深 圳 大 学 实 验 报 告

课程名称 算法设计与分析

项目名称 回溯法--地图填色问题

学　　院 计算机与软件学院

专　　业 软件工程

指导教师 卢亚辉

报 告 人 郑彦薇 学号 2020151022

实验时间 2022/4/18~2022/4/22

提交时间 2022/4/23

教务处制

一、实验目的与要求

1. 掌握回溯法算法设计思想。
2. 掌握地图填色问题的回溯法解法。

二、实验内容与方法

1. 背景知识

为地图或其他由不同区域组成的图形着色时，相邻国家/地区不能使用相同的颜色。我们可能还想使用尽可能少的不同颜色进行填涂。一些简单的“地图”（例如棋盘）仅需要两种颜色（黑白），但是大多数复杂的地图需要更多颜色。

每张地图包含四个相互连接的国家时，它们至少需要四种颜色。1852 年，植物学专业的学生弗朗西斯·古思里（Francis Guthrie）于 1852 年首次提出“四色问题”。他观察到四种颜色似乎足以满足他尝试的任何地图填色问题，但他无法找到适用于所有地图的证明。这个问题被称为四色问题。长期以来，数学家无法证明四种颜色就够了，或者无法找到需要四种以上颜色的地图。直到 1976 年德国数学家沃尔夫冈·哈肯（Wolfgang Haken）（生于 1928 年）和肯尼斯·阿佩尔（Kenneth Appel，1932 年-2013 年）使用计算机证明了四色定理，他们将无数种可能的地图缩减为 1936 种特殊情况，每种情况都由一台计算机进行了总计超过 1000 个小时的检查。

他们因此工作获得了美国数学学会富尔克森奖。在 1990 年，哈肯（Haken）成为伊利诺伊大学（University of Illinois）高级研究中心的成员，他现在是该大学的名誉教授。

四色定理是第一个使用计算机证明的著名数学定理，此后变得越来越普遍，争议也越来越小 更快的计算机和更高效的算法意味着今天您可以在几个小时内在笔记本电脑上证明四种颜色定理。

2. 问题描述

我们可以将地图转换为平面图，每个地区变成一个节点，相邻地区用边连接，我们要为这个图形的顶点着色，并且两个顶点通过边连接时必须具有不同的颜色。附件是给出的地图数据，请针对三个地图数据尝试分别使用 5 个（le450_5a），15 个（le450_15b），25 个（le450_25a）颜色为地图着色。

3. 实验方法

根据地图区域数以及相邻区域对数（即图中的边数）得到关于数据的邻接表，利用树的深度优先遍历思想和回溯法，按照要求对区域进行涂色，求出满足填色要求的所有解，统计该过程所需的时间，并对效率进行分析。

三、实验步骤与过程

（一）解题方法

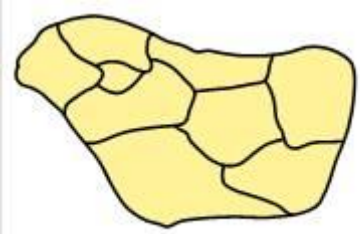
思路：该实验是对得到的图的顶点数、边数以及边信息对地图进行填色，且要求相邻区域填色不能相同。在解决该问题时，可以借助树的深度优先遍历，对每一块区域按树的高度从上往下逐层进行颜色选择，根据约束条件判断分枝是否满足情况，若满足，接着往下一层（即下一块区域）进行颜色选择，若不满足，利用回溯思想回到上一层，重新进行颜色选择，直到最后一层（即最后一块区域）完成涂色。

根据思路，进行编程，对主要模块伪代码展示如下：

```
//生成邻接表
int main()
{
    //从文件中获取边的起点和终点
    int v1 = getEdgeStart(file);
    int v2 = getEdgeEnd(file);
    //创建邻接表
    a[v1][0]++;//记录每个区域相邻区域数
    a[v1][a[v1][0]] = v2;
    a[v2][0]++;
    a[v2][a[v2][0]] = v1;
    node[v1].next_D++;//度
    node[v2].next_D++;
}
```

```
//深度+回溯
int DFS(Node *node, int v)
    if v==n;
        return node[v].colorEnableNum;
    else{
        int cnt = 0;
        for(i=1 to color_num)
        {
            if(paint_suit(v,i))//v可以涂色i
            {
                int temp = 0;
                paint(v,i);//给v涂色i
                deal_neighbor(v);//处理与v相邻区域的node信息
                temp = DFS(node,v+1);
                Recall;//数组回溯
                cnt += temp;
            }
        }
        return cnt;
    }
```

