

# 二叉堆、并查集和树状数组

刘汝佳

# 优先队列

- 优先队列(priority queue): 可以把元素加入到优先队列中, 也可以从队列中取出优先级最高的元素, 即以下ADT
  - **Insert(T, x):** 把x加入优先队列中
  - **DeleteMin(T, x):** 获取优先级最高的元素x, 并把它从优先队列中删除

# 堆的操作

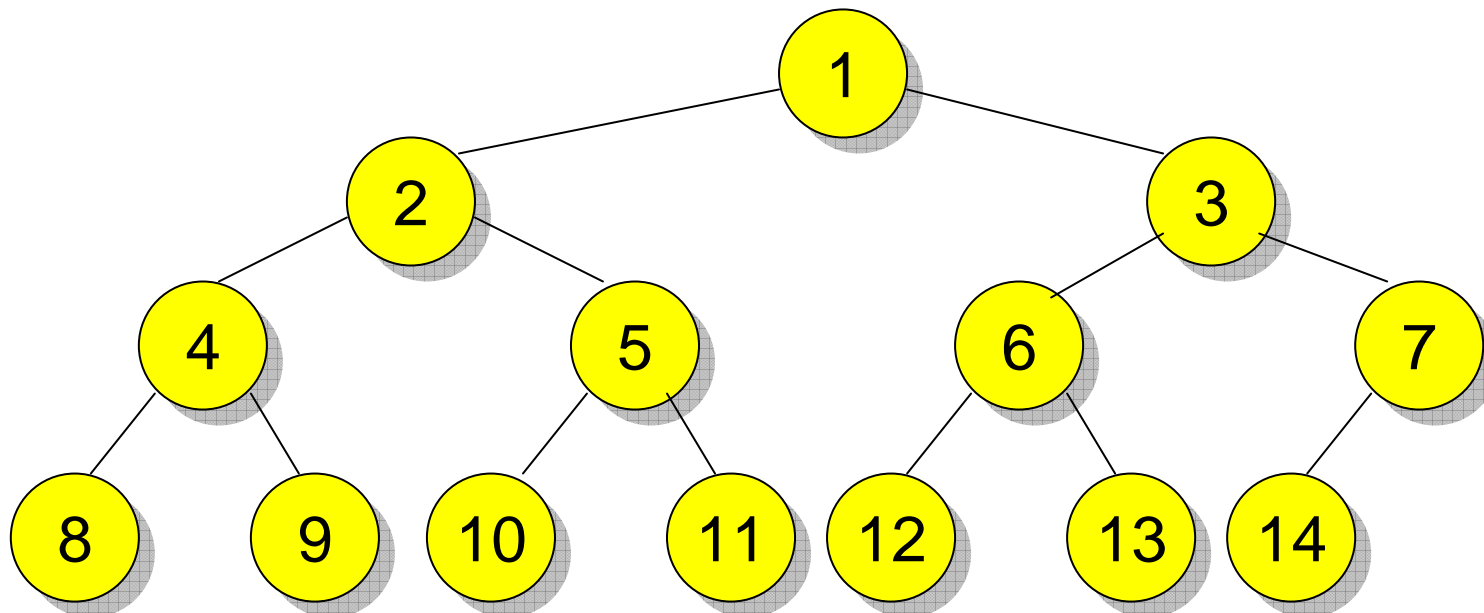
- 用二叉堆(binary heap)很容易实现优先队列
- 除了实现优先队列, 堆还有其他用途, 因此操作比优先队列多
  - **Getmin(T, x):** 获得最小值
  - **Delete(T, x):** 删除任意已知结点
  - **DecreaseKey(T, x, p):** 把x的优先级降为p
  - **Build(T, x):** 把数组x建立成最小堆

# 堆的定义

- 堆是一个完全二叉树
  - 所有叶子在同一层或者两个连续层
  - 最后一层的结点占据尽量左的位置
- 堆性质
  - 为空, 或者最小元素在根上
  - 两棵子树也是堆

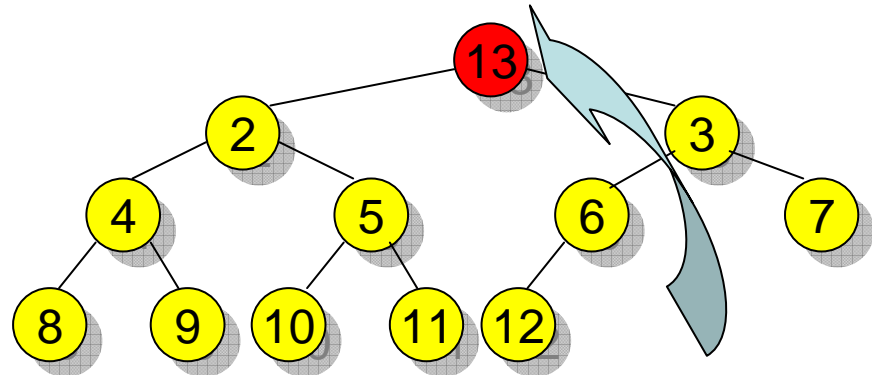
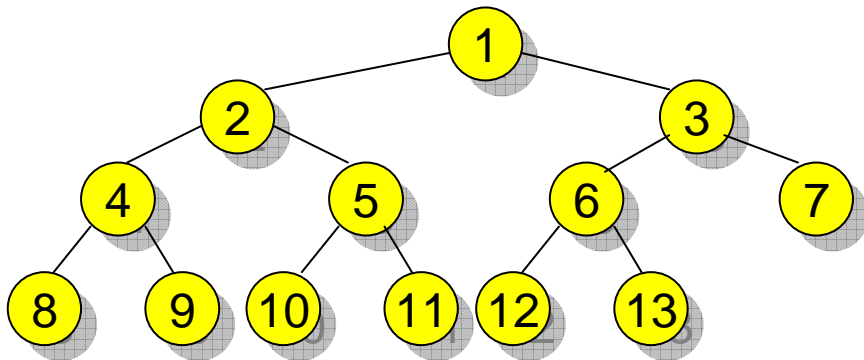
# 存储方式

- 最小堆的元素保存在`heap[1..hs]`内
  - 根在`heap[1]`
  - $K$ 的左儿子是 $2k$ ,  $K$ 的右儿子是 $2k+1$ ,
  - $K$ 的父亲是 $\lfloor k/2 \rfloor$

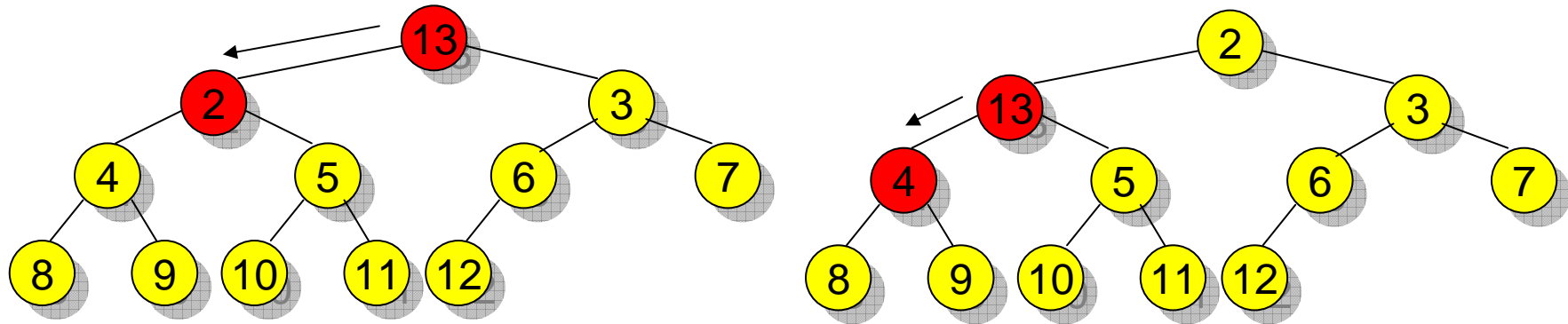


# 删除最小值元素

- 三步法
  - 直接删除根
  - 用最后一个元素代替根上元素
  - 向下调整



- 首先选取当前结点p的较大儿子. 如果比p大, 调整停止, 否则交换p和儿子, 继续调整



```
void sink(int p){  
    int q=p<<1, a = heap[p];  
    while(q<=hs){  
        if(q<hs&&heap[q+1]<heap[q])q++;  
        if(heap[q]>=a) break;  
        heap[p]=heap[q]; p=q; q=p<<1;  
    }  
    heap[p] = a;  
}
```

# 插入元素和向上调整

- 插入元素是先添加到末尾, 再向上调整
- 向上调整: 比较当前结点 $p$ 和父亲, 如果父亲比 $p$ 小, 停止; 否则交换父亲和 $p$ , 继续调整

```
void swim(int p){  
    int q = p>>1, a = heap[p];  
    while(q && a<heap[q]){ heap[p]=heap[q]; p=q; q=p>>1; }  
    heap[p] = a;  
}
```



# 堆的建立

- 从下往上逐层向下调整. 所有的叶子无需调整, 因此从 $hs/2$ 开始. 可用数学归纳法证明循环变量为 $i$ 时, 第 $i+1, i+2, \dots, n$ 均为最小堆的根

```
void insert(int a)
{ heap[++hs]=a; swim(hs); }
int getmin()
{ int r=heap[1]; heap[1]=heap[hs--];
  sink(1); return r; }
int decreaseKey(int p, int a)
{ heap[p]=a; swim(p); }
void build()
{ for(int i=hs/2;i>0;i--) sink(i); }
```

# 时间复杂度分析

- 向上调整/向下调整
  - 每层是常数级别, 共 $\log n$ 层, 因此 $O(\log n)$
- 插入/删除
  - 只调用一次向上或向下调整, 因此都是 $O(\log n)$
- 建堆
  - 高度为 $h$ 的结点有 $n/2^{h+1}$ 个, 总时间为

$$\sum_{h=0}^{\lfloor \log n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil \times O(h) = O\left( n \sum_{h=0}^{\lfloor \log n \rfloor} \frac{h}{2^h} \right)$$

# 例1. k路归并问题

- 把k个有序表合并成一个有序表.
- 元素共有n个.

