# 浅谈几何学与计算机的交叉应用

年级：1790级
专业：魔法V班
学号：17771026
姓名：Elaina
时间：1790年06月

# 摘要

就计算机在几何学领域的应用,本文介绍了**四色定理**(**Four Color Theorem**)及其历史与机器证明,以及**四色问题**的求解.就几何学在计算机领域的应用,本文介绍了三维**凸包**(**convex hull**)表面积问题的求解.

# 关键词

四色问题;凸包

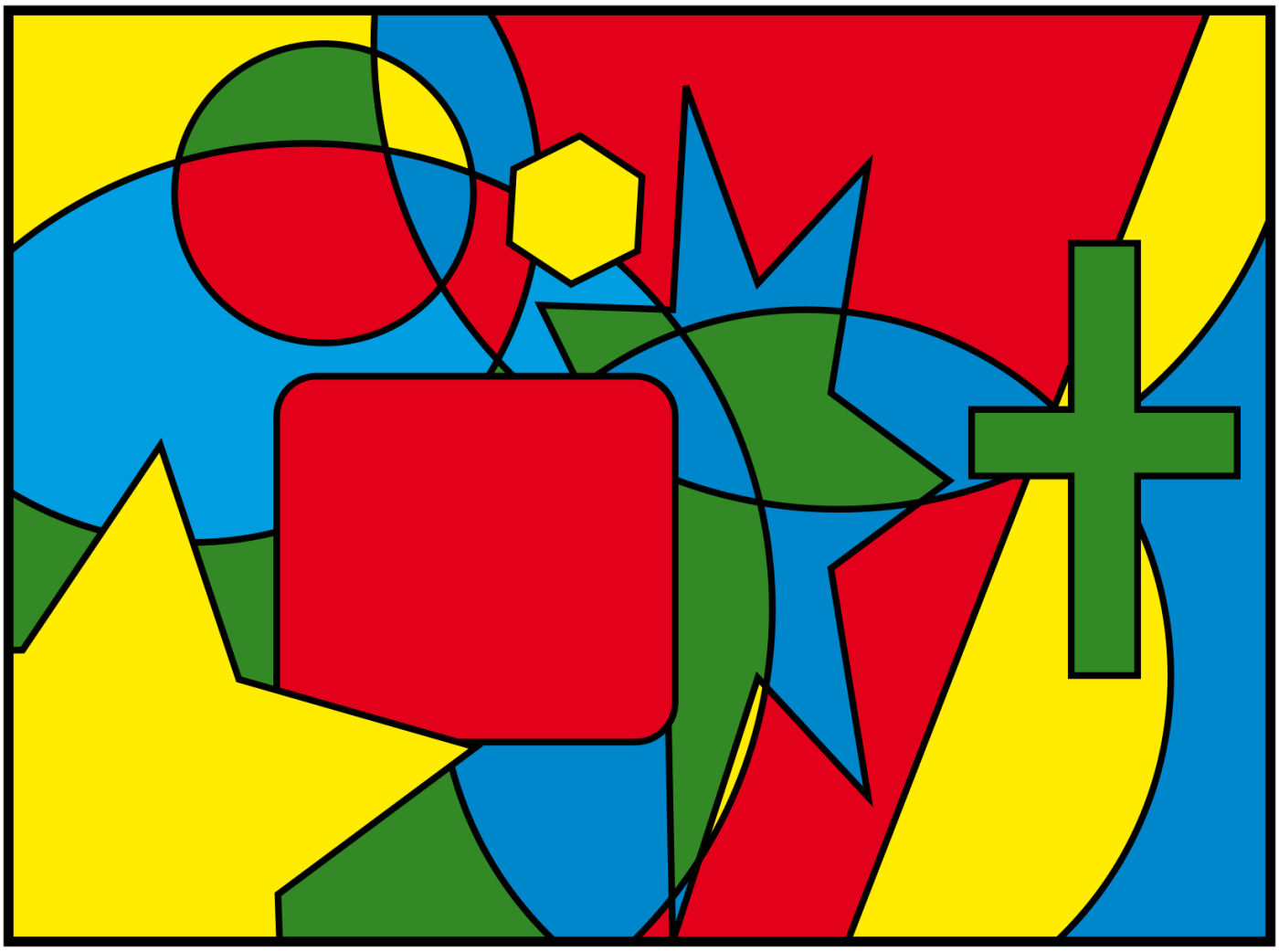
# 正文

国家的发展离不开日益密切的国际经济文化交流,学术也是一样.如今,**学科交叉**一词时常出现在我们眼前.在进行数学科学的研究时,市场需要计算机辅助；在进行计算机科学的研究时,常常需要转化称数学问题进行求解.如**四色定理**,开普勒猜想,NP-硬度的最小权重的三角测量,布尔毕达哥拉斯三元组问题等著名数学难题,均借助计算机得以证明；如**凸包**求解,**时间复杂度**(**time complexity**)计算,**空间复杂度**(**space complexity**)计算等问题.

几何学是数学的一大分支,本文将就**四色定理**和**凸包**求解问题,浅谈几何与计算机的交叉应用.

