目录

[Segment Tree 1](#_Toc119173715)

[推式子 2](#_Toc119173716)

[区间加等差数列，区间求和 2](#_Toc119173717)

[E. Tree Generator™ 3](#_Toc119173718)

[离线 7](#_Toc119173719)

[CF594D REQ 7](#_Toc119173720)

[动态开点 9](#_Toc119173721)

[主席树 11](#_Toc119173722)

[静态求小于等于k的个数 11](#_Toc119173723)

[静态整体K-th 12](#_Toc119173724)

[动态整体K-th 12](#_Toc119173725)

[静态区间K-th 13](#_Toc119173726)

[动态区间K-th 14](#_Toc119173727)

[区间修 17](#_Toc119173728)

[区间去重 19](#_Toc119173729)

[扫描线 21](#_Toc119173730)

[周长并 22](#_Toc119173731)

[线段树合并 24](#_Toc119173732)

[板子 25](#_Toc119173733)

[[Vani有约会]雨天的尾巴 /【模板】线段树合并 25](#_Toc119173734)

[线段树分裂 28](#_Toc119173735)

[模板 28](#_Toc119173736)

[二维线段树 31](#_Toc119173737)

[题目描述 31](#_Toc119173738)

[输入格式 31](#_Toc119173739)

[输出格式 31](#_Toc119173740)

[数据范围 31](#_Toc119173741)

[AC\_code 31](#_Toc119173742)

[区间加，区间覆盖维护历史最值 34](#_Toc119173743)

[区间取min/max 38](#_Toc119173744)

[区间取min 38](#_Toc119173745)

[区间取max 40](#_Toc119173746)

[吉老师线段树 41](#_Toc119173747)

[线段树优化建图 44](#_Toc119173748)

[Legacy 44](#_Toc119173749)

[树链剖分 48](#_Toc119173750)

[边权化点权 49](#_Toc119173751)

[Qtree1 49](#_Toc119173752)

[树剖+线段树上二分 50](#_Toc119173753)

[树 50](#_Toc119173754)

[需要维护路径中多条链上的多个信息。 51](#_Toc119173755)

[染色 51](#_Toc119173756)

[旅游 51](#_Toc119173757)

[有一些思维难度的题目 53](#_Toc119173758)

[Disruption P 53](#_Toc119173759)

[aaa被续 53](#_Toc119173760)

[[COCI2017-2018#5] Pictionary 54](#_Toc119173761)

[Mobile Phone Network 55](#_Toc119173762)

[[LNOI2014]LCA 56](#_Toc119173763)

[洛谷树 60](#_Toc119173764)

[有趣的游戏 64](#_Toc119173765)

[[USACO19DEC]Milk Visits G 67](#_Toc119173766)

[遥远的国度 69](#_Toc119173767)

[Jamie and Tree 73](#_Toc119173768)

[换根树剖 73](#_Toc119173769)

[动态开点线段树 81](#_Toc119173770)

[[SDOI2014]旅行 81](#_Toc119173771)

[树状数组 85](#_Toc119173772)

[单点修，求n阶前缀和 85](#_Toc119173773)

[Kruskal重构树 86](#_Toc119173774)

[实现过程 86](#_Toc119173775)

[性质 89](#_Toc119173776)

[应用 90](#_Toc119173777)

[堆 107](#_Toc119173778)

[模拟堆 107](#_Toc119173779)

[目的 107](#_Toc119173780)

[实现 107](#_Toc119173781)

[对顶堆 108](#_Toc119173782)

[目的 108](#_Toc119173783)

[实现原理 108](#_Toc119173784)

[实现 108](#_Toc119173785)

[可删除堆 109](#_Toc119173786)

[目的 109](#_Toc119173787)

[实现原理 109](#_Toc119173788)

[实现 109](#_Toc119173789)

[并查集 109](#_Toc119173790)

[可撤销并查集 109](#_Toc119173791)

[ST表 110](#_Toc119173792)

[二维数点 111](#_Toc119173793)

[离线 111](#_Toc119173794)

[在线 112](#_Toc119173795)

# Segment Tree

struct Node  
{  
 int l,r;  
 int add;  
 ll sum;  
}tr[N<<2];  
  
void pushup(int u)  
{  
 tr[u].sum = tr[u<<1].sum + tr[u<<1|1].sum;  
}  
  
void pushdown(int u)  
{  
 auto &root = tr[u],&left = tr[u<<1],&right = tr[u<<1|1];  
 if(root.add)  
 {  
 left.add += root.add,left.sum += root.add\*(left.r - left.l + 1);  
 right.add += root.add,right.sum += root.add\*(right.r - right.l + 1);  
 root.add = 0;  
 }  
}  
  
void build(int u,int l,int r)  
{  
 tr[u] = {l,r};  
 if(l==r)   
 {  
 tr[u].sum = a[l];  
 return ;  
 }  
 int mid = l + r >> 1;  
 build(u<<1,l,mid),build(u<<1|1,mid+1,r);  
}  
  
void modify(int u,int l,int r,int k)  
{  
 if(tr[u].l>r||tr[u].r<l) return ;  
 if(l<=tr[u].l&&tr[u].r<=r)  
 {  
 tr[u].sum += 1ll\*k\*(tr[u].r - tr[u].l + 1);  
 tr[u].add += k;  
 return ;  
 }  
 pushdown(u);  
 modify(u<<1,l,r,k),modify(u<<1|1,l,r,k);  
 pushup(u);  
}  
  
ll query(int u,int l,int r)  
{  
 if(tr[u].l>r||tr[u].r<l) return 0;  
 if(l<=tr[u].l&&tr[u].r<=r) return tr[u].sum;  
 pushdown(u);  
 return query(u<<1,l,r) + query(u<<1|1,l,r);  
}

## 推式子

### 区间加等差数列，区间求和

struct Node  
{  
 int l,r;  
 ll sum,val,d;  
}tr[N<<2];  
void build(int u,int l,int r)  
{  
 tr[u] = {l,r};  
 if(l==r) return ;  
 int mid = l + r >> 1;  
 build(u<<1,l,mid),build(u<<1|1,mid+1,r);  
}  
  
void pushup(int u)  
{  
 tr[u].sum = tr[u<<1].sum + tr[u<<1|1].sum;  
}  
  
void pushdown(int u)  
{  
 auto &root = tr[u],&left = tr[u<<1],&right = tr[u<<1|1];  
 if(root.val||root.d)  
 {  
 left.val += root.val,left.d += root.d;  
 left.sum += (2ll\*root.val + root.d\*(left.r-left.l))\*(left.r - left.l + 1)/2;  
 ll t = root.val + root.d\*(left.r - left.l + 1);  
 right.val += t,right.d += root.d;  
 right.sum += (2ll\*t + root.d\*(right.r-right.l))\*(right.r - right.l + 1)/2;  
 root.val = 0,root.d = 0;  
 }  
}  
  
void modify(int u,int l,int r,int a,int d)  
{  
 if(l>tr[u].r||r<tr[u].l) return ;   
 if(l<=tr[u].l&&tr[u].r<=r)  
 {  
 ll t = a + (tr[u].l - l)\*d;  
 tr[u].val += t;tr[u].d += d;  
 tr[u].sum += (2ll\*t + 1ll\*d\*(tr[u].r - tr[u].l))\*(tr[u].r - tr[u].l + 1)/2;  
 return ;  
 }  
 pushdown(u);  
 modify(u<<1,l,r,a,d);modify(u<<1|1,l,r,a,d);  
 pushup(u);  
}  
  
ll query(int u,int l,int r)  
{  
 if(l>tr[u].r||r<tr[u].l) return 0;  
 if(l<=tr[u].l&&tr[u].r<=r) return tr[u].sum;  
 pushdown(u);  
 return query(u<<1,l,r) + query(u<<1|1,l,r);  
}

### E. Tree Generator™

##### 分析

线段树好题，我们来一点点推理。

这里我们用括号序列来表示出一棵树，如果你对欧拉序这个概念有了解的话，括号序列就是对应的欧拉序。

本题你知道了欧拉序这个概念可能更好理解，但不懂也没事。

首先，我们需要知道。

##### 任取一段括号序列，左右括号匹配抵消后，剩下的括号数量对应了树中的一条链的长度。

这点，很好理解，因为，**左右括号匹配后，我们可以理解为，我们从一个点下去后又回来了，那我们可以理解为该点的影响消除掉（如果你对欧拉序有了解就会更明白这个意思）**

此时，我们要求的**树的直径就是树上所有路径的最大值**，我们记树上任意一条路径的长度为，则答案为

推导到这一步，我们需要知道两个关键点

1. **求树的直径其实就是求树上的所有路径的最大值，**
2. **将括号序列匹配的删去，最后剩下的序列的长度即为树上某条链的长度**

但知道这一点后，我们会发现这很难维护的，因此我们尝试进行接下来的推导。

##### 推导式子

首先，我们刚开始想到想消除匹配的括号，那肯定是想到能不能将(设为1，)设为-1,。

但是，很容易就能发现肯定不对啊，因为会把一些不匹配的也删去。

接下来的推导就比较奇妙了。

首先，我们在将括号转换结束后，我们假设[l,r]消完之后剩余x个右括号，y个左括号。我们进行求前缀和，可以得到数组，那么显然。

我们考虑，我们想求的路径的长度实际是**x+y**，但如果我们直接这样求完得到的值为**y-x**。

所以，我们为了得到答案，我们考虑将区间[l,r]在k处断开，则区间前半段左右相消后，一定为x个右括号，区间的后半段左右相消后，一定为y个左括号。

我们记sum(l,r)为[l,r]中所有数字和。那么有sum(l,k) = -x,sum(k+1,r) = y,还记得我们要求的值是什么嘛，sum(r,k+1) - sum(l,k) = y + x。同时我们可以将这个式子转化，sum(l,r) - 2\*sum(l,k)，我们可以发现，该式子的最值在sum(l,k) = -x处取到，因此，我们想要的答案即为整个序列的所有区间的该式子的最大值。

即，我们想要维护区间上sum(k+1,r) - sum(l,k)的最值，即，我们要维护的是，对于整个区间中，某一段满足，sum(y,z) - sum(x,y)的最大值。

##### 维护变量

我尽量以推理的角度，将维护的几个值合理推导一下，可能理解起来会更顺利一些。但为了理解的顺利，我会先将所有需要维护的值，在最前面列出来。

**只需要记住，我们维护前面七个，是为了维护最后一个，逻辑推导就会顺利一些**

* **sum表示sum(l,r)**
* **lmx表示maxsum(l,k)**
* **rmx表示maxsum(k,r)**
* **lmi表示minsum(l,k)**
* **rmi表示minsum(k,r)**
* **mx1表示maxsum(x,y) - sum(l,x),**
* **mx2表示maxsum(y,r) - sum(x,y),**
* **mx3表示maxsum(y,z) - sum(x,y),**

我们一个个来说

##### sum

这个就没什么特别值得说的，就直接求区间和就好

tr[u].sum = tr[u<<1].sum + tr[u<<1|1].sum

##### lmx与rmx

维护从l开始的最大和，**可以为0**

维护以r结尾的最大和，**可以为0**

tr[u].lmx = max(tr[u<<1].lmx,tr[u<<1].sum + tr[u<<1|1].lmx);  
tr[u].rmx = max(tr[u<<1|1].rmx,tr[u<<1|1].sum + tr[u<<1].rmx);

##### lmi与rmi

维护从l开始的最小和，**可以为0**

维护以r结尾的最小和，**可以为0**

tr[u].lmi = min(tr[u<<1].lmi,tr[u<<1].sum + tr[u<<1|1].lmi);  
tr[u].rmi = min(tr[u<<1|1].rmi,tr[u<<1|1].sum + tr[u<<1].rmi);

##### mx1

**维护的是，这个区间中，一些左端点为l的子区间的sum(y,x) - sum(l,x)的最值**

维护时，分为三种情况。

1. **x，y均在左区间。**

* 该情况最值为tr[u<<1].mx1

1. **x在左区间，y在右区间**

* 该情况最值为tr[u<<1|1].lmx + tr[u<<1].sum - 2\*tr[u<<1].lmi

1. **x，y均在右区间**

* 该情况最值为tr[u<<1|1].mx1-tr[u<<1].sum

##### mx2

**维护的是，这个区间中，一些右端点为r的子区间的sum(y,r) - sum(x,y)的最值**

维护时，依旧分为三种情况。

1. **x，y均在右区间**

* 该情况最值为tr[u<<1|1].mx2

1. **y在右区间，x在左区间**

* 该情况最值为tr[u<<1|1].sum - 2\*tr[u<<1|1].lmi - tr[u<<1].rmi

1. **x，y都在左区间**

* 该情况最值为tr[u<<1|1].sum + tr[u<<1].mx2

##### mx3

**维护的是，这个区间内，任选三个点x,y,z，sum(y,z) - sum(x,y)的最值，**

这次，我们分的情况较多，四种。

1. **x，y，z全部在左区间**

* tr[u<<1].mx3

1. **x，y在左区间，z在右区间**

* tr[u<<1|1].lmx + tr[u<<1].mx2

1. **x在左区间，y，z在右区间**

* -tr[u<<1].rmi + tr[u<<1].mx1

1. **x，y，z在右区间**

* tr[u<<1|1].mx3

修改操作就不多说了，直接改就是了。

##### AC\_code

#include<bits/stdc++.h>  
  
using namespace std;  
  
const int N = 2e5 + 10;  
/\*\*  
 \* mx1 = sum(x,y) - sum(l,x)  
 \* mx2 = sum(y,r) - sum(x,y)  
 \* mx3 = sum(y,z) - sum(x,y)  
 \*/  
struct Node  
{  
 int l,r,sum;  
 int lmx,rmx;  
 int lmi,rmi ;  
 int mx1,mx2,mx3;  
}tr[N<<2];  
char s[N];  
int n,q;  
  
void pushup(int u)  
{  
 tr[u].sum = tr[u<<1].sum + tr[u<<1|1].sum;  
 tr[u].lmx = max(tr[u<<1].lmx,tr[u<<1].sum+tr[u<<1|1].lmx);  
 tr[u].rmx = max(tr[u<<1|1].rmx,tr[u<<1|1].sum+tr[u<<1].rmx);  
 tr[u].lmi = min(tr[u<<1].lmi,tr[u<<1].sum+tr[u<<1|1].lmi);  
 tr[u].rmi = min(tr[u<<1|1].rmi,tr[u<<1|1].sum+tr[u<<1].rmi);  
 tr[u].mx1=max(tr[u<<1].mx1,max(-tr[u<<1].sum+tr[u<<1|1].mx1,tr[u<<1|1].lmx + tr[u<<1].sum - 2\*tr[u<<1].lmi));  
 tr[u].mx2=max(tr[u<<1|1].mx2,max(tr[u<<1|1].sum+tr[u<<1].mx2,-tr[u<<1].rmi+tr[u<<1|1].sum-2\*tr[u<<1|1].lmi));  
 tr[u].mx3=max(max(tr[u<<1].mx3,tr[u<<1|1].mx3),max(tr[u<<1].mx2+tr[u<<1|1].lmx,-tr[u<<1].rmi+tr[u<<1|1].mx1));  
}  
  
void build(int u,int l,int r)  
{  
 tr[u] = {l,r};  
 if(l==r)  
 {  
 if(s[l]=='(') tr[u].sum = 1,tr[u].lmx = tr[u].rmx = 1,tr[u].lmi = tr[u].rmi = 0;  
 else tr[u].sum = -1,tr[u].lmx = tr[u].rmx = 0,tr[u].lmi = tr[u].rmi = -1;  
 tr[u].mx1 = tr[u].mx2 = tr[u].mx3 = 1;  
 return ;  
 }  
 int mid = l + r >> 1;  
 build(u<<1,l,mid),build(u<<1|1,mid+1,r);  
 pushup(u);  
}  
  
void modify(int u,int x,int k)  
{  
 if(tr[u].l==tr[u].r)  
 {  
 tr[u].lmx = tr[u].rmx = max(k,0);  
 tr[u].lmi = tr[u].rmi = min(k,0);  
 tr[u].sum = k;  
 tr[u].mx1 = 1,tr[u].mx2 = 1,tr[u].mx3 = 1;  
 return ;  
 }  
 int mid = tr[u].l + tr[u].r >> 1;  
 if(x<=mid) modify(u<<1,x,k);  
 else modify(u<<1|1,x,k);  
 pushup(u);  
}  
  
int main()  
{  
 scanf("%d%d",&n,&q);  
 scanf("%s",s+1);  
 build(1,1,2\*(n-1));  
 printf("%d\n",tr[1].mx3);  
 while(q--)  
 {  
 int x,y;scanf("%d%d",&x,&y);  
 if(s[x]!=s[y])  
 {  
 swap(s[x],s[y]);  
 modify(1,x,(s[x]=='(')?1:-1);  
 modify(1,y,(s[y]=='(')?1:-1);  
 }  
 printf("%d\n",tr[1].mx3);  
 }  
 return 0;  
}

## 离线

### [CF594D REQ](https://www.luogu.com.cn/problem/CF594D)

#### 题目大意

给定序列，有个询问，每次给定，询问。对 取模。

.

