
数据结构
Data structures


堆
Heap

堆

- 二叉堆是一个基础的数据结构
- 它满足父结点的键值总是大于等于（或小于等于）子节点的键值。

- 堆的可以支持插入，删除或查找最大（或最小）键值的点。
- 且操作的时间复杂度为log级别。

- 基础例题：合并果子

简单应用

- 计算哈夫曼树
- 数据维护
- ……

NOI2015 荷马史诗

一部《荷马史诗》中有 n 种不同的单词，从 1 到 n 进行编号。其中第 i 种单词出现的总次数为 w_i。Allison 想要用 k 进制串 s_i 来替换第 i 种单词，使得其满足如下要求:

对于任意的 $1 \leq i, j \leq n$，$i \neq j$，都有：s_i 不是 s_j 的前缀。

现在 Allison 想要知道，如何选择 s_i，才能使替换以后得到的新的《荷马史诗》长度最小。在确保总长度最小的情况下，Allison 还想知道最长的 s_i 的最短长度是多少？

一个字符串被称为 k 进制字符串，当且仅当它的每个字符是 0 到 $k-1$ 之间（包括 0 和 $k-1$）的整数。

字符串 Str1 被称为字符串 Str2 的前缀，当且仅当：存在 $1 \leq t \leq m$，使得 $\text{Str1} = \text{Str2}[1..t]$。其中，$m$ 是字符串 Str2 的长度，$\text{Str2}[1..t]$ 表示 Str2 的前 t 个字符组成的字符串。

- n<=100000,k<=9
- 先考虑k=2怎么做
- 题目大意，给你n个数，就像合并果子一样，不过每次可以选k个合在一起，使得最后的贡献最小。

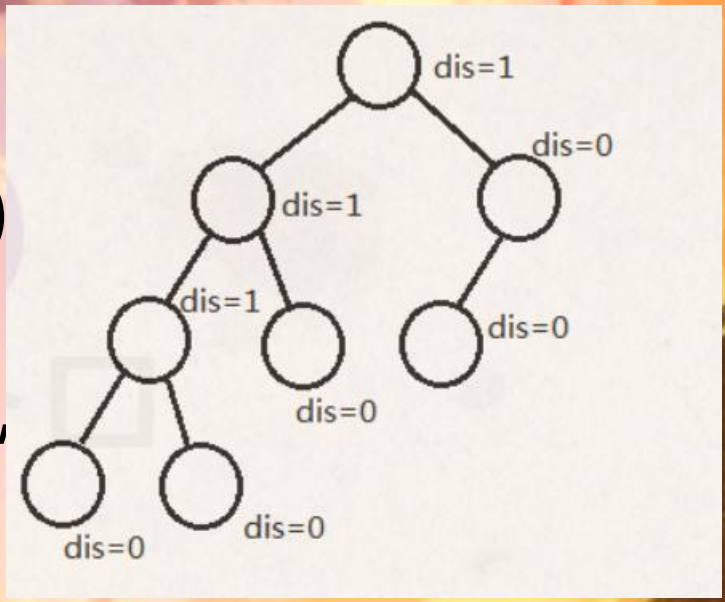
- 如果k=2，就是一个合并果子
- 将题目意思转化为：构造n个k进制字符串，第i个字符串有一个权值w[i]，长度为l[i]，要求输出∑l[i]*w[i]的最小值。
- 仔细想想，其实求解的过程也是一个k叉哈夫曼树的过程，每次只需要选择最小的k个进行合并。

- 合并到最后不到k个怎么办？
- 考虑哈夫曼树的性质，肯定先满足深度低的层满点后再往深的层加点，所以我们第一次合并的时候调整合并个数使得以后的合并都满足每次都能合并k个。

- 合并过程中的查找最小值，删除最小值，插入值用堆维护即可。

可并堆

- 可并堆，就是可以进行合并的堆，最常见的是左偏树。
- 它对于堆中每个节点维护一个dis，表示一直往右儿
- 子走走到叶子的步数。
- 左偏树需满足dis(left(x))<=dis(right(x))+1，dis(right(x))
- 容易知道满足这个性质树高就是log n级别的。
- 因为可以合并，所以插入不需要了，删除也容易了，
- 合并变成了基础操作。
- 接下来讲解可并堆如何实现合并操作。