（二）对下面这个小规模数据，利用四色填色测试算法的正确性



由于代码中不能处理图信息，需要将所得小规模地图进行转换。

1. 对地图中的区域按从左到右、从上到下进行标号：

2. 根据上图给出转换后的顶点个数、边个数和边信息：

p edge 9 17

e 1 2　　e 1 5　　e 1 6　　e 2 3　　e 2 5　　e 2 6

e 3 4　　e 3 6　　e 3 7　　e 4 7　　e 4 9　　e 5 6

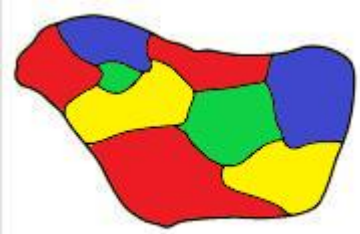
e 6 7　　e 6 8　　e 7 8　　e 7 9　　e 8 9

3. 运行代码，输入上述图信息，得到解（共 480 组，仅用其中一组进行验证）：

1 2 1 2 3 4 3 1 4（1~4 分别表示 4 种不同的颜色）

4. 正确性验证：

假设 1 表示红，2 表示蓝，3 表示绿，4 表示黄，按照上述解涂色结果如下：



可以看到以 4 种颜色对该小规模数据进行填涂，相邻区域颜色均不同，算法正确性得以验证。

（三）算法优化

对于（一）中提出的编程方法，在对小规模数据进行统计时，程序可以正常运行。但当数据增大到 450 个点之后，统计时间无法统计（达到 10 小时以上），故在对大规模数据进行填涂前需要先对基本回溯法进行优化。

1. 涂色选择搜索顺序(MRV)

(1)　思路：开始涂色前，将每一块区域的可涂颜色选择初始化为颜色数量（如小规模数据中，

颜色数量为 4），在对上一块区域进行涂色后，根据该区域与未着色区域是否相邻（即顶点间是否有边），更新未着色区域可涂颜色选择的数量，优先搜索未着色中可选颜色最少的点，把该点记为 A。若 A 的涂色需求无法满足，即 A 无法按约束条件进行着色时，进行剪枝。

区域编号	可涂颜色选择
1	4
2	4
3	4
4	4
5	4
6	4
7	4
8	4
9	4

区域1涂色

区域编号	可涂颜色选择
1	×
2	3
3	4
4	4
5	3
6	3
7	4
8	4
9	4

区域2涂色

区域编号	可涂颜色选择
1	×
2	×
3	3
4	4
5	2
6	2
7	4
8	4
9	4

区域5涂色

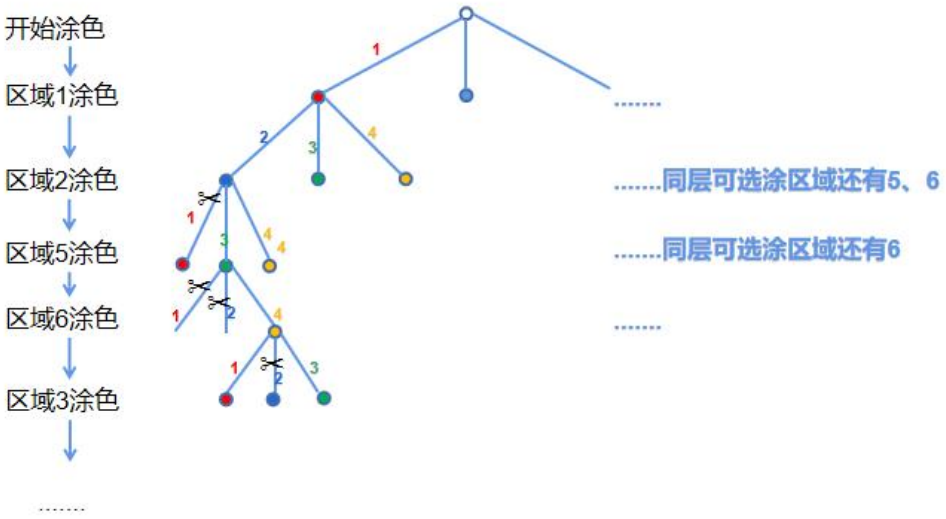
区域编号	可涂颜色选择
1	×
2	×
3	3
4	4
5	×
6	1
7	4
8	4
9	4