# 分析

- 每个表的元素都是从左到右移入新表
- 把每个表的当前元素放入二叉堆中, 每次删除最小值并放入新表中, 然后加入此序列的下一个元素
- 每次操作需要 $\log k$ 时间, 因此总共需要 $n \log k$ 的时间

## 例2. 序列和的前 $n$ 小元素

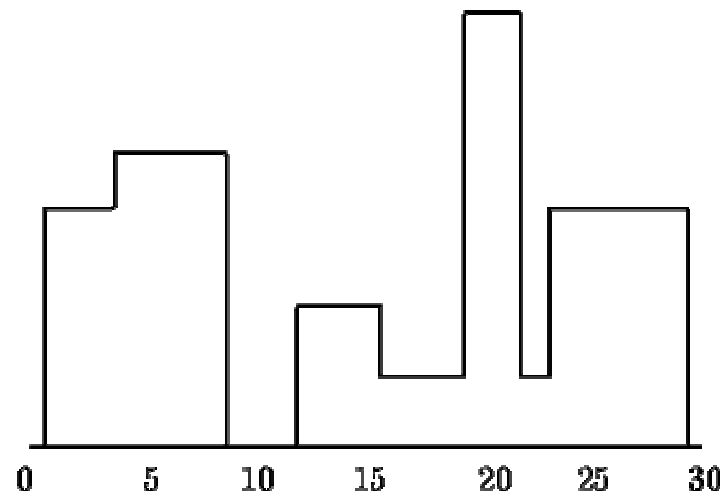
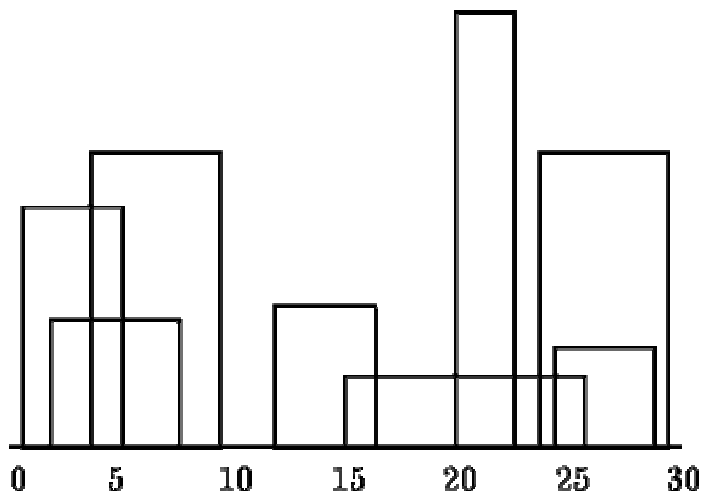
- 给出两个长度为 $n$ 的有序表A和B, 在A和B中各任取一个, 可以得到 $n^2$ 个和. 求这些和最小的 $n$ 个

# 分析

- 可以把这些和看成 $n$ 个有序表:
  - $A[1]+B[1] \leq A[1]+B[2] \leq A[1]+B[3] \leq \dots$
  - $A[2]+B[1] \leq A[2]+B[2] \leq A[2]+B[3] \leq \dots$
  - ...
  - $A[n]+B[1] \leq A[n]+B[2] \leq A[n]+B[3] \leq \dots$
- 类似刚才的算法, 每次 $O(\log n)$ , 共取 $n$ 次最小元素, 共 $O(n \log n)$

## 例3. 轮廓线

- 每一个建筑物用一个三元组表示(L, H, R), 表示左边界, 高度和右边界
- 轮廓线用X, Y, X, Y...这样的交替式表示
- 右图的轮廓线为: (1, 11, 3, 13, 9, 0, 12, 7, 16, 3, 19, 18, 22, 3, 23, 13, 29, 0)
- 给N个建筑, 求轮廓线



# 分析

- 算法一：用数组记录每一个元线段的高度
  - 离散化, 有 $n$ 个元线段
  - 每次插入可能影响 $n$ 个元线段,  $O(n)$ , 共 $O(n^2)$
  - 从左到右扫描元线段高度, 得轮廓线
- 算法二：每个建筑的左右边界为事件点
  - 把事件点排序, 从左到右扫描
  - 维护建筑物集合, 事件点为线段的插入删除
  - 需要求最高建筑物, 用堆, 共 $O(n\log n)$



## 例4. 丑数

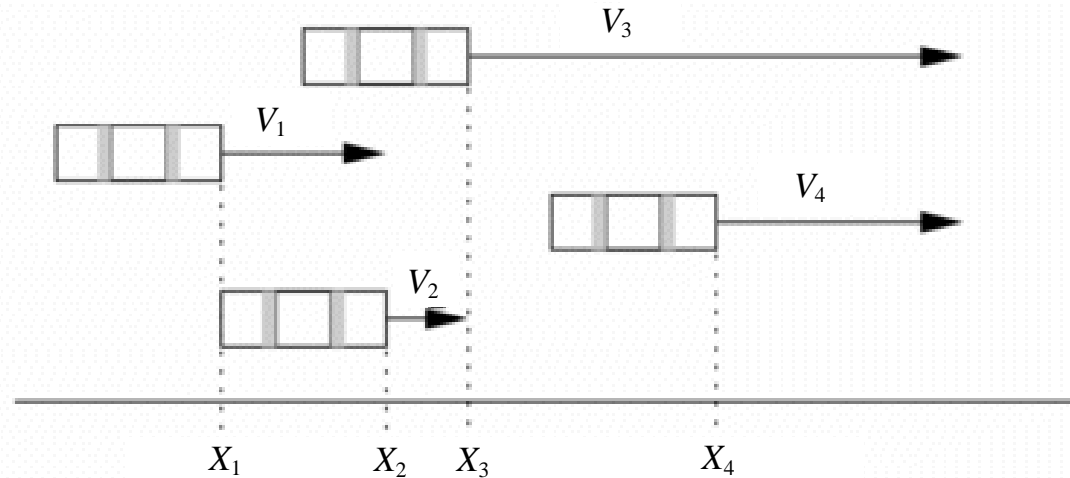
- 素因子都在集合{2, 3, 5, 7}的数称为ugly number
- 求第n大的丑数

# 分析

- 初始：把1放入优先队列中
- 每次从优先队列中取出一个元素 $k$ ，把 $2k$ ,  $3k$ ,  $5k$ ,  $7k$ 放入优先队列中
- 从2开始，取出的第 $n$ 个元素就是第 $n$ 大丑数
- 每取出一个数，插入4个数，因此任何堆里的元素是 $O(n)$ 的，时间复杂度为 $O(n \log n)$
- **思考：**如果集合元素个数 $m$ 与 $n$ 同阶，时间复杂度将变为怎样？如何优化？

## 例5. 赛车

- 有 $n$ 辆赛车从各不相同的地方以各种的速度(速度 $0 < v_i < 100$ )开始往右行驶，不断有超车现象发生。



- 给出 $n$ 辆赛车的描述（位置 $x_i$ ，速度 $v_i$ ），赛车已按照位置排序（ $x_1 < x_2 < \dots < x_n$ ）
- 输出超车总数以及按时间顺序的前 $m$ 个超车事件

# 分析

- 事件个数 $O(n^2)$ , 因此只能一个一个求
- 给定两辆车, 超越时刻预先可算出
- 第一次超车可能在哪些辆车之间?
  - 维护所有车的前方相邻车和追上时刻
    - 局部: 此时刻不一定是该车下个超车时刻!
    - 全局: 所有时刻的最小值就是下次真实超车时刻
- 维护: 超车以后有什么变化?
  - 相对顺序变化...改变三个车的前方相邻车
  - 重新算追上时刻, 调整三个权
  - 简单的处理方法: 删除三个再插入三个

## 例6. 可怜的奶牛

- 农夫John有 $n$  ( $n \leq 100\,000$ ) 头奶牛，可是由于它们产的奶太少，农夫对它们很不满意，决定每天把产奶最少的一头做成牛肉干吃掉。但还是有一点舍不得，John打算如果不止有一头奶牛产奶最少，当天就大发慈悲，放过所有的牛。
- 由于John的奶牛产奶是周期性的，John在一开始就能可以了解所有牛的最终命运，不过他的数学很差，所以请你帮帮忙，算算最后有多少头奶牛可以幸免于难。每头奶牛的产奶周期 $T_i$ 可能不同，但不会超过10。在每个周期中，奶牛每天产奶量不超过200。

# 分析

- 如果采用最笨的方法，每次先求出每头牛的产奶量，再求最小值，则每天的复杂度为 $O(n)$ ，总复杂度为 $O(Tn)$ ，其中 $T$ 是模拟的总天数。由于周期不超过10，如果有的牛永远也不会被吃掉，那么我们需要多模拟2520天（1, 2, 3, ..., 10的最小公倍数）才能确定
- 周期同为 $t$ 的奶牛在没有都被吃掉之前，每天的最小产奶量也是以 $t$ 为周期的。因此如果把周期相同的奶牛合并起来，每天只需要比较10类奶牛中每类牛的最小产奶量就可以了，每天的复杂度为 $O(k)$ ，其中 $k$ 为最长周期