可并堆的合并

- 现在合并x与y，先比较权值看x与y谁该当根。
- 假设x当根，那么递归合并x的右子树与y。
- 然后如果x的右子树的dis大于左子树的dis，我们交换x的左右子树。
- 接着更新一发x的dis，即dis(x)=dis(right(x))+1
- 有了合并就可以闯天下了。

合并操作的代码

```
int merge(int x,int y)
{
        if (!x||!y) return x+y;
        pushdown(x);
        pushdown(y);
        if (val[x]<val[y]) swap(x,y);
        int z=newnode(x,1);
        r[z]=merge(r[z],y);
        if (dep[l[z]]<dep[r[z]]) swap(l[z],r[z]);
        dep[z]=dep[r[z]]+1;
        return z;
}
```

APIO2012 派遣

给一颗N个点的树，每个点有权值c[i],l[i],现在需要你找一个节点j，选出一些它子树中的点满足c的总和小于M，得到A=l[j]*选的点数，求最大的A。

N<=100000

- 我们把i的子树及自己的元素全部排一下序，然后累加，去找最多的元素，sumc<=M,那么答案即为 Li*元素个数
- 那我们就可以枚举所有的点，每次都做一下这个过程，取max
- 如何快速从子树中得到信息呢？ 对于这个排序，我们可以用可合并堆来搞. 每次把当前点堆与儿子堆合并，维护大根堆，然后pop最大元素，直到sumc<=m 这时ans=l[i]*堆的大小。
- 时间复杂度o(n log n)


LCA

LCA

- LCA中文全名最近公共祖先，指树上两个节点相同祖先中深度最深的点。
- 快速求LCA是解决树上问题的一大基础

Tarjan算法求LCA

- 只要是支持离线的，就可以用tarjan算法，把询问分别挂在两个端点上。
- solve(x)表示解决x子树内所有询问，先递归子树。
- 我们维护并查集，希望对一个节点x进行getfather可以找到当前x的最高祖先y的父亲满足y的子树全被访问过（即满足子树全被访问过的最高祖先y，我们要得到的y的父亲）。那么每递归完x一个儿子y，令fa[y]=x。
- 递归完子树后，我们来尝试解决挂在x上的所有询问，假设另外一点是y，我们分两种情况：
- 1、y未被访问过，那么该询问不管
- 2、y已被访问过，那么我们直接对ygetfather，就是lca了。
- 正确性显然，这个方法是线性的。

倍增在线求LCA

- 对于树上的每个点我们先预处理出它们的深度，然后我们就可以轻易想到一种暴力求解的方法：
- 对于要求公共祖先的两个点u，v，每次让深度深的点往父亲方向走直到第一次u=v，即是最近公共祖先了。
- 但是若树是一条倒v的链，每次询问都可能要o（n）的复杂度，
- 有没有更快的方法呢？

倍增在线求LCA

- 我们处理出一个数组f,f[i][j]表示节点i向上走2^j步到达哪个点，这样我们就不用一步一步走了。
- 我们先利用处理好的f，将u和v上升
- 到同一深度，然后一起以2^i的步伐
- 往上走，直到u=v
- 预处理复杂度o(n log n)
- 单次询问复杂度o(log n)

```
int up(int x,int b)
{
	fd(i,20,0)
		if ((1<<i)<=b)
		{
			x=f[x][i];
			b-=1<<i;
		}
	return x;
}

int getlca(int x,int y)
{
	if (dep[x]>dep[y]) x=up(x,dep[x]-dep[y]); else y=up(y,dep[y]-dep[x]);
	if (x==y) return x;
	fd(i,20,0)
		if (f[x][i]!=f[y][i])
		{
			x=f[x][i];
			y=f[y][i];
		}
	return f[y][0];
}
```

模板题

- 给一棵大小为n的树，树边有边权，m次询问每次询问两点的路径上边的最大值
- 1<=n<=50000， 1<=m<=75000， 0<=边权<=1000

题解

- 在LCA的基础上记录s[i][j]表示从节点i往上走2^j步路上的边最大值是多少，在找LCA的过程中我们就可以跟着得到最大值的答案。

进阶
POJ3728The merchant

- 给你一颗n个点有点权的树，q次询问，每次询问从u走到v路上，买卖一次东西所赚的最大差价为多少。

- N,q<=100000

- 把u到lca的路径叫左链，v到lca的路径叫右链。
- 最大差值要么在左链，要么在右链，要么穿过最近公共祖先。所以我们可以用倍增维护最大值，最小值，从上到下的最大差值，从下到上的最大差值。对于询问走一遍两点间的路径，答案为左链最大差值，右链最大差值，和右链最大值减左链最小值三者最大值。


