

Algorithm Fundamentals

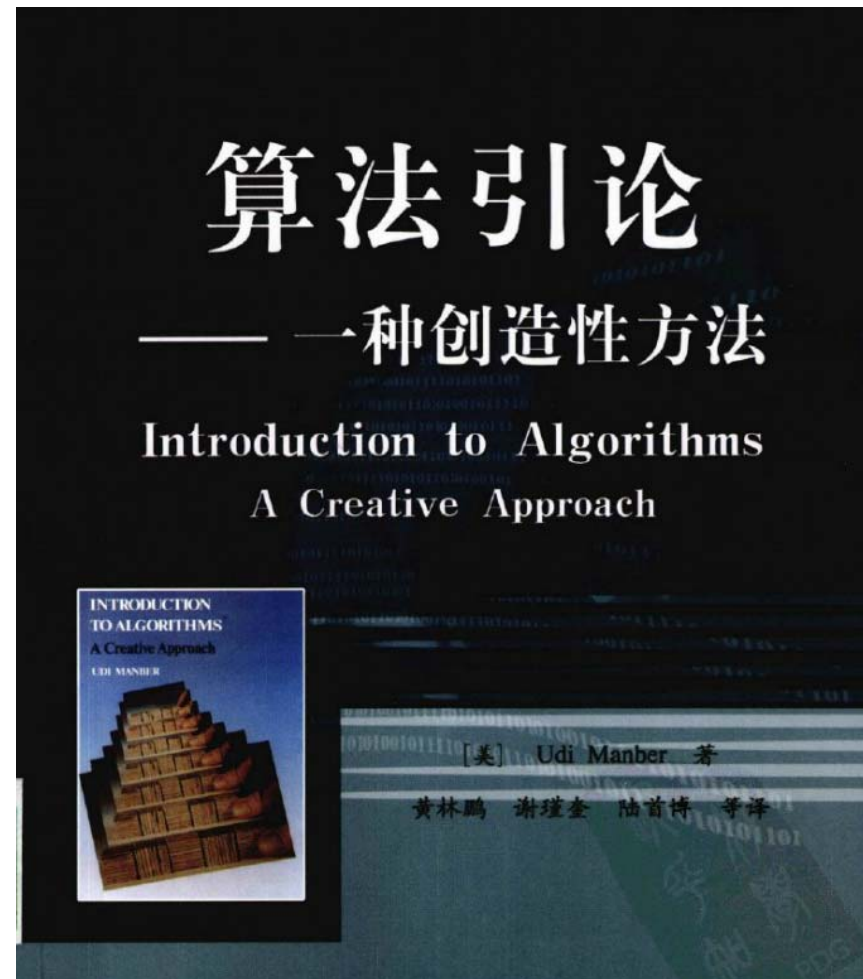
Instructor: Shizhe Zhou

Course Code:00125401

Text book

- Introduction to Algorithms:
A creative Approach

The author: Udi Manber
(The developer of GLIMPSE)



Programming Language

- C++
- You can use JAVA or C#..

考核与作业

作业: 30%(书面作业, 编程作业, 论文阅读[交叉评阅])
期末考试: 60%
出勤: 10%

课件下载: staff.ustc.edu.cn/~szhou/course/algorithmFundamentals2014/

a b c.. About algorithms

- a. 一台计算机可接受的指令:定义明确,长度有限的基本操作序列.
- b. 问题的规模很重要! Scale matters first! 不要忽视算法的复杂度和有效性.
- c. 人脑精力有限, 权衡利弊! Spending hours and days to save a few seconds, NOT a good designing.
- d. 设计不等于证明.设计算法需要定义概念, 挖掘逻辑, 发现模式. 数学归纳的重要性.