## 四色定理

**四色定理**,又称**四色猜想**,是**世界三大数学猜想**之一.



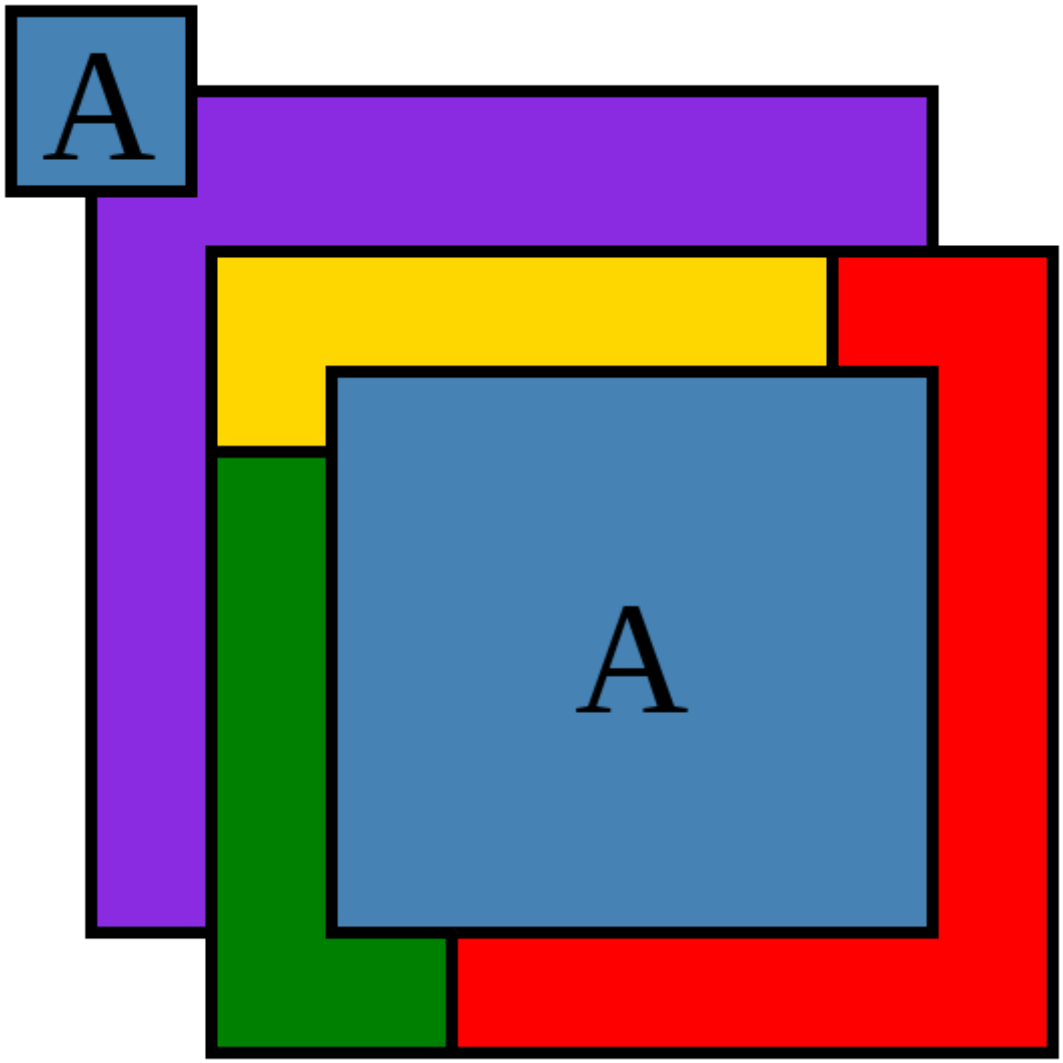
四色地图的一个例子

## 定理内容

如果在平面上划出一些邻接的有限区域,那么可以用四种颜色来给这些区域染色,使得每两个邻接区域染的颜色都不一样.

### 另一种通俗的阐述

任意一个无**外飞地**(**exclave**)的地图都可以用四种颜色染色,使得没有两个相邻国家染的颜色相同.
其中,**外飞地**为人文地理中的**外飞地**概念.若某国家 $A$ 拥有一块与本国分离的领土 $a$ , $a$ 被其他国家包围,则 $a$ 称为 $A$ 的**外飞地**.




**外飞地**的一个例子

### 拓扑学阐述

设有一平面或其一部分,将其划分为互不重叠的区域的集合.
定义**地图**:一个**地图**为以下方式的划分:
将平面划分为有限个区域,使得任意两个区域的交集是空集,所有的区域的并集是整个平面;所有区域中,只有一个区域是**无界区域**,其余区域都是**有界区域**.
其中,**有界区域**指能够用一个长和宽都有限的矩形覆盖的区域;**无界区域**则是指不能用这样的矩形覆盖的区域.
每个区域相当于通俗说法中的**国家**.
区域之间的边界相当于通俗说法中的**国界线**,定义为连续不自交的曲线,也称为**连续简单曲线**.
**连续简单曲线**是指一个连续函数

$$c:[0,1]\to\mathbb{R}^2$$

的像集

$$C=\{c(t)|t\in[0,1]\},$$

且满足

$$c(t_1)\neq c(t_2),\forall 0\leqslant t_1<t_2\leqslant 1,(t_1,t_2)\neq(0,1),$$

这样说明曲线不与自身相交.
若 $c(0)\neq c(1)$, 则称曲线为**弧**,否则称为**圈**.
平面$\mathbb{R}^2$中的一个**地图**是指一个有限个**连续简单曲线**的集族

$$L=\bigcup_{i=1}^{m}\{C_i\},m\in\mathbb{N}\cap[2,+\infty).$$

其中对$\forall i\neq j\in\mathbb{N}\cap[1,m]$,

$$C_i=\{c_i(t)|t\in[0,1]\}$$

是连续函数

$$c_i:[0,1]\to\mathbb{R}^2$$

的像集,且满足

$$c_i(t_1)\neq c_i(t_2),\forall 0\leqslant t_1<t_2\leqslant 1,(t_1,t_2)\neq(0,1),$$

且

$$|C_i\cap C_j|\in\{0,1\}.$$

定义**边**: $L$ 中每一个**连续简单曲线**.
定义**顶点**:任意**边** $C$ 的端点 $c(0),c(1)$ 称**顶点**.

