数据结构与算法

实验报告（三）

实验题目： 地图染色问题

姓 名： 魏靖

学 号： 2021213513

日 期： 2022.05.29

自我评分： 【 A+ 】95

自我评分说明：A+，A，B+，B，B-，C，D，分别对应分数95、90、85、80、75、70、60

诚信声明

本人郑重承诺：本实验程序和实验报告均是本人独立学习和工作所获得的成果。尽我所知，实验报告中除特别标注的地方外，不包含其他同学已经发表或撰写过的成果；实验程序中对代码工作的任何帮助者所作的贡献均做了明确的说明，并表达了谢意。

如有抄袭，本人原因承担因此而造成的任何后果。

特此声明。

签名：魏靖 日期：2022.05.29

程序引用说明

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 引用项 | 来源 | 相同代码行数 |
| 1 | 查找函数 | 《书名》 |  |
| 2 | 构造有序链表函数 | 互联网网址 |  |
| 3 |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
| 小计 | | |  |

总代码行数\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_; 引用占比\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**1、实验简介**

通过具体代码实现地图染色，掌握图的存储结构表示，用代码表现图案，最好还能实现最后的输出为图片（当然这个需要进一步深入研究）。

1. **程序框架**

（1）实现思路：

①输入一张图片（邻接关系），我们用无向图表示地图上相邻区域

②依次遍历顶点（**遍历方法怎么选？**）

③设置储存顶点颜色的数组int color【n】（n为顶点数），（颜色可用整型数表示）

④对顶点进行“染色”，然后对比是否与相邻顶点为同一颜色。如果相同，重新选取（选取的时候也是从前往后选，使用的颜色尽可能少）；如果不同，则继续遍历。

1. 问题提取：遍历方法怎么选？

因为不涉及权值的比较，无关生成最小连通路径的问题，这一点要想清楚。

而对简单的图的深度遍历或广度遍历方法来说，假设每个顶点的每个链接都去比较颜色是否相同，则使用的颜色最少就为最优，二者皆是如此，看不出孰优孰劣。而相对来说不好直观判断两顶点是否相邻，我们想到对顶点自行排序，依顺序进行遍历和颜色选取。如果一个顶点的度越多，则它受到的限制也越多，影响别的顶点的因素也更大；度越小，约束越小，颜色数组的颜色就可以按顺序选下去，这样引申出来颜色就会相对更多，而度大的时候更像一种被迫依次选颜色的过程，这样得到的用颜色量就会更少。

然而若出现两个顶点的度一样，又如何排序？

经过20余组图验证，似乎没有太大影响。（当然应该是会有特殊情况的，但是那种图我目前还没有做出来），所以咱们暂且不考虑，而选择最简单的方式（本身比较度大小时存在的循环顺序，不做多余处理）

结构：

#include<iostream>

using namespace std;

const int n = 7; 顶点数

struct NODE

{

int ID;

int Link;

}; 顶点结构

bool Color(int m[n][n], int k, int color[], int n, NODE Node[]) 判断是否相同颜色函数

void Line(int m[][n], NODE Node[]) 排序函数

void Coloring(int m[n][n], int color[n], int n, NODE Node[]) 染色函数

1. **关键代码实现**

**3.1图的输入**

int m[n][n] = {

0,1,1,1,1,1,1,

1,0,1,0,0,0,0,

1,1,0,1,1,0,0,

1,0,1,0,0,0,0,

1,0,1,0,0,1,0,

1,0,0,0,1,0,0,

1,0,0,0,0,0,0

};

**（写入图的邻接矩阵）可拓深：（实现任意一个简单地图的染色）**

(取消前面说的const int n=7)

cout<<”请输入你的地图由几个板块”<<endl;

cin<<n;

int m[n][n]={0};

for（int i=0;i<n;i++)

{

cout<<”请输入与”<<i+1<<”相接的板块”<<endl;

cin>>k;

m[i][k-1]=m[k-1][i]=1;

}

**3.2排序与遍历**

void Line(int m[][n], NODE Node[])

{

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

{

Node[i].ID = i;

Node[i].Link = 0;

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

Node[i].Link += m[i][j]; //利用邻接矩阵得到Link（度数）

}

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

for (int j = 0; j < n - 1; j++)

if (Node[j].Link < Node[j + 1].Link)

//按ID依次比较度，大的排到前面，小的移到后面，其余情况下不改变

{

NODE tmp = Node[j];

Node[j] = Node[j + 1];

Node[j + 1] = tmp;

}

}

1. **不足**

（1）没有探索出度相同时最优解不一致的情况，理论分析又好像达不到可以解决的高度，有一点点意犹未尽，特别是看资料说对于大部分都是成立的，那这一少部分数据到底到哪去找去发现还是个难题（好像貌似是理论上研究得不够）。

（2）有想法但是没有时间去实现从色号设定到图片输出的代码了，所以总还是觉得心里有些遗憾吧，不过下来会具体看一看图片编码的相关内容也为研究计算机视觉做准备。

1. **心得体会**

（1）图的存储结构很直观，并且能够轻松理解是否相关，可以把一些问题简化等等，是很不错的一种数据存储结构。

（2）思考一个问题并把它总结出来是一个锻炼思维的好办法，很有自豪感。

**代码：**

#include<iostream>

using namespace std;

const int n = 7;

struct NODE

{

int ID;

int Link;

};

bool Color(int m[n][n], int k, int color[], int n, NODE Node[])

{

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

if ((m[k][Node[i].ID]) && (color[k] == color[Node[i].ID]))

return false;

return true;

}

void Line(int m[][n], NODE Node[])

{

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

{

Node[i].ID = i;

Node[i].Link = 0;

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

Node[i].Link += m[i][j];

}

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

for (int j = 0; j < n - 1; j++)

if (Node[j].Link < Node[j + 1].Link)

{

NODE tmp = Node[j];

Node[j] = Node[j + 1];

Node[j + 1] = tmp;

}

}

void Coloring(int m[n][n], int color[n], int n, NODE Node[])

{

color[Node[0].ID] = 1;

for (int k = 1; k < n; k++)

{

color[Node[k].ID] = 1;

while (!Color(m, Node[k].ID, color, k, Node))

color[Node[k].ID] += 1;

}

}

int main()

{

int color[n] = { 0 };

int m[n][n] = {

0,1,1,1,1,1,1,

1,0,1,0,0,0,0,

1,1,0,1,1,0,0,

1,0,1,0,0,0,0,

1,0,1,0,0,1,0,

1,0,0,0,1,0,0,

1,0,0,0,0,0,0

};

NODE Node[n];

Line(m, Node);

Coloring(m, color, n, Node);

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

cout << i + 1 << ":" << color[i] << endl;

return 0;

}

**调试结果：**