#### 分析

比较早期的CF的数据结构题。层层buff叠加下，这题显然不会太难。

我们来简单分析一下。首先看到只有询问，那刻在DNA里的，就要想**离线**操作。

我们考虑计算，等于计算

我们最大的问题时，**如何不重复的计算区间内的每个质数**，第二次DNA一动，**去重**又是**离线**的经典操作。

我们维护**树状数组**。

我们对区间**右端点**进行排序，每次扫到一个点i时，将该点的值，以及其所有的质数所对应的贡献加入树状数组，总的来说就是，。

如何去重呢？我们只需要记录当前位置加的质数上次出现的位置的0贡献清空即可保证去重。我们维护一个数组pre[x]表示x这个数字上次出现的位置。

还有一个小问题，对于每一个位置，如果我们每次扫到之后再进行求所有的质数，这时间复杂度为。因此我们利用**积性函数**的性质，O(n)求出。具体实现可以看代码。

#### AC\_code

#include<bits/stdc++.h>  
#define ios ios::sync\_with\_stdio(false); cin.tie(0), cout.tie(0)  
using namespace std;  
typedef long long LL;  
typedef pair<int,int> PII;  
const int N = 2e5 + 10,M = 1e6 + 10,mod = 1e9 + 7;  
  
struct Node  
{  
 int l,r,id;  
 bool operator<(const Node& W) const   
 {  
 return r<W.r;  
 }  
};  
  
int tr[N];  
vector<int> fact[M];  
bool st[M];  
int primes[M],cnt;  
int a[N],ans[N],pre[M],inv[M];  
int n,q;  
  
int ksm(int a,int b)  
{  
 int res = 1;  
 while(b)  
 {  
 if(b&1) res = 1ll\*res\*a%mod;  
 a = 1ll\*a\*a%mod;  
 b >>= 1;  
 }  
 return res;  
}  
  
void init()  
{  
 fact[1].push\_back(1);st[1] = 1;  
 for(int i=2;i<M;i++)  
 {  
 if(!st[i])   
 {  
 fact[i].push\_back(i);  
 primes[++cnt] = i;  
 inv[i] = ksm(i,mod-2);  
 inv[i-1] = ksm(i-1,mod-2);  
 }  
 for(int j=1;1ll\*i\*primes[j]<M;j++)  
 {  
 st[primes[j]\*i] = 1;  
 fact[primes[j]\*i] = fact[i];  
 if(i%primes[j]==0) break;  
 fact[primes[j]\*i].push\_back(primes[j]);  
 }  
 }  
}  
  
void add(int x,int c)  
{  
 while(x<=n)  
 {  
 tr[x] = 1ll\*tr[x]\*c%mod;  
 x += x & -x;  
 }  
}  
  
int sum(int x)  
{  
 int res = 1;  
 while(x)  
 {  
 res = 1ll\*res\*tr[x]%mod;  
 x -= x & -x;  
 }  
 return res;  
}  
  
int main()  
{  
 ios;  
 init();  
 cin>>n;  
 for(int i=1;i<=n;i++) tr[i] = 1;  
 for(int i=1;i<=n;i++) cin>>a[i];  
 cin>>q;  
 vector<Node> query;  
 for(int i=0;i<q;i++)   
 {  
 int l,r;cin>>l>>r;  
 query.push\_back({l,r,i});  
 }  
 sort(query.begin(),query.end());  
 int now = 0;  
 for(int i=0;i<q;i++)  
 {  
 int l = query[i].l,r = query[i].r,id = query[i].id;  
 while(now<r)  
 {  
 now++;  
 if(a[now]==1) continue;  
 add(now,a[now]);  
 for(auto x:fact[a[now]])  
 {  
 if(pre[x]) add(pre[x],inv[x-1]),add(pre[x],x);  
 pre[x] = now;  
 add(now,x-1),add(now,inv[x]);  
 }  
 }  
 if(l!=1) ans[id] = 1ll\*sum(r)\*ksm(sum(l-1),mod-2)%mod;  
 else ans[id] = sum(r);  
 }  
 for(int i=0;i<q;i++) cout<<ans[i]<<'\n';  
 return 0;  
}

## 动态开点

**这里给的板子，中间有pushdown操作，因此最好开单点修的双倍点数，如果不行，逼近极限的开**，

**如果结构体其中有多个不同种类的属性，建议直接拆开写，省一些内存**

struct Node  
{  
 int l,r;  
 int sum,add;  
}tr[N<<6];  
int n,idx,root;  
  
int init(int u)  
{  
 tr[u].l = tr[u].r = tr[u].add = tr[u].sum = 0;  
 return u;  
}  
  
void pushup(int u)  
{  
 tr[u].sum = tr[tr[u].l].sum + tr[tr[u].r].sum;  
}  
  
void pushdown(int u,int l,int r)  
{  
 if(!tr[u].l) tr[u].l = init(++idx);  
 if(!tr[u].r) tr[u].r = init(++idx);  
 auto &root = tr[u],&left = tr[tr[u].l],&right = tr[tr[u].r];  
 int mid = l + r >> 1;  
 if(root.add)  
 {  
 left.add += root.add;  
 right.add += root.add;  
 left.sum += (mid - l + 1)\*root.add;  
 right.sum += (r - mid)\*root.add;  
 root.add = 0;  
 }  
}  
  
void modify(int &u, int l, int r, int L, int R, int k) {  
 if(R<l||L>r) return ;  
 if(!u) u = init(++idx);  
 if(L <= l && R >= r) {  
 int len = r - l + 1;  
 tr[u].sum += len \* k;  
 tr[u].add += k;  
 return ;  
 }  
 pushdown(u, l, r);  
 int mid = l + r >> 1;  
 modify(tr[u].l, l, mid, L, R, k);modify(tr[u].r, mid + 1, r, L, R, k);  
 pushup(u);  
}  
  
int query(int u,int l,int r,int L,int R)  
{  
 if(R<l||L>r) return 0;  
 if(!u) return 0;  
 if(L<=l&&r<=R) return tr[u].sum;  
 pushdown(u,l,r);  
 int mid = l + r >> 1;  
 return query(tr[u].l,l,mid,L,R) + query(tr[u].r,mid+1,r,L,R);  
}

## 主席树

**如果结构体其中有多个不同种类的属性，建议直接拆开写，省一些内存**

**有时候，如果初始化中，节点属性有一些特殊的，可以考虑将root[0]的树也建出来**

**首先要说，这里给的例子，都是建立的权值线段树，因此我们将权值给离散化降低了常数**

**但是，如果是可持久化的数组的话，即以下标去持久化的话，就不用离散化了。**

### 静态求小于等于k的个数

这里给的例子是，**小于等于k**。

**这里给的下标是从0开始标的，若从1开始记得+1**

我们分别给出四种情况，我们对get以及最后求解的一点点改变。

1. **小于等于**，则get中为upper\_bound(nums.begin(),nums.end(),x) - nums.begin() - 1，因为我们要找到**小于等于k的第一个位置**。
2. **小于**，则get中为lower\_bound(nums.begin(),nums.end(),x) - nums.begin() - 1，因为我们要找到**小于k的第一个位置**。
3. **大于等于**，**等价于len-cnt(小于)**
4. **大于**，**等价于len-cnt(小于等于)**

#include<bits/stdc++.h>  
using namespace std;  
const int N = 2e5 + 10;  
struct Node  
{  
 int l,r;  
 int cnt;  
}tr[N\*4+N\*17];  
int root[N],idx;  
int a[N];  
vector<int> nums;  
int n,m;  
  
int find(int x)  
{  
 return lower\_bound(nums.begin(),nums.end(),x) - nums.begin();  
}  
  
int get(int x)  
{  
 return upper\_bound(nums.begin(),nums.end(),x) - nums.begin() - 1;  
}  
  
int insert(int p,int l,int r,int x)  
{  
 int q = ++idx;  
 tr[q]=tr[p];  
 if(l==r)  
 {  
 tr[q].cnt++;  
 return q;  
 }  
 int mid = l + r >> 1;  
 if(x<=mid) tr[q].l=insert(tr[q].l,l,mid,x);  
 else tr[q].r=insert(tr[q].r,mid+1,r,x);  
 tr[q].cnt=tr[tr[q].l].cnt+tr[tr[q].r].cnt;  
 return q;  
}  
  
int query(int q,int p,int l,int r,int k)  
{  
 if(l==r)  
 {  
 if(l<=k) return tr[q].cnt - tr[p].cnt;  
 return 0;  
 }  
 int mid = l + r >> 1;  
 int cnt = tr[tr[q].l].cnt - tr[tr[p].l].cnt;  
 if(mid<k) return cnt + query(tr[q].r,tr[p].r,mid+1,r,k);  
 return query(tr[q].l,tr[p].l,l,mid,k);  
}  
  
int main()  
{  
 int T,cas = 0;cin>>T;  
 while(T--){  
 scanf("%d%d", &n, &m);  
 for(int i=1;i<=n;i++)   
 {  
 scanf("%d",&a[i]);  
 nums.push\_back(a[i]);  
 }  
  
 sort(nums.begin(),nums.end());  
 nums.erase(unique(nums.begin(),nums.end()),nums.end());  
  
 for(int i=1;i<=n;i++) root[i] = insert(root[i-1],0,nums.size()-1,find(a[i]));  
 printf("Case %d:\n",++ cas);  
 while(m--)  
 {  
 int l,r,k;   
 scanf("%d%d%d",&l,&r,&k);l++,r++;  
 printf("%d\n",query(root[r],root[l-1],0,nums.size()-1,get(k)));  
 }  
 nums.clear();  
 idx = 0;  
 }  
 return 0;  
}

### 静态整体K-th

**直接sort**

### 动态整体K-th

**单点修，整体查。**

我们维护一颗，权值线段树。节点维护的值sum即为，该区间对应的所有值有多少个。

然后需要的时候，直接树上二分就好。

### 静态区间K-th

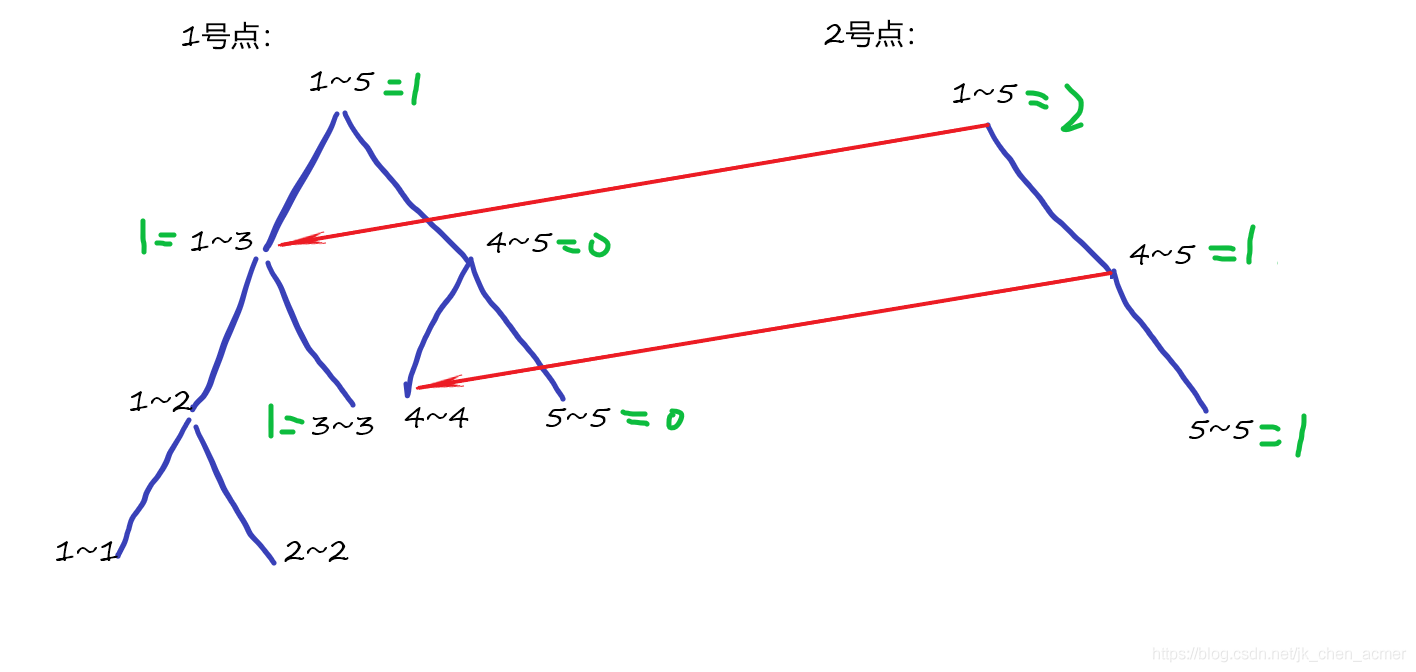
**静态第k大等价求len-k+1小**

#include<bits/stdc++.h>  
using namespace std;  
const int N = 2e5 + 10;  
struct Node  
{  
 int l,r;  
 int cnt;  
}tr[N\*4+N\*17];  
int root[N],idx;  
int a[N];  
vector<int> nums;  
int n,m;  
  
int find(int x)  
{  
 return lower\_bound(nums.begin(),nums.end(),x)-nums.begin();  
}  
  
int build(int l,int r)  
{  
 int p = ++idx;  
 if(l==r) return p;  
 int mid = l + r >> 1;  
 tr[p].l=build(l,mid),tr[p].r=build(mid+1,r);  
 return p;  
}  
  
int insert(int p,int l,int r,int x)  
{  
 int q = ++idx;  
 tr[q]=tr[p];  
 if(l==r)  
 {  
 tr[q].cnt++;  
 return q;  
 }  
 int mid = l + r >> 1;  
 if(x<=mid) tr[q].l=insert(tr[q].l,l,mid,x);  
 else tr[q].r=insert(tr[q].r,mid+1,r,x);  
 tr[q].cnt=tr[tr[q].l].cnt+tr[tr[q].r].cnt;  
 return q;  
}  
  
int query(int q,int p,int l,int r,int k)  
{  
 if(l==r) return r;  
 int mid = l + r >> 1;  
 int cnt = tr[tr[q].l].cnt - tr[tr[p].l].cnt;  
 if(k<=cnt) return query(tr[q].l,tr[p].l,l,mid,k);  
 else return query(tr[q].r,tr[p].r,mid+1,r,k-cnt);  
}  
  
int main()  
{  
 scanf("%d%d", &n, &m);  
 for(int i=1;i<=n;i++)   
 {  
 scanf("%d",&a[i]);  
 nums.push\_back(a[i]);  
 }  
  
 sort(nums.begin(),nums.end());  
 nums.erase(unique(nums.begin(),nums.end()),nums.end());  
  
 for(int i=1;i<=n;i++) root[i]=insert(root[i-1],0,nums.size()-1,find(a[i]));  
  
 for(int i=0;i<m;i++)  
 {  
 int l,r,k;  
 scanf("%d%d%d",&l,&r,&k);  
 printf("%d\n",nums[query(root[r],root[l-1],0,nums.size()-1,k)]);  
 }  
 return 0;  
}