定义**中性点**: 所有属于**边**或**顶点**的点.其集合

$$N_L = \{x | x \in C_i, i \in \mathbb{N} \cap [1, m]\}.$$

定义**国家**: $L$ 将非**中性点**划分成的若干个道路连通的开集.其集族 $A_L$.
可见,每个**国家**是 $\mathbb{R}^2 - N_L$ 的一个极大连通子集,是没有**外飞地**的;取一个非中性点 $x$, 所有能够从 $x$ 经过一条不含**中性点**的**弧**到达的点构成的集合,就是一个**国家**.
定义**n-染色**:**地图** $L$ 的一个**n-染色**方案指一个函数

$$f : A_L \to \mathbb{N} \cap [1, n].$$

若

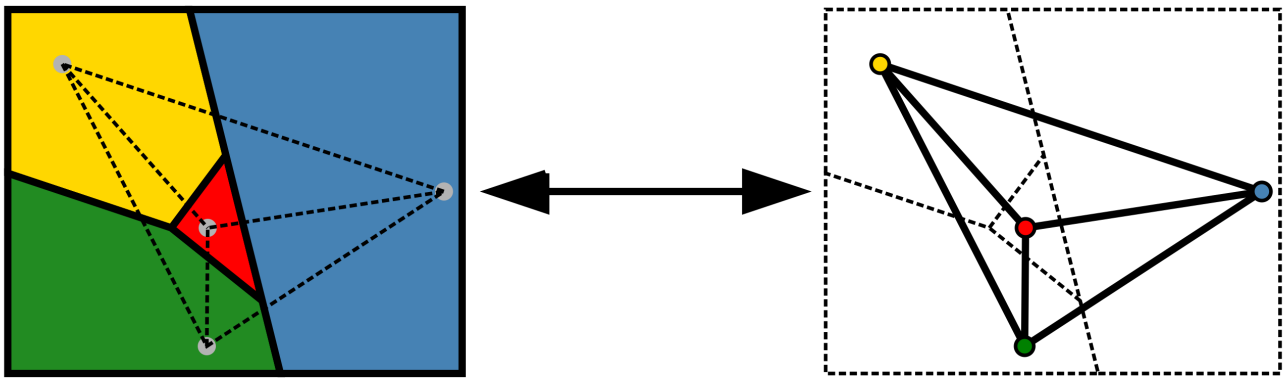
$$f(A_i) \neq f(A_j), \forall A_i, A_j \in A_L, |\overline{A_i} \cap \overline{A_j}| \in \{0, 1\},$$

则称它是**可行的**.
**四色定理**:每个**地图**都存在**可行的4-染色方案**,即

$$\forall L, \exists f : A_L \to \{1, 2, 3, 4\}, s.t., f(A_i) \neq f(A_j), \forall A_i, A_j \in A_L, |\overline{A_i} \cap \overline{A_j}| \in \{0, 1\}.$$

**图论阐述**

将上述**地图**转化为图论中的一个**无向平面图**,即**地图**中的每一个国家用其内部的一个点代表,作为一个顶点.如果两个国家相邻,就在两个顶点之间连一条线.



地图与无向平面图互相转化的一个例子

**四色定理**:必然可以用四种颜色给**无向平面图**的顶点染色,使得相连的顶点颜色不同.

## 历史

1852年,古德里(Francis Guthrie)提出猜想:"只需要四种颜色为地图着色".
1879年,肯普(Alfred Kempe)宣称证明了**四色定理**.11年后,肯普的证明被希伍德(P. J. Heawood)推翻.他举

出了反例,并提出了**五色定理**.但近代,该反例又被董德周推翻.
1913年,伯克霍夫(George David Birkhoff)证明了由不超过 $12$ 个国家构成的地图都能用四色染色.9年后,他的学生富兰克林(Philip Franklin)证明了由不超过 $25$ 个国家构成的地图都能用四色染色.
1976年,阿佩尔(K. Appe)和哈肯(W. Haken)使用计算机辅助证明了**四色定理** ,但是他们的证明在当时并未获得普遍认可.主要原因为使用计算机的部分无法用手算核实,应该用手算核实的部 分因过于繁琐,无人能够独立完成.
1994年,西缪尔(P. Seymour)等人对阿佩尔和哈肯的证明进行了修正和改进,同时证明了其正确性.
2004年,龚提尔(Georges Gonthier)使用证明验证程序**Coq**来对当时交由计算机运算的算法程序进行了形式上的可靠性验证.验证表明,**四色定理**的机器验证程序确实有效地验证所有构形的可约性,完成了证明中的要求.

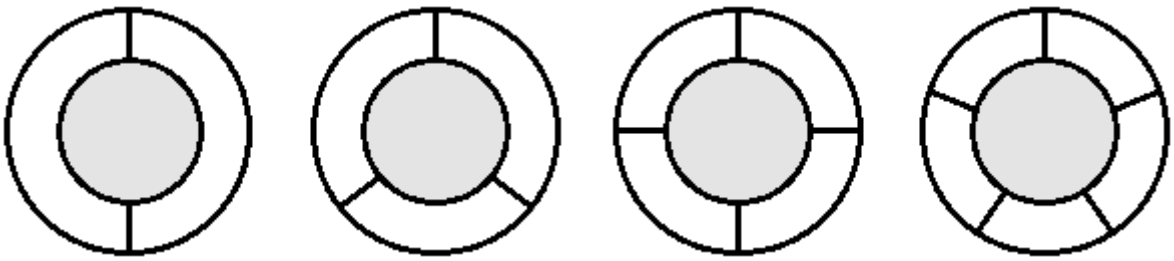
## 计算机辅助证明

### 可约构形

考虑 $n+1$ 个国家中邻国数目最小的国家 $A$ , $A$ 的邻国的数目不超过 $5$. 若 $A$ 的邻国数目不超过 $3$, 那么可以把 $A$ **去掉**(如和其中一个邻国连成一体),形成一个 $n$ 个国家的地图,这个地图可以用四种颜色着色,而原来的 $3$ 个邻国至多用了三种颜色.这时候将 $A$ 还原,染上第四种颜色,就等于找到给用四种颜色原地图着色的方法.
这种能够**去掉**一个国家,减少国家数的局部称**可约构形**(**reducible configuration**).