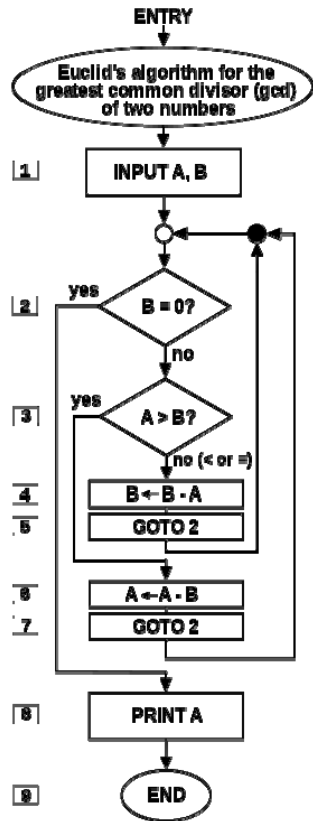
Mathematical Induction

- Induction hypothesis (归纳假设)
- Inductive reasoning (归纳推理)



example

- 设计函数 $\text{gcd}(A,B)$ 求最大公约数 greatest common divisor



```
int gcd_v0(int a, int b){
    while(1){
        if( b>=a ){ /*如果在while循环中有某个分支可能不改变任何数值，也不退出，就会导致一个死循环*/
            b -= a;
            if(b==0) return a;
        }
        else{
            a -= b;
            if(a==0) return b;
        }
    }
}
```

```
int gcd_v1(int a, int b){
    while(1){
        if(b==0) return a;
        if(a==0) return b;
        else{
            if( b>=a ){
                b -= a;
                if(b==0) return a;
            }
            else{
                a -= b;
                if(a==0) return b;
            }
        }
    }
}
```

```
int gcd_v2(int a, int b){
    while(1){
        if(b==0) return a;
        if(a==0) return b;
        else{
            if( b>=a ) b -= a;
            else a -= b;
        }
    }
}
```

example

- 设计函数 gcd(A,B) 求最小公倍数 smallest common multiple

```
/*least common multiple*/
int lcm(int a, int b){
    if(a==0) return 0;
    if(b==0) return 0;
    int ma = a, mb = b;
    while(1){
        if(mb==ma) return ma;
        else{
            if( mb>=ma ) ma += a;
            else      mb += b;
        }
    }
}
```

or

$$\text{lcm}(a, b) = \frac{|a \cdot b|}{\text{gcd}(a, b)}$$

example

- 对任意自然数 x 和 n , $x^n - 1$ 能被 $x - 1$ 整除.

证明 1:

对 n 做归纳: 假设 $x^{n-1} - 1$ 能被 $x - 1$ 整除

$$\text{分解 } x^n - 1 = (x^{n-1} - 1)x + (x - 1)$$


证明 2:

对 x 做归纳: 假设 $x^{n-1} - 1$ 能被 $x - 1$ 整除

$$\begin{aligned} \text{分解 } x^n - 1 &= [(x - 1) + 1]^n - 1 = \sum_{i=0}^n C_n^i (x - 1)^{n-i} \cdot 1^i - 1 \\ &= \sum_{i=0}^{n-1} C_n^i (x - 1)^{n-i} \cdot 1^i + 1 - 1 = \sum_{i=0}^{n-1} C_n^i (x - 1)^{n-i} \end{aligned}$$

归纳法: if $n-1$ establish $\rightarrow n$ establish

归纳法的变种:

- Begin with $n=1$;
If $n-1$ establish, n establish \rightarrow all establish.
- Begin with $n=1$; (强归纳假设)
If for any $m < n$ establish, n establish \rightarrow all establish.
- Begin with $n=1, n=2$; 
If $n-2$ establish, n establish. \rightarrow all establish.
- Begin with $n=1$;
If $n/2$ establish, n establish \rightarrow for all $n=2^k$ establish.