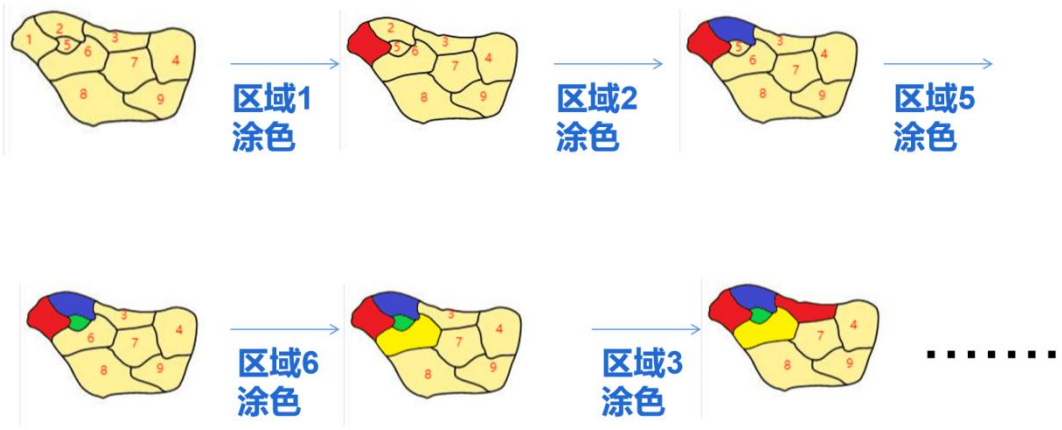
区域6涂色

区域编号	可涂颜色选择
1	×
2	×
3	2
4	4
5	×
6	×
7	3
8	3
9	4

区域3涂色

.....





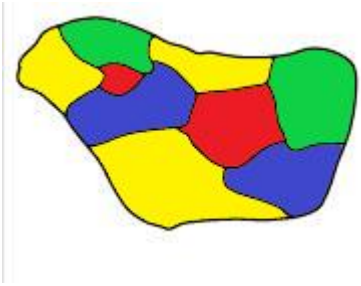
(2) 伪代码：根据上述思路，在普通回溯法的基础上，获取未着色区域当前可涂颜色选择的数量，选出选择最少的区域进行涂色。

```
int MRV(Node *node)
{
    int min → ∞;
    int index → 0;
    for(i = 1 to n)
    {
        if(node[i].color==0)
        {
            int min1 = node[i].colorEnableNum;
            if(min>min1)
            {
                min = min1;
                index =i;
            }
        }
    }
    return index;
}
```

