كليـــــة .....

كونترول الفرقة .....

العـــام الجامعـــى 2019 / 2020 – دور مايـــو

الغلاف الخارجى للبحث

أولاً: البيانات الخاصة بالطالب				
الفرقة الدراسية	الثانية		التخصص	عام
اسم القسم	عام			
اسم المقرر	Algorithms			
استاذ المقرر	Dr \ Marwa M. A. Elfattah			
ثانياً: البيانات الخاصة بالبحث				
عنوان البحث	Graph coloring			
طبيعة المشاركة	بحث فردى		بحث جماعى	
ارسال البحث	بواسطة البريد الالكتروني amr.alaa.fcih@gmail.com			
اسماء الطلاب المشاركين فى البحث (يكتب الاسم رباعيا)	م	الاسم رباعى	رقم الجلوس	الرقم القومى
	1			
	2			
	3			
	4			
	5			
تاريخ الإرسال	2020 / 6 /			
ثالثاً: البيانات الخاصة بالكونترول				
النتيجة	ناجح		راسب	
أعضاء لجنة تقييم البحث		الاسماء		التوقيع
	1			
	2			
	3			

فى حالة عدم قبول البحث يرجى ذكر الأسباب	- ..... - ..... - ..... - .....

- **Problem statement:**
  - **Introduction and challenges that this project aims to solve:**

A graph coloring is a special case of graph labeling. This is a way of coloring a graph's vertices in its simplest form, so that no two adjacent vertices are of the same color.

The idea of using colors stems from the color of a map's nations. Coloring the faces of a graph embedded in the plane was generalized. Typical for mathematical and computer representations is the use of the first few positive or non-positive.

To solve this problem there are 2 famous algorithms:

- **Greedy coloring:**
  The greedy algorithm considers the vertices in a specific order. The quality of the resulting coloring depends on the chosen ordering. There exists an ordering that leads to a greedy coloring with the optimal number of colors.
- **Backtracking coloring:**
  The idea is to assign colors one by one to different vertices, starting from the vertices 0. Check for validation by considering already assigned colors to the adjacent vertices.

– Pseudocode:

- **Solution 1: using Backtracking**

Beginning: takes a graph matrix and a number of colors

For every color:

If *current color is not taken by current vertex's neighbors*:

Assign current color to the current vertex

If *we can apply the rest of colors to the neighbor vertices by calling the function again:*

Return Array of colors

Else:

Remove the color for the current vertex and true a new one

Else:

Return false

- **Solution 2: using Greedy**

Beginning: takes a graph matrix and a number of colors

For every vertex:

For each vertex neighbor:

Mark their color as unavailable

For each color:

If color **is** available:

Assign it to the current vertex

If all vertices **are** colored:

Return color array

Else:

Return empty array as false

- **Time complexity**:

  - **Solution 1: using Backtracking**

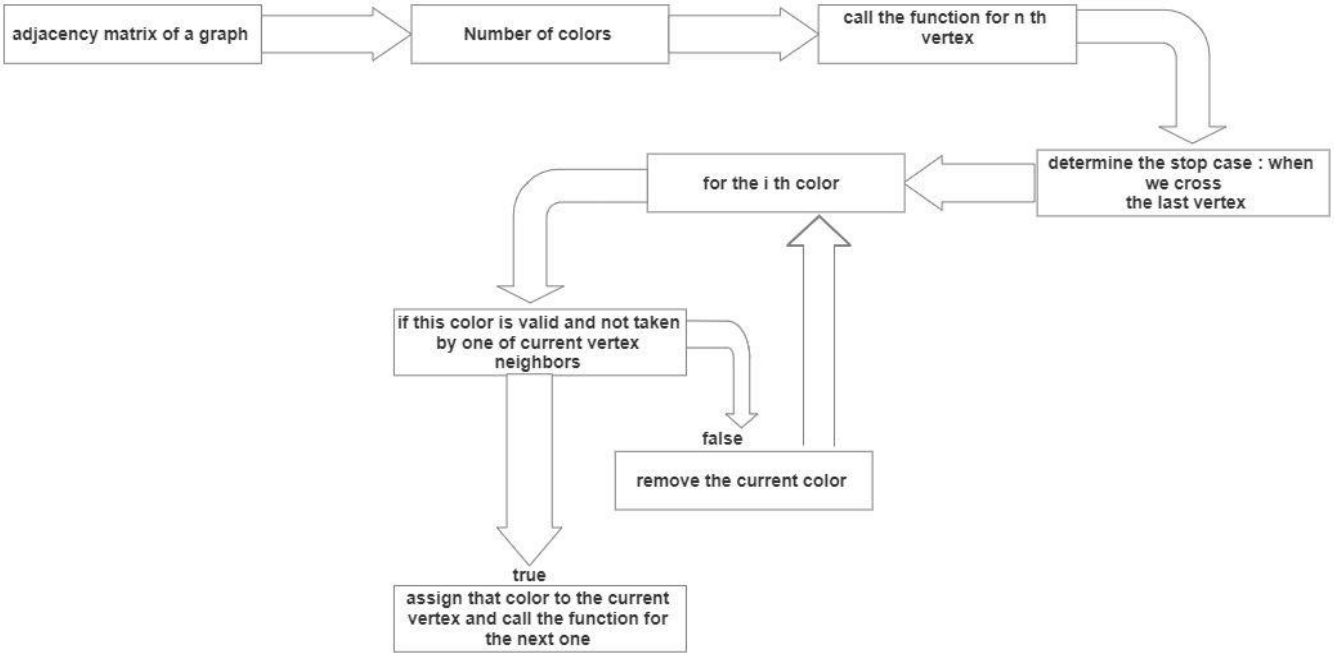
    There are n recurrence levels, and the recursion branches V ways each level, so the time is $O(n^V)$.
    V -> number of vertices
    n -> number of colors

  - **Solution 2: using Greedy Coloring**

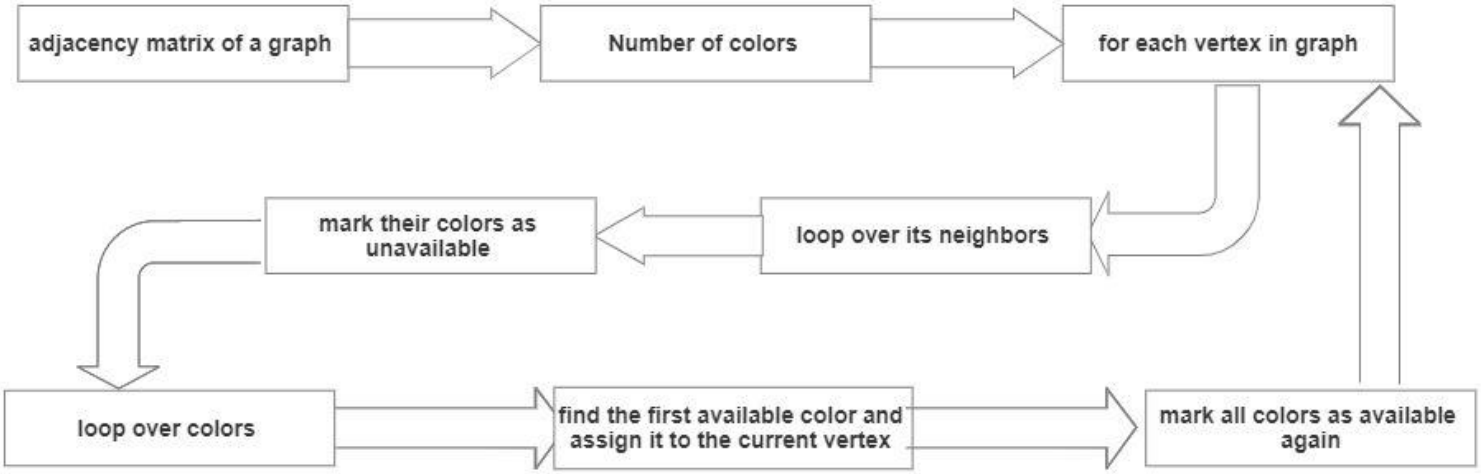
    $\sum_0^V(\sum_0^V 1 + \sum_1^n 1) = V^2 + Vn = O(V^2)$