- 结构归纳(Structural Induction)

一般用于找到反命题在结构上的矛盾之处

这种论证的一个实例：考虑一下所有二叉树的集合。我们将证明在完全二叉树中叶子的数目比内部节点的数目多一个。假设该命题错误，则必然存在反例情形；对所有的反例情形而言，一定存在含有极小可能数目的内部节点的一个树。记这个反例树为 C ， C 有 n 个内部节点和 l 个叶子，满足 $n+1 \neq l$ 。

首先 C 是非平凡的，因为平凡的树 $n = 0, l = 1$ ，因此平凡树不是反例。故， C 至少含有其亲代交点是一个内部节点的一个叶子。从树上删掉这个叶子和他的父辈，将被删叶子的节点的兄弟节点提升到被删叶子从前父辈节点所占有的位置。这样做将 n 和 l 同时 -1 ，因此新的树也有 $n+1 \neq l$ ，这样就得到了一个更小的反例。但是在归纳假设中， C 已经是最小的反例了；因此，一开始所谓“存在反例情形”之猜想必然是错误的。

使用归纳法注意陷阱

所有观察到的乌鸦都是黑的



所有乌鸦都是黑的。

除非我们见过所有的乌鸦，
但是我们不可能都知道！
可能还有些罕见的白乌鸦。



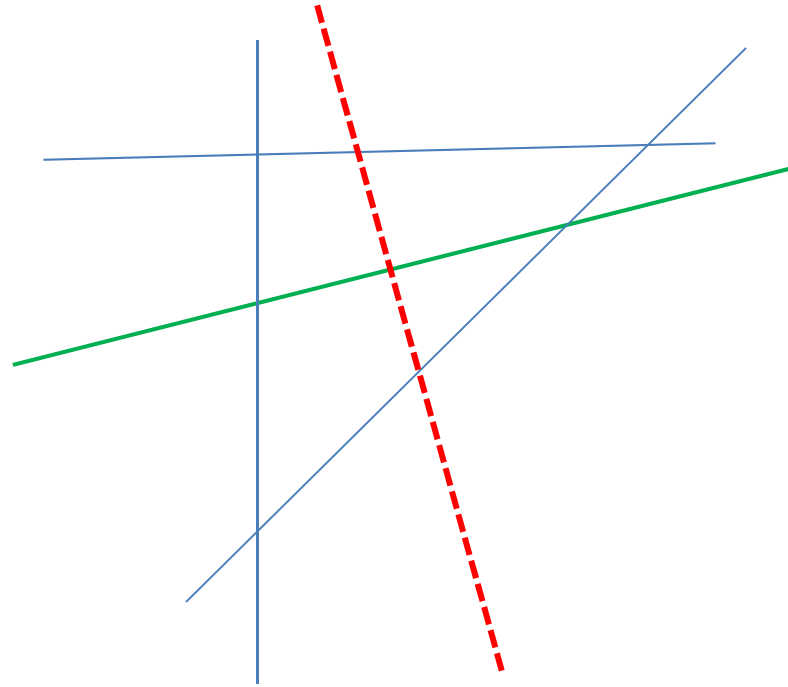
归纳假设的普遍化往往隐藏漏洞

Theorem 2.5 #Region partition by lines

n lines in general position divide the plane into $n(n+1)/2+1$ regions.

general position:

- 1.任何两条不平行
- 2.任何三条不交于一点



p8

求证：平面上存在居于一般位置的 n 条直线, 添加第 $n+1$ 条直线将增加 $n+1$ 个区域.

归纳假设：已存在居于一般位置的 $n-1$ 条直线, 添加第 n 条直线时将增加 n 个区域.

即命题对 $n-1$ 是成立的.

Look at an individual region when only $n-1$ lines exist.