(3) 利用小规模数据进行算法的正确性验证：

运行代码，得到小规模数据的总方案数为 **480** 种，其中一组解为：**4 3 4 3 1 2 1 4 2**

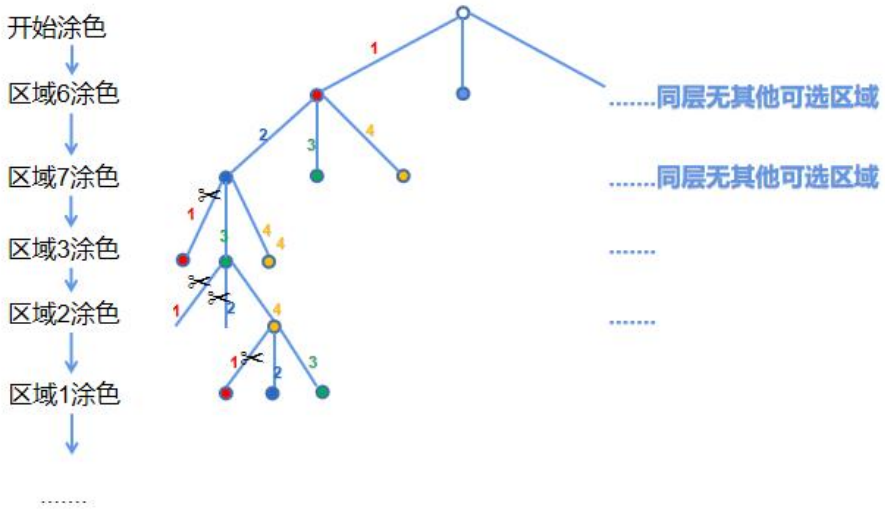
1 表示红，2 表示蓝，3 表示绿，4 表示黄，按照上述解涂色结果如下：

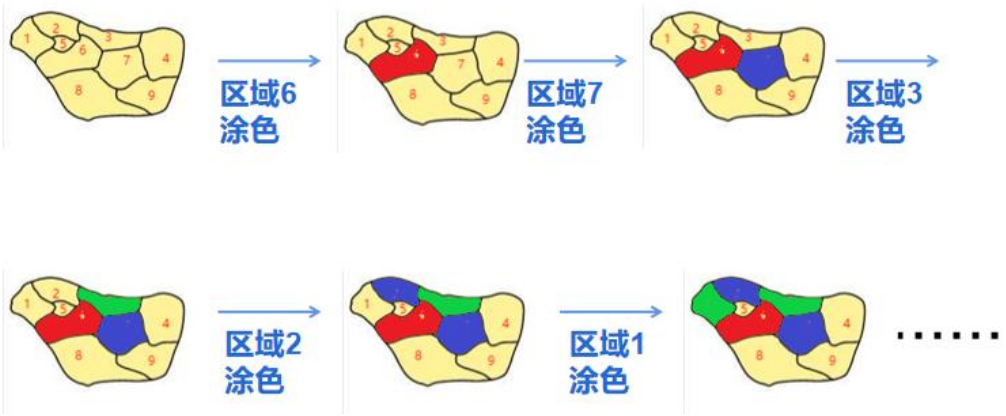


可以看到以 4 种颜色对该小规模数据进行填涂，相邻区域颜色均不同，算法正确性得以验证。

2．顶点搜索顺序(MRV+DH)

(1) 思路：通过区域的涂色选择按从小到大进行排序，优先选择可涂选择少的区域进行涂色，可以对整个回溯过程进行优化，缩短运行时间。处理过程中发现，当前可涂选择最少的区域可能有多个，这个时候进行进一步限制：在可涂选择最少的点中，选择受相邻区域影响最多（即顶点度最大）优先进行涂色。这样其实是对约束条件多的区域进行处理，优先对容易导致失败的变量进行赋值，可以更早的发现错误，以缩短运行时间。





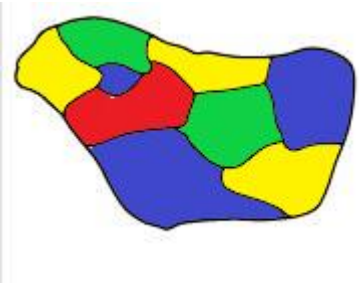
(2) 伪代码：根据上述思路，在涂色选择顺序的基础上，获取未着色区域可涂颜色最少中，度最大的区域进行涂色。

```
int MRVandDH(Node *node)
{
    int min → ∞;
    int index → 0;
    for(i = 1 to n)
    {
        if(node[i].color==0)
        {
            int min1 = node[i].colorEnableNum;
            if(min>min1)
            {
                min = min1;
                index =i;
            }
            else if(min==min1)
            {
                if(node[i].next_D>node[index].next_D)//next_D为结点的度
                    index = i;
            }
        }
    }
    return index;
}
```

(3) 利用小规模数据进行算法的正确性验证：

运行代码，得到小规模数据的总方案数为 **480** 种，其中一组解为：**4 3 4 2 2 1 3 2 4**

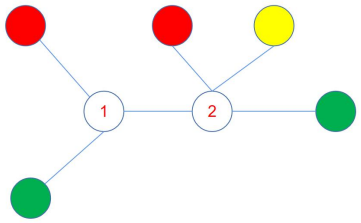
1 表示红，2 表示蓝，3 表示绿，4 表示黄，按照上述解涂色结果如下：



可以看到以 4 种颜色对该小规模数据进行填涂，相邻区域颜色均不同，算法正确性得以验证。

3. 向前探查搜索顺序(MRV+DH+FORWARD)