并查集

并查集

- 并查集是一种树型的数据结构，用于处理一些不相交集合（Disjoint Sets）的合并及查询问题。常常在使用中以森林来表示。
- 集就是让每个元素构成一个单元素的集合，也就是按一定顺序将属于同一组的元素所在的集合合并。
- 并查集有link和find，如果朴素的话，并查集容易形成一条链，那样就会复杂度爆炸。
- 我们来学习并查集的两个优化：按秩合并与路径压缩。

只进行按秩合并

- 按秩合并，就是给每个节点一个秩，rank[x]表示x的秩。秩的定义其实就是树高。
- 合并两个集合时，我们把秩小的合并到秩大的里去，若两个集合秩相同就可以更新秩。这样子，并查集的树高是log n的。
- 我们用归纳法证明，秩为i的至多只有2^i个节点。
- 秩为0，只有一个节点，显然成立。
- 若秩为n时成立，至多有2^n个节点，那么一颗秩为n+1的并查集需要两个秩为n的并查集合并，至多有2^(n+1)个节点。
- 所以有n个节点的并查集秩至多为log n，树高就是log n的。

只进行路径压缩

- 路径压缩，每次find后，将被访问点到根节点路径上所有节点的父亲设为根节点。
- 路径压缩的复杂度是均摊log n的，由于证明比较复杂而且对解题没有帮助，在这里就不给大家证明了，有兴趣的同学可以自行上网搜索相关证明。

TJOI&HEOI2016树

- 一颗树，除根节点外初始都是白点，根节点是黑点。
- 每次染黑一个结点或者询问一个结点的最近黑色祖先。

- n<=10^5

- 我们发现没有强制我们在线……那就倒过来做，那么每次是染白一个节点。
- 我们设fa[i]表示从i出发往上最近的黑祖先。
- 染白一个点就把fa[x]设为x的父亲。
- 然后这个fa可以并查集维护

冷战

- 给定一副 N 个点的图。动态的往图中加边，并且询问某两个点最早什么时候联通。

- 强制在线

- N<=500000

- 考虑并查集。并查集实际上维护了一棵树。那么假如我们按秩合并，
- 这棵树的深度是 O(log n) 的。我们将一个点连向其父亲的边权设为这条边
- 加入的时间，那么每次询问时，暴力查询树上从 u 到 v 所经过边权的最大
- 值即可。时间复杂度为 O(n log n)，常数较小。

SCOI2016萌萌哒

- 有一个无前导0的n位数，有m个限制形如[l1,r1]=[l2,r2]，问满足条件的数有多少种，答案模10^9+7。

- N<=100000,m<=100000

倍增并查集

- 我们用ST表，f[i][j]表示[i][i+2^j-1]这一段。
- 那么初始时每一段单独成一个集合。
- 对于一个限制可以拆成log 份，然后进行集合合并。
- 然后呢，如果任意f[s][t]和f[i][j]属于同一集合，那么f[s][t-1]与f[i][j-1]以及f[s+2^(t-1)-1][t-1]和f[i+2^(j-1)-1][j-1]都应该属于同一集合。
- 为了满足这个限制，只要最后再一层一层的做，把下一层的合并了即可。
- 合并注意必须让编号大的合进编号小的里。
- 统计答案就很简单啦，答案是9*10^(集合个数-1)