### 动态区间K-th

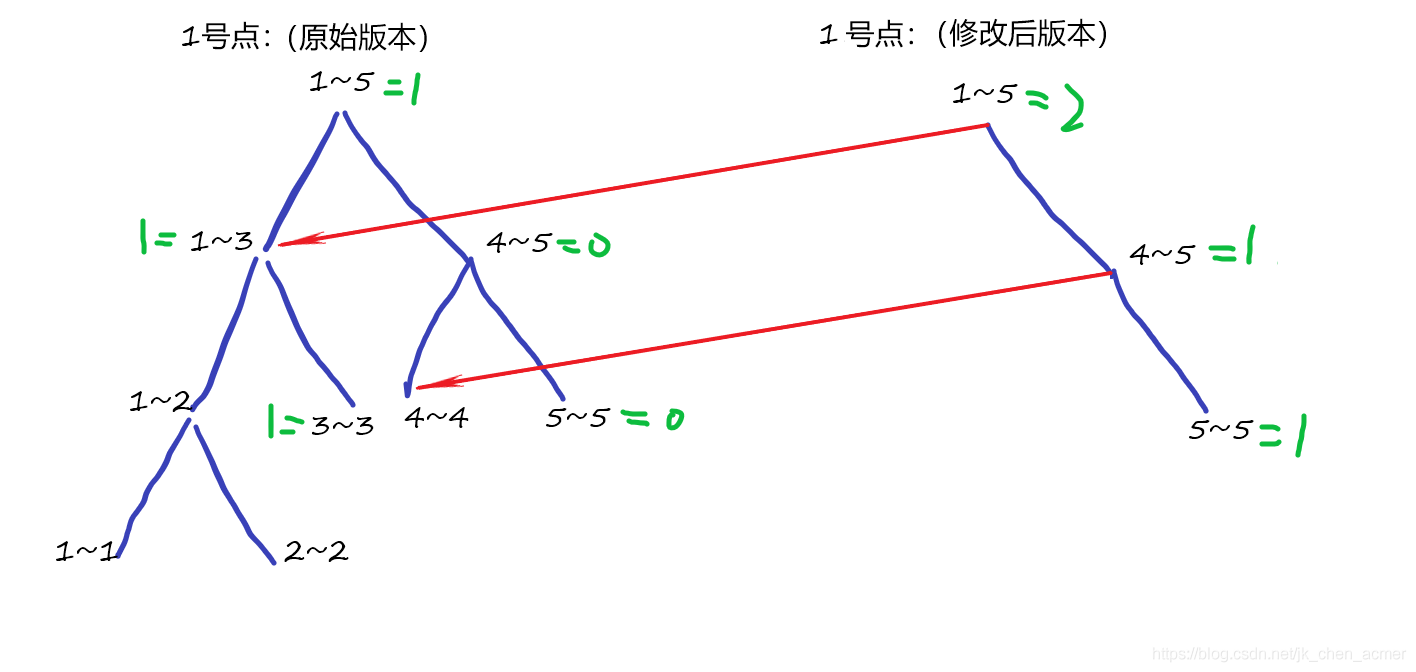
**单点修，区间查**

**时间复杂度**



在这里插入图片描述

直接对比不带修的主席树，其实我们就是对于点对应的线段树直接更新了，而不去创建一个新版本。



在这里插入图片描述

**简单来说，就是，我们使用树状数组套权值线段树，每次修改时，直接修改每一个位置对应的线段树的log(n)个节点**

#include<bits/stdc++.h>  
#define ios ios::sync\_with\_stdio(false); cin.tie(0), cout.tie(0)  
using namespace std;  
const int N = 1e5 + 10;  
  
struct Node  
{  
 int l,r;  
 int sum;  
}tr[N\*400];//开的大小为 询问次数\*log(线段树长度(指离散化后的长度))^2 本题是1e5询问，离散化后长度为20，因此N\*400  
  
struct Ask  
{  
 char op;  
 int l,r,k;  
}ask[N];  
  
int n,m,idx;  
int a[N],root[N];//原序列值，每个位置对应的根节点  
vector<int> alls;//离散化  
int ctl,ctr,L[N],R[N];//存储每次询问时，左边与右边需要改变的树时哪些，其实不用开到N，没那么大  
  
int get(int x)  
{  
 return lower\_bound(alls.begin(),alls.end(),x) - alls.begin() + 1;  
}  
  
void pushup(int u)  
{  
 tr[u].sum = tr[tr[u].l].sum + tr[tr[u].r].sum;  
}  
  
//last存的是之前的节点下标，now存的是当前的节点下标，用now替换last的。  
int modify(int last,int now,int l,int r,int x,int k)  
{  
 tr[now].l = tr[last].l,tr[now].r = tr[last].r;  
 if(l==r)  
 {  
 tr[now].sum = tr[last].sum + k;  
 return now;  
 }  
 int mid = l + r >> 1;  
 if(x<=mid) tr[now].l = modify(tr[now].l,++idx,l,mid,x,k);  
 else tr[now].r = modify(tr[now].r,++idx,mid+1,r,x,k);  
 pushup(now);  
 return now;  
}  
  
//x是改变的位置，val是x对应的权值线段树改变的位置，v是改变的值  
void add(int x,int val,int v)  
{  
 int k = get(val);  
 while(x<=n)  
 {  
 //我们每次操作就是将x对应的线段树的log(alls.size())的节点更新一下。  
 root[x] = modify(root[x],++idx,1,alls.size(),k,v);  
 x += x & -x;  
 }  
}  
  
void query\_init(int l,int r)  
{  
 ctl = ctr = 0;l--;//--是因为用的是sum(r) - sum(l-1)  
 while(l)  
 {  
 L[++ctl] = root[l];  
 l -= l & -l;  
 }  
 while(r)  
 {  
 R[++ctr] = root[r];  
 r -= r & -r;  
 }  
}  
  
int query(int l,int r,int k)//类似于主席树去记忆  
{  
 if(l==r) return l;  
 int mid = l + r >> 1;  
 int v = 0;  
 for(int i=1;i<=ctl;i++) v -= tr[tr[L[i]].l].sum;  
 for(int i=1;i<=ctr;i++) v += tr[tr[R[i]].l].sum;  
 if(v>=k)  
 {  
 for(int i=1;i<=ctl;i++) L[i] = tr[L[i]].l;  
 for(int i=1;i<=ctr;i++) R[i] = tr[R[i]].l;  
 return query(l,mid,k);  
 }  
 else   
 {  
 for(int i=1;i<=ctl;i++) L[i] = tr[L[i]].r;  
 for(int i=1;i<=ctr;i++) R[i] = tr[R[i]].r;  
 return query(mid+1,r,k-v);  
 }  
}  
  
int main()  
{  
 ios;  
 cin>>n>>m;  
 for(int i=1;i<=n;i++) cin>>a[i],alls.push\_back(a[i]);  
 for(int i=0;i<m;i++)  
 {  
 char op;int l,r,k;cin>>op>>l>>r;  
 if(op=='Q') cin>>k,ask[i] = {op,l,r,k};  
 else ask[i] = {op,l,r},alls.push\_back(r);  
 }  
 sort(alls.begin(),alls.end());  
 alls.erase(unique(alls.begin(),alls.end()),alls.end());  
 for(int i=1;i<=n;i++) add(i,a[i],1);  
 for(int i=0;i<m;i++)  
 {  
 auto [op,l,r,k] = ask[i];  
 if(op=='Q')  
 {  
 query\_init(l,r);  
 cout<<alls[query(1,alls.size(),k)-1]<<'\n';  
 }  
 else   
 {  
 add(l,a[l],-1);  
 a[l] = r;  
 add(l,r,1);  
 }  
 }  
 return 0;  
}

### 区间修

**区间加，区间询问和**

我们用**标记永久化**，也就是说，我们只在需要的时候，直接将标记当做参数下传来避免pushdown。

#include<bits/stdc++.h>  
#define ios ios::sync\_with\_stdio(false); cin.tie(0), cout.tie(0)  
using namespace std;  
using ll = long long;  
const int N = 1e5 + 10;  
  
struct Node  
{  
 int l,r;  
 ll sum,add;  
}tr[N\*25];//正常主席树大小  
  
int n,m;  
int a[N],root[N],idx;  
  
void pushup(int u,int l,int r)//计算每一层的pushdown时，因为标记未下传，需要直接将标记的影响一起算上。  
{  
 tr[u].sum = tr[tr[u].l].sum + tr[tr[u].r].sum + tr[u].add\*(r - l + 1);  
}  
  
void build(int &u,int l,int r)  
{  
 u = ++idx;  
 if(l==r)  
 {  
 tr[u].sum = a[l];  
 return ;  
 }  
 int mid = l + r >> 1;  
 build(tr[u].l,l,mid),build(tr[u].r,mid+1,r);  
 pushup(u,l,r);  
}  
  
int insert(int p,int l,int r,int L,int R,ll k)//k区间加的值  
{  
 if(l>R||r<L) return p;  
 int q = ++idx;  
 tr[q]=tr[p];  
 if(L<=l&&r<=R)  
 {  
 tr[q].sum += (r - l + 1)\*k;//将k的影响加上  
 tr[q].add += k;  
 return q;  
 }  
 int mid = l + r >> 1;  
 tr[q].l=insert(tr[q].l,l,mid,L,R,k);tr[q].r=insert(tr[q].r,mid+1,r,L,R,k);  
 pushup(q,l,r);  
 return q;  
}  
  
ll query(int q,int l,int r,int L,int R,ll k)  
{  
 if(l>R||r<L) return 0;  
 if(L<=l&&r<=R) return tr[q].sum + (r-l+1)\*k;//将父节点下传的标记的影响加上  
 int mid = l + r >> 1;  
 return query(tr[q].l,l,mid,L,R,k+tr[q].add) + query(tr[q].r,mid+1,r,L,R,k+tr[q].add);  
 //不断增加标记的值  
}  
  
int main()  
{  
 ios;  
 int T = 0;  
 while(cin>>n>>m){  
 if(T) cout<<'\n';  
 for(int i=1;i<=n;i++) cin>>a[i],root[i] = 0;  
 build(root[0],1,n);  
 int id = 0;  
 while(m--)  
 {  
 char op;int l,r,d;  
 cin>>op>>l;  
 if(op=='C')//区间加一个值  
 {  
 cin>>r>>d;id++;  
 root[id] = insert(root[id-1],1,n,l,r,d);  
 }  
 else if(op=='Q') //询问当前树下的[l,r]的值  
 {  
 cin>>r;  
 cout<<query(root[id],1,n,l,r,0)<<'\n';  
 }  
 else if(op=='H') //询问d版本的树下的[l,r]的值  
 {  
 cin>>r>>d;  
 cout<<query(root[d],1,n,l,r,0)<<'\n';  
 }  
 else root[id] = root[l],id = l;//将版本回溯  
 }  
 for(int i=1;i<=idx;i++) tr[i] = {0,0,0,0};//清空  
 idx = 0;T++;  
 }  
 return 0;  
}

### 区间去重

这里给的例子是，区间去重后算区间和。

求区间[l,r]中出现过的数字的和（相同的只算一次）。粘上去重我们一般都考虑能否离线，但是这题强制在线。如果不能离线，那这类问题我们通常用主席树去做。我们来看看怎么用主席树去做。

首先，我们设表示[1,i-1]中最后一次出现的位置，没有出现则为0。这个可以求到。

对于一个，若，说明[l,i-1]中出现了一个与相等的数，那么这个i是不能算贡献的。

问题就又变成了，。考虑建立主席树，第j棵树维护的是的数的和。(的位置上都为0)。

我们需要做的是，对于某个值，如果是第一次出现，我们直接在root[0]的位置进行修改。否则，我们设表示为下一个与相同的位置是哪，我们到这个位置后，直接新建一棵树，直接单点修，该位置改为。

简单来说，我们是用这种方式，用空间把所有情况预处理出来了。最后的结果，就是对第i棵树，我们求其上的区间和[l,r](要求l大于等于i)就是，对于下标i后面的所有数完成了去重。

所有的本可以离线但强制在线的题目都是类似的思路。

#include<bits/stdc++.h>  
#define ios ios::sync\_with\_stdio(false); cin.tie(0), cout.tie(0)  
using namespace std;  
  
using ll = long long;  
const int N = 2e5 + 10;  
struct Node  
{  
 int l,r;  
 ll sum;  
}tr[N\*20];  
  
int n,m,a[N],b[N],rt[N],pos[N],idx;  
  
void pushup(int u)  
{  
 tr[u].sum = tr[tr[u].l].sum + tr[tr[u].r].sum;  
}  
  
void modify1(int &u,int l,int r,int x,int y)  
{  
 if(!u) u = ++idx;  
 if(l==r)  
 {  
 tr[u].sum = y;  
 return ;  
 }  
 int mid = l + r >> 1;  
 if(x<=mid) modify1(tr[u].l,l,mid,x,y);  
 else modify1(tr[u].r,mid+1,r,x,y);  
 pushup(u);  
}  
  
void modify2(int u,int &v,int l,int r,int x,int y)  
{  
 v = ++idx;  
 tr[v] = tr[u];  
 if(l==r)  
 {  
 tr[v].sum = y;  
 return ;  
 }  
 int mid = l + r >> 1;  
 if(x<=mid) modify2(tr[u].l,tr[v].l,l,mid,x,y);  
 else modify2(tr[u].r,tr[v].r,mid+1,r,x,y);  
 pushup(v);  
}  
  
ll query(int u,int l,int r,int x,int y)  
{  
 if(l>y||r<x) return 0;  
 if(!u) return 0;  
 if(x<=l&&r<=y) return tr[u].sum;  
 int mid = l + r >> 1;  
 return query(tr[u].l,l,mid,x,y) + query(tr[u].r,mid+1,r,x,y);  
}  
  
void buildseg()  
{  
 for(int i=1;i<=n;i++)  
 {  
 if(!pos[a[i]]) modify1(rt[0],1,n,i,a[i]);  
 else b[pos[a[i]]] = i;  
 pos[a[i]] = i;  
 }  
 for(int i=1;i<=n;i++)  
 {  
 if(b[i]) modify2(rt[i-1],rt[i],1,n,b[i],a[b[i]]);  
 else rt[i] = rt[i-1];  
 }  
}  
  
int main()  
{  
 ios;cin>>n>>m;  
 for(int i=1;i<=n;i++) cin>>a[i];  
 buildseg();  
 while(m--)  
 {  
 int l,r;cin>>l>>r;  
 cout<<query(rt[l-1],1,n,l,r)<<'\n';  
 }  
 return 0;  
}

## 扫描线

#### 面积并

#include<bits/stdc++.h>  
#define ios ios::sync\_with\_stdio(false); cin.tie(0), cout.tie(0)  
using namespace std;  
using ll = long long;  
const int N = 1e6 + 10;  
  
int n;  
ll xl,yl,xr,yr,alls[N<<1];  
  
struct ScanLine{  
 ll l,r,h;  
 int mark;//mark用于保存权值(1/-1)  
 bool operator<(const ScanLine &rhs) const{  
 return h < rhs.h;  
 }  
}line[N<<1];  
  
struct Node  
{  
 int l,r,sum;  
 ll len;  
 //sum：被完全覆盖的次数  
 //len：区间内被截的长度  
}tr[N<<2];  
  
void build(int u,int l,int r)  
{  
 tr[u] = {l,r};  
 if(l==r) return ;  
 int mid = l + r >> 1;  
 build(u<<1,l,mid),build(u<<1|1,mid+1,r);  
}  
  
void pushup(int u)  
{  
 int l = tr[u].l,r = tr[u].r;  
 if(tr[u].sum) tr[u].len = alls[r + 1] - alls[l];//也就是说被覆盖过，更新长度  
 else tr[u].len = tr[u<<1].len + tr[u<<1|1].len;//合并儿子信息  
}  
  
void modify(int u,ll L,ll R,int k)  
{  
 int l = tr[u].l,r = tr[u].r;  
 //注意，l，r和L，R的意义完全不同  
 //l，r表示这个节点管辖的下标范围  
 //而L，R则表示需要修改的真是下标  
 if(alls[r+1]<=L||R<=alls[l]) return ;  
 //这里加等号的原因是  
 //假设现在考虑[2,5]，[5,8]两条线段，要修改[1,5]区间的sum  
 //很明显，虽然5在这个区间内，[5,8]却并不是我们希望修改的线段。  
 //所以总价一下，就加上了等号  
 if(L<=alls[l]&&alls[r+1]<=R)  
 {  
 tr[u].sum += k;  
 pushup(u);  
 return ;  
 }  
 modify(u<<1,L,R,k);modify(u<<1|1,L,R,k);  
 pushup(u);  
}  
  
int main()  
{  
 ios;  
 cin>>n;  
 for(int i=1;i<=n;i++)  
 {  
 cin>>xl>>yl>>xr>>yr;  
 alls[2\*i - 1] = xl,alls[2\*i] = xr;  
 line[2\*i-1] = {xl,xr,yl,1};  
 line[2\*i] = {xl,xr,yr,-1};  
 //一条线段包含两个端点，一个矩形的上下边都需要扫描线扫过  
 }  
 n<<=1;  
 sort(line+1,line+n+1);  
 sort(alls + 1,alls + 1 + n);  
 int tot = unique(alls + 1,alls + 1 + n) - alls - 1;  
 build(1,1,tot - 1);  
 //为什么是tot - 1  
 //因为右端点的对应关系已经被篡改了，现在是左闭右开  
 ll ans = 0;  
 for(int i=1;i<n;i++)//最后一条不需要加  
 {  
 modify(1,line[i].l,line[i].r,line[i].mark);  
 //先把扫描线的信息导入线段树  
 ans += tr[1].len \* (line[i+1].h - line[i].h);  
 //统计面积  
 }  
 cout<<ans<<'\n';  
 return 0;  
}