(1) 思路：向前探查是根据当前待涂色结点的下一个节点的可涂情况来判断当前结点涂色是否导致失败。举一个例子说明，如对 1 涂蓝色后，向前一步探查 2 无颜色可涂或者 2 仅剩蓝色可涂而导致冲突，出现上述两种情况即可提前预知失败，提前停止回溯，以减少运行时间。



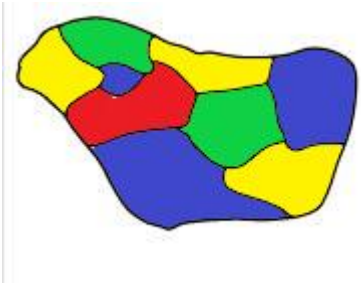
(2) 真代码：根据上述思路，在涂色选择顺序的基础上，获取未着色区域可涂颜色最少中度最大的区域进行涂色。

```
//v表示当前涂色点，j表示向前探查的点，s为当前颜色
bool FORWARD(Node *node, int s, int j, int v)
{
    node[j].colorEnable[s] = -v;
    node[j].colorEnableNum--;
    node[j].next_D--;
    if(node[j].colorEnableNum <= 0)
        return false;
    return true;
}
```

(3) 利用小规模数据进行算法的正确性验证：

运行代码，得到小规模数据的总方案数为 **480** 种，其中一组解为：**4 3 4 2 2 1 3 2 4**

1 表示红，2 表示蓝，3 表示绿，4 表示黄，按照上述解涂色结果如下：



可以看到以 4 种颜色对该小规模数据进行填涂，相邻区域颜色均不同，算法正确性得以验证。

（四）对附件中给定的地图数据填涂

使用最终优化方案对给定地图数据进行填涂，得到运行时间统计如下：

数据	le450_5a	le450_15a(提前结束)	le450_25a(提前结束)
运行时间(s)	18.59	11.06	3.882
总方案数	3840	?	?

分析：对于数据 le450_5a，一开始运行时间超过 10 小时，通过优化后，运行时间只需 18.59s，便能将所有的解求出。对于 15 色和 25 色的数据，由于解的数量十分大，在统计运行时间时，设置判断条件--若解的数量已经超过一亿个，则提前结束。根据结果可以看到 15 色得到超过一亿组解的运行时间高于 25 色的运行时间，这是因为颜色选择多时，找到一组合法解更容易，也就是能够在更短的时间内得到可以正确涂色的方案，所以找到数量级相同的解，颜色数量为 25 的会更快。

（五）随机产生不同规模的图，分析算法效率与图规模的关系（四色）

随机生成边信息：

在随机生成数据时，需要注意几个情况。一是生成的起点和终点不能为 0 且不能超出点数范围；二是生成的起点和终点不能重复；三是生成的边不能重复，即新生成的边的起点终点不能与前面已经生成的边的起点终点重复。根据上述三个限制条件，给出编程方法如下：

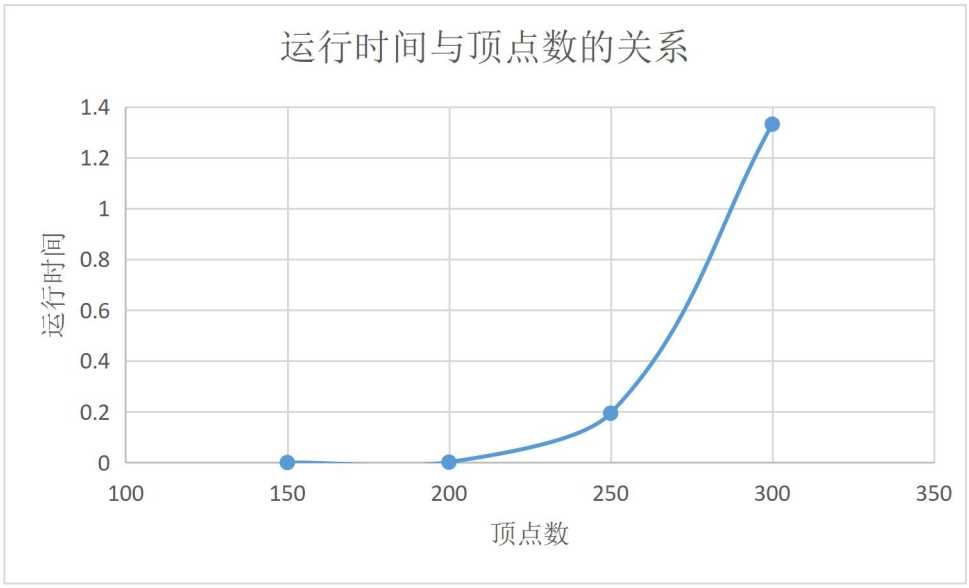
```
void CreateRandom()
{
    int e[n + 1][n + 1];
    int v1,v2;//边的起点和终点
    int sum = 1;//统计当前的边数
    memset(e, 0, sizeof(e));
    srand((unsigned int) time(0));
    while (sum <= edge)
    {
        while (1)
        {
            v1 = rand() % n + 1;
            v2 = rand() % n + 1;
            if (v1 != v2 && e[v1][v2] != 1)
                break;//起点终点不相等且边不重复，成功生成
        }
        e[v1][v2] = e[v2][v1] = 1;//表示以v1v2为端点的边已生成
        CreateList(v1,v2);//创建邻接表
        sum += 1;//继续生成新的边
    }
}
```