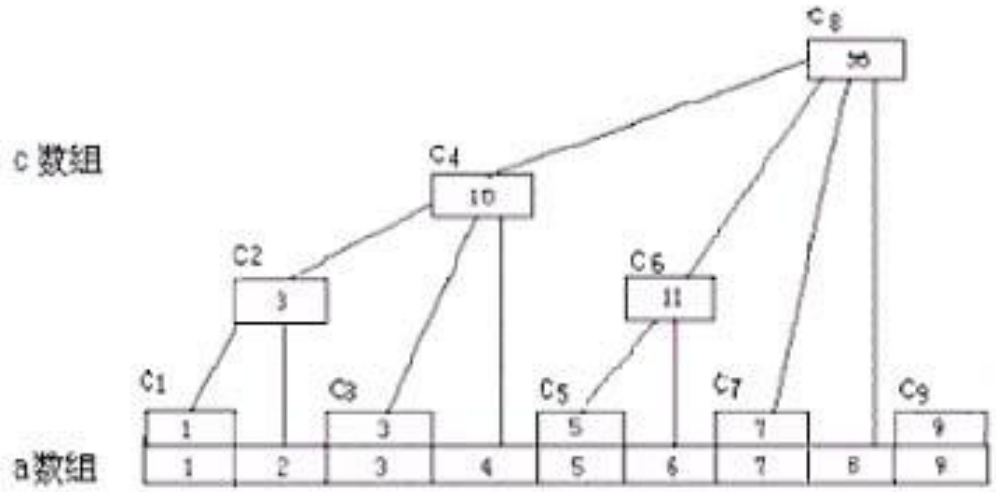
树状数组

树状数组

- 树状数组是一种常数小，空间o(n)的数据结构，支持区间和查询与单调修改。
- 首先你要知道什么是lowbit函数。
- lowbit(x)表示将x拆成二进制后，找到从低到高x的第一个1，然后后面部分就是x的lowbit。比如二进制是10101110000，那么lowbit是10000化为十进制数是16。
- 树状数组保留了C数组，其中Ci=A[i-lowbit(i)+1]+……+A[i]
- lowbit怎么求？一种简便方法是lowbit(x)=x&-x
- -x的二进制其实是把x的二进制取反然后+1。
- 例如10101110000，取反是1……1 0101010000
- 你发现lowbit部分取反后变成01……1，然后+1就变成10……0，于是和原来一致。
- 而lowbit前的部分，自然就和原来相反，and的值为0。

树状数组

- 看图吧。
- 然后我们发现，要求1~i的
- 前缀和，我们先加上C[i]，也就是
- 加了A[i-lowbit(i)+1]~A[i]，那么我们还要
- 求1~i-lowbit(i)的前缀和，继续处理子
- 问题即可。这就是查询啦，因为
- lowbit一直在减小，所以是log n。
- 修改呢？可以发现修改i，首先Ci要修
- 改，其次i+lowbit(i)的lowbit一定比i大，
- 所以也要修改C[i+lowbit(i)]，同理还要修改
- C[i+lowbit(i)+lowbit(i+lowbit(i))……]
- 一直往上走，就是修改啦。因为lowbit一
- 直增加，所以也是log n的。



树状数组的拓展

- 事实上，树状数组存在许多拓展。
- 比如最值维护，只要只需要查询前缀最值，且修改只涉及取最值操作，也是可以做的。
- 而树状数组可以区间修改，这里不进行讨论。
- 树状数组可以和主席树搭配使用，超出noip范围，也不讨论。

基础题

- 已知一个数列，你需要进行下面两种操作：
- 1.将某一个数加上x
- 2.求出某区间每一个数的和

- N<=100000

树状数组部分

```
int lowbit(int x){return x&-x;}
ll sum(int i){  //区间询问
        ll sum = 0;
        while(i){
                sum += d[i];
                i -= lowbit(i);
        }
        return sum;
}

void add(int i,int x){    //构造线段树,在第i个位置上加x
        while(i<=n){
                d[i] += x;
                i += lowbit(i);
        }
}
```

主代码部分

```
while(m--){
        int flag,x,y;
        scanf("%d%d%d",&flag,&x,&y);
        if(flag == 1)   add(x,y);
        else printf("%lld\n",sum(y) - sum(x-1));
```