# 分析

- 假设周期为6的牛有4头，每次只需要比较 $k$ 组牛的“代表”就可以了，每天模拟的时间复杂度为 $O(k)$ 。

项 目	第 $6n+1$ 天	第 $6n+2$ 天	第 $6n+3$ 天	第 $6n+4$ 天	第 $6n+5$ 天	第 $6n+6$ 天
牛1	2	5	3	5	7	4
牛2	3	1	6	7	5	4
牛3	5	3	3	5	3	9
牛4	4	4	3	8	8	2
合并结果	2（牛1）	1（牛2）	3（多）	5（多）	3（牛3）	2（牛4）

# 分析

- 只要周期为6的牛都不被吃掉，这个表一直是有效的。但是在吃掉一头奶牛后，我们需要修改这个表，使它仍然记录着每天的最小产奶量
  - 方法一: 重新计算，时间 $O(h)$ ，其中 $h$ 是该组的牛数
  - 方法二: 把一个周期中每天的最小产奶量组织成堆，每次删除操作的复杂度是 $O(k \log h)$
- 由于每头奶牛最多被吃掉一次，因此用在维护“最小产奶量结构”的总复杂度不超过 $O(nk \log n)$ 。每天复杂度为 $O(k)$ ，总复杂度为 $O(Tk + nk \log n)$



## 例7. 黑匣子

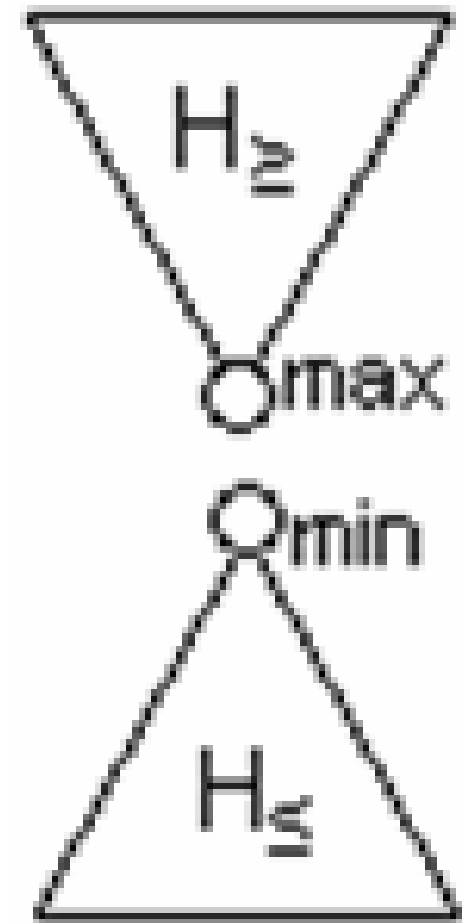
- 我们使用黑匣子的一个简单模型。它能存放一个整数序列和一个特别的变量 $i$ 。在初始时刻，黑匣子为空且 $i$ 等于0。这个黑匣子执行一序列的命令。有两类命令：
- **ADD( $x$ )**: 把元素 $x$ 放入黑匣子；
- **GET**:  $i$ 增1的同时，输出黑匣子内所有整数中第 $i$ 小的数。牢记第 $i$ 小的数是当黑匣子中的元素以非降序排序后位于第 $i$ 位的元素

## 例7. 黑匣子

编号	命令	i	黑匣子内容	输出
1	ADD(3)	0	3	
2	GET	1	<b>3</b>	3
3	ADD(1)	1	1, 3	
4	GET	2	1, <b>3</b>	3
5	ADD(-4)	2	-4, 1, 3	
6	ADD(2)	2	-4, 1, 2, 3	
7	ADD(8)	2	-4, 1, 2, 3, 8	
8	ADD(-1000)	2	-1000, -4, 1, 2, 3, 8	
9	GET	3	-1000, -4, <b>1</b> , 2, 3, 8	1
10	GET	4	-1000, -4, 1, <b>2</b> , 3, 8	2
11	ADD(2)	4	-1000, -4, 1, 2, 2, 3, 8	

# 分析

- 降序堆 $H_{\geq}$ 和升序堆 $H_{\leq}$ 如图放置
- $H_{\geq}$ 根节点的值 $H_{\geq}[1]$ 在堆 $H_{\geq}$ 中最大,  
 $H_{\leq}$ 根节点的值 $H_{\leq}[1]$ 在堆 $H_{\leq}$ 中最小,  
并满足
  - $H_{\geq}[1] \leq H_{\leq}[1]$
  - $\text{size}[H_{\geq}] = i$
- **ADD( $x$ ):** 比较 $x$ 与 $H_{\geq}[1]$ , 若 $x \geq H_{\geq}[1]$ , 则将 $x$ 插入 $H_{\leq}$ , 否则从 $H_{\geq}$ 中取出 $H_{\geq}[1]$ 插入 $H_{\leq}$ , 再将 $x$ 插入 $H_{\geq}$
- **GET:**  $H_{\leq}[1]$ 就是待获取的对象。输出 $H_{\leq}[1]$ , 同时从 $H_{\leq}$ 中取出 $H_{\leq}[1]$ 插入 $H_{\geq}$ , 以维护条件(2)

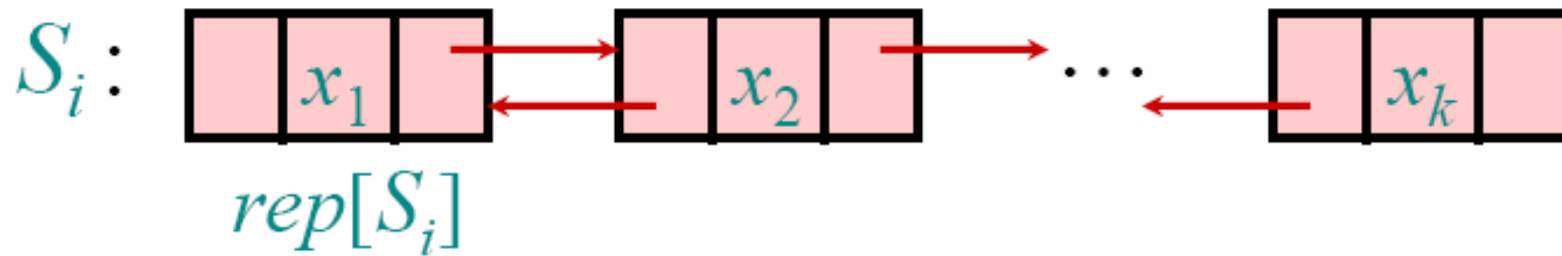


# 并查集

- 并查集维护一些不相交集合  $S = \{S_1, S_2, \dots, S_r\}$ , 每个集合  $S_i$  都有一个特殊元素  $\text{rep}[S_i]$ , 称为集合代表. 并查集支持三种操作
  - **Make-Set(x)**: 加入一个集合  $\{x\}$  到  $S$ , 且  $\text{rep}[\{x\}] = x$ . 注意,  $x$  不能被包含在任何一个  $S_i$  中, 因为  $S$  里任何两个集合应是不相交的
  - **Union(x, y)**: 把  $x$  和  $y$  所在的两个不同集合合并. 相当于从  $S$  中删除  $S_x$  和  $S_y$  并加入  $S_x \cup S_y$
  - **Find-Set(x)**: 返回  $x$  所在集合  $S_x$  的代表  $\text{rep}[S_x]$

# 链结构

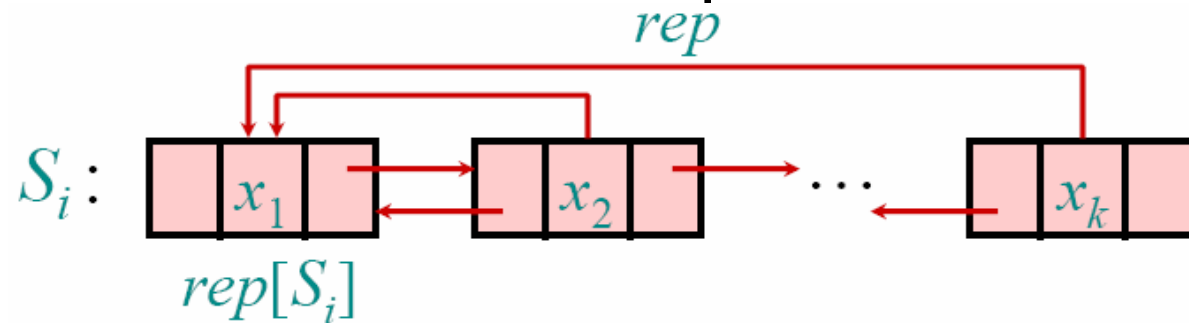
- 每个集合用双向链表表示,  $\text{rep}[S_i]$  在链表首部



- **Make-Set(x)**: 显然是 $O(1)$ 的
- **Find-Set(x)**: 需要不断往左移, 直到移动到首部. 最坏情况下是 $O(n)$ 的
- **Union(x, y)**: 把 $S_y$ 接在 $S_x$ 的尾部, 代表仍是  $\text{rep}[S_x]$ . 为了查找链表尾部, 需要 $O(n)$