### 不可避免的可约构形集

伯克霍夫等人的证明是肯普的方法的延续和系统化,归纳为寻找一个**不可避免的可约构形集**(**an unavoidable set of reducible configurations**,简称**不可避免集**).



肯普使用的不可避免集

### 放电法

将无向平面图其看作是平面的**电网**,并将每个点按照**度数**(**degree**,连出的边数)分类.首先在每个**度数**为 $k$ 的节点放置 $6-k$ 的**电荷**.
**放电**过程(**discharging procedure**)指将这些**电荷**以特殊的规则进行重新分配,从而找出**电网**结构上的特性,创建**不可避免集**.具体来说,可以假设某些构形全不存在,然后构造一个**放电**过程,使得接受**电荷**的总量不再等于释放**电荷**的总量,从而导出矛盾.每个良好设计的**放电**过程都能证明一个**不可避免集**的存在.

### 阿佩尔和哈肯的证明

二人通过纸笔运算,得到了一个由 $1936$ 个构形构成的**不可避免集**,对应的**放电**规则由 $487$ 条规则构成.经过电脑 $1200h$ 的验证,得出这 $1936$ 个构形均为**可约构形**,**四色定理**得证.

### 西缪尔等人的证明

经过修正和改进,**不可避免集**大小减至 $633$ ,**放电**法则减至 $20$ 个,图着色算法由 $4$ 次时间算法优化为 $2$ 次时间算法,证明所需机时由 $1200h$ 缩短到 $24h$ .此外,第二代证明达到了人工复核的要求,如果用计算机验证只需 $5min$ 就能完成,且没有第一代证明中只有计算机才能完成的验证过程.

### 机器证明的可靠性问题

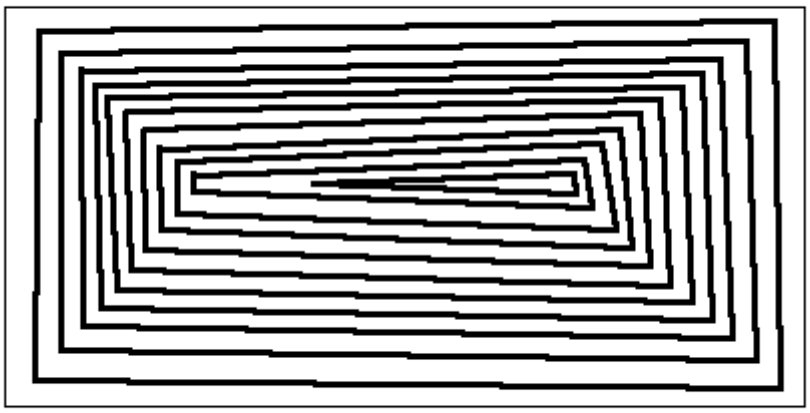
2004年,龚提尔(Georges Gonthier)使用证明验证程序**Coq**来对当时交由计算机运算的算法程序进行了形式上的可靠性验证.证明验证程序是一个由法国开发的软件,能够从逻辑上验证一段电脑程序是否正常运行,并且是否达到了它应该达到的逻辑目的.验证表明,**四色定理**的机器验证程序确实有效地验证所有构形的可约性,完成了证明中的要求.至此,**四色定理**的理论部分和计算机证明算法部分都得到了验证.

## 四色问题

称求得一个给定的无**外飞地**的**地图**的一种**可行的4-染色方案**的问题为**四色问题**.
**四色定理**已证,意味着**四色问题**总是有解的,在求解**四色问题**时,不需要考虑答案存在性问题.
下面给出一种使用计算机求解**四色问题**的方法.

### 输入

输入数据为一张对比度较高的图片.

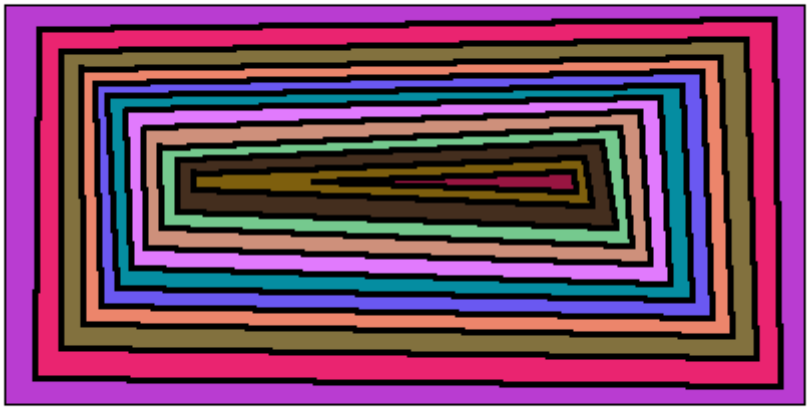


输入数据的一个例子

将输入空地图转化为 `bool` 型矩阵储存,矩阵内每一个变量数值与像素点的颜色有关.