### 周长并

#include <iostream>  
#include <stdio.h>  
#include <algorithm>  
#define lson (x << 1)  
#define rson (x << 1 | 1)  
using namespace std;  
const int MAXN = 2e4;  
int n, X[MAXN << 1];  
int x1, y1, x2, y2, pre = 0; /\* 先初始化为 0 \*/  
  
struct ScanLine {  
 int l, r, h, mark;  
 bool operator<(const ScanLine& rhs) const  
 {  
 if(h == rhs.h) return mark > rhs.mark;  
 return h < rhs.h;  
 }  
// 注意！这里是后来被 hack 掉以后加上去的  
// 在此感谢 @leprechaun\_kdl 指出问题  
// 如果出现了两条高度相同的扫描线，也就是两矩形相邻  
// 那么需要先扫底边再扫顶边，否则就会多算这条边  
// 这个对面积并无影响但对周长并有影响  
// hack 数据：2 0 0 4 4 0 4 4 8 输出应为：24  
} line[MAXN];  
  
struct SegTree {  
 int l, r, sum, len, c;  
// c表示区间线段条数  
 bool lc, rc;  
// lc, rc分别表示左、右端点是否被覆盖  
// 统计线段条数(tree[x].c)会用到  
} tree[MAXN << 2];  
  
void build\_tree(int x, int l, int r) {  
 tree[x].l = l, tree[x].r = r;  
 tree[x].lc = tree[x].rc = false;  
 tree[x].sum = tree[x].len = 0;  
 tree[x].c = 0;  
 if(l == r)  
 return;  
 int mid = (l + r) >> 1;  
 build\_tree(lson, l, mid);  
 build\_tree(rson, mid + 1, r);  
}  
  
void pushup(int x) {  
 int l = tree[x].l, r = tree[x].r;  
 if(tree[x].sum) {  
 tree[x].len = X[r + 1] - X[l];  
 tree[x].lc = tree[x].rc = true;  
 tree[x].c = 1;  
// 做好相应的标记  
 }  
 else {  
 tree[x].len = tree[lson].len + tree[rson].len;  
 tree[x].lc = tree[lson].lc, tree[x].rc = tree[rson].rc;  
 tree[x].c = tree[lson].c + tree[rson].c;  
// 如果左儿子左端点被覆盖，那么自己的左端点也肯定被覆盖；右儿子同理  
 if(tree[lson].rc && tree[rson].lc)  
 tree[x].c -= 1;  
// 如果做儿子右端点和右儿子左端点都被覆盖，  
// 那么中间就是连续的一段，所以要 -= 1  
 }  
}  
  
void edit\_tree(int x, int L, int R, int c) {  
 int l = tree[x].l, r = tree[x].r;  
 if(X[l] >= R || X[r + 1] <= L)  
 return;  
 if(L <= X[l] && X[r + 1] <= R) {  
 tree[x].sum += c;  
 pushup(x);  
 return;  
 }  
 edit\_tree(lson, L, R, c);  
 edit\_tree(rson, L, R, c);  
 pushup(x);  
}  
  
ScanLine make\_line(int l, int r, int h, int mark) {  
 ScanLine res;  
 res.l = l, res.r = r,  
 res.h = h, res.mark = mark;  
 return res;  
}  
// POJ 不这样做就会CE，很难受  
  
int main() {  
 scanf("%d", &n);  
 for(int i = 1; i <= n; i++) {  
 scanf("%d %d %d %d", &x1, &y1, &x2, &y2);  
 line[i \* 2 - 1] = make\_line(x1, x2, y1, 1);  
 line[i \* 2] = make\_line(x1, x2, y2, -1);  
 X[i \* 2 - 1] = x1, X[i \* 2] = x2;  
 }  
 n <<= 1;  
 sort(line + 1, line + n + 1);  
 sort(X + 1, X + n + 1);  
 int tot = unique(X + 1, X + n + 1) - X - 1;  
 build\_tree(1, 1, tot - 1);  
 int res = 0;  
 for(int i = 1; i < n; i++) {  
 edit\_tree(1, line[i].l, line[i].r, line[i].mark);  
 res += abs(pre - tree[1].len);  
 pre = tree[1].len;  
// 统计横边  
 res += 2 \* tree[1].c \* (line[i + 1].h - line[i].h);  
// 统计纵边  
 }  
 res += line[n].r - line[n].l;  
// 特判一下枚举不到的最后一条扫描线  
 printf("%d", res);  
 return 0;  
}

## 线段树合并

**其主要用于合并一堆权值线段树，与动态开点结合**

其主要写法有两种。

int merge(int a,int b,int x,int y) {  
 if(!a) return b;if(!b) return a;  
 if(x==y) {d[a]+=d[b];t[a]=x;return a;}  
 int mid=x+y>>1;  
 l[a]=merge(l[a],l[b],x,mid),r[a]=merge(r[a],r[b],mid+1,y);  
 pushup(a);return a;  
}

把b合并到a上。

但是我们这样直接把b合并过来的话，在以后继续合并a的时候可能合并过程就会破坏b的结构，所以这种方法只适合离线下来，合并完成之后立即询问。

我们也可以像主席树那样，合并不在原来的树上，而是新开节点，这样就不需要离线了，我们可以一边询问一遍用。

int merge(int a,int b,int x,int y) {  
 if(!a) return b;if(!b) return a;  
 int root=++cnt;  
 if(x==y) {d[root]=d[a]+d[b];t[root]=x;return root;}  
 int mid=x+y>>1;  
 l[root]=merge(l[a],l[b],x,mid),r[root]=merge(r[a],r[b],mid+1,y);  
 pushup(root);return root;  
}

缺点就是非常炸空间。

我们假设初始化是n次，操作m次，维护的线段树长度为len

则建立的线段树节点，第一个的空间复杂度为O(nlogn)，第二个空间复杂度即为O((n+m)loglen)。

### 板子

我们给一道例题

### [Vani有约会]雨天的尾巴 /【模板】线段树合并

#### 题目描述

首先村落里的一共有 座房屋，并形成一个树状结构。然后救济粮分 次发放，每次选择两个房屋 ，然后对于 到 的路径上(含 和 )每座房子里发放一袋 类型的救济粮。

然后深绘里想知道，当所有的救济粮发放完毕后，每座房子里存放的最多的是哪种救济粮。

#### 输入格式

输入的第一行是两个用空格隔开的正整数，分别代表房屋的个数 和救济粮发放的次数 。

第 到 第 行，每行有两个用空格隔开的整数 ，代表存在一条连接房屋 和 的边。

第 到第 行，每行有三个用空格隔开的整数 ，代表一次救济粮的发放是从 到 路径上的每栋房子发放了一袋 类型的救济粮。

#### 输出格式

输出 行，每行一个整数，第 行的整数代表 号房屋存放最多的救济粮的种类，如果有多种救济粮都是存放最多的，输出种类编号最小的一种。

如果某座房屋没有救济粮，则输出 。

* 对于 的数据，保证 。
* 对于 的数据，保证 。
* 对于 测试数据，保证 ，，。

#### 分析

我们直接对每一个节点，建立一颗权值线段树。

离线计算。利用差分完成区间加。

则对于某一个点，其上某一种救济粮的数量，即为其子树所有对应位置的线段树的合并。

#### AC\_code

#include<bits/stdc++.h>  
#define ios ios::sync\_with\_stdio(false); cin.tie(0), cout.tie(0)  
using namespace std;  
const int N = 1e5 + 10,M = N<<1;  
int h[N],e[M],ne[M],idx;  
int sz[N],son[N],dep[N],fa[N];  
int top[N];  
struct Node  
{  
 int l,r;  
 int mx,col;  
}tr[N\*60];  
struct Ask  
{  
 int x,y,z;  
};  
int ans[N],root[N];  
vector<Ask> op;  
int n,m,R,cnt;  
  
void add(int a,int b)  
{  
 e[idx] = b,ne[idx] = h[a],h[a] = idx++;  
}   
  
void dfs1(int u,int pa)  
{  
 sz[u] = 1;  
 for(int i=h[u];~i;i=ne[i])  
 {  
 int j = e[i];  
 if(j==pa) continue;  
 fa[j] = u;dep[j] = dep[u] + 1;  
 dfs1(j,u);  
 sz[u] += sz[j];  
 if(sz[j]>sz[son[u]]) son[u] = j;  
 }  
}  
  
void dfs2(int u,int tp)  
{  
 top[u] = tp;  
 if(!son[u]) return ;  
 dfs2(son[u],tp);  
 for(int i=h[u];~i;i=ne[i])  
 {  
 int j = e[i];  
 if(j==fa[u]||j==son[u]) continue;  
 dfs2(j,j);  
 }  
}  
  
int lca(int u,int v)  
{  
 while(top[u]!=top[v])  
 {  
 if(dep[top[u]]<dep[top[v]]) swap(u,v);  
 u = fa[top[u]];  
 }  
 return dep[u] < dep[v]?u:v;  
}  
  
void pushup(int u)  
{  
 if(tr[tr[u].l].mx>=tr[tr[u].r].mx) tr[u].col = tr[tr[u].l].col,tr[u].mx = tr[tr[u].l].mx;  
 else tr[u].col = tr[tr[u].r].col,tr[u].mx = tr[tr[u].r].mx;  
}  
  
void modify(int &u,int l,int r,int x,int k)  
{  
 if(!u) u = ++cnt;  
 if(l==r)  
 {  
 tr[u].mx += k;  
 tr[u].col = x;  
 return ;  
 }  
 int mid = l + r >> 1;  
 if(x<=mid) modify(tr[u].l,l,mid,x,k);  
 else modify(tr[u].r,mid+1,r,x,k);  
 pushup(u);  
}  
  
int merge(int u,int v,int l,int r)  
{  
 if(!u) return v;if(!v) return u;  
 if(l==r)   
 {  
 tr[u].mx += tr[v].mx;  
 tr[u].col = l;  
 return u;  
 }  
 int mid = l + r >> 1;  
 tr[u].l = merge(tr[u].l,tr[v].l,l,mid),tr[u].r = merge(tr[u].r,tr[v].r,mid+1,r);  
 pushup(u);return u;  
}  
  
void dfs(int u)  
{  
 for(int i=h[u];~i;i=ne[i])  
 {  
 int j = e[i];  
 if(j==fa[u]) continue;  
 dfs(j);  
 root[u] = merge(root[u],root[j],1,R);  
 }  
 if(tr[root[u]].mx) ans[u] = tr[root[u]].col;  
}  
  
int main()  
{  
 ios;cin>>n>>m;  
 memset(h,-1,sizeof h);  
 for(int i=0;i<n-1;i++)  
 {  
 int u,v;cin>>u>>v;  
 add(u,v),add(v,u);  
 }  
 dfs1(1,-1),dfs2(1,1);  
 for(int i=0;i<m;i++)  
 {  
 int x,y,z;cin>>x>>y>>z;  
 R = max(R,z);  
 op.push\_back({x,y,z});  
 }  
 for(auto [x,y,z]:op)  
 {  
 int anc = lca(x,y);  
 modify(root[x],1,R,z,1);modify(root[y],1,R,z,1);  
 modify(root[anc],1,R,z,-1);if(fa[anc]) modify(root[fa[anc]],1,R,z,-1);  
 }  
 dfs(1);  
 for(int i=1;i<=n;i++) cout<<ans[i]<<'\n';  
 return 0;  
}

## 线段树分裂

用处为，用在维护权值线段树，其中维护的是具体i的个数，区间的则是总和。

其中的关键代码为

其中假设，u为原树，v为分裂出的新的树，k则为，我们要将原树中前总和为k的部分保留，后面的部分拆分出来给v的新树。

void split(int u,int &v,ll k)  
{  
 if(!u) return ;  
 v = ++idx;  
 ll cnt = tr[tr[u].l].cnt;  
 if(k>cnt) split(tr[u].r,tr[v].r,k-cnt);  
 else swap(tr[u].r,tr[v].r);  
 if(k<cnt) split(tr[u].l,tr[v].l,k);  
 tr[v].cnt = tr[u].cnt - k;  
 tr[u].cnt = k;  
}

### 模板

给出一个可重集 （编号为 ），它支持以下操作：

0 p x y：将可重集 中大于等于 且小于等于 的值移动到一个新的可重集中（新可重集编号为从 开始的正整数，是上一次产生的新可重集的编号+1）。

1 p t：将可重集 中的数放入可重集 ，且清空可重集 （数据保证在此后的操作中不会出现可重集 ）。

2 p x q：在 这个可重集中加入 个数字 。

3 p x y：查询可重集 中大于等于 且小于等于 的值的个数。

4 p k：查询在 这个可重集中第 小的数，不存在时输出 -1。

#include<bits/stdc++.h>  
#define ios ios::sync\_with\_stdio(false); cin.tie(0), cout.tie(0)  
using namespace std;  
using ll = long long;  
const int N = 4e5 + 10;  
  
struct Node  
{  
 int l,r;  
 ll cnt;  
}tr[N\*30];  
  
int n,m,idx;  
int root[N],cnt;  
  
void pushup(int u)  
{  
 tr[u].cnt = tr[tr[u].l].cnt + tr[tr[u].r].cnt;  
}  
  
void modify(int &u,int l,int r,int x,int k)  
{  
 if(!u) u = ++idx;  
 if(l==r)   
 {  
 tr[u].cnt+=k;  
 return ;  
 }  
 int mid = l + r >> 1;  
 if(x<=mid) modify(tr[u].l,l,mid,x,k);  
 else modify(tr[u].r,mid+1,r,x,k);  
 pushup(u);  
}  
  
int merge(int u,int v,int l,int r)  
{  
 if(!u) return v;if(!v) return u;  
 if(l==r)  
 {  
 tr[u].cnt += tr[v].cnt;  
 return u;  
 }  
 int mid = l + r >> 1;  
 tr[u].l = merge(tr[u].l,tr[v].l,l,mid);tr[u].r = merge(tr[u].r,tr[v].r,mid+1,r);  
 pushup(u);  
 return u;  
}  
  
void split(int u,int &v,ll k)  
{  
 if(!u) return ;  
 v = ++idx;  
 ll cnt = tr[tr[u].l].cnt;  
 if(k>cnt) split(tr[u].r,tr[v].r,k-cnt);  
 else swap(tr[u].r,tr[v].r);  
 if(k<cnt) split(tr[u].l,tr[v].l,k);  
 tr[v].cnt = tr[u].cnt - k;  
 tr[u].cnt = k;  
}  
  
ll query(int u,int l,int r,int x,int y)  
{  
 if(!u) return 0;  
 if(x>r||y<l) return 0;  
 if(x<=l&&r<=y) return tr[u].cnt;  
 int mid = l + r >> 1;  
 return query(tr[u].l,l,mid,x,y) + query(tr[u].r,mid+1,r,x,y);  
}  
  
int queryk(int u,int l,int r,int k)  
{  
 if(l==r) return l;  
 int mid = l + r >> 1;  
 if(tr[tr[u].l].cnt>=k) return queryk(tr[u].l,l,mid,k);  
 return queryk(tr[u].r,mid+1,r,k-tr[tr[u].l].cnt);  
}  
  
int main()  
{  
 ios;cin>>n>>m;  
 for(int i=1;i<=n;i++)  
 {  
 int x;cin>>x;  
 modify(root[1],1,n,i,x);  
 }  
 cnt = 1;  
 while(m--)  
 {  
 int op,x,y,z;  
 cin>>op>>x>>y;  
 if(!op)  
 {  
 cin>>z;  
 ll q1 = query(root[x],1,n,1,z),q2 = query(root[x],1,n,y,z);  
 int a = 0;  
 split(root[x],root[++cnt],q1-q2);  
 split(root[cnt],a,q2);  
 root[x] = merge(root[x],a,1,n);  
 }  
 else if(op==1) root[x] = merge(root[x],root[y],1,n);  
 else if(op==2)   
 {  
 cin>>z;  
 modify(root[x],1,n,z,y);  
 }  
 else if(op==3)   
 {  
 cin>>z;  
 cout<<query(root[x],1,n,y,z)<<'\n';  
 }  
 else   
 {  
 if(tr[root[x]].cnt<y) cout<<"-1\n";  
 else cout<<queryk(root[x],1,n,y)<<'\n';  
 }  
 }  
 return 0;  
}