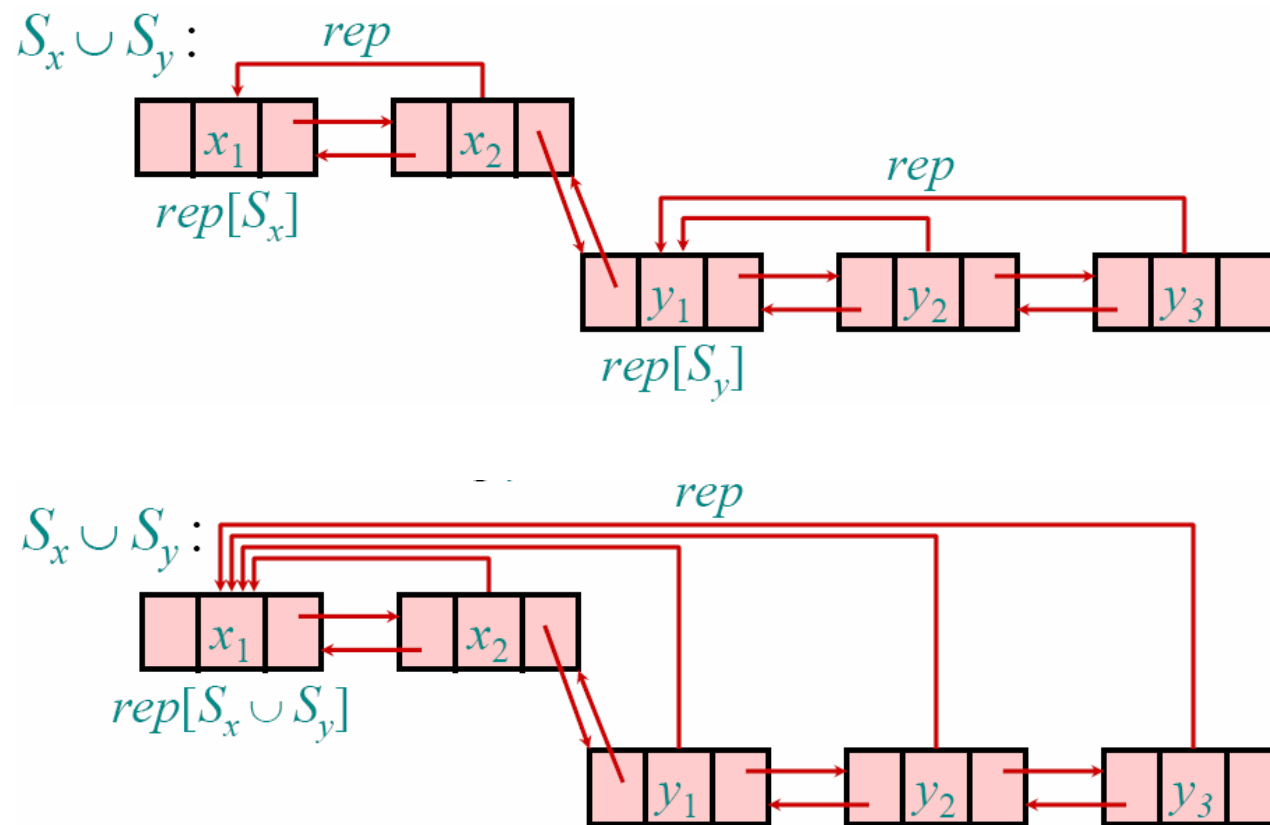
# 增强型链结构

- 给每个结点增加一个指回rep的指针



- Make-Set(x):** 仍为常数
- Find-Set(x):** 降为常数(直接读rep)
- Union(x, y):** 变得复杂: 需要把 $S_y$ 里所有元素的rep指针设为 $rep[S_x]$ !

# 增强型链结构的合并



- 可以把x合并到y中，也可以把y合并在x中

# 技巧1: 小的合并到大的中

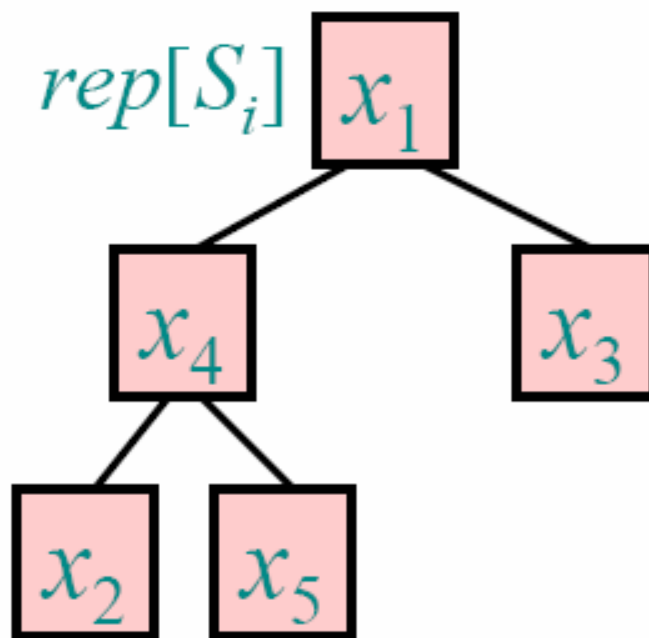
- 显然, 把小的合并到大的中, 这一次Union操作会比较节省时间, 更精确的分析?
- 用 $n$ ,  $m$ ,  $f$ 分别表示Make-Set的次数, 总操作次数和Find-Set的次数, 则有
- **定理:** 所有Union的总时间为 $O(n \log n)$
- **推论:** 所有时间为 $O(m + n \log n)$
- **证明:** 单独考虑每个元素 $x$ , 设所在集合为 $S_x$ , 则修改 $\text{rep}[x]$ 时,  $S_x$ 至少加倍. 由于 $S_x$ 不超过 $n$ , 因此修改次数不超过 $\log_2 n$ , 总 $n \log n$



# 树结构

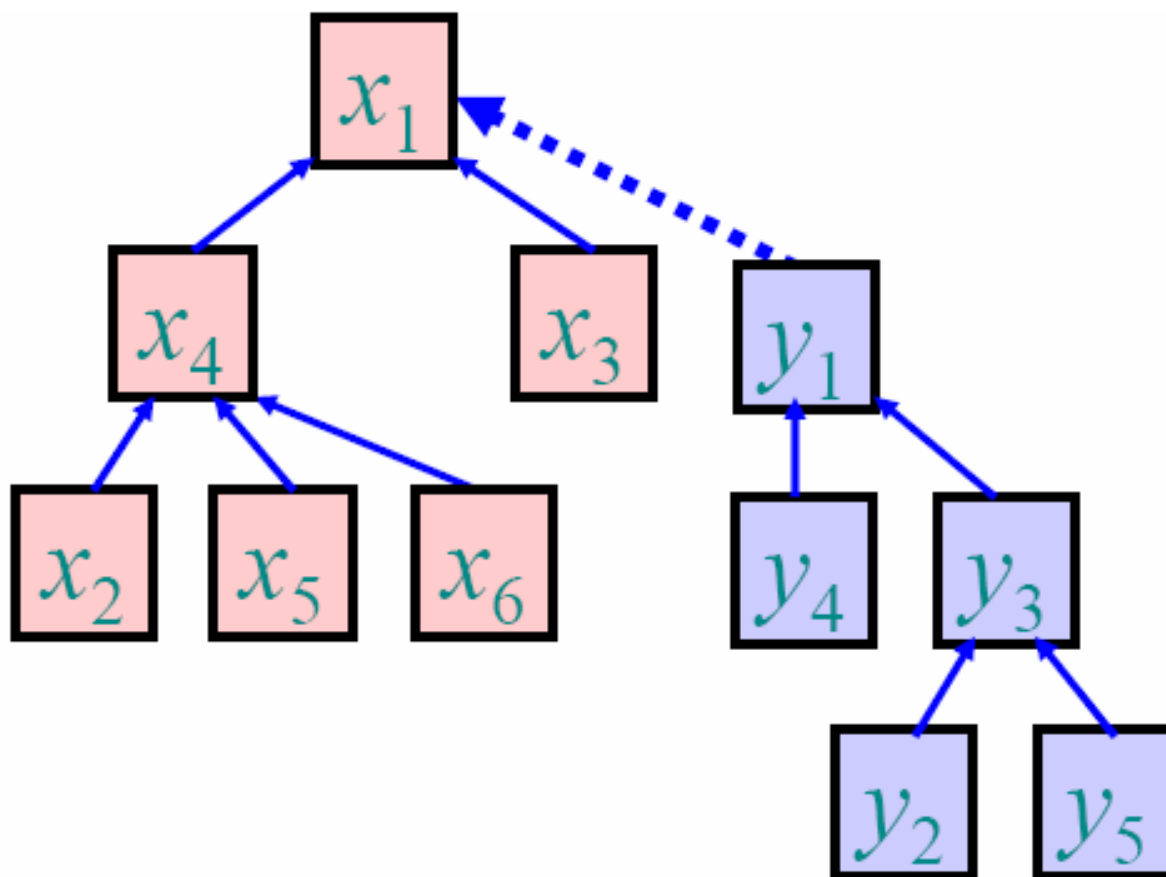
- 每个集合用一棵树表示, 根为集合代表

$$S_i = \{x_1, x_2, x_3, x_4, x_5\}$$



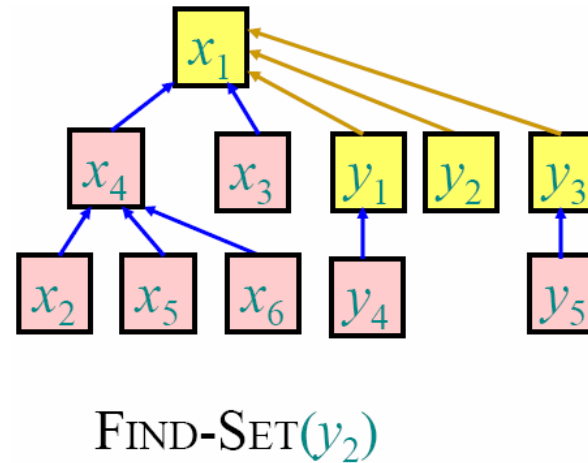
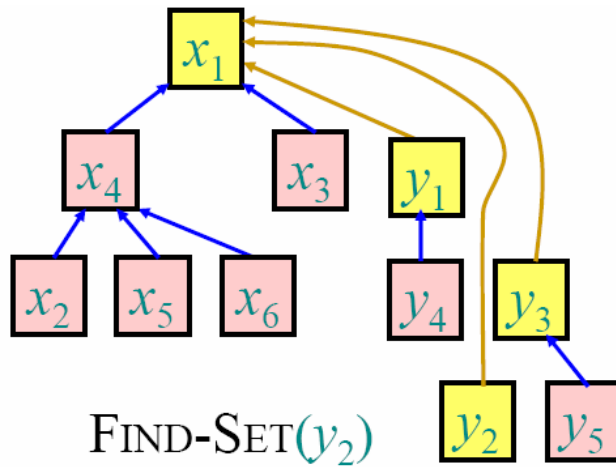
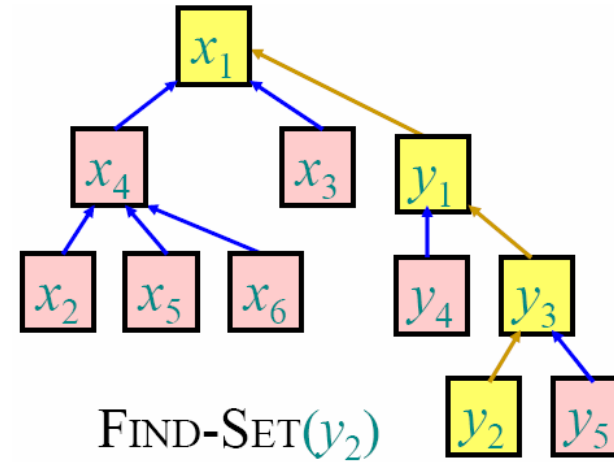
# 树结构的合并