**国家处理**

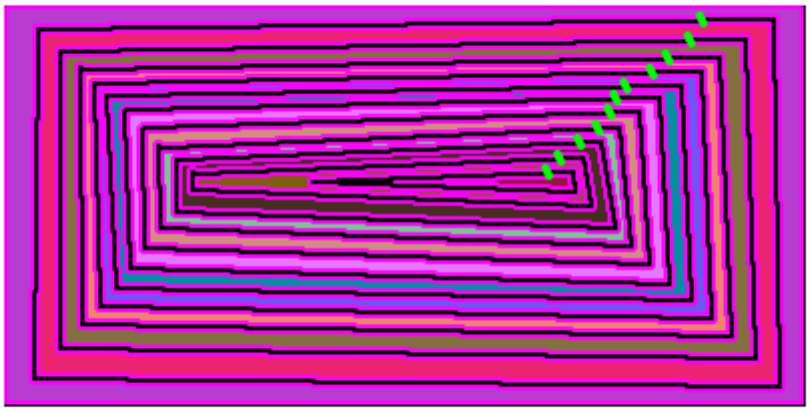
使用**深度优先搜索**(**BFS**)标记国家.该标记过程可视为将每个**国家**染上不同颜色.



上例国家处理后

**邻国处理**

若来自不同**国家**的两个像素(矩阵内变量)**足够近**,那么认为这两个**国家**是邻国.
已知每个**国家**必然有邻国;每条边的可视为多个矩形相连;同时不可能出现违反**四色定理**的地图.通过上述规则的约束,通过**二分法**确定认为**足够近**的距离数值.



上例邻国处理后(绿点为两个国家最近的像素处)

**颜色填充**

填充颜色有多种算法,这里使用**威尔士-鲍威尔算法**(**Welsh-Powell Algorithm**,下文简称**WP**)和基于**广度优先搜索**(**DFS**)的**回溯填色算法**(下文简称**D**)两种即可.
定义**最优解**:**四色问题**的一个**最优解**是一个存在**可行的n-染色方案**但不存在**可行的(n-1)-染色方案**的**地图**的一个**可行的n-染色方案**.
定义**简单图**:可用**WP**求解**四色问题**的**地图**.大部分**地图**都是**简单图**.
**WP**较快,但是在面对非**简单图**时不能成功填色;同时,面对一些存在**可行的n-染色方案**( $1 \leqslant n \leqslant 3$ )的地图时,仍然需要四种颜色,即所得解不是**最优解**.
**D**较慢,但是适用于任何**地图**,且得到的是**最优解**.
填色时,可以选择两种方案:

方案一:先使用**WP**,若失败,则再使用**D**.

方案二:直接使用**D**.

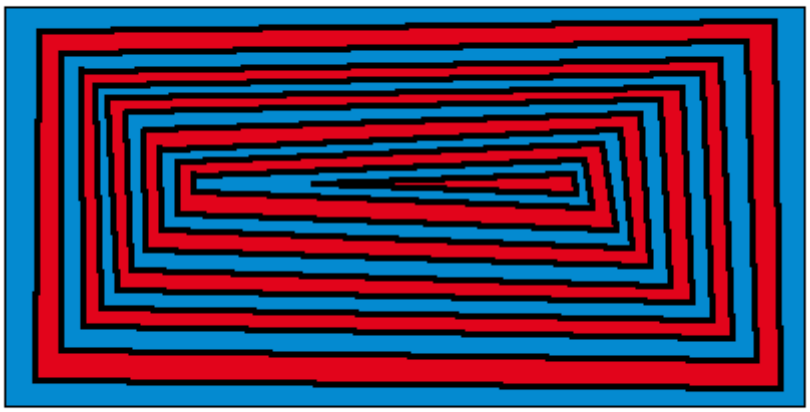
**WP**和**D**的**时间复杂度** $O(WP), O(D)$ 差异是以数量级体现的,设 $n = \dfrac{O(D)}{O(WP)}$ , $n$ 与**地图**的复杂程度正相关.再假设**简单图**占比为 $q > 50\%$.
利用假设的数据,通过计算,发现方案一二的**时间复杂度**之比为

$$F = \frac{1 * q + (1 + n)(1 - q)}{n} = 1 - q + \frac{1}{n}.$$

当**地图**趋于复杂时

$$F_0 = \lim_{n \to \infty} 1 - q + \frac{1}{n} = 1 - q < 0.5.$$

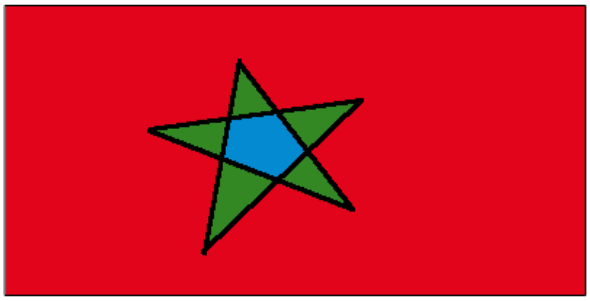
故选择方案一较快.