## 二维线段树

### 题目描述

最近，有人发明了一种三维版的俄罗斯方块。和二维版本类似，一些立方体按照一定的顺序掉落，直到碰到别的方块或是地面才会停止掉落。立方体停止掉落后会一直保持掉落时的位置，直到游戏结束。

你的朋友决定以这个新版本的俄罗斯方块为背景，出一道题。给出每个立方体的掉落顺序和其掉落的轨迹，在所有方块完成掉落后求出最高方块的高度。在这个游戏中，方块均垂直下落，且方块不会旋转或翻转。为了方便描述，我们会建立一个空间直角坐标系，该坐标系的原点为地面的一角，并且坐标轴与地面边缘平行。

现在轮到你解决这个问题了。

### 输入格式

第一行三个整数 ，分别为地面的长度，宽度，和将要掉落的立方体数量。

接下来 行，每行五个整数 ，描述一个掉落的立方体。其中 分别代表立方体的长，宽，高。立方体的底面（即长 宽的那一面）将正对地面。立方体底面四个角在地面的投影坐标分别为 ，，，。

### 输出格式

输出一个整数，即方块掉落结束后最高方块的高度。

### 数据范围

，，，，，。

**空间复杂度**

**时间复杂度**

### AC\_code

#include<iostream>  
#include<cstring>  
#include<cstdio>  
#define maxn 1010  
#define slu ((o<<2)+1)  
#define sru ((o<<2)+2)  
#define sld ((o<<2)+3)  
#define srd ((o<<2)+4)  
#define midx ((l+r)>>1)  
#define midy ((u+d)>>1)  
using namespace std;  
  
  
struct segtree{  
 int maxx,lazy;  
}tree[(maxn<<2)\*(maxn<<2)];  
  
  
inline void maki(int &s)  
{  
 s=0;char c=getchar();while(c>'9' || c<'0') c=getchar();  
 while(c<='9' && c>='0')s\*=10,s+=c-'0',c=getchar();return;  
}  
inline int honoka(int a,int b)  
{return a>b?a:b;}  
  
  
inline void buildtree(int l,int r,int u,int d,int o)  
{  
 tree[o].lazy=0;  
   
 if(l==r && u==d)  
 {  
 tree[o].maxx=0;  
 return;  
 }  
   
 buildtree(l,midx,u,midy,slu);  
 tree[o].maxx=honoka(tree[o].maxx,tree[slu].maxx);  
   
 if(l!=r)  
 {  
 buildtree(midx+1,r,u,midy,sru);  
 tree[o].maxx=honoka(tree[o].maxx,tree[sru].maxx);  
   
 if(u!=d)//此处隐含了l!=r这个条件  
 {  
 buildtree(midx+1,r,midy+1,d,srd);  
 tree[o].maxx=honoka(tree[o].maxx,tree[srd].maxx);  
 }  
 }  
   
 if(u!=d)  
 {  
 buildtree(l,midx,midy+1,d,sld);  
 tree[o].maxx=honoka(tree[o].maxx,tree[sld].maxx);  
 }  
 return;  
}  
  
  
inline void pushdown(int l,int r,int u,int d,int o)  
{  
 if(tree[o].lazy==0)  
 return;  
   
 tree[slu].maxx=tree[o].lazy;  
 tree[slu].lazy=tree[o].lazy;  
   
 if(l!=r)  
 {  
 tree[sru].maxx=tree[o].lazy;  
 tree[sru].lazy=tree[o].lazy;  
   
 if(u!=d)//同建树里面的结构优化  
 {  
 tree[srd].maxx=tree[o].lazy;  
 tree[srd].lazy=tree[o].lazy;  
 }  
 }  
   
 if(u!=d)  
 {  
 tree[sld].maxx=tree[o].lazy;  
 tree[sld].lazy=tree[o].lazy;  
 }  
 tree[o].lazy=0;  
 return;  
}  
  
  
inline void update(int ql,int qr,int qu,int qd,int l,int r,int u,int d,int o,int add)  
{  
 if(l>=ql && r<=qr && u>=qu && d<=qd)  
 {  
 tree[o].maxx=add;  
 tree[o].lazy=add;  
 return;  
 }  
   
 pushdown(l,r,u,d,o);  
   
 if(ql<=midx)  
 {  
 if(qu<=midy)  
 {  
 update(ql,qr,qu,qd,l,midx,u,midy,slu,add);  
 tree[o].maxx=honoka(tree[slu].maxx,tree[o].maxx);  
 }  
   
 if(qd>midy && u!=d)  
 {  
 update(ql,qr,qu,qd,l,midx,midy+1,d,sld,add);  
 tree[o].maxx=honoka(tree[sld].maxx,tree[o].maxx);  
 }  
 }  
   
 if(qr>midx && l!=r)  
 {  
 if(qu<=midy)  
 {  
 update(ql,qr,qu,qd,midx+1,r,u,midy,sru,add);  
 tree[o].maxx=honoka(tree[sru].maxx,tree[o].maxx);  
 }  
   
 if(qd>midy && u!=d)  
 {  
 update(ql,qr,qu,qd,midx+1,r,midy+1,d,srd,add);  
 tree[o].maxx=honoka(tree[srd].maxx,tree[o].maxx);  
 }  
 }  
 return;  
}  
  
  
inline int query(int ql,int qr,int qu,int qd,int l,int r,int u,int d,int o)  
{  
 if(l>=ql && r<=qr && u>=qu && d<=qd)return tree[o].maxx;  
 pushdown(l,r,u,d,o);  
 int ans=-1;  
   
 if(ql<=midx)  
 {  
 if(qu<=midy)  
 ans=honoka(query(ql,qr,qu,qd,l,midx,u,midy,slu),ans);  
   
 if(qd>midy && u!=d)  
 ans=honoka(query(ql,qr,qu,qd,l,midx,midy+1,d,sld),ans);  
 }  
   
 if(qr>midx)  
 {  
 if(l!=r)  
 {  
 if(qu<=midy)  
 ans=honoka(query(ql,qr,qu,qd,midx+1,r,u,midy,sru),ans);  
   
 if(qd>midy && u!=d)  
 ans=honoka(query(ql,qr,qu,qd,midx+1,r,midy+1,d,srd),ans);  
 }  
 }  
 return ans;  
}  
  
  
int main()  
{  
 int D,S,N;  
 maki(D),maki(S),maki(N);  
 while(N--)  
 {  
 int d,s,h,x,y;  
 maki(d),maki(s),maki(h),maki(x),maki(y);  
 int add=query(x+1,x+d,y+1,y+s,1,D,1,S,0);  
 add+=h;  
 update(x+1,x+d,y+1,y+s,1,D,1,S,0,add);  
 }  
 printf("%d\n",query(1,D,1,S,1,D,1,S,0));  
 return 0;  
}

## 区间加，区间覆盖维护历史最值

**求历史最值，我们都考虑能否将多个懒标记糅合成一个懒标记**

给出 **T** 个数，**E** 个操作（）

* Q X Y：询问从 **X** 到 **Y** 的**当前**最大值
* A X Y：询问从 **X** 到 **Y** 的**历史**最大值（出现过的最大数）
* P X Y Z：将 **X** 到 **Y** 这段区间加 **Z**
* C X Y Z：将 **X** 到 **Y** 这段区间赋值为 **Z**

#include<bits/stdc++.h>  
using namespace std;  
const int N = 1e5 + 10;  
struct Node   
{  
 int l,r,mx,his;  
 bool iscov;//是有区间覆盖  
 int add,cov;//从上次清空后，区间加，区间覆盖累积的值  
 int mxadd,mxcov;//从上次清空后，区间加，区间覆盖所积累过程中的最大值  
 //所以其实这里用了四个懒标记，下边两个的作用，我会尽量在下边说清楚  
}tr[N<<2];  
int n,m;  
  
void pushup(int u)//pushup非常好说，就直接用子节点更新父节点即可  
{  
 tr[u].mx = max(tr[u<<1].mx,tr[u<<1|1].mx);  
 tr[u].his = max(tr[u<<1].his,tr[u<<1|1].his);  
}  
  
void build(int u,int l,int r)  
{  
 if(l==r)  
 {  
 int x;  
 scanf("%d",&x);  
 tr[u] = {l,r,x,x};  
 return ;  
 }  
 tr[u] = {l,r};  
 int mid = l + r >> 1;  
 build(u<<1,l,mid),build(u<<1|1,mid+1,r);  
 pushup(u);  
}  
  
/\*\*  
 \* 在这里，我还是要反复说，将复杂的懒标记操作直接封装成函数。  
 \* 这样会大大减少我们犯错误的可能。  
 \* 在下传懒标记的时候，我们要考虑对所有维护的变量的影响。  
 \* 记住，向下传递的操作是，用父区间改变子区间  
 \*/  
  
void changeadd(Node &u,int k,int mxk)//对区间加操作的实现，k为这次区间加的值，而mxk则为父区间历史最大区间加的值，  
{  
 if(u.iscov)//判断该区间是否已经有赋值操作了。  
 {  
 //若有，则接下来直接将区间加操作，改为对区间赋值操作的改变  
 u.mxcov = max(u.mxcov,u.cov+mxk);//更新子区间历史最大赋值  
 u.his = max(u.his,u.mx+mxk);//更新子区间历史最大值  
 u.mx += k;//加上本次要加的值  
 u.cov += k;//加上本次要加的值  
 }  
 else  
 {  
 //若没有就继续累加区间加操作。  
 u.mxadd = max(u.mxadd,u.add+mxk);//更新子区间历史最大加和值  
 u.his = max(u.his,u.mx+mxk);//更新子区间历史最大值  
 u.mx += k;//加上本次要加的值  
 u.add += k;//加上本次要加的值  
 }  
}  
  
void changecov(Node &u,int k,int mxk)//对区间加操作的实现，k为这次区间赋值的值，而mxk则为父区间历史最大赋值的值，  
{  
 if(u.iscov)  
 {  
 //若已经被赋值  
 u.mxcov = max(u.mxcov,mxk);//则更新子区间的历史最大赋值  
 u.his = max(u.his,mxk);//更新子区间的历史最大值  
 }  
 else  
 {  
 u.iscov = 1;//标记已经覆盖  
 u.mxcov = mxk;//第一次覆盖，则对于子区间来说，历史最大覆盖就是mxk  
 u.his = max(u.his,mxk);//更新子区间的历史最大值  
 }  
 u.mx = u.cov = k;//更新一下mx和cov  
}  
  
void pushdown(int u)  
{  
 auto &root = tr[u],&left = tr[u<<1],&right = tr[u<<1|1];//在我们合并后的操作序列中，先加后赋值  
 changeadd(left,root.add,root.mxadd);  
 changeadd(right,root.add,root.mxadd);  
 root.mxadd = root.add = 0;//操作后，清空两个标记  
 if(root.iscov)//是否有覆盖  
 {  
 changecov(left,root.cov,root.mxcov);  
 changecov(right,root.cov,root.mxcov);  
 root.iscov = 0;//更新后，标记无覆盖了。  
 //这里没有清空关于区间覆盖的两个值，因为没必要，我们都是直接改变区间值的。  
 }  
}  
  
void modifyadd(int u,int l,int r,int k)  
{  
 if(l<=tr[u].l&&tr[u].r<=r)  
 {  
 changeadd(tr[u],k,k);//改变该区间的和，可以直接当做一个历史加和最大值为k，且本次要加的也为k的父区间  
 return ;  
 }  
 pushdown(u);  
 int mid = tr[u].l + tr[u].r >> 1;  
 if(l<=mid) modifyadd(u<<1,l,r,k);  
 if(r>mid) modifyadd(u<<1|1,l,r,k);  
 pushup(u);  
}  
  
void modifycov(int u,int l,int r,int k)  
{  
 if(l<=tr[u].l&&tr[u].r<=r)  
 {  
 changecov(tr[u],k,k);//与上同理  
 return ;  
 }  
 pushdown(u);  
 int mid = tr[u].l + tr[u].r >> 1;  
 if(l<=mid) modifycov(u<<1,l,r,k);  
 if(r>mid) modifycov(u<<1|1,l,r,k);  
 pushup(u);  
}  
  
int querymx(int u,int l,int r)  
{  
 if(l<=tr[u].l&&tr[u].r<=r) return tr[u].mx;  
 pushdown(u);  
 int mid = tr[u].l + tr[u].r >> 1;  
 int res = -INT\_MAX;  
 if(l<=mid) res = max(res,querymx(u<<1,l,r));  
 if(r>mid) res = max(res,querymx(u<<1|1,l,r));  
 return res;  
}  
  
int queryhis(int u,int l,int r)  
{  
 if(l<=tr[u].l&&tr[u].r<=r) return tr[u].his;  
 pushdown(u);  
 int mid = tr[u].l + tr[u].r >> 1;  
 int res = -INT\_MAX;  
 if(l<=mid) res = max(res,queryhis(u<<1,l,r));  
 if(r>mid) res = max(res,queryhis(u<<1|1,l,r));  
 return res;  
}  
  
int main()  
{  
 scanf("%d",&n);  
 build(1,1,n);  
 scanf("%d",&m);  
 while(m--)  
 {  
 char op[2];  
 int x,y,z;  
 scanf("%s%d%d",op,&x,&y);  
 if(\*op=='Q') printf("%d\n",querymx(1,x,y));  
 else if(\*op=='A') printf("%d\n",queryhis(1,x,y));  
 else if(\*op=='P')   
 {  
 scanf("%d",&z);  
 modifyadd(1,x,y,z);  
 }  
 else   
 {  
 scanf("%d",&z);  
 modifycov(1,x,y,z);  
 }  
 }  
 return 0;  
}

## 区间取min/max

**时间复杂度**

### 区间取min

我们其实无法做到快速的去区间取min，暴力的去求，我们只能递归到叶节点才能更新。

但是这样我们的时间复杂度，最坏情况下，单次修改将会是的。

我们考虑换个思路，**将区间内大于v的数都减去一个数，将这些数都变为v。**

因为不同的数要减去不同的数才能等于v，显然我们不能设很多的tag来表示区间内不同的数要减去的数。

那我们退一步来说，**如果区间内大于v的数只有一种**，我们可以怎么做呢？

只需要维护一个tag就好了，我们只用知道这一个数需要减的值就好了。

那怎样才能使得区间内大于v的数只有一种呢？

只需要多递归几次就行了。

我们在线段树每个节点设以下三个变量，分别为：

* mx：该区间的最大值
* se：该区间的严格次大值
* cnt：该区间的最大值的个数

因为只有在区间只有一种数大于k的时候我们才能快速更新，即只有在满足的节点上更新。

struct Node  
{  
 int l,r;  
 int mx,se,cnt;  
 int mxadd,add;  
}tr[N<<2];  
int n,m,a[N];  
  
void pushup(int u)  
{  
 auto &root = tr[u],&left = tr[u<<1],&right = tr[u<<1|1];  
 root.mx = max(left.mx,right.mx);  
 if(right.mx==left.mx)  
 {  
 root.se = max(left.se,right.se);   
 root.cnt = left.cnt + right.cnt;  
 }  
 else if(left.mx>right.mx)  
 {  
 root.se = max(left.se,right.mx);  
 root.cnt = left.cnt;  
 }  
 else   
 {  
 root.se = max(left.mx,right.se);  
 root.cnt = right.cnt;  
 }  
}  
  
void build(int u,int l,int r)  
{  
 tr[u] = {l,r};  
 if(l==r)  
 {  
 tr[u].mx = a[l];  
 tr[u].cnt = 1;tr[u].se = -inf;   
 return ;  
 }  
 int mid = l + r >> 1;  
 build(u<<1,l,mid),build(u<<1|1,mid+1,r);  
 pushup(u);  
}  
  
void change(Node &u,int mxadd,int add)  
{  
 u.mx += mxadd;//区间最值变化为u.mx+mxadd  
 if(u.se!=-inf) u.se += add;//若存在次小值，区间次大值变为u.se + add  
 u.mxadd += mxadd,u.add += add;//接下来，更新当前区间此时的u.mxadd与u.add  
}  
  
void pushdown(int u)  
{  
 auto &root = tr[u],&left = tr[u<<1],&right = tr[u<<1|1];  
 if(root.mxadd||root.add){  
 int mx = max(left.mx,right.mx);   
 //如果子区间的最值等于此时区间的最值，则用来更新的子区间的mxadd标记就用root.mxadd  
 //否则就，用root.add。  
 if(left.mx==mx) change(left,root.mxadd,root.add);  
 else change(left,root.add,root.add);  
 if(right.mx==mx) change(right,root.mxadd,root.add);  
 else change(right,root.add,root.add);  
 root.mxadd = root.add = 0;  
 }  
}  
  
void modfiymin(int u,int l,int r,int k)  
{  
 if(l>tr[u].r||tr[u].l>r||k>=tr[u].mx) return ;//若此时区间的最大值已经小于等于k，直接返回  
 //只有当满足k>tr[u].se时才更新  
 //更新当前区间，更新标记为mxadd = k-tr[u].mx,add = 0,hismxadd = k-tr[u].mx,hisadd = 0;  
 //这里只对最大值操作，因为，只有当se<k<mx时，我们才对区间内所有的最大值操作，让他减去(tr[u].mx-k)  
 if(l<=tr[u].l&&tr[u].r<=r&&tr[u].se<k) return change(tr[u],k-tr[u].mx,0);  
 pushdown(u);  
 modfiymin(u<<1,l,r,k),modfiymin(u<<1|1,l,r,k);  
 pushup(u);  
}