- 和链结构类似, 小的合并到大的中



## 技巧2: 路径压缩

- 查找结束后顺便把父亲设置为根, 相当于有选择的设置rep指针而不像链结构中强制更新所有rep



# 路径压缩的分析

- 设 $w[x]$ 为 $x$ 的子树的结点数, 定义势能函数

$$\phi(x_1, \dots, x_n) = \sum_i \lg \text{weight}[x_i]$$

- **Union( $x_i, x_j$ )**增加势能. 最多会让 $w[\text{rep}[x_i]]$  增加  $w[\text{rep}[x_j]] \leq n$ , 因此势能增加不超过 $\lg n$
- **Find-Set( $x$ )**减少势能. 把路径压缩看作是从根到结点 $x$ 的向下走过程, 则除了第一次外的其他向下走的步骤 $p \rightarrow c$ 会让 $c$ 的子树从 $p$ 的子树中移出, 即 $w[p]$ 减少 $w[c]$ , 而其他点的 $w$ 值保持不变

# 路径压缩的分析

- **Find-Set**除了第一次外的其他向下走的步骤  $p \rightarrow c$  会让  $c$  的子树从  $p$  的子树中移出
  - 情况一:  $w[c] \geq w[p]/2$ , 则势能将至少减少1
  - 情况二:  $w[c] < w[p]/2$ , 这种情况最多出现  $\log n$  次, 因为  $w[p]$  最多进行  $\log n$  次除2操作就会得到1
- **Union**操作积累起来的  $m \log n$  的势能将被 **Find-Set** 消耗, 情况一最多消耗  $m \log n$  次, 情况二本身不超过  $m \log n$  次, 因此
- **定理:** Find-Set 的总时间为  $O(m \log n)$

# 路径压缩的分析

- **定理:** 如果所有**Union**发生在**Find-Set**之前, 则所有操作的时间复杂度为 $O(m)$
- **证明:** 每次**Find-Set**将会让路径上除了根的所有结点为根的儿子. 所有结点只会有一次改变, 因此总时间复杂度为 $O(m)$
- 也就是说
  - 只使用技巧1(启发式合并):  $O(m+n\log n)$
  - 只使用技巧2(路径压缩):  $O(m\log n)$
- 同时使用呢?

# Ackermann函数及其反函数

Define  $A_k(j) = \begin{cases} j+1 & \text{if } k=0, \\ A_{k-1}^{(j+1)}(j) & \text{if } k \geq 1. \end{cases}$  – iterate  $j+1$  times

$$A_0(j) = j + 1$$

$$A_0(1) = 2$$

$$A_1(j) \sim 2^j$$

$$A_1(1) = 3$$

$$A_2(j) \sim 2^j 2^j > 2^j$$

$$A_2(1) = 7$$

$$A_3(1) = 2047$$

$$A_3(j) > 2^{2^{2^{\dots^{2^j}}}} \quad \left. \vphantom{2^{2^{2^{\dots^{2^j}}}}} \right\} j$$

$A_4(j)$  is a lot bigger.

$$A_4(1) > 2^{2^{2^{\dots^{2^{2047}}}}} \quad \left. \vphantom{2^{2^{2^{\dots^{2^{2047}}}}} \right\} 2048$$

Define  $\alpha(n) = \min \{k : A_k(1) \geq n\} \leq 4$  for practical  $n$ .

# 树结构的完整结论

- **定理:**  $m$ 个操作的总时间复杂度为  $O(m\alpha(n))$

```
void makeset(int x){ rank[x] = 0; p[x]=x; }
int findset(int x){
    int i, px = x;
    while (px != p[px]) px = p[px];
    while (x != px) { i = p[x]; p[x] = px; x = i; }
    return px;
}
void unionset (int x , int y){
    x = findset(x); y = findset(y);
    if(rank[x] > rank[y]) p[y] = x;
    else { p[x] = y; if(rank[x] == rank[y]) rank[y]++; }
}
```

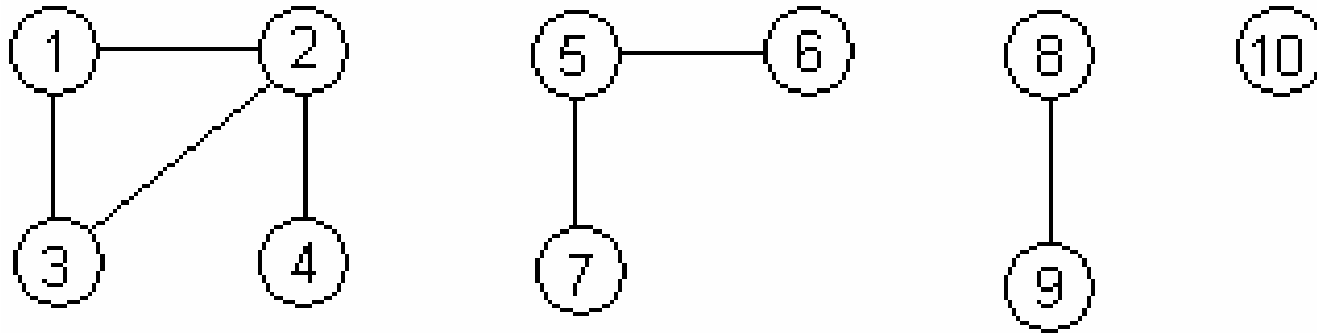


# 例1. 亲戚

- 或许你并不知道，你的某个朋友是你的亲戚。他可能是你的曾祖父的外公的女婿的外甥女的表姐的孙子。如果能得到完整的家谱，判断两个人是否亲戚应该是可行的，但如果两个人的最近公共祖先与他们像个好几代，使得家谱十分庞大，那么检验亲戚关系实非人力所能及。在这种情况下，最好的帮手就是计算机。
- 为了将问题简化，你将得到一些亲戚关系的信息，如同Marry和Tom是亲戚，Tom和Ben是亲戚，等等。从这些信息中，你可以推出Marry和Ben是亲戚。请写一个程序，对于我们的关于亲戚关系的提问，以最快的速度给出答案。

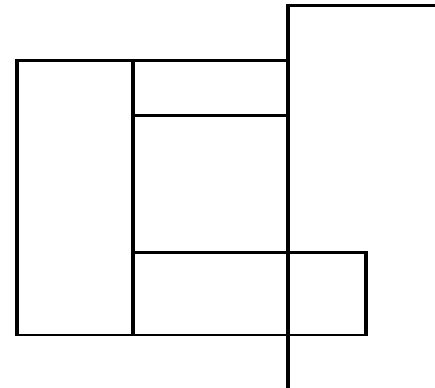
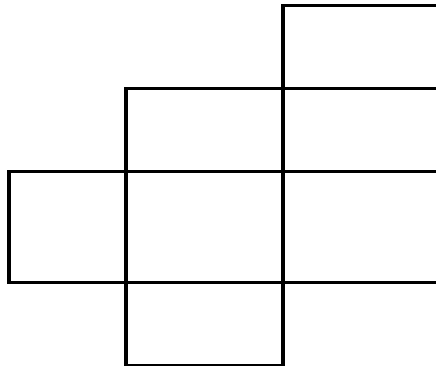
# 分析

- 本质: 是否在图的同一个连通块
- 问题: 图太庞大, 每次还需要遍历
- 解决: 用并查集



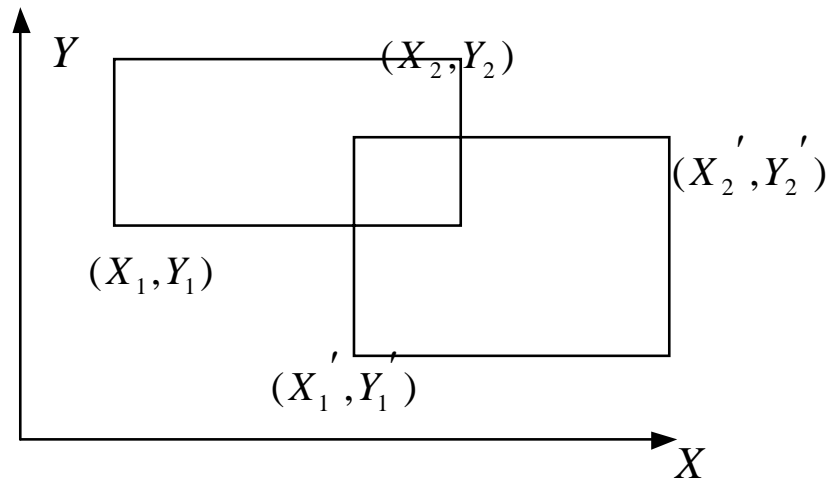
## 例2. 矩形

- 在平面上画了N个长方形，每个长方形的边平行于坐标轴并且顶点坐标为整数。我们用以下方式定义印版：
  - 每个长方形是一个印版；
  - 如果两个印版有公共的边或内部，那么它们组成新的印版，否则这些印版是分离的
- 数出印版的个数. 左图有两个，右图只有一个