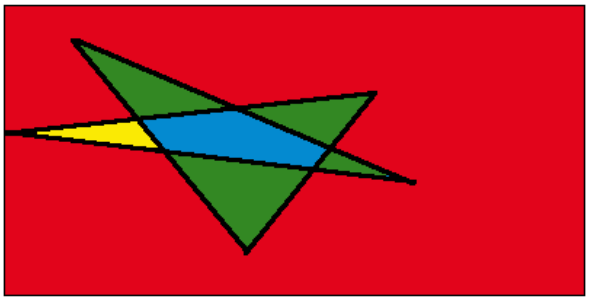


上例染色完成

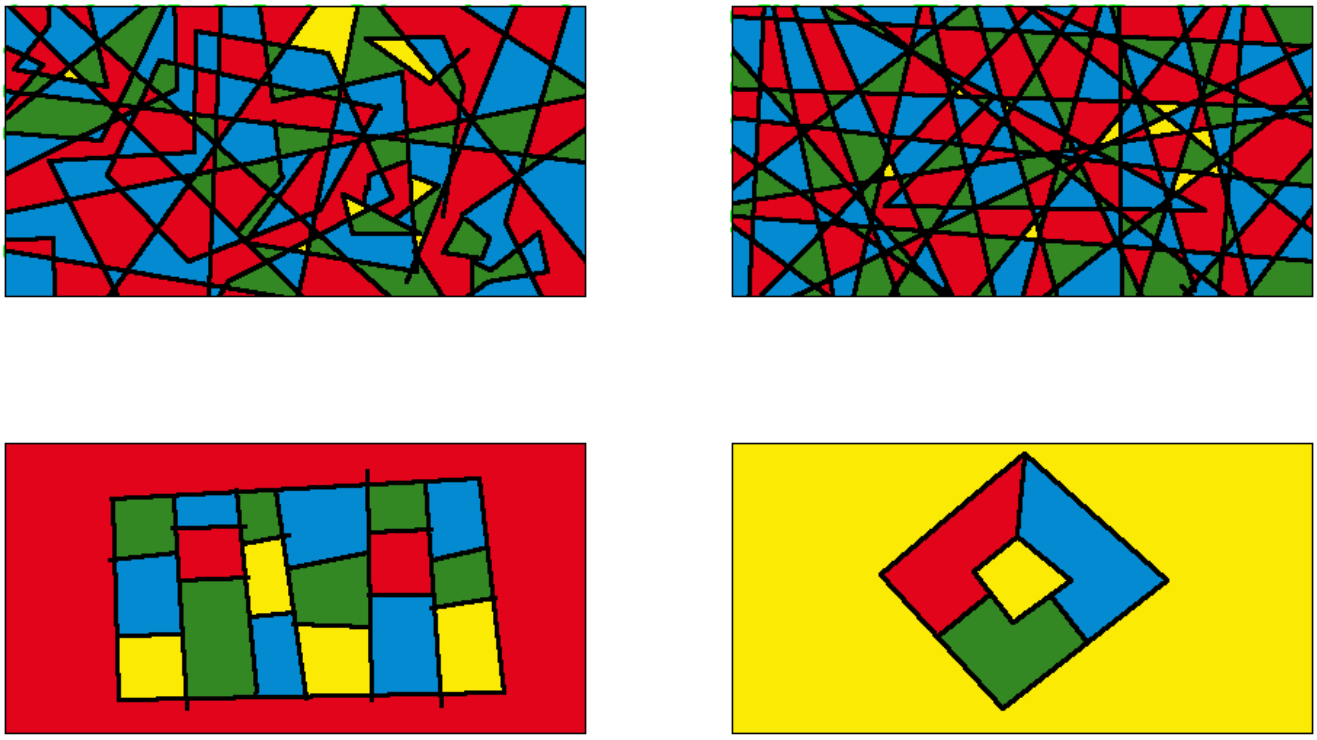
```
Loading pixels...
Analyzing areas & finding nodes...
Found a total of 12 areas/nodes.
Analyzing marginal points & finding edges...
Found marginal points for all areas.
Found a total of 11 edges.
Building & solving graph, stand by...
Calculating valence...
Node 1 has highest valence: 2
Coloring successful with Welsh-Powell algorithm!
Finished.
```

上例求解时所得中间结果日志





拓扑同构的图形在WP下均未得到**最优解**且颜色数目不同



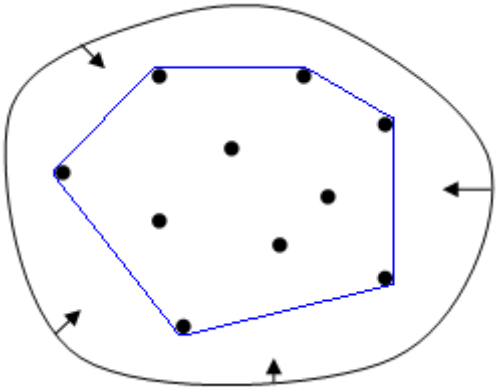
其它**地图**染色完成示例

# 三维凸包

在点集拓扑学与$\mathbb{R}^J$中,**凸集**(**convex set**)是一个点集合,其中每两点之间的直线点都落在该点集合中.
在一个实数向量空间 $V$ 中,对于给定集合 $X$ ,所有包含 $X$ 的**凸集**的交集 $S = \bigcap_{X \subseteq K \subseteq V} K$ 被称为 $X$ 的**凸包**.
可以用 $X$ 内所有点 $(x_1, x_2, \cdots, x_n)$ 的线性组合来构造 $X$ 的**凸包**

$$S = \{\sum_{i=1}^{n} t_j x_j | x_j \in X, \sum_{i=1}^{n} t_j = 1, t_j \in [0,1]\}.$$

在$\mathbb{R}^2$中,**凸包**可想象为一条刚好包着所有点的橡皮圈.



蓝色即为这10个点所得**凸包**

## 联系

在算法竞赛中,经常要编程求二维**凸包**,偶尔会出现三维**凸包**.
数学上求**凸包**时,可用增量法,Jarvis步进法,Graham扫描法,单调链法,分治法,Akl-Toussaint启发式快包法.
算法竞赛中求**凸包**时,常用增量法,Jarvis步进法,Graham扫描法.
而这里给出增量法求$\mathbb{R}^3$中一些点的三维**凸包**的表面积过程.