### 区间取max

非常类似的操作。

只需要多递归几次就行了。

我们在线段树每个节点设以下三个变量，分别为：

* mi：该区间的最小值
* se：该区间的严格次小值
* cnt：该区间的最小值的个数

因为只有在区间只有一种数小于k的时候我们才能快速更新，即只有在满足的节点上更新。

struct Node  
{  
 int l,r;  
 int mi,se,cnt;  
 int miadd,add;  
}tr[N<<2];  
int n,m,a[N];  
  
void pushup(int u)  
{  
 auto &root = tr[u],&left = tr[u<<1],&right = tr[u<<1|1];  
 root.mi = min(left.mi,right.mi);  
 if(right.mi==left.mi)  
 {  
 root.se = min(left.se,right.se);   
 root.cnt = left.cnt + right.cnt;  
 }  
 else if(left.mi<right.mi)  
 {  
 root.se = min(left.se,right.mi);  
 root.cnt = left.cnt;  
 }  
 else   
 {  
 root.se = min(left.mi,right.se);  
 root.cnt = right.cnt;  
 }  
}  
  
void build(int u,int l,int r)  
{  
 tr[u] = {l,r};  
 if(l==r)  
 {  
 tr[u].mi = a[l];  
 tr[u].cnt = 1;tr[u].se = inf;   
 return ;  
 }  
 int mid = l + r >> 1;  
 build(u<<1,l,mid),build(u<<1|1,mid+1,r);  
 pushup(u);  
}  
  
void change(Node &u,int miadd,int add)  
{  
 u.mi += miadd;//区间最值变化为u.mi+miadd  
 if(u.se!=inf) u.se += add;//若存在次小值，区间次大值变为u.se + add  
 u.miadd += miadd,u.add += add;//接下来，更新当前区间此时的u.miadd与u.add  
}  
  
void pushdown(int u)  
{  
 auto &root = tr[u],&left = tr[u<<1],&right = tr[u<<1|1];  
 if(root.miadd||root.add){  
 int mi = max(left.mi,right.mi);   
 //如果子区间的最值等于此时区间的最值，则用来更新的子区间的miadd标记就用root.miadd  
 //否则就，用root.add。  
 if(left.mi==mi) change(left,root.miadd,root.add);  
 else change(left,root.add,root.add);  
 if(right.mi==mi) change(right,root.miadd,root.add);  
 else change(right,root.add,root.add);  
 root.miadd = root.add = 0;  
 }  
}  
  
void modfiymax(int u,int l,int r,int k)  
{  
 if(l>tr[u].r||tr[u].l>r||k<=tr[u].mi) return ;//若此时区间的最小值已经大于等于k，直接返回  
 //只有当满足k<tr[u].se时才更新  
 //更新当前区间，更新标记为miadd = k - tr[u].mi,add = 0;  
 //这里只对最小值操作，因为，只有当mi<k<se时，我们才对区间内所有的最小值操作，让他加上k - tr[u].mi  
 if(l<=tr[u].l&&tr[u].r<=r&&tr[u].se>k) return change(tr[u],k-tr[u].mi,0);  
 pushdown(u);  
 modfiymax(u<<1,l,r,k),modfiymax(u<<1|1,l,r,k);  
 pushup(u);  
}

## 吉老师线段树

区间求和，区间取min，区间历史最值，区间最值

**modifymin时间复杂度是**

#include<bits/stdc++.h>  
#define ios ios::sync\_with\_stdio(false); cin.tie(0), cout.tie(0)  
using namespace std;  
using ll = long long;  
const int N = 5e5 + 10,inf = 2e9;  
/\*\*  
 \* @brief   
 \* sum 区间和  
 \* mx 区间最大值，se区间严格次大值，cnt区间最大值个数，hismx区间历史最大值  
 \* mxadd 区间最大值的懒标记 add 区间非最大值的懒标记 hismxadd 区间最大值的懒标记的历史最大值 hisadd 区间非最大值的懒标记的历史最大值  
 \*/  
struct Node  
{  
 int l,r;  
 ll sum;  
 int mx,se,cnt,hismx;  
 int mxadd,add,hismxadd,hisadd;  
}tr[N<<2];  
  
void pushup(int u)  
{  
 auto &root = tr[u],&left = tr[u<<1],&right = tr[u<<1|1];  
 root.sum = left.sum + right.sum;  
 root.mx = max(left.mx,right.mx);  
 root.hismx = max(left.hismx,right.hismx);  
 if(right.mx==left.mx)  
 {  
 root.se = max(left.se,right.se);   
 root.cnt = left.cnt + right.cnt;  
 }  
 else if(left.mx>right.mx)  
 {  
 root.se = max(left.se,right.mx);  
 root.cnt = left.cnt;  
 }  
 else   
 {  
 root.se = max(left.mx,right.se);  
 root.cnt = right.cnt;  
 }  
}  
  
void build(int u,int l,int r)  
{  
 tr[u] = {l,r};  
 if(l==r)  
 {  
 int x;cin>>x;  
 tr[u].sum = tr[u].mx = tr[u].hismx = x;  
 tr[u].cnt = 1;tr[u].se = -inf;   
 return ;  
 }  
 int mid = l + r >> 1;  
 build(u<<1,l,mid),build(u<<1|1,mid+1,r);  
 pushup(u);  
}  
  
void change(Node &u,int mxadd,int add,int hismxadd,int hisadd)  
{  
 u.sum += 1ll\*mxadd\*u.cnt + 1ll\*add\*(u.r - u.l + 1 - u.cnt);  
 u.hismx = max(u.hismx,u.mx + hismxadd);//历史最值更新是，用当前的最值+(mxadd的最大值)  
 u.mx += mxadd;//区间最值变化为u.mx+mxadd  
 if(u.se!=-inf) u.se += add;//若存在次小值，区间次大值变为u.se + add  
 u.hismxadd = max(u.hismxadd,u.mxadd + hismxadd);//hismxadd用当前区间此时的u.mxadd+hismxadd来更新  
 u.hisadd = max(u.hisadd,u.add + hisadd);//hisadd用当前区间此时的u.add+hisadd来更新  
 u.mxadd += mxadd,u.add += add;//接下来，更新当前区间此时的u.mxadd与u.add  
}  
  
void pushdown(int u)  
{  
 auto &root = tr[u],&left = tr[u<<1],&right = tr[u<<1|1];  
 if(root.mxadd||root.add||root.hismxadd||root.hisadd){  
 int mx = max(left.mx,right.mx);   
 //如果子区间的最值等于此时区间的最值，则用来更新的子区间的mxadd标记就用root.mxadd，hismxadd就用root.hismxadd  
 //否则就，用root.add与root.hisadd。  
 if(left.mx==mx) change(left,root.mxadd,root.add,root.hismxadd,root.hisadd);  
 else change(left,root.add,root.add,root.hisadd,root.hisadd);  
 if(right.mx==mx) change(right,root.mxadd,root.add,root.hismxadd,root.hisadd);  
 else change(right,root.add,root.add,root.hisadd,root.hisadd);  
 root.mxadd = root.add = root.hismxadd = root.hisadd = 0;  
 }  
}  
  
void modfiyadd(int u,int l,int r,int k)  
{  
 if(tr[u].l>r||tr[u].r<l) return ;  
 //更新当前区间，更新标记为mxadd = k,add = k,hismxadd = k,hisadd = k;  
 if(l<=tr[u].l&&tr[u].r<=r) return change(tr[u],k,k,k,k);  
 pushdown(u);  
 modfiyadd(u<<1,l,r,k),modfiyadd(u<<1|1,l,r,k);  
 pushup(u);  
}  
  
void modfiymin(int u,int l,int r,int k)  
{  
 if(l>tr[u].r||tr[u].l>r||k>=tr[u].mx) return ;//若此时区间的最大值已经小于等于k，直接返回  
 //只有当满足k>tr[u].se时才更新  
 //更新当前区间，更新标记为mxadd = k-tr[u].mx,add = 0,hismxadd = k-tr[u].mx,hisadd = 0;  
 //这里只对最大值操作，因为，只有当se<k<mx时，我们才对区间内所有的最大值操作，让他减去(tr[u].mx-k)  
 if(l<=tr[u].l&&tr[u].r<=r&&tr[u].se<k) return change(tr[u],k-tr[u].mx,0,k-tr[u].mx,0);  
 pushdown(u);  
 modfiymin(u<<1,l,r,k),modfiymin(u<<1|1,l,r,k);  
 pushup(u);  
}  
  
ll querysum(int u,int l,int r)  
{  
 if(tr[u].l>r||tr[u].r<l) return 0;  
 if(l<=tr[u].l&&tr[u].r<=r) return tr[u].sum;  
 pushdown(u);  
 return querysum(u<<1,l,r) + querysum(u<<1|1,l,r);  
}  
  
int querymx(int u,int l,int r)  
{  
 if(tr[u].l>r||tr[u].r<l) return -inf;  
 if(l<=tr[u].l&&tr[u].r<=r) return tr[u].mx;  
 pushdown(u);  
 return max(querymx(u<<1,l,r), querymx(u<<1|1,l,r));  
}  
  
int queryhismx(int u,int l,int r)  
{  
 if(tr[u].l>r||tr[u].r<l) return -inf;  
 if(l<=tr[u].l&&tr[u].r<=r) return tr[u].hismx;  
 pushdown(u);  
 return max(queryhismx(u<<1,l,r), queryhismx(u<<1|1,l,r));  
}  
  
int n,m;  
  
int main()  
{  
 ios;  
 cin>>n>>m;  
 build(1,1,n);  
 while(m--)  
 {  
 int op,l,r,k;cin>>op>>l>>r;  
 if(op==1)   
 {  
 cin>>k;  
 modfiyadd(1,l,r,k);  
 }  
 else if(op==2)  
 {  
 cin>>k;  
 modfiymin(1,l,r,k);  
 }  
 else if(op==3) cout<<querysum(1,l,r)<<'\n';  
 else if(op==4) cout<<querymx(1,l,r)<<'\n';  
 else cout<<queryhismx(1,l,r)<<'\n';  
 }  
 return 0;  
}

## 线段树优化建图

### [Legacy](https://www.luogu.com.cn/problem/CF786B)

#### 分析

模板题

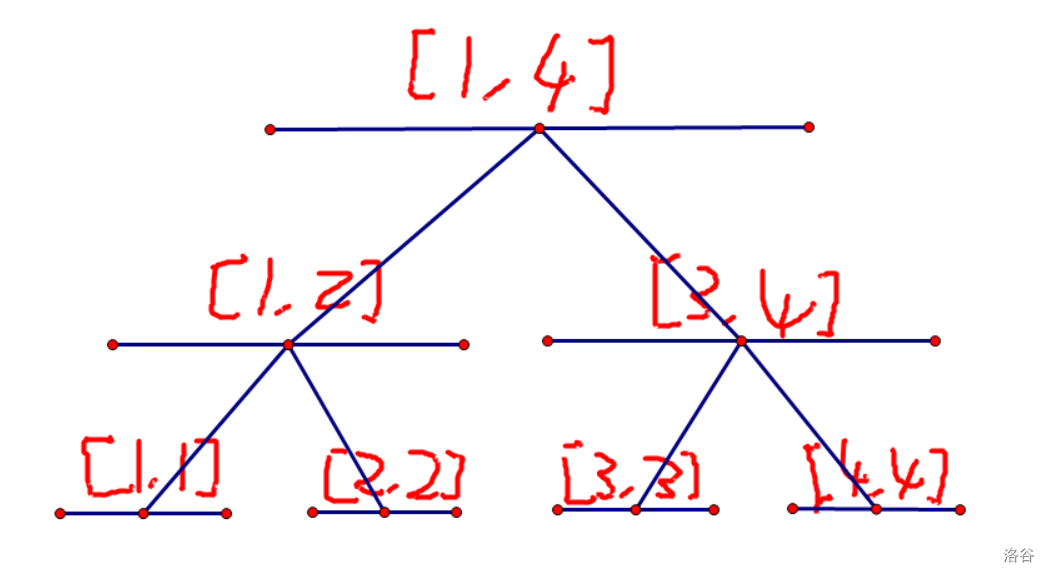
利用线段树完成优化建图。

如果按照题目的要求去建边，我们直接的不论是时间还是空间都炸了，。

我们看到其中的第一二个操作都是从**某个单点**向**区间**连边。

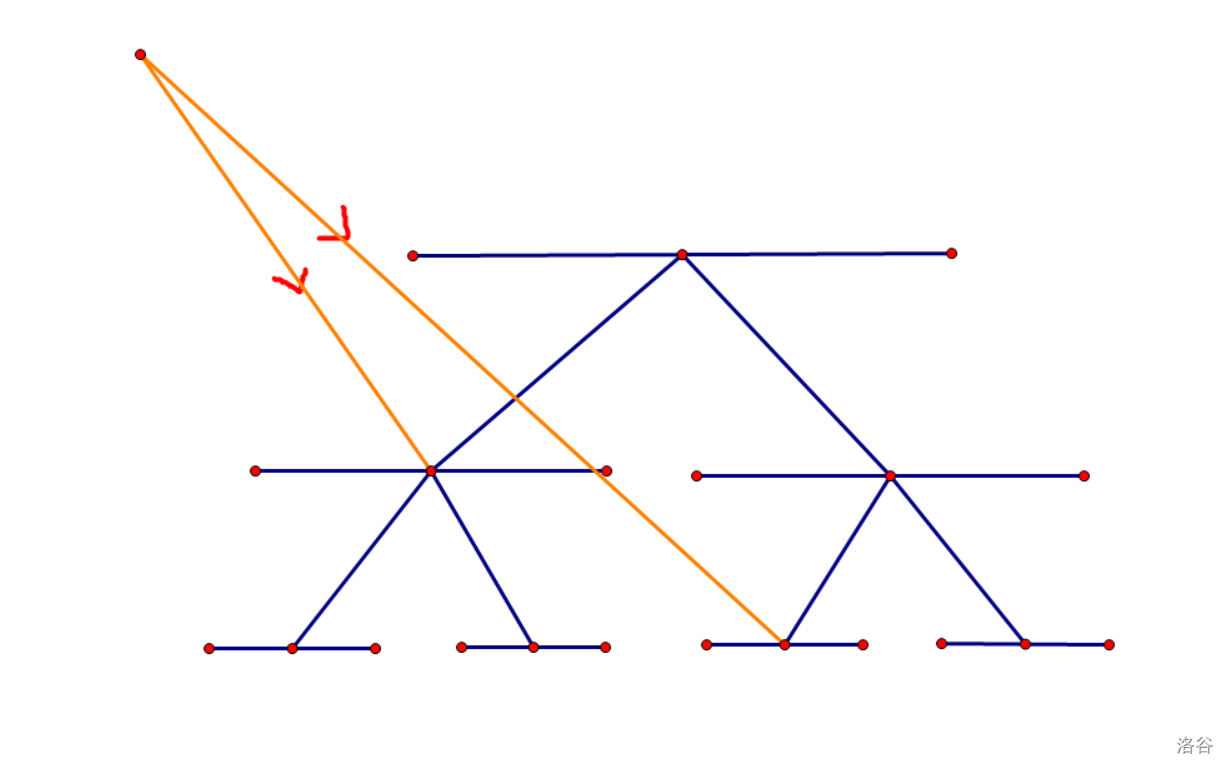
这种区间操作，我们考虑一下，能否利用，**每一个区间都可以表示为线段树上logn个区间**来减少边的个数。