# 分析

- 把矩形看作点，有公共边的矩形连边，问题转化为求连通分量的个数
- 判断方法：



## 例3. 代码等式

- 由元素0和1组成的非空的序列称为一个二进制代码。一个代码等式就是形如 $x_1x_2\cdots x_l=y_1y_2\cdots y_r$ , 这里 $x_i$ 和 $y_j$ 是二进制的数字（0或1）或者是一个变量（如英语中的小写字母）
- 每一个变量都是一个有固定长度的二进制代码，它可以在代码等式中取代变量的位置。我们称这个长度为变量的长度
- 对于每一个给出的等式，计算一共有多少组解。
- 例: a,b,c,d,e的长度分别是4,2,4,4,2, 则 $1bad1 = acbe$ 有16组解

# 分析

- 长度为**k**的变量拆成**k**个长度为**1**的变量
- 每位得到一个等式
  - $1=1$ 或者 $0=0$ : 冗余等式
  - $1=0$ 或者 $0=1$ : 无解
  - $a=b$ : **a**和**b**相等 (**a**为变量**b**可以为常数)
- 相等关系用并查集处理, 最后统计集合数为**n**, 答案为 $2^n$ 。

## 例4. 围墙

- 按顺序给出 $M$ 个整点组成的线段，找到最小的 $k$ ，使得前 $k$ 条线段构成了封闭图形。  
（任意两条线段只可能在端点相交）

# 分析

- 将所有出现过的坐标用整数表示，初始时每个独立成树。读入连接**A**和**B**的线段后，将**A**、**B**所在的树和并。如果**A**、**B**在同一棵树，那么就出现了封闭图形（因为**x**个点**x**条边的图必定出现圈）
- 把坐标转换成编号的步骤，可以通过对坐标进行排序，再删除重复。
- 时间:  $O(M\log M)$



## 例5. 可爱的猴子

- 树上挂着 $n$ 只可爱的猴子（ $n \leq 2 \cdot 10^5$ ）。
- 猴子1的尾巴挂在树上
- 每只猴子有两只手，每只手可以抓住最多一只猴子的尾巴，也可以不抓。猴子想抓谁一定抓得到
- 所有猴子都是悬空的，因此如果一旦脱离了树，猴子会立刻掉到地上。
- 第0, 1, ...,  $m$ （ $1 \leq m \leq 400\,000$ ）秒中每一秒都有某个猴子把他的某只手松开，因此常有猴子掉在地上
- 请计算出每个猴子掉到地上的时间

# 分析

- 并查集？
- “时光倒流”
- 如何标记每只猴子的时间？
  - 枚举并查集的元素
  - 需要访问兄弟/儿子？
  - 链表即可
  - 不用指针

## 例6. 奇数偶数

- 你的朋友写下一个由0和1组成的字符串，并告诉你一些信息，即某个连续的子串中1的个数是奇数还是偶数。你的目标是找到尽量小的 $i$ ，使得前 $i+1$ 条不可能同时满足
  - 例如，序列长度为10，信息条数为5
  - 5条信息分别为1 2 even, 3 4 odd, 5 6 even, 1 6 even, 7 10 odd
- 正确答案是3，因为存在序列(0,0,1,0,1,1)满足前3条信息，但是不存在满足前4条的序列

# 分析

- 可以从前缀和s的奇偶性恢复整个序列
  - $a \ b \text{ even}$  等价于  $s[b], s[a-1]$  同奇偶
  - $a \ b \text{ odd}$  等价于  $s[b], s[a-1]$  不同奇偶
- 集合
  - $\text{same}[i]$ : 已知与i在同一个等价类中的元素集合
  - $\text{opp}[i]$ : 已知与i不在同一个等价类中的元素集合
- 初始  $\text{same}[i]=\{i\}$ ,  $\text{opp}[i]=\{\}$

# 分析

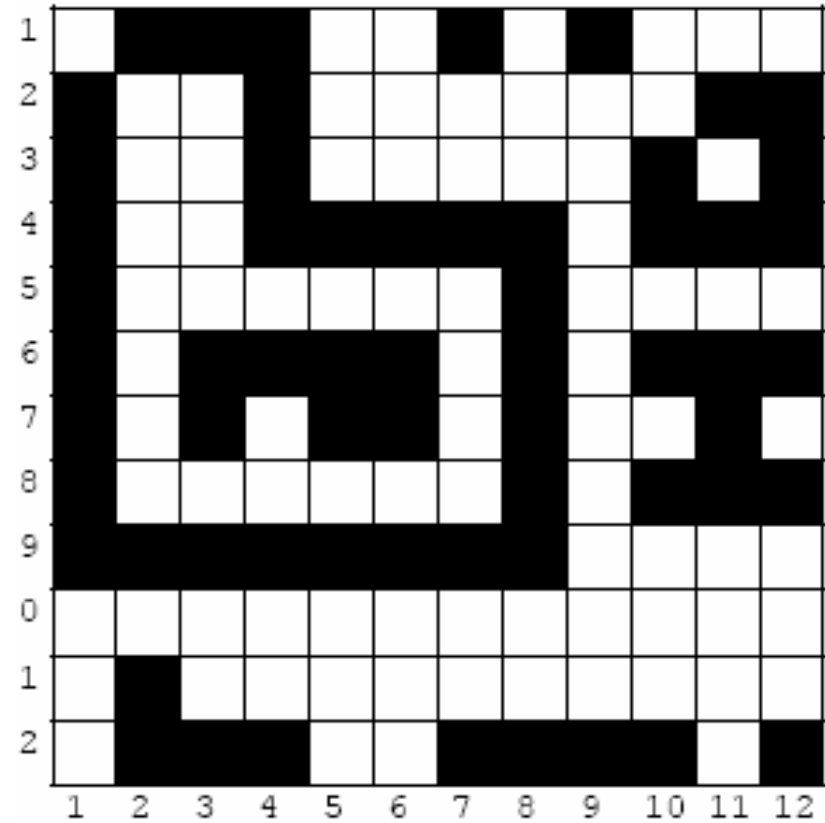
- 则两种条件意味着集合合并
- $s[i]$ 和 $s[j]$ 同奇偶
  - 合并 $same[i]$ 和 $same[j]$
  - 合并 $opp[i]$ 和 $opp[j]$
- $s[i]$ 和 $s[j]$ 不同奇偶
  - 合并 $same[i]$ 和 $opp[j]$
  - 合并 $same[j]$ 和 $opp[i]$

## 例7. 团伙

- 如果两个认识, 那么他们要么是朋友要么是敌人, 规则如下
  - 朋友的朋友是朋友
  - 敌人的敌人是敌人
- 给出一些人的关系(朋友、敌人), 判断一共最多可能有多少个团伙

## 例8. 船

- 给一个01矩阵, 黑色代表船. 右图有一个29吨的, 3个7吨的, 两个4吨的和三个一吨的.
- 输入行数 $N$ ( $<30000$ )和每行的黑色格子(区间数和每个区间)
- 输出每种重量的个数
- 一共不超过1000个船, 每个的重量不超过1000



## 例9. 离线最大值

- 设计一个集合, 初始为空, 每次可以插入一个 $1 \sim n$ 的数( $1 \sim n$ 各恰好被插入一次), 也可以删除最大值, 要求 $m$ 次操作的总时间尽量小.



# 分析

- 在最后加入 $n-m$ 次虚拟的MAX操作, 并记第 $i$ 个MAX操作为 $M_i$ , 记 $M_1$ 之前的插入序列为 $S_1$ ,  $M_{i-1}$  ( $1 < i \leq m$ ) 和 $M_i$ 之间的插入序列为 $S_i$
- 如果 $n$ 在 $S_j$ 中被插入, 则 $M_j$ 的输出一定是 $n$ . 然后删除 $M_j$ , 即把 $S_j$ 合并到 $S_{j+1}$ 中, 然后再查找 $n-1$ 所在的序列 $S_k$ , 则 $M_k$ 的输出为 $n-1$ ...如此下去, 从 $n$ 到 $1$ 依次查找每个数所在序列, 就可以得到它后面的MAX操作的结果, 并把它和紧随其后的序列合并

## 例10. 合并队列

- 初始时 $n$ 个数 $1 \sim n$ 各在单独的一列中, 需要执行两个操作
  - **Move**( $i, j$ ): 把 $i$ 所在列接到 $j$ 所在列的尾部
  - **Check**( $i, j$ ): 询问 $i$ 和 $j$ 是否在同一列, 如果是, 输出二者之间的元素个数

# 分析

- 每个数 $i$ 的 $p[i]$ 表示 $i$ 和 $p[i]$ 在同一队列, 且 $p[i]$ 是 $i$ 之前的第 $d[i]$ 个元素
- 对于队首 $x$ , 有 $p[x]=x$ , 附加变量 $tot[x]$ 表示以 $x$ 为首的队列一共有多少个元素
  - Move需要进行两次查找和一次合并
  - Check需要两次查找
- FIND: 修改 $p[i]$ 时要修改 $d[i]$
- MERGE: 可以启发式合并么???