    Where is V–> vertex, n –> number of colors

- **Block Diagram:**
  - **Solution 1: using Backtracking**



  - **Solution 2: using Greedy Coloring**



- **Output:**

```java
package com.algo;

import java.util.Arrays;

public class Main {
    public static void main(String[] args) {
        int[][] arr={
                {0,1,1,0,0,1},
                {1,0,1,1,0,0},
                {1,1,0,1,0,1},
                {0,1,1,0,1,0},
                {0,0,0,1,0,1},
                {1,0,1,0,1,0}
        };
        Coloring coloring=new Coloring(arr, m: 4);
        Coloring2 coloring2=new Coloring2(arr, numberOfColors: 4);
        System.out.println(Arrays.toString(coloring2.greedyColoring()));
        System.out.println(Arrays.toString(coloring.colorIt()));
    }
}
```

```
"C:\Program Files\Java\jdk-13\bin\j
[1, 2, 3, 1, 2, 4]
[1, 2, 3, 1, 2, 4]

Process finished with exit code 0
```

- **Source Code:**

  - **Solution 1: using Backtracking**

```
package com.algo;

import java.util.Arrays;

public class Coloring {
   private int[][] matrix;
   private int numberOfColors;
   private int[] colors;

   public Coloring(int[][] matrix,int m){
      this.matrix=matrix;
      this.numberOfColors=m;
      colors=new int[matrix.length];
   }

   private boolean isValid(int currentVertex,int currentColor){
      for (int i=0;i<matrix.length;i++){
         if(matrix[currentVertex][i]==1&&currentColor==colors[i]){
            return false;
         }
      }
      return true;
   }

   private boolean canBeColoredWith(int currentVertex){

      if(currentVertex==matrix.length){
         return true;
      }
      for (int i=1;i<=numberOfColors;i++){
         if (isValid(currentVertex,i)){
            colors[currentVertex]=i;
            if (canBeColoredWith(currentVertex+1)){
               return true;
            }
            colors[currentVertex]=0;
         }
      }
      return false;
   }

   public int[] colorIt(){
      Arrays.fill(colors,0);
      if (!canBeColoredWith(0)){
         return new int[]{};
      }
      return colors;
   }

}
```

- **Solution 2: using Greedy Coloring**

```
public class Coloring2 {
   private HashMap<Integer, ArrayList<Integer>> vertex=new HashMap<>();
   private boolean[] availableColors;
   private int[] listOfColors;
   private int numberOfColors;
   private int[] colors;
   public Coloring2(int[][] matrix,int numberOfColors){
      this.availableColors=new boolean[numberOfColors];
      this.listOfColors=new int[numberOfColors];
      this.colors=new int[matrix.length];
      this.numberOfColors=numberOfColors;
      for (int i=0;i<numberOfColors;i++){
         listOfColors[i]=i+1;
      }
      for (int i=0;i<matrix.length;i++){
         ArrayList<Integer> temp=new ArrayList<>();
         for(int j=0;j<matrix[i].length;j++){
            if(matrix[i][j]==1){
               temp.add(j);
            }
         }
         vertex.put(i,temp);
      }
   }
   private boolean canBeColoredWith(int[] colors){
      for (int color:colors){
         if(color==0){
            return false;
         }
      }
      return true;
   }
   public int[] greedyColoring(){
      Arrays.fill(availableColors,true);  //m
      Arrays.fill(colors,0); //m
      for (int currentVertex: vertex.keySet()){  //v
         for(int i=0;i<vertex.get(currentVertex).size();i++){ //v
            ArrayList<Integer> temp=vertex.get(currentVertex);
            if(colors[temp.get(i)]!=0){
               availableColors[colors[temp.get(i)]-1]=false;
            }
         }
         for (int j=0;j<numberOfColors;j++){  //m
            if (availableColors[j]){
               colors[currentVertex]=listOfColors[j];
               break;
            }
         }
         Arrays.fill(availableColors,true);
      }
      if (canBeColoredWith(colors)){
         return colors;
      }else {
         return new int[]{};
```