我们就拿2操作来举例。现在假设假如有一个点要与[l,3]的点连边权为w的边，那么我们建出线段树：



img

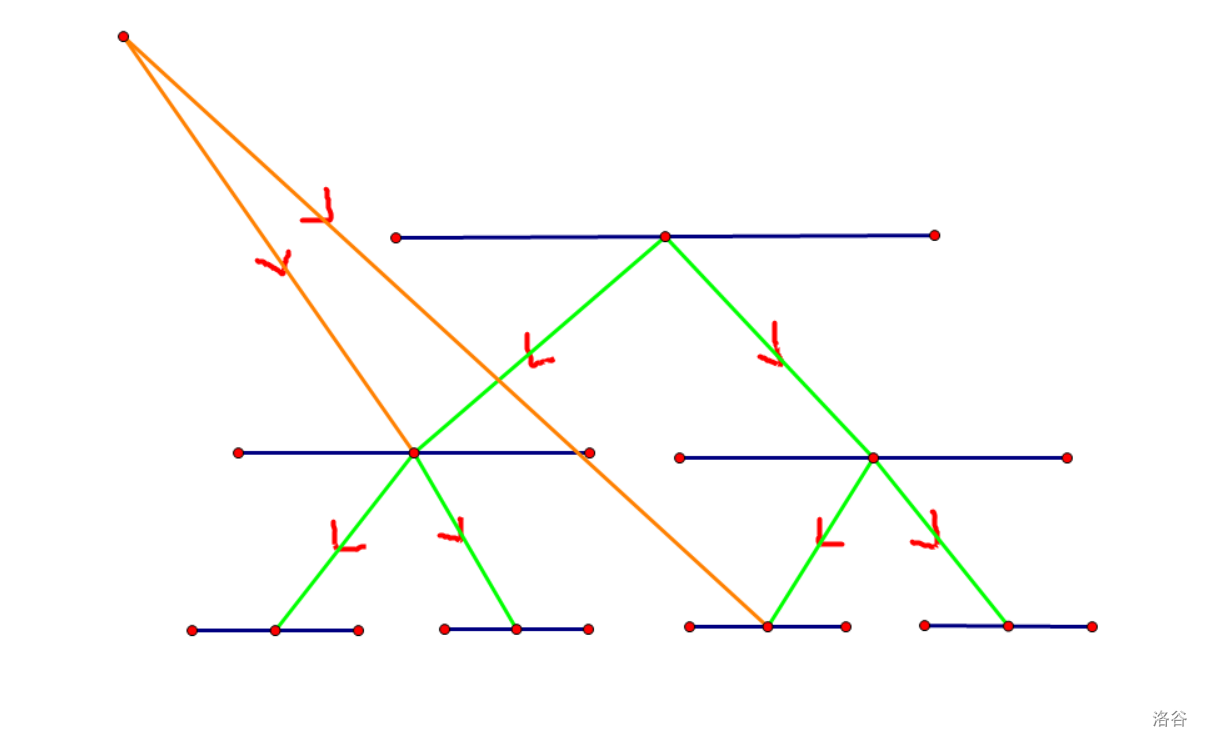
将[1,3]拆成[1,2]与[3,3]然后分别连边，边权为w（图中橙色的边）：



img

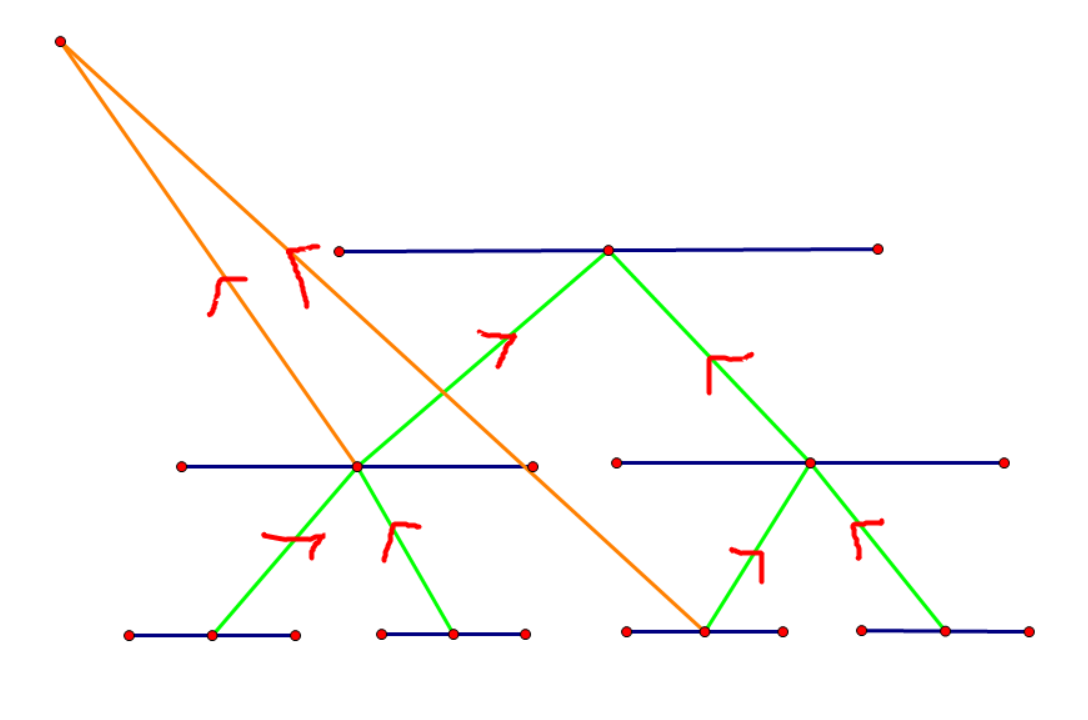
但是仅仅只连着两条边是肯定不够的，因为你只将这个点与**一个区间**表示的点连了边，并没有将其连到**具体的单点**上。

因此我们还从每个区间想起子区间连边，由于你向下走，从一个大区间对应到一个小区间**没有代价**，因此这些边的边权为0。



img

操作三也同理，只不过要将之前所有连的边都反向。

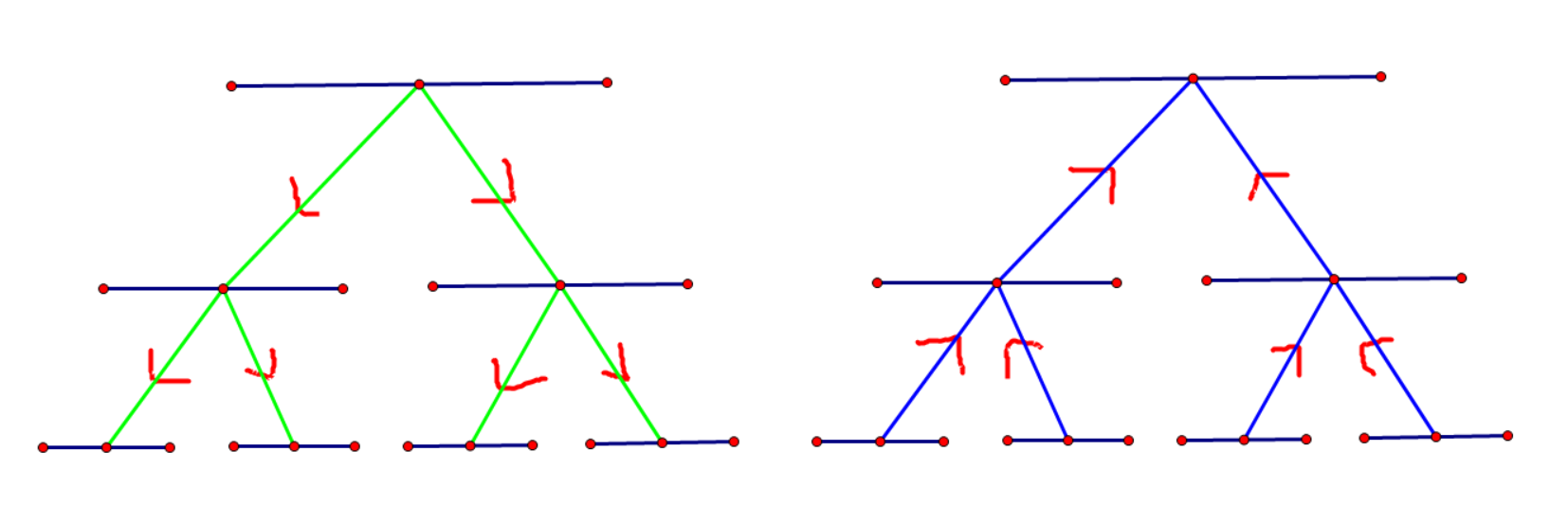


img

以上是操作2与操作3分开来考虑的情形，那么操作2与操作3相结合怎么办？

显然你不能把它们揉到一起去，因为你线段树上每条边向上向下边权都为0，故从源点到每个点的最短路也为0，这肯定是不太行。

因此我们建两颗线段树，第一棵树只连从上到下的边，第二课只连自下而上的边：



img

对于2操作，你就从第二课线段树的叶子结点向第一颗线段树中的对应区间连边（下边橙色的边）。

对于3操作，你就从第二课线段树的对应区间向第一课线段树中的叶子结点连边（下边粉色的边）。

**其实就是把右边的所有叶节点当做起点，左边的所有点当做起点，我们看能不能到达所有的点。**

#### AC\_code

#include<bits/stdc++.h>  
#define ios ios::sync\_with\_stdio(false); cin.tie(0), cout.tie(0)  
using namespace std;  
using ll = long long;  
typedef pair<ll,int> pii;  
const int N = 1e5 + 10,M = 1e6 + 10,D = 5e5;  
const ll inf = 0x3f3f3f3f3f3f3f3f;  
  
struct Node  
{  
 int l,r;  
}tr[N<<2];  
  
int n,m,st;  
ll dist[M];  
int leaf[N];  
bool vis[M];  
vector<pair<int,int>> g[M];  
  
void build(int u,int l,int r)  
{  
 tr[u] = {l,r};  
 if(l==r)  
 {  
 leaf[l] = u;  
 return ;  
 }  
 int mid = l + r >> 1;  
 g[u].push\_back({u<<1,0}),g[u].push\_back({u<<1|1,0});  
 g[(u<<1)+D].push\_back({u+D,0}),g[(u<<1|1)+D].push\_back({u+D,0});  
 build(u<<1,l,mid),build(u<<1|1,mid+1,r);  
}  
  
void connect(int u,int l,int r,int v,int w,int tp)  
{  
 if(l<=tr[u].l&&tr[u].r<=r)  
 {  
 if(tp) g[u+D].push\_back({v,w});  
 else g[v].push\_back({u,w});  
 return ;  
 }  
 int mid = tr[u].l + tr[u].r >> 1;  
 if(r<=mid) connect(u<<1,l,r,v,w,tp);  
 else if(l>mid) connect(u<<1|1,l,r,v,w,tp);  
 else connect(u<<1,l,mid,v,w,tp),connect(u<<1|1,mid+1,r,v,w,tp);   
}  
  
int main()  
{  
 ios;  
 cin>>n>>m>>st;  
 build(1,1,n);  
 for(int i=1;i<=m;i++)  
 {  
 int op;cin>>op;  
 if(op==1)   
 {  
 int v,u,w;cin>>v>>u>>w;  
 g[leaf[v]].push\_back({leaf[u],w});  
 }  
 else   
 {  
 int v,l,r,w;cin>>v>>l>>r>>w;  
 connect(1,l,r,leaf[v],w,op&1);  
 }  
 }  
 for(int i=1;i<=n;i++) g[leaf[i]].push\_back({leaf[i]+D,0}),g[leaf[i]+D].push\_back({leaf[i],0});  
 priority\_queue<pii,vector<pii>,greater<pii>> q;  
 memset(dist,0x3f,sizeof dist);  
 dist[leaf[st]+D] = 0;  
 q.push({0,leaf[st]+D});  
 while(q.size())  
 {  
 auto t = q.top();q.pop();  
 int ver = t.second;  
 if(vis[ver]) continue;  
 vis[ver] = 1;  
 for(auto it:g[ver])  
 {  
 int v = it.first,w = it.second;  
 if(dist[ver]+w<dist[v])  
 {  
 dist[v] = dist[ver] + w;  
 q.push({dist[v],v});  
 }  
 }  
 }  
 for(int i=1;i<=n;i++)  
 {  
 if(dist[leaf[i]]==inf) cout<<"-1 ";  
 else cout<<dist[leaf[i]]<<' ';  
 }  
}

# 树链剖分

int h[N],e[M],ne[M],idx;  
int w[N],fa[N],son[N],dep[N],sz[N];  
int nw[N],dfn[N],top[N],ts;  
int n,m;  
  
void add(int a,int b)  
{  
 e[idx]=b,ne[idx]=h[a],h[a]=idx++;  
}  
  
void dfs1(int u,int pa,int depth)  
{  
 dep[u]=depth,fa[u]=pa,sz[u]=1;  
 for(int i=h[u];~i;i=ne[i])  
 {  
 int j = e[i];  
 if(j==pa) continue;  
 dfs1(j,u,depth+1);  
 sz[u]+=sz[j];  
 if(sz[son[u]]<sz[j]) son[u]=j;  
 }  
}  
  
void dfs2(int u,int tp)  
{  
 dfn[u] = ++ts,nw[ts] = w[u],top[u]=tp;  
 if(!son[u]) return;  
 dfs2(son[u],tp);  
 for(int i=h[u];~i;i=ne[i])  
 {  
 int j = e[i];  
 if(son[u]==j||j==fa[u]) continue;  
 dfs2(j,j);  
 }  
}

## 边权化点权

**因为我们的树剖中维护的是点权，因此需要维护树上权值的时候，我们只需将边权放到这条边，深度较大的点即可。**

### Qtree1

考查知识点

**LCA+树剖+线段树**

倒也不算难，我们看到，要做树上的单点修改，并且需要动态的知道一条路径上的最大值。

那就是这三个知识点了，因为如果若是一段连续的区间，那就不用LCA了，直接线段树+树剖就可以做了

利用树剖，将树拆成区间问题，同时还能去做LCA。

这里想说的是，如何存边。

不难想到，我们将边的权值转化为维护节点的权值，每个节点维护的是其上面的那条边。

但是有一个小问题，此时我们就不能将LCA的点权纳入考虑了，因为LCA是其上边的边的边权。

解决方案也很好想，有树剖LCA的特点，因此只需分两种情况

1. 两个点不是一个点，那到最后时，两个点一定不会是一个点，此时只需要将深度较小的点(即LCA)的编号+1，再查询即可
2. 若是一个点，那就直接输出0（如果不加特判，将会死循环，因为l>r了）

## 树剖+线段树上二分

**题目通常要求我们在点u到root的路径间寻找距离点u最近（最远）的符合某些条件的点**

**因此，我们在搜索时，对区间的左子树和右子树是有判断条件的递归**

### 树

写这篇题解是因为好像用了个跟大家都不一样的思路去写了这道题目。

这题用的知识点依旧是：**树剖+线段树**

关键在于，我们维护的是什么？在这里，我们考虑维护**一个区间是否存在染色的点**

接下来，我们分别说说这两个操作中具体怎么操作。

#### 标记操作

这里很好说，我们只用进行单点修改即可

void modify(int u,int x)  
{  
 if(tr[u].l==tr[u].r)//当查询到对应结点时停止。  
 {  
 tr[u].st = 1;  
 return ;  
 }  
 int mid = tr[u].l + tr[u].r >> 1;  
 if(x<=mid) modify(u<<1,x);  
 else modify(u<<1|1,x);  
 pushup(u);  
}

#### 询问操作

接下来，就是重头戏了。我们如何进行询问？

在这里我用了类似于深搜的方式进行询问。

对于一个点y，我们想求其最近的被染色的祖先，需要我们不断的沿链向上回溯。

当然，这步很基础。接下来，我们具体说说如何实现搜索式的查询。

* **想找到距离y最近的被染色祖先，那我们在递归时就先从线段的右儿子找起（若右儿子有被染色的节点）则我们直接向其中递归，找到那个节点。**
* **否则，若是右儿子里，我们就去左儿子里面找（如果左儿子中有的话）**
* **若左右儿子都没有的话，我们直接返回-1表示该区间没有我们想要找到的点**