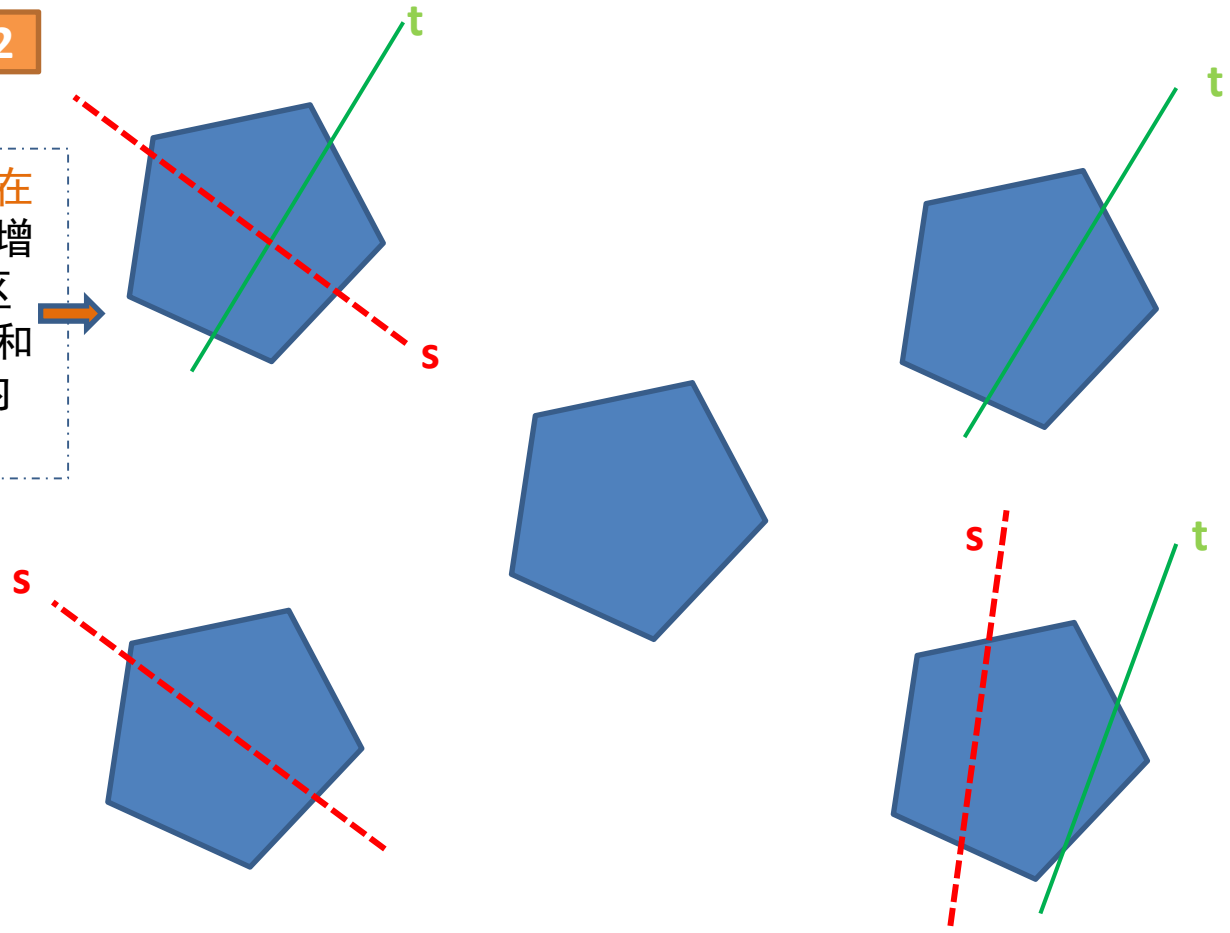
归纳假设：已存在居于一般位置的 $n-1$ 条直线，添加第 n 条直线时将增加 n 个区域。即命题对 $n-1$ 是成立的。

证明：

选定原来 n 条中的任一条直线 s (红色虚线)，并将之移除。我们标记移除后存在的区域 G 为蓝色。此时我们可重复使用归纳假设：添加的第 $n+1$ 条直线 t (绿色) 将增加 n 个区域。然后，将 s 放回去，所有蓝色区域 G 都可依据它和直线 s 和 t 的位置关系划分为5类：

$n+2$

仅有这一类区域， s 的存在使得 t 对 G 的区域数量的增加量从1变为2。而这类区域有且仅有一个（因为 s 和 t 必须相交于某个区域内部的某一点）。得证。



Another simpler proof

- 新添加的第 n 条直线上必有 $n-1$ 个交点,他们将这条直线划分为 n 段,其中的每一段都对应于一个新出现的区域(且这些区域间互不相同),因此第 n 条直线将使得区域的数量增加 n .