## 算法设计

初始,先选三个不共线的点,组成一个面.其实应该是正反两面的,即将正反两面均加入**凸包**的面集.逐次将点加入,然后检查之前的点是否在新**凸包**上.
当加入点 $P_r$ 时,想象从该点**看**向旧**凸包** $CH(P_{r-1})$ ,删除 $CH(P_{r-1})$ 中 $P_r$ 可以**看**到的面,然后将所有 $P_r$ 能看到的点与 $P_r$ 连线,其中连续两点和 $P_r$ 构成三角形,得到新**凸包** $CH(P_r)$ .

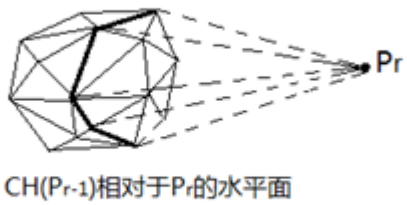



图1

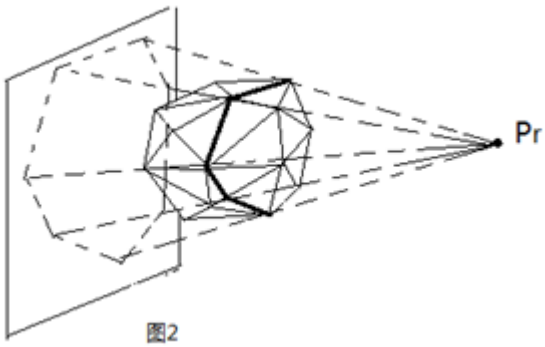



图2

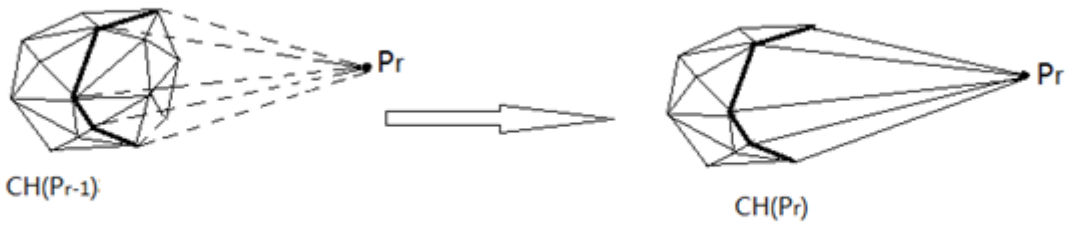



图3

加入点同时更新**凸包**的过程

由于每次都要检查所有之前的点,时间复杂度为 $O(n^2)$.
为进一步优化算法,笔者在误差允许范围内将各点的位置震荡,以达到使所得**凸包**的各面均为三角形的目的.
在求三维**凸包**的过程中,出现了两种向量:一种是由一个点指向另一个点的向量,另一种是由两个已知求叉积所得向量.
其中,第一种向量需要额外记录其始终两点.虽然终点和可由其始点和向量方向和长度计算得来,但算法竞赛中通常牺牲空间来节省时间,故均需开辟空间记录.
得到两个向量类:

```
class Vector1{
public:
        double x,y,z;
    double l;
        double len(){return (l=sqrt(x*x+y*y+z*z));}
    double lx,ly,lz,rx,ry,rz;
};
class Vector2{
public:
        double x,y,z;
    double l;
        double len(){return (l=sqrt(x*x+y*y+z*z));}
};
```

其中 `len()` 函数的作用是计算向量长度.
由于算法设计,笔者将这两类向量以及给出的点均用同一个类 `P` 存储:

```
class P{
public:
        double x,y,z;
        void ck(){x+=rv;y+=rv;z+=rv;}
        void pin(){scanf("%lf%lf%lf",&x,&y,&z);ck();}
        void ot(){printf("%.5lf %.5lf %.5lf\n",x,y,z);}
        double len(){return sqrt(x*x+y*y+z*z);}
        P operator+(P r){return (P){x+r.x,y+r.y,z+r.z};}
        P operator-(P r){return (P){x-r.x,y-r.y,z-r.z};}
        P operator/(P r){return (P){y*r.z-z*r.y,z*r.x-x*r.z,x*r.y-y*r.x};}
        double operator*(P r){return x*r.x+y*r.y+z*r.z;}
};
```

其中, `ck()` 为震荡函数,重载 `+` 和重载 `-` 为向量加和向量减,重载 `/` 为叉积,重载 `*` 为点积.
而对**凸包**的面,笔者用类 `S` 存储:

```
class S{
public:
        int v[3];
        void ot(){printf("%d %d %d\n",v[0],v[1],v[2]);}
        P sp(){return (p[v[1]]-p[v[0]])/(p[v[2]]-p[v[0]]);}
        double sa(){return sp().len()/2.0;}
        int ck(P pp){return ((pp-p[v[0]])*sp()>0);}
};
```

我们知道,三角形面积等于其任意两边的叉积长度的一半,即其中函数 `sa()` .
计算**凸包**所有面的面积之和,即为其表面积.
设该**凸包**面集大小为 $F$ ,由于该**凸包**所有面均为三角形,那么其边集大小