加强版

- 给你N个数，有两种操作：
- 1：给区间[a,b]的所有数增加X
- 2：询问区间[a,b]的数的和。

- N<=100000

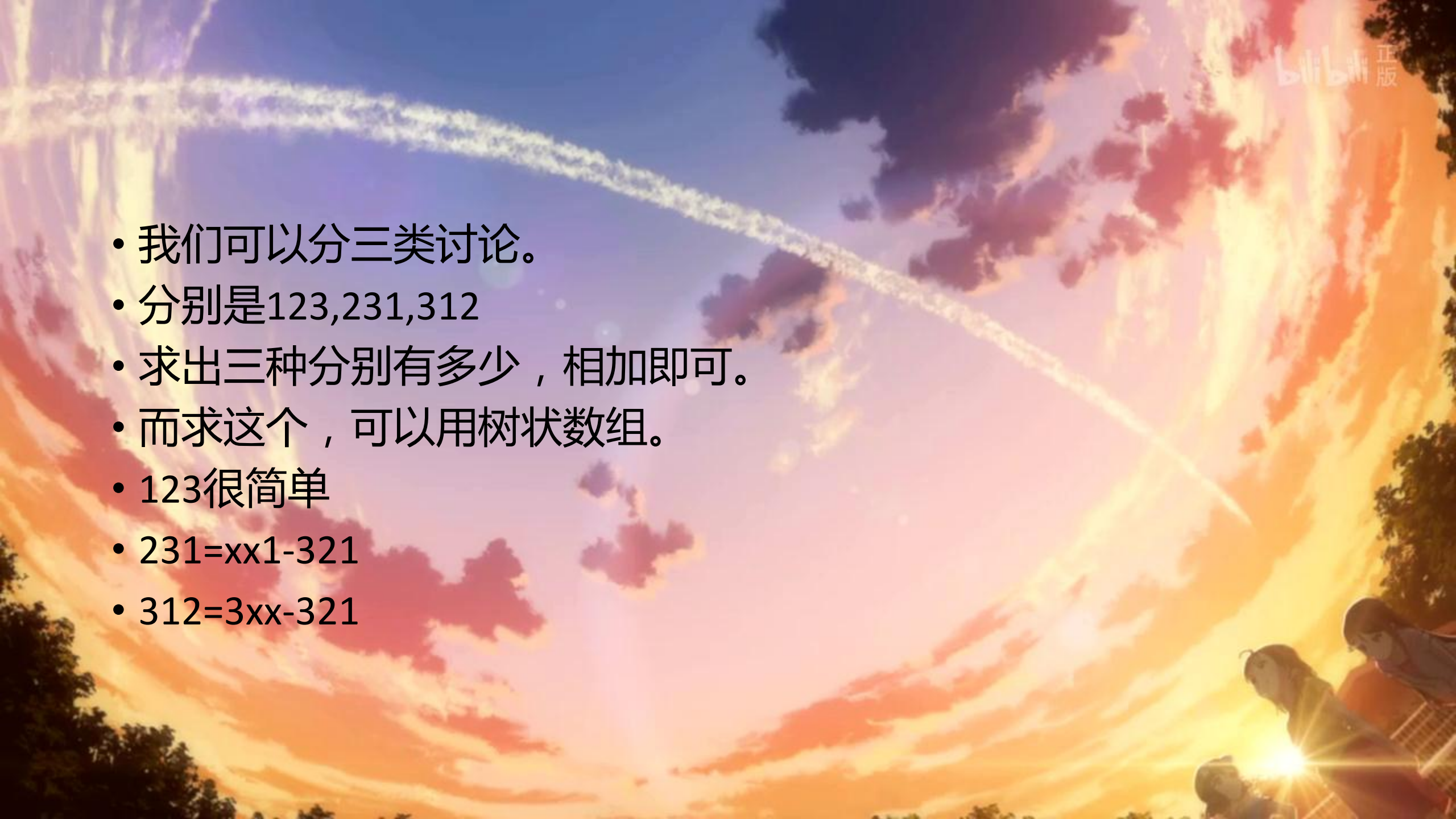
- 令c[i]=a[i]-a[i-1]
- 则a[1]+a[2]+a[3]+..+a[n]=c[1]+(c[1]+c[2])+(c[1]+c[2]+c[3])+……+(c[1]+c[2]+……+c[n]) = n*(c[1]+c[2]+……+c[n])-(0*c[1]+1*c[2]+2*c[3]+……+(n-1)*c[n])
- 用树状数组维护c[1]+c[2]+…+c[n],和0*c[1]+1*c[2]+2*c[3]…即可

朗格拉日计数

在平面上以圆周等分排列着 N 个带标号 (标号为 $1..N$) 的点，你需要计算有多少个三元组 (a,b,c) 其中满足 $a<b<c$, 而且标号为 a,b,c 的点在圆上分布的顺序为顺时针顺序。(分布顺序为 a,b,c 的意思是顺时针表示，从标号为 a 点出发，顺时针在圆上遍历一圈，标号为 b 的点先遍历到，标号为 c 的点后遍历到)(所以分布顺序可以说只有两种一种 a,b,c 一种 a,c,b)

- 其实问题要求，有多少个三元组(i,j,k)满足ai<aj<ak,且i,j,k在圆上是顺时针排列

- 题目不难，
- 求一个最优解法
- 常数最小。
- N<=200000


• 我们可以分三类讨论。
• 分别是123,231,312
• 求出三种分别有多少，相加即可。
• 而求这个，可以用树状数组。
• 123很简单
• 231=xx1-321
• 312=3xx-321