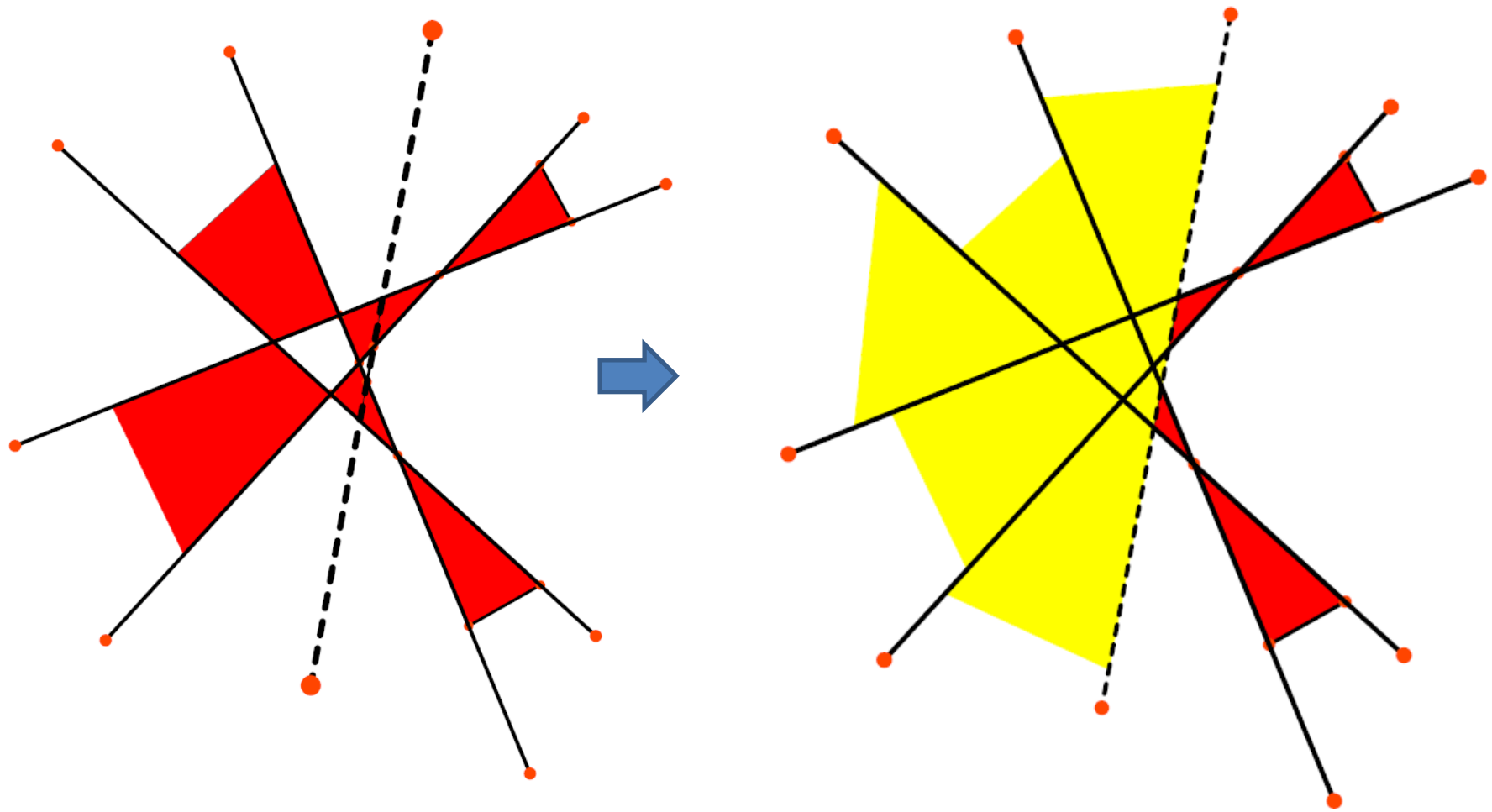
Conclusion:

n lines in general position gives divided regions = $2+2+3\dots+n = (n+1)n/2+1$

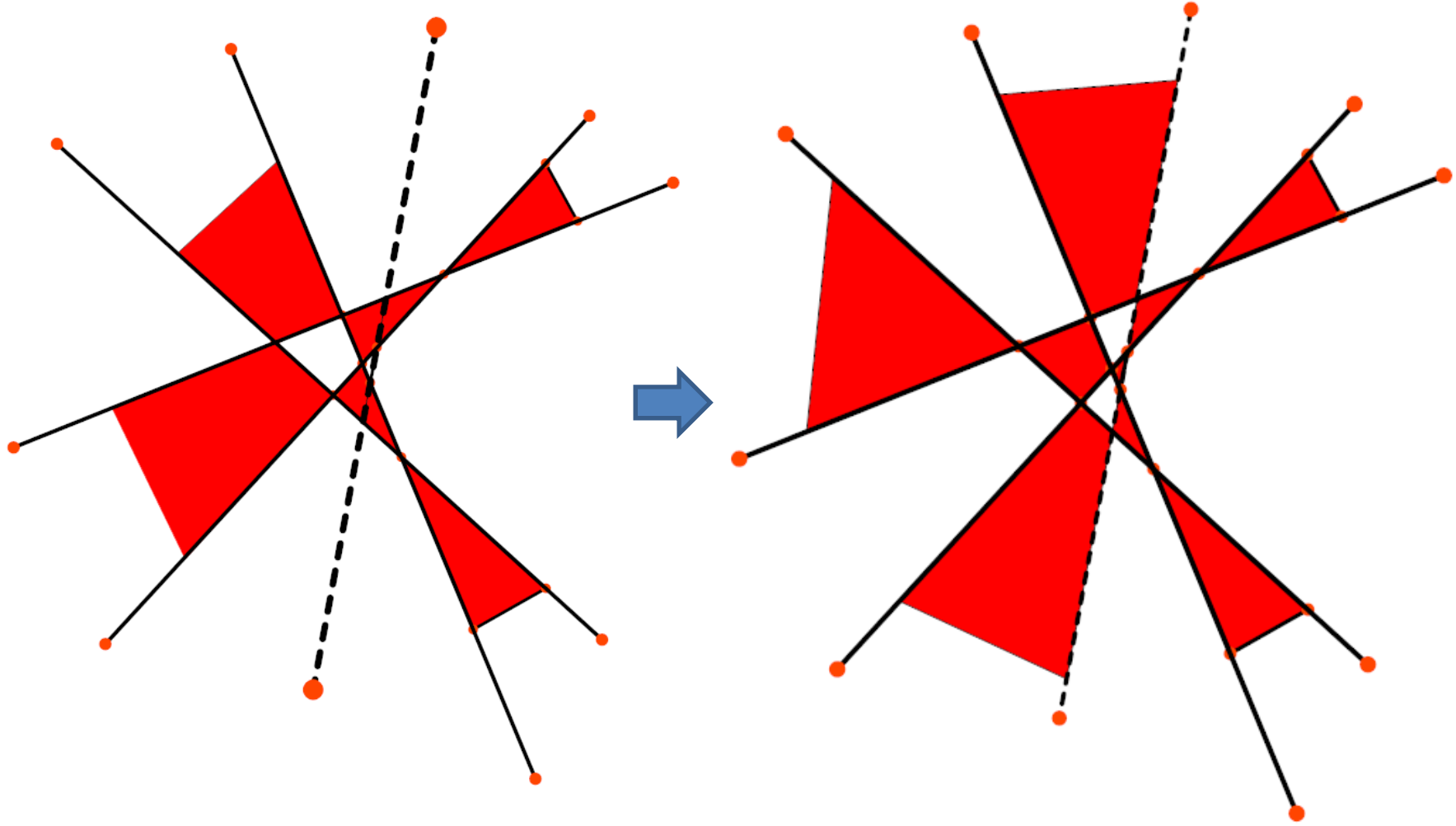
Theorem 2.6

任意条直线构成的区域可用2种颜色着色.