按不同顶点数和边数的要求，随机生成不同的图，对运行时间进行统计：

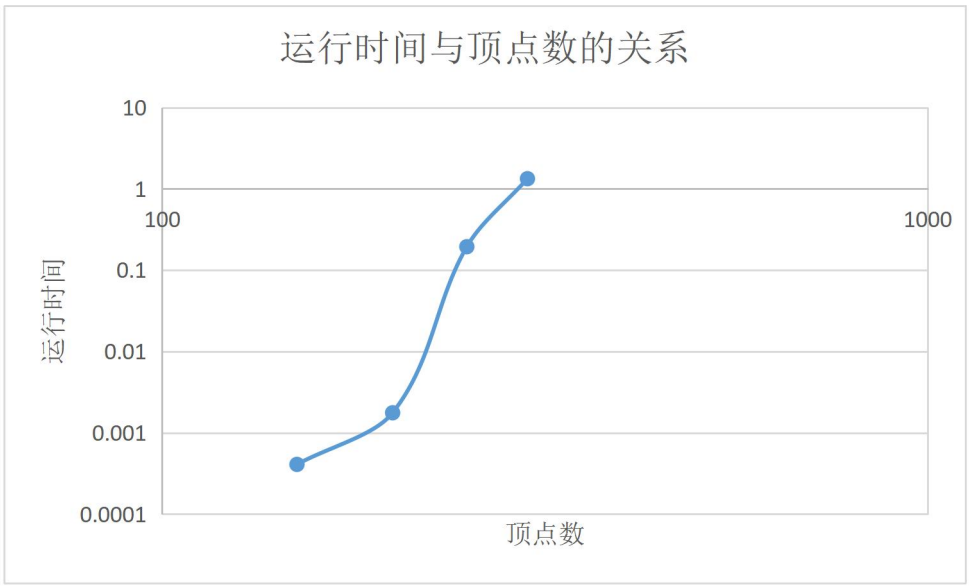
1. 固定边数，点数逐渐增大

固定边数2000(最少有64个点)