线段树
Segment tree

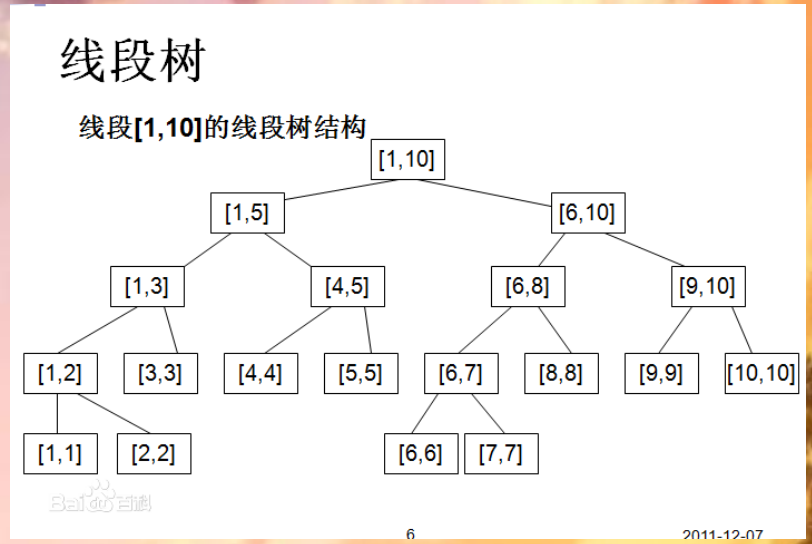
基本概念

- 线段树是一种二叉搜索树，与区间树相似，它将一个区间划分成一些单元区间，每个单元区间对应线段树中的一个叶结点。 [1]
- 对于线段树中的每一个非叶子节点[a,b]，它的左儿子表示的区间为[a,(a+b)/2]，右儿子表示的区间为[(a+b)/2+1,b]。因此线段树是平衡二叉树，最后的子节点数目为N，即整个线段区间的长度。
- 使用线段树可以快速的查找某一个节点在若干条线段中出现的次数，时间复杂度为O(logN）。而未优化的空间复杂度为2N，因此有时需要离散化让空间压缩。

举例说明

- 右图是线段树的一个直观形式，每一个节点代表一个区间的信
- 息，当我们要查询区间信息时只需要拿出logn级别个数的点合
- 并就可以了，例如[4,9]=[4,5]+[6,8]+[9,9]。

- 以树状数组的例题来说，每个节点我们存区间和，
- 修改的时候我们自顶向下修改相关的每一个节点，
- 查询时找出对应区间的节点将他们的和加起来。
- 每次操作均为logn时间复杂度



构建，修改与查询的代码

```
void build(int pos,int x,int y)
{
    if (x==y)
    {
        sum[pos]=a[x];
        return;
    }
    int mid=(x+y)/2;
    build(pos+pos,x,mid);
    build(pos+pos+1,mid+1,y);
    sum[pos]=sum[pos+pos]+sum[pos+pos+1];
}
```

```
void modify(int pos,int x,int y,int w)
{
    if (x==y)
    {
        sum[pos]=a[y];
        return;
    }
    int mid=(x+y)/2;
    if (w<=mid) modify(pos+pos,x,mid,w);
    else modify(pos+pos+1,mid+1,y,w);
    sum[pos]=sum[pos+pos]+sum[pos+pos+1];
}
```

```
int query(int pos,int x,int y,int xx,int yy)
{
    if (x==xx&&y==yy) return sum[pos];
    int mid=(x+y)/2;
    if (yy<=mid) return query(pos+pos,x,mid,xx,yy);
    else
    if (xx>mid) return query(pos+pos+1,mid+1,y,xx,yy);
    else return query(pos+pos,x,mid,xx,mid)+query(pos+pos+1,mid+1,y,mid+1,yy);
}
```

