Experiment No.9

Graph Colouring Problem Using Back Tracking

Registration No.:-24141028

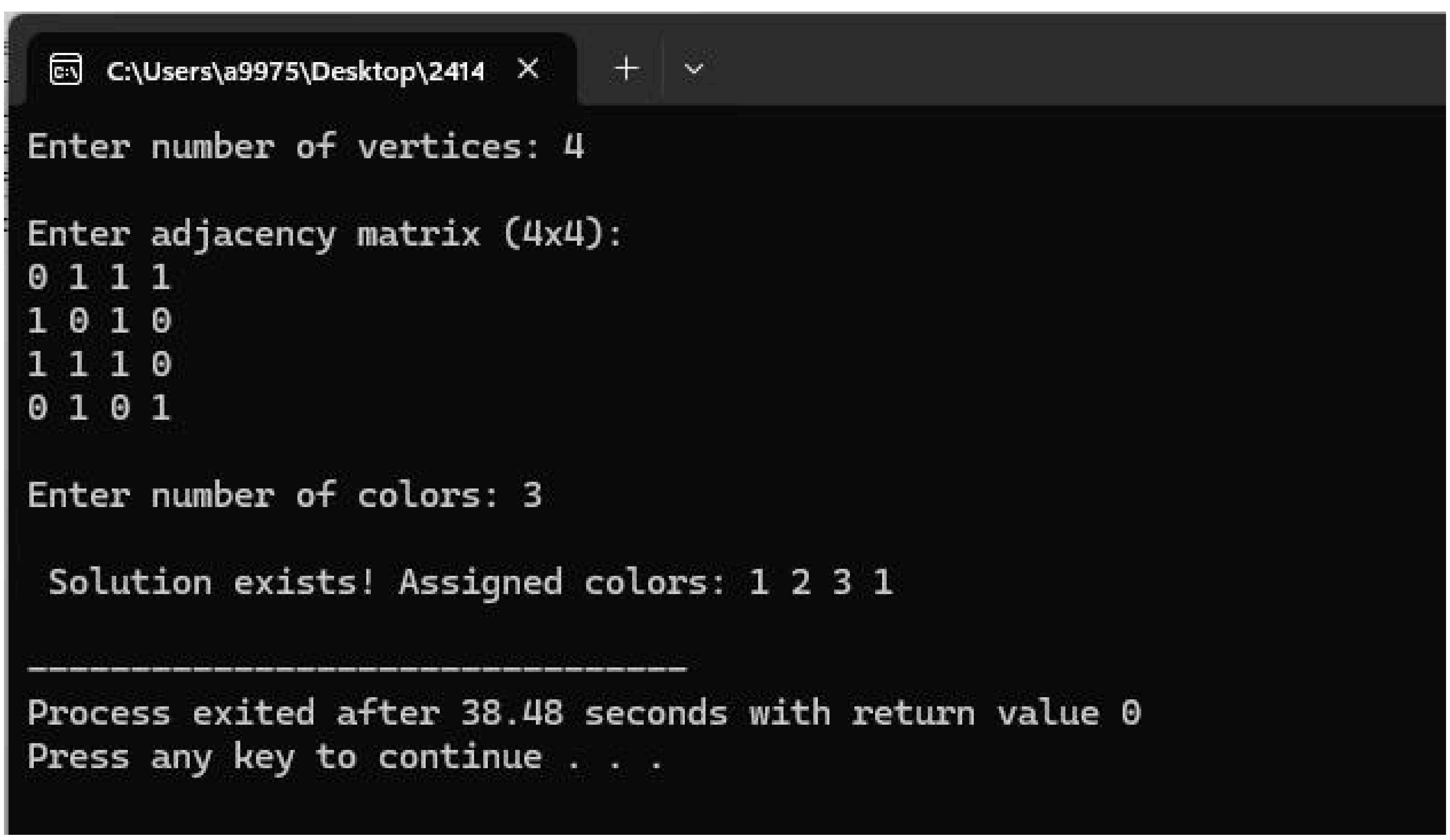
Program:-

```
#include<iostream>
#include<vector>
using namespace std;
bool isSafe(int v, vector<vector<int>>&graph, vector<int>&color, int c) {
int i;
   for ( i = 0; i<graph.size(); i++) {
      if (graph[v][i]&&color[i] == c)
         return false;
   }
   return true;
}
bool graphColoringUtil(vector<vector<int>>&graph, int m, vector<int>&color, int v) {
int c ;
   if (v == graph.size())
      return true;
   for ( c = 1; c<= m; c++) {
      if (isSafe(v, graph, color, c)) {
         color[v] = c;
         if (graphColoringUtil(graph, m, color, v + 1))
            return true;
         color[v] = 0;
      }
   }
   return false;
}
bool graphColoring(vector<vector<int>>&graph, int m) {
   int V = graph.size();
   vector<int>color(V, 0);

   if (!graphColoringUtil(graph, m, color, 0)) {
      cout<<"\nNo solution exists ??"<<endl;
      return false;
   }
   int i;
   cout<<"\n Solution exists! Assigned colors: ";
   for (i=0; i<V ;i++)
      cout<<color[i] <<" ";
   cout<<endl;
   return true;
```

```
}
int main() {
   int i,j,V;
   cout<<"Enter number of vertices: ";
   cin>>V;

   vector<vector<int>>graph(V, vector<int>(V));
   cout<<"\nEnter adjacency matrix ("<<V<<"x"<<V<<"):\n";
   for (i = 0; i<V; i++) {
      for ( j = 0; j<V; j++) {
         cin>>graph[i][j];
      }
   }

   int m;
   cout<<"\nEnter number of colors: ";
   cin>>m;
   graphColoring(graph, m);

   return 0;
}
```

Output:-

Algorithm: Graph Colouring using Backtracking



Algorithm :-

Input: A graph G(V, E) represented as an adjacency matrix. Number of colours m.

Output: A colour assignment for each vertex such that no two adjacent vertices have the same colour, or report failure

Step 1: Initialize an array color[V] to 0 (no colours assigned).

Step 2: Call the recursive function solve(v) starting with vertex v = 0.

Step 3:

solve(v):1. If v == V, all vertices are coloured → return true.

2. For each colour c in {1, 2, … , m}:

1. If isSafe(v, c) is true: Assign color[v] = c. If solve(v + 1) returns true, return true. Else, reset color[v] = 0 (backtrack).

3. If no valid colour is found, return false.

Step 4: isSafe(v, c): For every vertex i adjacent to v,

if graph[v][i] == 1 and color[i] == c, return false. Return true if no conflicts exist.

Step 5: If solve(0) returns true, output the color[] array;

else, output "No solution exists".

Time Complexity: O(m^V)

Space Complexity: O(V)

List of Applications:-

1. Register allocation in compilers
2. Scheduling problems

3. Map colouring
4. Frequency assignment in mobile networks
5. Timetable scheduling
6. Sudoku solving
7. Circuit design
8. Task assignment in parallel processing
9. Pattern matching in image segmentation
10. Channel assignment in wireless communication systems