顶点数	150	200	250	300
运行时间(s)	0.000408	0.00176	0.194	1.33105



取对数刻度表示后：

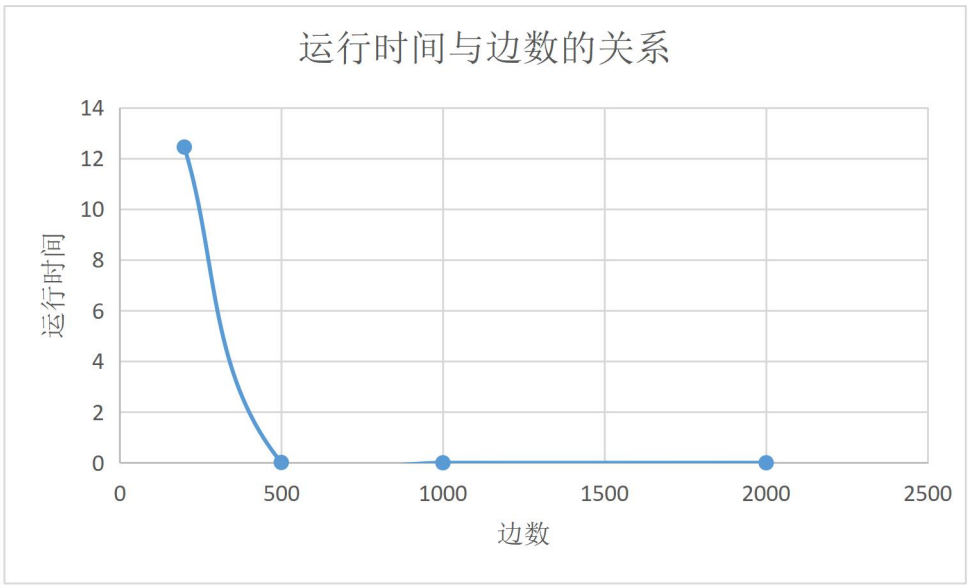


可以看到固定边数时，随着顶点数的增多，运行时间也逐渐增大。这是因为边数固定，点数增多，图的边密度也随之减小，对于填涂的约束条件少，剪枝效率低，用四色对地图进行填涂得到的解也更多，因此总体运行时间会增大。

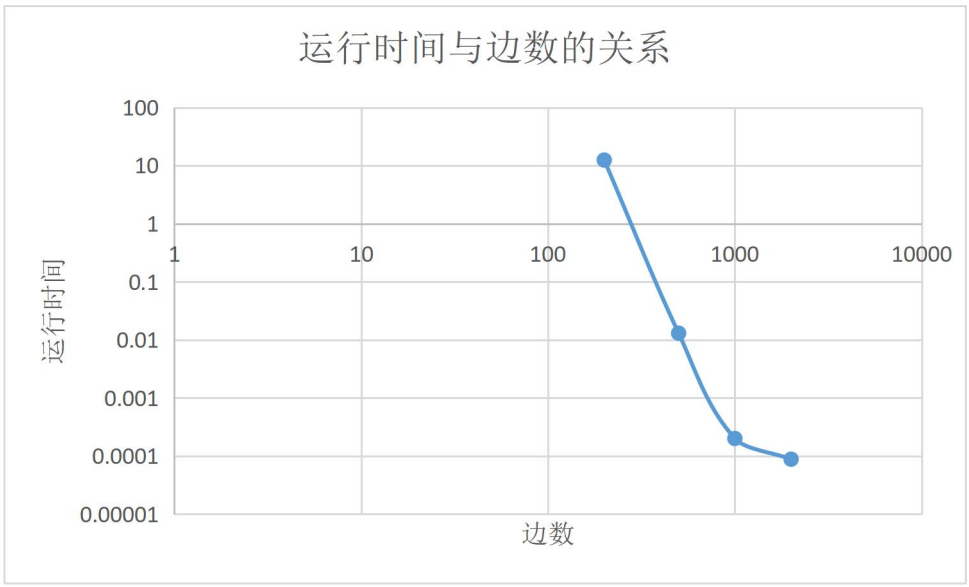
2. 固定点数，边数逐渐增大

固定点数100(最多可以有4950条边)