$$E = \frac{3}{2}F.$$

由**欧拉公式**

$$F + V - E = 2$$

得,其顶点集大小

$$V = 2 + E - F = \frac{1}{2}F + 2.$$

## 代码实现

```
//Code By 1677
#include<bits/stdc++.h>
#define LMX 9223372036854775807
#define IMX 2147483647
#define M0 1000000007
#define M9 998244353
#define M8 19260817
#define M7 7221457
#define PI 3.1415926
#define E 2.71828
#define LL long long
#define doublw double
#define ini int
#define itn int
#define calss class
#define pt putchar
#define gt getchar
#define lowbit(a) (a&(~a+1))
#define cmm(a,b) if(a^b){a^=b;b^=a;a^=b;}
#define lcm(a,b) (a/gcd(a,b)*b)
#define gcd(a,b) exgcd(a>b?a:b,a>b?b:a)
#define mn(a,b) (a<b?a:b)
#define mx(a,b) (a>b?a:b)
#define ab(a) (a>=0?a:(~a+1))
#define ED(a) (~scanf("%d",&a))
#define I(b,a,c) (a>=b&&a<=c)
inline LL exgcd(LL a,LL b){if(!b)return a;return exgcd(b,a%b);}
#define MXN 2333
#define var (1e-9)
#define rnd ((double)rand()/(double)RAND_MAX)
#define rv ((rnd-0.5)*var)
int v[MXN][MXN];
double ans;
class P{
public:
        double x,y,z;
        void ck(){x+=rv;y+=rv;z+=rv;}
        void pin(){scanf("%lf%lf%lf",&x,&y,&z);ck();}
        void ot(){printf("%.5lf %.5lf %.5lf\n",x,y,z);}
        double len(){return sqrt(x*x+y*y+z*z);}
        P operator+(P r){return (P){x+r.x,y+r.y,z+r.z};}
        P operator-(P r){return (P){x-r.x,y-r.y,z-r.z};}
        P operator/(P r){return (P){y*r.z-z*r.y,z*r.x-x*r.z,x*r.y-y*r.x};}
        double operator*(P r){return x*r.x+y*r.y+z*r.z;}
}p[MXN];
class S{
public:
        int v[3];
        void ot(){printf("%d %d %d\n",v[0],v[1],v[2]);}
        P sp(){return (p[v[1]]-p[v[0]])/(p[v[2]]-p[v[0]]);}
        double sa(){return sp().len()/2.0;}
```

```
        int ck(P pp){return ((pp-p[v[0]])*sp()>0);}
}s[MXN],ss[MXN];
int main(){
        int n,t=2;
        scanf("%d",&n);
        memset(v,0,sizeof(v));
        for(int i=0;i<n;i++)p[i].pin();
        s[0]=(S){0,1,2};
        s[1]=(S){2,1,0};
        int tt=0;
        for(int i=3;i<n;i++){
                int b;
                for(int j=0;j<t;j++){
                        if(!(b=s[j].ck(p[i])))ss[tt++]=s[j];
                        v[s[j].v[0]][s[j].v[1]]=v[s[j].v[1]][s[j].v[2]]=v[s[j].v[2]][s[j].v[0]]=
                }
                for(int j=0;j<t;j++){
                        #define x s[j].v[0]
                        #define y s[j].v[1]
                        if(v[x][y]&&!v[y][x])ss[tt++]=(S){x,y,i};
                        #undef x
                        #undef y
                        #define x s[j].v[1]
                        #define y s[j].v[2]
                        if(v[x][y]&&!v[y][x])ss[tt++]=(S){x,y,i};
                        #undef x
                        #undef y
                        #define x s[j].v[2]
                        #define y s[j].v[0]
                        if(v[x][y]&&!v[y][x])ss[tt++]=(S){x,y,i};
                        #undef x
                        #undef y
                }
                for(int j=0;j<tt;j++)s[j]=ss[j];
                t=tt;tt=0;
        }
        if(t&1)printf("Error!");
        printf("These points' convex hull have %d vertexes, %d edges, %d faces.\n",2+(t>>1),t+(t
        for(int i=0;i<t;i++)ans+=s[i].sa();
        printf("And its superficial area is %.5lf.\n",ans);
}
```

## 运行结果

### 输入数据

```
4
0 0 0
1 0 0
0 1 0
0 0 1
```

## 输出数据

```
These points' convex hull have 4 vertexes, 6 edges, 4 faces,
And its superficial area is 2.36603.
```

# 参考资料


[1]Wikipedia. Computer-assisted proof [EB/OL]. https://en.wikipedia.org/wiki/Computer-assisted_proof, 2020, 6.

[2]Rudolf Fritsch, Gerda Fritsch. The Four-Color Theorem: History, Topological Foundations, and Idea of proof [M]. New York Berlin Heidelberg: Springer-Verlag, 1998.

[3]Alfred Bray Kempe. On the geographical problem of the four colors [J]. American Journal of Mathematics, 1879, 2(3): 193–200.

[4]Norman L. Biggs, E. Keith Lloyd, Robin J. Wilson. Graph Theory 1736-1936 [J]. Oxford University Press, 1999.

[5]K. Appel, W. Haken. Every planar map is four colorable. Part I: Discharging [J]. Illinois Journal of Mathematics, 1977, 21(3): 429–490.

[6]K. Appel, W. Haken, J. Koch. Every planar map is four colorable. Part II: Reducibility [J]. Illinois Journal of Mathematics, 1977, 21(3): 491–567.

[7]Richard Zach. Four Color Theorem Verified in Coq [EB/OL]. University of Calgary, 2005.

[8]Wikipedia. Four color theorem [EB/OL]. https://en.wikipedia.org/wiki/Four_color_theorem, 2020, 5.

[9]董德周. 否定希伍德的“有名反例”和他证明的“五色定理” [J]. 前沿科学, 2011, 5(17): 78-85.

[10]董德周. 关于最大平面图着色的探讨 [A]. 科学新知 [C]. 18-19.

[11]邓硕, 王献芬. 四色定理获证历程及对图论的影响 [J]. 科技视界, 2012: 125-126.

[12]王献芬, 胡作玄. 四色定理的三代证明 [J]. 自然辩证法通讯, 2010, 32(4): 42-48.

[13]KSkun. 三维凸包 [EB/OL]. Luogu, 2020, 6.

[14]Wikipedia. Convex set [EB/OL]. https://en.wikipedia.org/wiki/Convex_set, 2020, 6.