Not has to be in general position!



反转直线n左侧所有区域颜色



金字塔求和

$$\begin{array}{rcccccc} & & & & & & 1 & = & 1 \\ & & & & & & 3 & + & 5 & = & 8 \\ & & & & & 7 & + & 9 & + & 11 & = & 27 \\ & & & 13 & + & 15 & + & 17 & + & 19 & = & 64 \\ 21 & + & 23 & + & 25 & + & 27 & + & 29 & = & 125 \end{array}$$

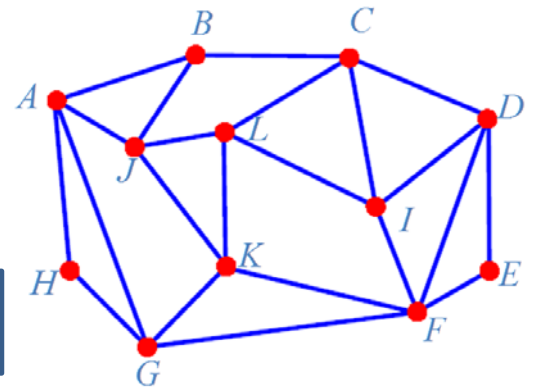
p10

证明上述三角形中第*i*行的和是*i*³

逆向证明

- 转为证明相邻两行的差是 $(i+1)^3 - i^3 = 3i^2 + 3i + 1$
 - 每行对应数字差为 $2i$, 因此第*i*+1行比第*i*行前*i*个数大 $2i^2$.
- 转为证明 *i*+1 行的最后一个数等于 $i^2 + 3i + 1$ 即可.
 - 计算相邻两行最后一个数字之差: $i^2 + 3i + 1 - [(i-1)^2 + 3(i-1) + 1] = 2i + 2$.
- 转为证明 *i*+1 行的最后一个数比 *i* 行的最后一个数大 $2i + 2$ → 直接成立.

欧拉公式



- 任意连通平面图: $\#V + \#F = \#E + 2$.

$\#V$ =节点数; $\#F$ =面数; $\#E$ =边数; 图的外部区域算一个面(外部面)

要点: 对双参数($\#V, \#F$)依次归纳证明.

先对只有一个面的情形论证, 然后对 n 个面的情形论证. 只有一个面时, $\#E$ 可变.

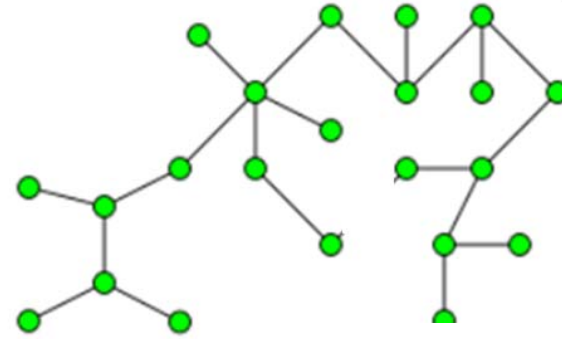
归纳假设1: $\#F=1$ 的连通平面图满足 $\#V+1 = \#E+2$.

归纳假设2: $\#F=n$ 的连通平面图满足 $\#V+n = \#E+2$.

归纳假设1: $\#F=1$ 的连通平面图满足 $\#V = \#E+1$.

等价于证明: 对任意树有 $\#V=\#E+1$.