边数	200	500	1000	2000
运行时间(s)	12.44	0.013	0.0002	0.000088



取对数刻度表示后：

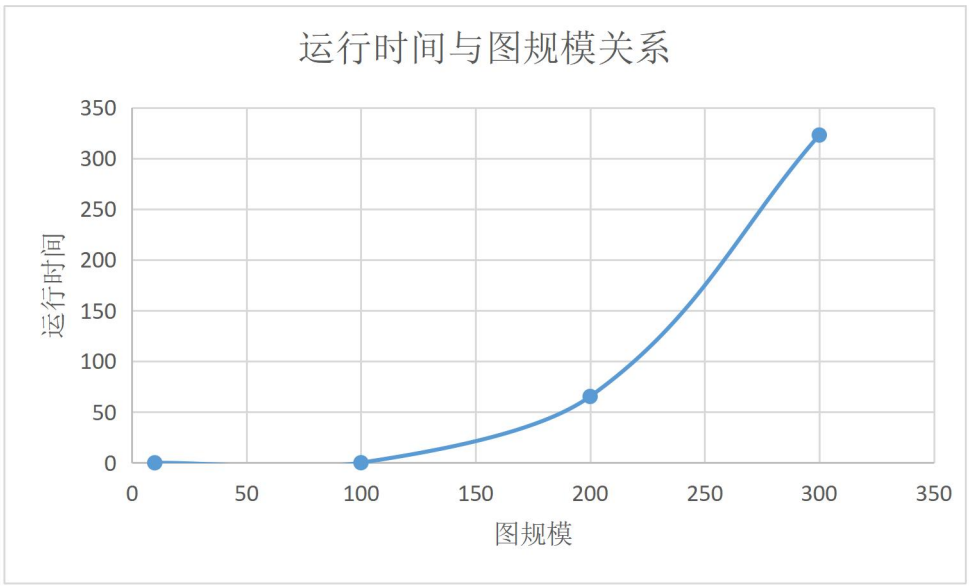


可以看到固定点数时，运行时间对着边数的增大而减少。这是因为图的边密度增大，图的合法解也就更少，通过对回溯过程进行剪枝，减少了许多错误情况，剪枝效率高，解的数目少，总体求解的运行时间也就更快。

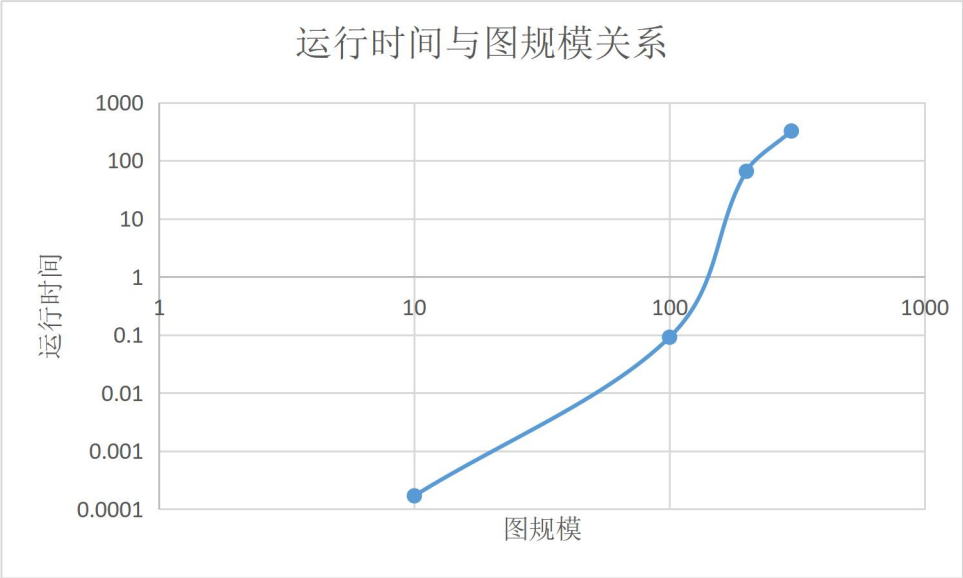
3. 图规模增大

边规模为点规模的5倍

规模	10	100	200	300
运行时间(s)	0.000169	0.0909	65.352	322.853



取对数刻度表示后：



可以看到随着图规模的增大，运行时间也就更长。边与顶点的关系固定为两倍关系，图的规模增大，需要填涂的区域也就更多，求解过程中需要判断的情况增多，运行时间也就更长。

四、实验结论或体会

1. 实验结论：

在进行地图填色时，点数、边数、颜色选择数，对于实际解决问题都有影响。

边数固定，点数增大时，由于需要进行填涂的区域数变多，约束条件少，剪枝效率低，同时合法解也更多，运行效率也就更慢；

点数固定，边数增加时，由于地图边密度的增大，相邻区域多，对填涂颜色的约束也就更大，对于优化后的算法而言，这样的约束可以使剪枝效率增大，运行效率更快；

颜色选择数多，填色时更容易得到正确解，得到一组解的速度更快。

2. 实验体会：

通过这次实验，对使用回溯法有了更深的认识以及学会如何借助此方法解决一些实际问题。对于解决问题时需要对每一块区域（或是个体）逐一突破时，可以借助回溯法进行解决。在实际运用过程中可以根据约束条件对算法进行优化，也就是剪枝，学会根据实验实际运行时间不断优化实验的算法，得以更高效的解决问题。

指导教师批阅意见： 成绩评定： 指导教师签字： 年　　月　　日
备注：

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。

2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。