进阶

- 对于区间修改，涉及到多个线段树节点的改动，如果每次操作都改所有相关节点，则时间复杂度就会爆炸。
- 这里引入懒标记，可以使时间复杂度减低到logn级别。

例题

- 给你N个数，有Q次操作，有两种操作：
- 1：给区间[a,b]的所有数增加X。
- 2：询问区间[a,b]的数最大的数是多少。

- N,Q<=100000

题解

- 显然线段树节点记录该区间的最大值，查询的时候就找到相应的节点，取最大值就可以了。
- 但是修改呢？不可能每个点都修改，这样时间复杂度就爆炸了。

- 考虑到每次询问线段树节点时都是自上而下走，所以我们修改时并不需要每次都改完相应的节点，只需要找到对应区间的节点修改并打上标记，在以后的访问中要用到这个节点的信息时再修改，这样就能保证每次修改的时间复杂度为logn

- 每次进入先处理标记
- 修改时打标记但不下传

```
void modify(int pos,int x,int y,int xx,int yy,int add)
{
    push_down(pos);
    if (x==xx&&y==yy)
    {
        mx[pos]+=add;
        if (x!=y)
        {
            tag[pos+pos]+=add;
            tag[pos+pos+1]+=add;
        }
        return;
    }
    int mid=(x+y)/2;
    if (yy<=mid) modify(pos+pos,x,mid,xx,yy);
    else
    if (xx>mid) modify(pos+pos+1,mid+1,y,xx,yy);
    else
    {
        modify(pos+pos,x,mid,xx,mid);
        modify(pos+pos+1,mid+1,y,mid+1,yy);
    }
    mx[pos]=max(mx[pos+pos],mx[pos+pos+1]);
}
```

bzoj3211花神游历各国

- 给你N个数，有Q次操作，有两种操作
  1：区间[a,b] 的所有数开方下取整。
-
  2：询问区间[a,b]的和是多少。

- **n,q<=1e5,Ai<=1e9**

题解

- 开方？一个数最多被开几次？
- 于是我们可以暴力开方，但注意1和0开方之后会变成自己不能保证复杂度
- 于是线段树维护一个标记表示这个区间里是否全都是1/0,如果是就不递归下去操作
- 本题还可以支持区间加法操作，较难，如有兴趣可以自己思考。

公路建设

- 给出一张n个点，m条边的无向图，每条边有边权.
- 给出q次询问，每次询问一个编号区间[l,r]的边所形成的最小生成树（森林）的边权和

- N<=100,M<=10^5,Q<=15000

题解

- 我们可以用线段树维护每一个区间的最小生成树
- 注意到n只有100，可以直接把最小生成树的边开一个数组按边权从小到大存下来
- 合并是直接暴力O(n)归并，用并查集得到新的最小生成树
- 总复杂度O(qn log m)