# 维护前缀和

- 包含 $n$ 个元素的整数数组 $A$ ，每次可以
  - $C(i, j)$ : 修改一个元素 $A[i] = j$
  - $Q(i)$ : 询问前缀 $S_i = A_1 + A_2 + \dots + A_i$ 的值
- 如何设计算法，使得修改和询问操作的时间复杂度尽量低？

# LOWBIT

- 设  $C[i] = a[i - 2^k + 1] + \dots + a[i]$ , 其中  $k$  为  $i$  在二进制下末尾0的个数, 令  $LOWBIT(i) = 2^k$ 
  - 例如,  $i = 1001010110010000$ , 则  $k = 4$
- 不难得到LOWBIT公式
  - $LOWBIT(x) = x \text{ and } (x \text{ xor } (x - 1))$

```
inline int lowbit(int x)
{
    return x & (x ^ ( x - 1 ) );
}
```

# 修改A[x]的后果

- 修改A[x], 可能有很多C随之修改
- 例如 $x=76=(1001010)_2$ , 可以得到:

$p_1 = 1001010$

$p_2 = 1001100$

$p_3 = 1010000$

$p_4 = 1100000$

$p_5 = 10000000$

$P_1 = x$

$P_{i+1} = P_i + \text{LOWBIT}(P_i)$

如何证明 $P_i$ 和 $P_{i+1}$ 之间的C值都不改变?

- 则需要依次修改 $C[p_1], C[p_2], \dots$

## 前缀和 $A[1]+\dots+A[x]$ 的计算

- 首先累加 $C[x]$ , 因为它的定义是以 $x$ 结尾的连续和. 它的连加起点是 $C[i-\text{LOWBIT}(i)+1]$ , 因此问题转化为了求 $A[1]+\dots+A[i-\text{LOWBIT}(i)]$
- 由此, 我们得到递推式

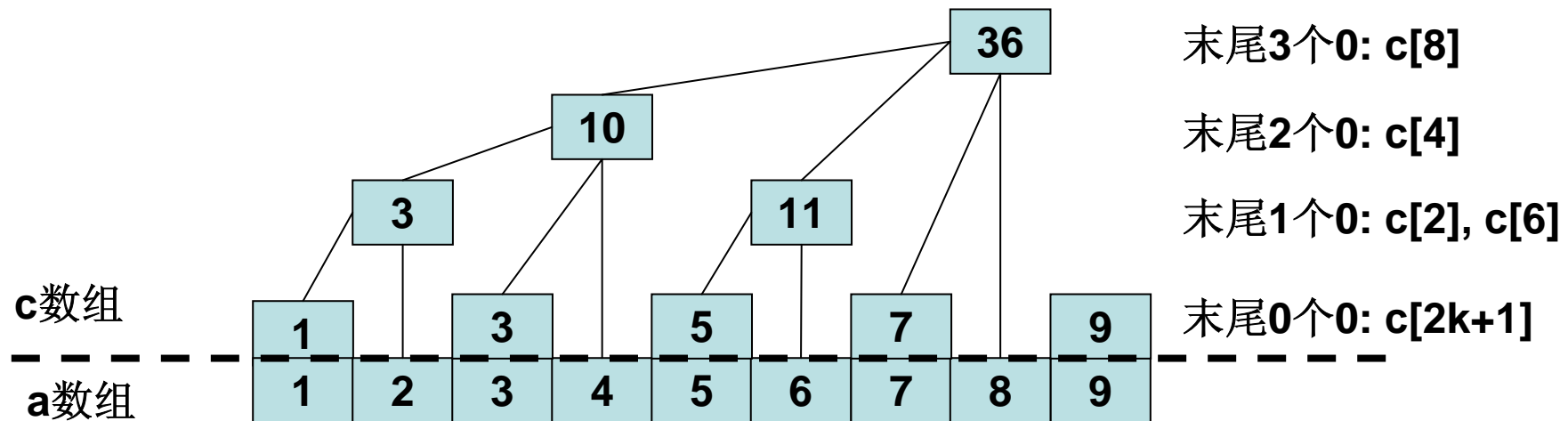
$$P_1=x$$

$$P_{i+1}=P_i-\text{LOWBIT}(P_i)$$

- 则只需要累加 $C[p_1], C[p_2], \dots$

# C的分层树状结构

- 数组  $c[i] = a[i-2^k+1] + a[i-2^k+2] + \dots + a[i]$ 
  - $k$  为  $i$  在二进制形式下末尾0的个数
  - 起点是把  $i$  的最后一个1变为0再加1
- $c$  数组的分层表示和递推关系如下图

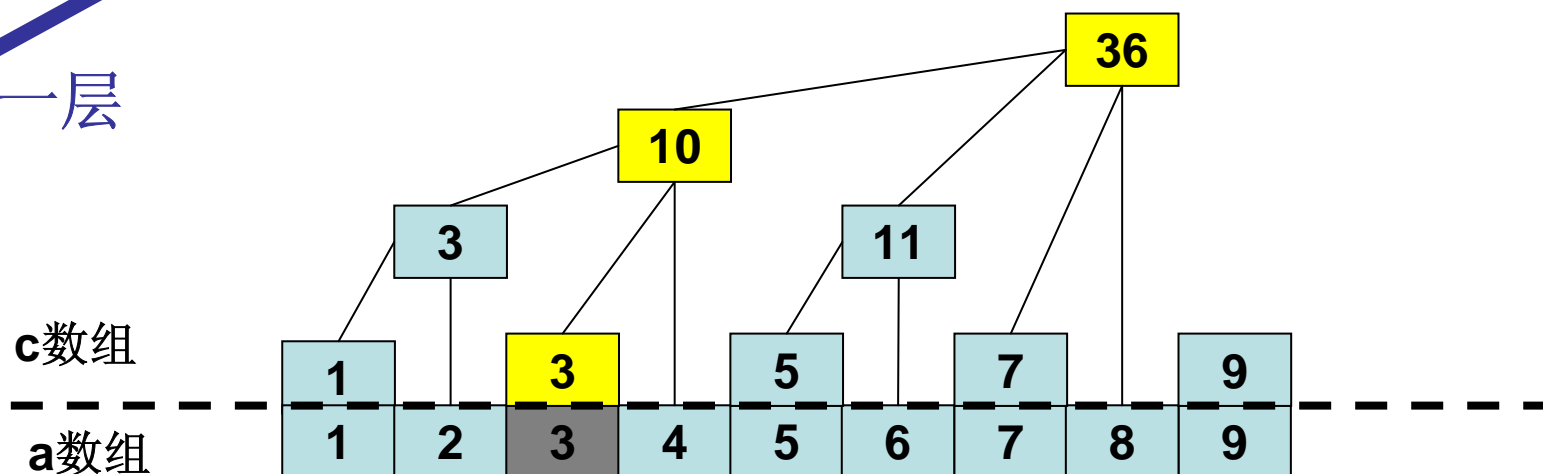


第  $i$  层末尾有  $i$  个零，度数为  $i+1$ ，定义式  $2^i$  项



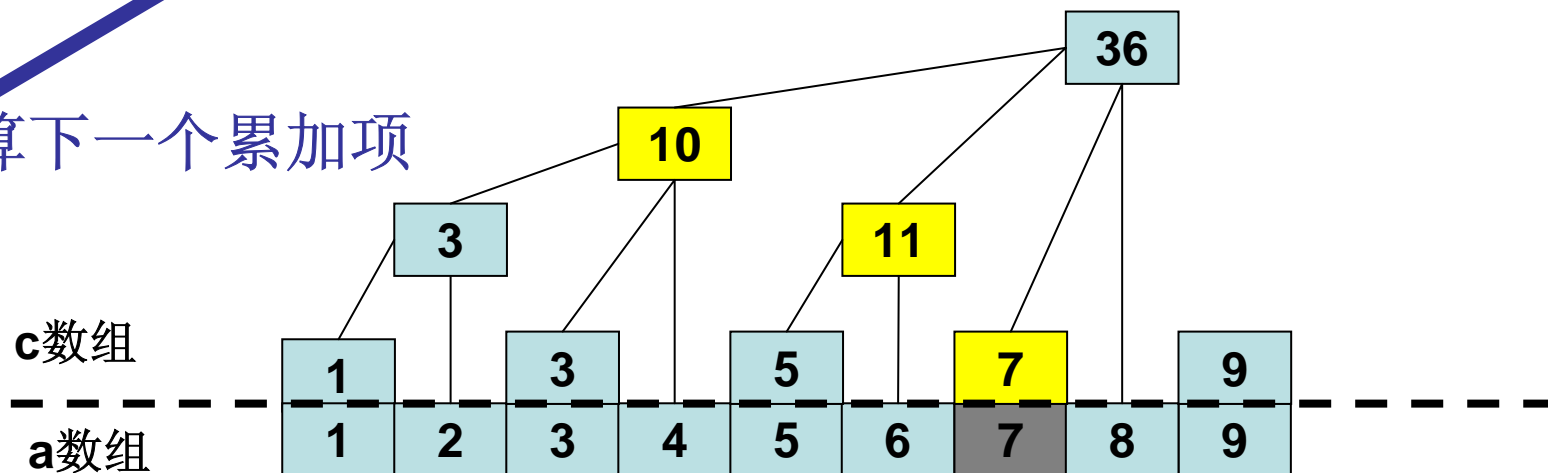
```
void Add(int p , int d){  
    while(p <= n){  
        C[p] += d;  
        p += lowbit(p);  
    }  
}
```

提升一层



```
int Sum(int p){  
    int ret = 0;  
    while( p ){  
        ret += C[p];  
        p -= lowbit(p);  
    }  
    return ret;  
}
```