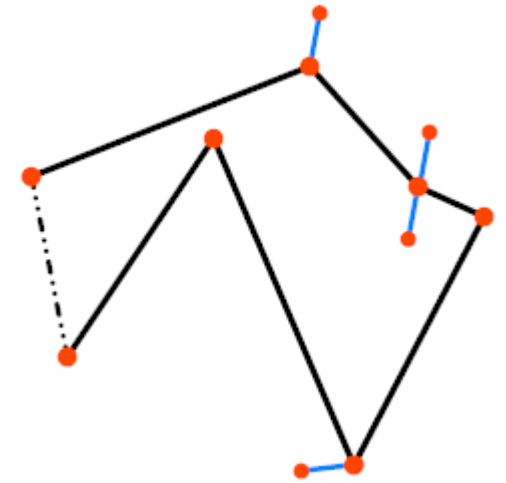
仅有一个面(无回路)的连通图就是树



证明:

归纳假设: 当节点 n 时成立.

在树中一定存在一个点度数为1. 若不然, 即有每一节点的度数都 ≥ 2 . 从某一节点出发沿它的一条边步进到下一个节点, 因为每一节点的度数都 ≥ 2 , 可以保证每次都走与上一次不同的边, 这个序列可以无限长. 然而我们只有有限个(n 个)节点, 如果形成了无限长的序列, 其中必含有回路. 矛盾.



所以存在度数为1的节点, 取出该点, 边数减少1, 且图仍然连通(因原图中其余两点间的任何路径都不经过该点).

利用归纳假设, 仍然有 $\#V = \#E+1$.

归纳假设2: $\#F=n$ 的连通平面图满足 $\#V+n=\#E+2$.

证明:

以归纳假设1为基础,我们有 $\#F=1$ 时,命题成立
现假设 $\#F=n$ 时成立,

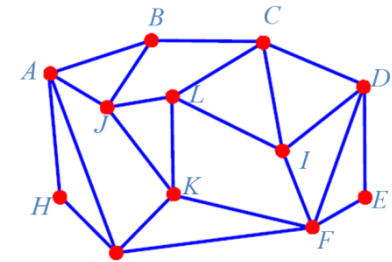
对 $\#F=n+1$ 的连通平面图 G ,一定存在一个面 f 与外部面相邻.

去掉 f 与外部面相邻的边,因为去掉回路中任一条边,回路中任何两点**依旧保持连通**.所以剩下的图 G' 仍然是连通的平面图.

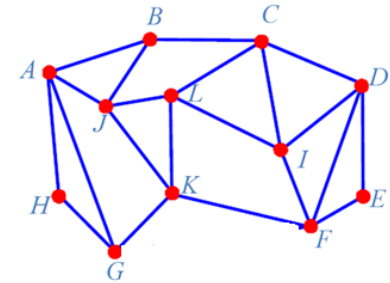
利用归纳假设, $\#V(G')+n = \#E(G')+2$;

G' 与 G 相比,面数减少1,边数减少1,而点数不变,所以有 $\#V(G) + n = \#E(G)+2$.

而 n 是 $\#F$, 所以 $\#V + \#F = \#E + 2$.



Caution:要利用归纳假设,需先验证连通性!



三个参数依序
结构化证明。

循环不变量

- Loop invariant : a property, a [logical assertion](#), that holds before (and after) each repetition.
- Knowing its invariant(s) is essential for understanding the effect of a loop.

```
1. int max(int n, const int a[n]) {
2.     int m = a[0];
3.     // m equals the maximum value in a[0...0]
4.     int i = 1;
5.     while (i != n) {
6.         // m equals the maximum value in a[0...i-1]
7.         if (m < a[i])
8.             m = a[i];
9.         // m equals the maximum value in a[0...i]
10.        ++i;
11.        // m equals the maximum value in a[0...i-1]
12.    }
13.    // m equals the maximum value in a[0...i-1], and i==n
14.    return m;
15. }
```