-406-G1-MT:~/Desktop$ g++ hamil1.cpp
guest-pwsjin@iiit-HP-406-G1-MT:~/Desktop$ ./a.out
Solution Exists: Following is one Hamiltonian Cycle
1 2 8 7 6 5 4 3 0
guest-pwsjin@iiit-HP-406-G1-MT:~/Desktop$