计算下一个累加项

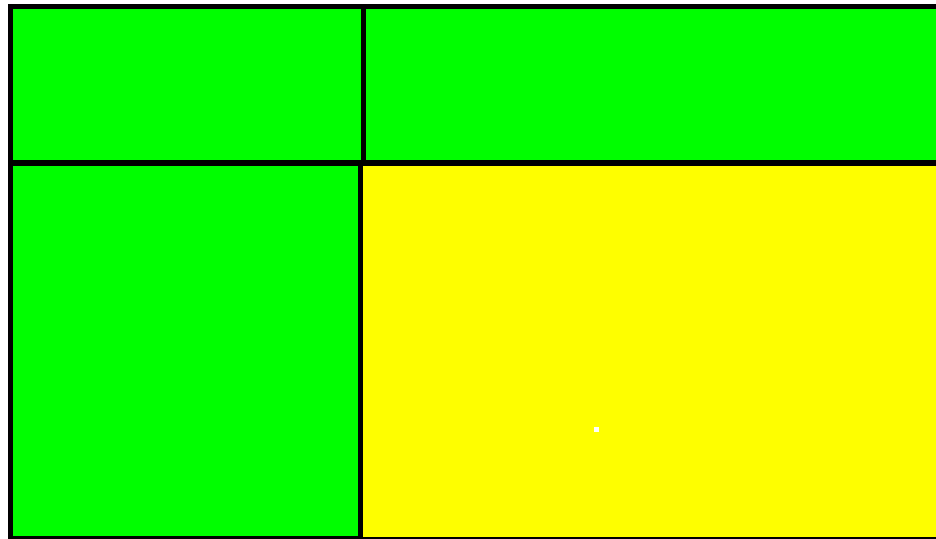


# 例1. 手机

- Tampere地区被划分成 $N*N$ 个单元，形成一个 $N*N$ 的表格，行列坐标均为0到 $N-1$ ，每个单元中有一个移动电话信号发射基地
  - $C(X,Y,A)$ : 基地 $(X,Y)$ 移动电话数的变化值为 $A$
  - $Q(L,T,R,B)$ : 询问矩形区域内移动电话总数
- 注意C操作中 $A$ 可正可负

# 分析

- 任意矩形转化为四个前缀矩形
- 转化为二维前缀和
- 二维独立, 分别处理, 每次操作 $\log^2 n$



## 例2. 01矩阵

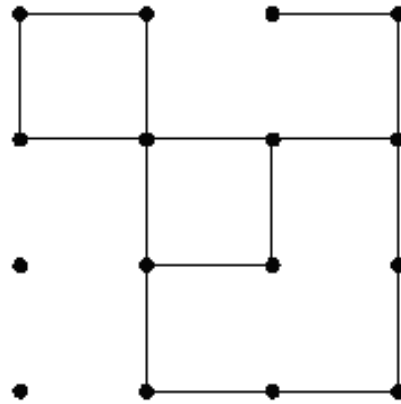
- 给 $n*n$ 的01矩阵,支持
  - $C(x_0, y_0, x_1, y_1)$ : 改变矩形（每个元素取反）
  - $Q(x, y)$ : 查询 $(x, y)$ 的值

# 分析

- 构造辅助01矩阵 $C'$ ，初始为0
- 矩形分解:  $C(x_0, y_0, x_1, y_1)$ 等价于改变以下4点的值
  - $C'(x_0, y_0)$ ,  $C'(x_0, y_1)$ ,  $C'(x_1, y_0)$ ,  $C'(x_1, y_1)$
- 元素 $(x, y)$ 的最终值完全取决于在 $C'$ 中 $(x, y)$ 的右下方的元素和的奇偶性
- 维护二维前缀和→二维数状数组

### 例3. 方格问题

- 在一个 $N*N$ 格点方阵的给了一些长度为1的线段, 每条线段的格式为 $(X, Y, D)$ , 其中 $D$ 可以是H (往右) 或者V (往下)



- 计算出一共有多少个正方形。上图共有3个正方形, 两个边长为1, 一个边长为2

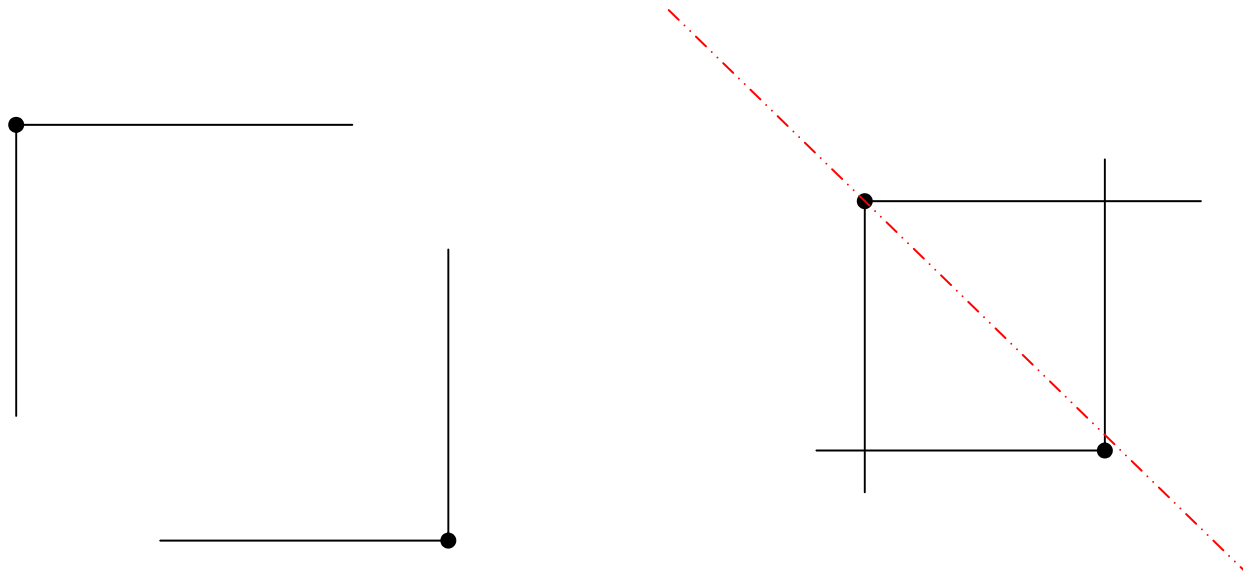
# 分析

- 用一个01矩阵表示每条元线段是否存在
- 算法一：然后枚举左上角顶点和边长，检查每条元线段是否都存在。时间复杂度 $O(n^4)$
- 算法二：预处理计算出每个点往下，往右可以延伸多长，判断整条边降为 $O(1)$ ，总时间降为 $O(n^3)$
- 设左上角为 $(x, y)$ ，若存在边长为 $K$ 的正方形
  - 不一定存在 $K-1$ 的正方形
  - 但是我们至少可以确定， $K-1$ 正方形的其中两条边是必然存在的



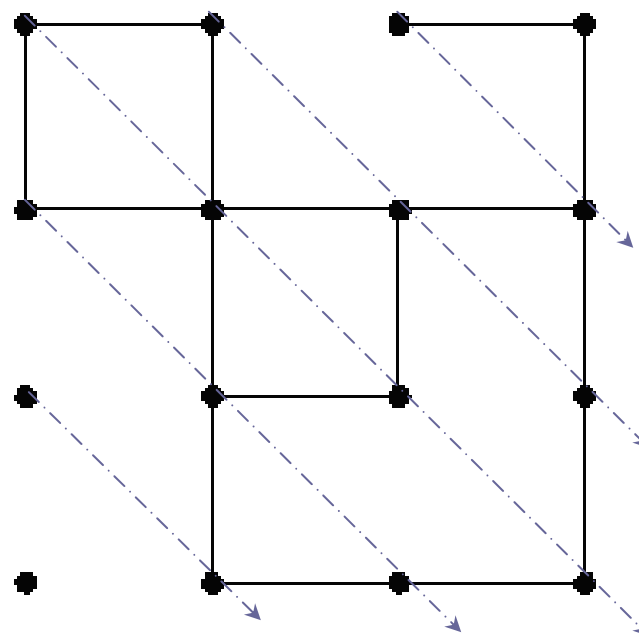
# 充要条件

- 把一个正方形拆成左上和右下两部分, 则两个部分可以构成一个正方形的充要条件是
  - 两个顶点在同一条倾斜**135度**的直线上
  - 右下部分的两条边与左上部分两条边有交点



# 算法

- 依次处理各条斜线, 每条线自左上到右下依次考虑各个点
  - 累加新点往左上方向“作用范围”内的点数
  - 删除往右下方方向已经延伸不到当前位置的点
- 用树状数组维护连续和,  $O(n^2 \log n)$



## 例4. 队伍选择

- IOI要来了，BP队要选择最好的选手去参加。为了选出的选手是最好的，教练组织了三次竞赛并给出每次竞赛排名。所有N名选手都参加了每次竞赛并且每次竞赛都没有并列的。当A在所有竞赛中名次都比B前，我们就说A是比B更好。如果没有人比A更好，我们就说A是优秀的。
- 求：优秀选手的个数

# 分析

- **朴素算法：**依次判断每个选手*i*是否优秀。判断的方法是枚举所有其他选手*j*，看是否*j*比*i*更好。时间复杂度为 $O(n^2)$
- 三次比赛很容易降低为两次：按第一次比赛的排名重新给选手编号（第一名为1号...），则*i*比*j*好当且仅当*i*<*j*且 $a[i]<a[j]$ 且 $b[i]<b[j]$ ，其中*a*和*b*分别是第二次和第三次的比赛名次

# 分析

- 在平面中依次插入各个 $(a[i], b[i])$ ，每次计算 $x < a[i]$ 的点中 $y$ 的最小值 $\min$ ，则 $i$ 是优秀的当且仅当 $\min$ 比 $b[i]$ 大
- 需要维护前缀最小值 $\min(x[1 \dots i])$ 。由于点只插入不删除，所以**所有前缀最小值都不会变大**，树状数组的框架仍然适用!
- 时间复杂度： $O(n \log n)$