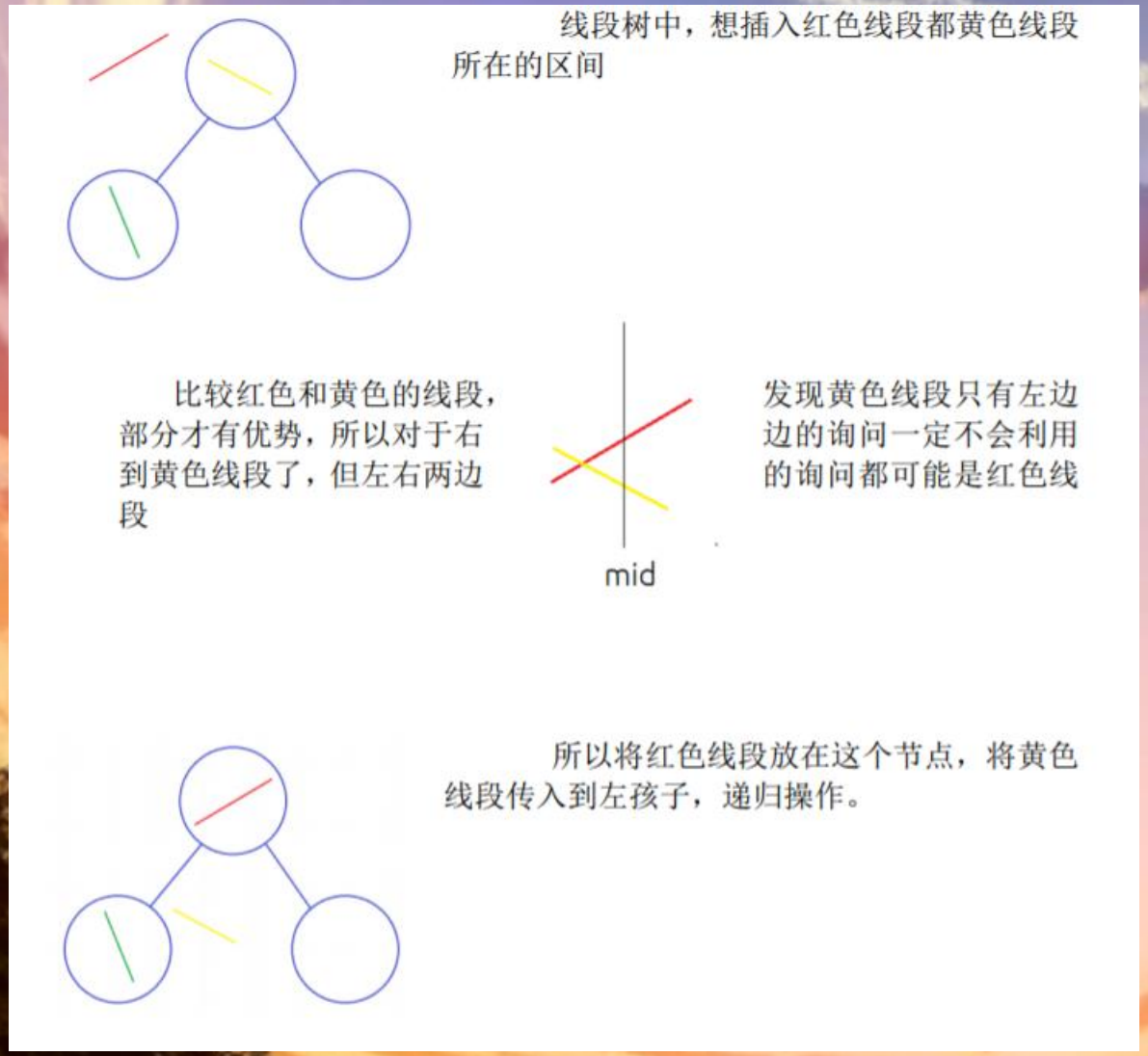
查询

- 给出若干条线段，用(x1,y1)，(x2,y2)表示其两端点坐标，现在要求支持两种操作：
  0 x1 y1 x2 y2
  表示加入一条新的线段，(x1,y1)-(x2,y2)
  1 x0
  询问所有线段中，x坐标在x0处的最高点的y坐标是什么，如果对应位置没有线段，则输出0。
- 原先有n条线段，m次操作。
- n<=50000,m<=150000,保证坐标在[-1e6,1e6]范围内且都为整数

题解

- 我们发现由于询问是整数，所以实际上最终只会询问 1e5 个位置，我们对于 这些位置建一颗线段树。 线段树一个节点为一个容器，可以存放一条题目中的线段。考虑插入一条线 段时，即在线段树上区间插入，当找到要插入的区间节点是，如果这个节点的容 器中没有东西，则直接将线段放在这个容器中。否则和容器中的线段比较，找到 较高部分较多的那一条线段，将存放在容器中，另外一条直接递归下去比较，最 后到叶子节点直接修改即可。

具体操作



- 最终询问的时候，只需对于所有包含询问位置的区间的节点上线段，一个一 个求出 y 坐标，然后取最大即可。这样一次插入，在线段树上最多覆盖满 log 个区间，每个区间递归下去 log 层。查询时间复杂度为一个 log。时间复杂度为O(nlogn2)

楼房重建

- 有n条线段(i,0)-(i,hi)
- m次操作每次修改一条线段长度，并要求你输出该次操作后从（0,0）可以看到的线段数量。

- N,m<=10^5,Hi<=1e9

题解

- 第i条线段可以被看到是什么情况？其顶端点与原点的连接线段的斜率在1~i中最大。
- 我们用线段树维护一段区间的最大斜率，另外线段树中的区间会被划分成两半(l,mid)和(mid+1,r)，我们预处理出一个值：当右半区间的左边只有左半区间的时候（即不考虑1到l-1），(0,0)能看到右半区间的线段的个数。
- 怎么维护？

- 我们考虑一个函数f(l,r,p)表示当（1,l-1）的最大斜率为p时能看到区间中的线段数目。
- 显然当左区间最大值大于p时可以直接用预处理的值，然后递归进左区间，否则直接递归进右区间，这样可以在log^n时间复杂度内得到答案。
- 修改时，在线段树中修改并在合并时调用改函数更新线段树数据，再在全局中调用一次改函数得到答案。
- 总复杂度o(n log n^2)