## 需要维护路径中多条链上的多个信息。

**对于某些题目，上跳过程中，维护两边目前已经跳过的部分的信息是常见手段，每次查询到当前段的信息和原有信息合并规则往往和线段树区间信息合并规则一致。**

**请注意，有些题目中，可以随意交换两边所维护的结构体（可能没懂我在说什么，写一些就知道了，hhh）**

### 染色

我们来根据操作来讨论一下，需要维护的值有什么。

**将节点 a 到节点 b 的路径上的所有点（包括 a 和 b）都染成颜色 c。**

很明显，我们需要维护一下tag，来保存该区间是否发生了**整体被某种颜色覆盖**

这并不困难，我们把眼光放到第二个操作上

**询问节点 a 到节点 b 的路径上的颜色段数量。**

此时，我们很明显需要维护一个sum，表示该段上不同颜色段的数量。

同时为了维护合并后区间的sum，我们需要维护两个值**lc**和**rc**分别表示左端点的颜色和右端点的颜色。

若合并区间时，左区间的右端点颜色 = 右区间的左端点颜色，则该区间的颜色段数量减1

同时需要注意的是，**在从一个条链跳到另外一条链时，可能会发生颜色连续的事情，从而使答案减1**

具体解决方案就是，可以在全局开一个**Lc**和**Rc**变量，用来记此时该条链的左端点颜色和右端点颜色。

同时维护两个变量**ans1**和**ans2**，用来分别统计u的上一条链的左端点颜色和v的上一条链的左端点颜色。

还需要注意的是，**当top[u]==top[v]，即u,v在同一条链时，因为此时区间的两个端点分别为u,v，需要分别对u,v的上一条链的左端点颜色进行对比，若相同则减1**

### 旅游

这题和[染色](https://www.cnblogs.com/aitejiu/p/16030688.html)，是非常类似的题型。我们从头来看。

本题要求我们动态维护一条链上，**从起点到终点中，挑选一个点为买，再挑选一个点为卖，能得到的最大利润（若利润为负则为0，即可以理解为不买不卖）**

需要注意的是，因为**买的点编号一定要比卖的点靠前（最多刚买的就卖，但这跟不买是一样的）**，因此我们无法简单的维护一个最大最小值就结束了。

那我们维护什么呢？从**最大子段和中得到的提醒**，我们维护以下几个值

* mx,mi，即一段的最大最小值
* lmx，表示为从左向右，能得到的利润最大值。（线段树中的从左到右，即为编号从小到大）
* rmx，表示为从右向左，能得到的利润最大值。（线段树中的从右向左，记为编号从大到小）
* tag，很常规的区间加懒标记，不多解释了。

其中，关于lmx和rmx的维护，我们来看一下

但请一定记住这点：**注意！注意！注意！，是有顺序问题的，因为我们是从起点到终点，是有方向的**

* lmx的维护，我们分为两部分：
  + 不跨左右区间，则取左右区间lmx即可，即为max(left.lmx,right.lmx)
  + 跨左右区间，则为**从左区间到右区间**，则取右区间的最大值-左区间的最小值即可，即为right.mx - left.mi
* rmx的维护，也分为两部分
  + 不跨左右区间，则取左右区间rmx即可，即为max(left.rmx,right.rmx)
  + 跨左右区间，则为**从右区间到左区间**，则取左区间的最大值-右区间的最小值即可，即为left.mx-right.mi

我们需要维护**从起点翻上来的所有链的信息**，我们记为L,还需维护**从终点翻上来的所有链的信息**，我们记为R

我们首先完成一下初始化

L.lmx = L.rmx = 0;  
L.mx = -INF,L.mi = INF;  
R = L;

接下来，就要说卡了我超久的一点了。记得我们之前提到的方向问题了嘛？

当我们在更新L，R时，我们需要注意两点：

1. 一定要分清楚，更新的**是从起点上来的L，还是从终点上来的R**。
2. 同时在更新L或R时，因为此时用来更新的L或R的链肯定比原来的L，R的编号小，因此都是在L，R左边，要按新区间为左区间，L，R为右区间去更新。

在完成这些之后，最后的答案由两类决定，取其中最大值：

1. 不跨区间，则去L的rmx与R的lmx的最大值，即max(L.rmx,R.lmx)
2. 跨区间，则取R的最大值-L的最小值，即R.mx - L.mi

**这题，很重要的一点就是，有方向问题，方向问题在树链剖分中也比较常见，需要注意此时操作的是起点的链还是终点的链了**

**另外在查询的时候，我们需要注意，只有在查询区间横跨左右区间时，我们才进行合并操作，否则直接返回左区间，右区间或整个区间即可**

## 有一些思维难度的题目

### Disruption P

还是蛮套路的，我们来总结一下这题的推理过程

首先考虑，**从每一条边去找哪一条边是我们需要的最短的替换路**。发现太麻烦了，则**正难则反**，我们**从新加的每一条路去考虑**。

**对于新加的边，其能更新其两个端点在树中的路径**

依据此，我们的问题即转化为了，**如何快速的对树中路径中的可以替换的最小值进行更新**

嗷，还有一个小问题，**边权化点权**，可以在这里看到操作方法[P4114 Qtree1](https://www.cnblogs.com/aitejiu/p/16028289.html)

不难想到利用**线段树+树剖**

问题就结束了，时间复杂度为

### aaa被续

**题目描述**

由于aaa没有完成HansBug的任务。所以HansBug开始计划着如何续aaa。

现在HansBug手里有 个aaa，每个aaa有一个码力值。一共存在 条连接两个aaa的边，故这 个aaa构成一棵**有根树**，根为1号aaa。

现在HansBug想要续了这 个aaa。HansBug所采用的策略是，对于第 个aaa，先让他和他的各级子aaa们~~乖乖♂站好~~成一队，然后依次续掉。

经过长期对于aaa码力值的研究，HansBug发现，**对于每一队aaa**，设有 个，码力值依次为，则续了队伍里的第 个aaa所能获得的码力值为 。

然而，aaa之间的关系树相当的复杂，HansBug的智商早已不够用，于是这个任务就交给了你。不过HansBug知道，任何一个aaa都不会有超过5个的直接子aaa

HansBug想要知道在每次排♂队方法最优的情况下，续了这些aaa最多可以获得的码力值~~，事成之后分给你100000 % 10点码力值~~。

这题不算很麻烦，我们直接来说思路。

**算法：树剖+线段树**

首先不难想到，我们肯定是从最大值开始算起，接下来考虑该点对答案的**贡献**。

对于枚举的点，我们假设点为i

其对答案的贡献是，i到1的路径中，所经过的所有节点的子树中的i在从小到大的排位\*的和

转化一下，则对于枚举的点，我们要干的事情是知道从i到1的路径中，在路径中节点的子树下的排位的和。

这个问题不难解决，我们可以初始化线段树中的每个节点的sum为sz[i]，这样我们从大到小枚举点的时候，直接算出i到1的路径权值和即可。在枚举该点后，要将路径中的所有点点权-1，因为此时从对于路径中每个点的子树中的点的点权从小到大的排位要再少一个了。

总结下，我们要干的是两件事：

* 求i到1的路径中的权值和\*即可以得到i对答案的贡献
* 接着不要忘记将位次-1，即为将从i到1的路径中的所有点点权-1

则，我们对线段树的操作时，求区间和，和区间加

### [COCI2017-2018#5] Pictionary

**题目描述**

在宇宙一个不为人知的地方，有一个星球，上面有一个国家，只有数学家居住。  
在这个国家有个数学家，有趣的是，每个数学家都住在自己的城市，且城市间无道路相连，因为他们可以在线交流。当然，城市有从到的编号。

一位数学家决定用手机发论文，而手机将“不言而喻”自动更正成了“猜谜游戏”。  
不久之后，这个国家就发现了猜谜游戏。他们想要见面一起玩，于是这个国家就开始了修路工程。  
道路修建会持续天。对于第天，若，则和城市间会修一条路。

由于数学家们忙于建筑工作，请你来确定一对数学家最早什么时候能凑到一起玩。

啊，也不算很麻烦，我们主要展示思路过程。

首先，看到q次询问中，每次询问使得a和b的联通的最早时间。那我们自然想到，可以直接求a和b的gcd，然后就可以得到一定能将其连接起来的日子了。

然后发现，这没什么用，因为在这之前的某天可能已经将两个点分别所在的连通块连起来过了。

那，我们就转换思路，首先，我们能想到的是。

对于每一天i而言，如果想将GCD等于m-i+1的城市全部连接起来，那我们只需要将城市m-i+1与其倍数的城市全部连接起来，这样就可以办到了，不难证明，时间复杂度是的。想想为什么这样的图与原来那样建图等价(提示：我们只考虑联通性)，将边权设为建立的那天

接下来，我们转化一下题目，**就是求任意两点之间，使其联通的所有路径中的最大边权的最小值。**

这就是个经典问题了，解决方法很多，这里提一下，但不会都实现。其中包括Kruskal重构树，建立最小生成树，求两点之间路径的最大值

可以直接去看看[P1967 [NOIP2013 提高组] 货车运输](https://www.luogu.com.cn/problem/P1967)

但不论第一种还是第二种，我们都要用到Kruskal算法，将边权从小到大排序，再建树。

而正好，我们在遍历天数的时候，就是边权最小到大建边的，因此直接按顺序枚举跑Kruskal就可以了。

对于第一种，这就是一个板子，建完树求LCA就是答案了。代码会稍微短一些。可以尝试一下。

我们这里说的是第二个方法。

先建立最小生成树，接着求两点之间路径的最大值。解决方案，也很多了。

主要体现在你用什么办法维护最大值。

如果你用倍增法，直接多维护一个数组维护路径最值即可。

如果你用树剖，就得用一下线段树了。

我们总结一下：

这题最难的点在于，建图的时候，如果把原图转化。我们根据的是只需要连通性，因此直接从m-i+1城市直接连向其倍数，就能更跟原图创造一样的连通性。

最后就是经典问题了。

### Mobile Phone Network

**题面翻译**

网络有 个节点。  
你的竞争者在一些节点之间提供了连接。竞争者提供了 个连接，对于提供的第 个连接，联通了 和 ，价格为 。  
你想提供 个连接。（保证这些连接不成环。）第 个连接联通了 和 ，它们的价格还没有决定。  
你想设置合适的价格，使得：

* 你的 个连接都会被顾客选择。
* 个连接的价格总和最大。

所有连接都是双向的。  
输出这个最大值。如果最大值无穷大，输出 。

我们来重新着重分析一下题目要求。

题目需要我们满足下边两个要求:

* **你的 k 个连接都会被顾客选择。**
* **k 个连接的价格总和最大。**

我们一个一个来解决

**你的k个连接都会被顾客选择**

最后形成的是一棵树，则一定生成的是一个最小生成树。

为使我们的**k**个连接一定被选择，则我们初始时将k条边边权设置为0，直接跑最小生成树。

**k个连接的价格总和最大**

我们考虑对于一条未在树上的边u->v，对于这条边而言，它的边权一定是要大于等于在树上路径u->v的任意一条边的边权。

则，我们的思路就出来了，**我们利用未使用的边，去更新树上路径的所有边的边权最小值**

做法如下：

* 标记**m**条边中，被选择的边。
* 接下来枚举，**m**条边中未被使用的边，去更新树上路径

由于要动态更新，区间最小值。

所以我们考虑使用树剖+线段树

结束啦~~

### [LNOI2014]LCA

**题目描述**

给出一个 个节点的有根树（编号为 到 ，根节点为 ）。

一个点的深度定义为这个节点到根的距离 。

设 表示点i的深度， 表示 与 的最近公共祖先。

有 次询问，每次询问给出 ，求 。

**本题要计算的就是l~r与z的LCA的深度之和**

我们来看看，**是否可以将求多个dep转化一下**

我们先对**dep有一个理解，dep就是从i到root总共有多少点**

我们从整体上考虑，发现对于一个询问：l , r , z 来说，所有的 lca 都在 z 到根的路径上。**从而有一些点，它们对很多的 lca 的深度都有贡献**，而这个贡献等于在这个点下面的 lca 的个数，所以我们可以把每个 lca 到根的路径上的每个点的权值都加一。然后从 z 向上走到根，沿路统计的权值就是答案了。

总结一下，**若只有一次询问，则我们想统计答案，只需要将l~r中所有点，从其位置向根节点中的路径的所有点点权+1，最后从z向根节点求一个区间和，即为答案**

是时候考虑优化的问题了：我们每次的操作都是从某个点到根的，所以树链剖分+线段树就好了。

但是考虑到每次统计时，不能很好的排除 l ~ r 区间之外的点对 z->根 这条路径的贡献，所以我们每次都要清空线段树。

我们每次清空线段树，然后从 l ~ r 再添加一遍，树剖+线段树的复杂度就是的，还要做 q 次，复杂度依然不理想。

看数据范围，应该就是正解了，现在要想办法优化掉最后的那个 q 的复杂度。

我们看到区间 l~r ，**我们需要考虑的是，如何排除l~r之外区间的影响?**

**我们联想一下类似于主席树的思路，我们从1开始以此将从该节点到根节点的路径中的所有点全部+1，则对于[l,r]的区间影响，我们可以通过用[1,r]的版本减去[1,l-1]版本的影响**

**我们可以将询问的区间拆开为两个，将询问离线查询。按照右端点从小到大排序（左端点都是根），然后按从小到大的顺序添加点，每遇到一个询问就查询一次，从而排除掉区间之外的点的影响，也就优化掉一个 q 的复杂度。**

具体的实现过程可以看一看代码。

##### Ac\_code

#include<bits/stdc++.h>  
using namespace std;  
const int N = 5e4 + 10,mod = 201314;  
typedef pair<int,int> PII;  
typedef long long LL;  
struct Node  
{  
 int l,r,sum,tag;  
}tr[N<<2];  
struct Query  
{  
 int r,z,id;//每一个询问拆成两部分，分别存储一个询问的l-1,r，以及z，询问的编号，以及是左端点还是右端点  
 bool f;  
 bool operator<(const Query& W)const  
 {  
 return r<W.r;  
 }  
}que[N<<1];  
int h[N],e[N],ne[N],idx;  
int sz[N],son[N],fa[N],dep[N];  
int top[N],id[N],ts;  
PII ans[N];  
//对于每一个ans的first存储的是对于1~l-1的版本中，从z到根的权值和。  
//而每一个ans的second存储的是对于1~r的版本中，从z到根的权值和  
//两者相减，则可以排除掉其余区间对z到根节点的权值和的影响。  
int n,q;  
  
void add(int a,int b)  
{  
 e[idx] = b,ne[idx] = h[a],h[a] = idx++;  
}  
  
void dfs1(int u,int depth)  
{  
 sz[u] = 1,dep[u] = depth++;  
 for(int i=h[u];~i;i=ne[i])  
 {  
 int j = e[i];  
 dfs1(j,depth+1);  
 sz[u] += sz[j];  
 if(sz[j]>sz[son[u]]) son[u] = j;  
 }  
}  
  
void dfs2(int u,int tp)  
{  
 top[u] = tp,id[u] = ++ts;  
 if(!son[u]) return ;  
 dfs2(son[u],tp);  
 for(int i=h[u];~i;i=ne[i])  
 {  
 int j = e[i];  
 if(j==son[u]) continue;  
 dfs2(j,j);  
 }  
}  
  
void pushup(int u)  
{  
 tr[u].sum = (LL)(tr[u<<1].sum + tr[u<<1|1].sum)%mod;  
}  
  
void pushdown(int u)  
{  
 auto &root = tr[u],&left = tr[u<<1],&right = tr[u<<1|1];  
 if(root.tag)  
 {  
 left.tag = (LL)(left.tag + root.tag)%mod;  
 right.tag = (LL)(right.tag + root.tag)%mod;  
 left.sum = (LL)(left.sum + 1ll\*(left.r - left.l + 1)\*root.tag%mod)%mod;  
 right.sum = (LL)(right.sum + 1ll\*(right.r - right.l + 1)\*root.tag%mod)%mod;  
 root.tag = 0;  
 }  
}  
  
void build(int u,int l,int r)  
{  
 tr[u] = {l,r};  
 if(l==r) return ;  
 int mid = l + r >> 1;  
 build(u<<1,l,mid),build(u<<1|1,mid+1,r);  
}  
  
void modify(int u,int l,int r)  
{  
 if(l<=tr[u].l&&tr[u].r<=r)  
 {  
 tr[u].tag ++ ;  
 tr[u].sum = (LL)(tr[u].sum + tr[u].r - tr[u].l + 1)%mod;  
 return ;  
 }  
 pushdown(u);  
 int mid = tr[u].l + tr[u].r >> 1;  
 if(l<=mid) modify(u<<1,l,r);  
 if(r>mid) modify(u<<1|1,l,r);  
 pushup(u);  
}  
  
int query(int u,int l,int r)  
{  
 if(l<=tr[u].l&&tr[u].r<=r) return tr[u].sum;  
 int res = 0;  
 pushdown(u);  
 int mid = tr[u].l + tr[u].r >> 1;  
 if(l<=mid) res = (LL)(res + query(u<<1,l,r))%mod;  
 if(r>mid) res = (LL)(res + query(u<<1|1,l,r))%mod;  
 return res;  
}  
  
void modify\_chain(int x)  
{  
 while(top[x]!=1)  
 {  
 modify(1,id[top[x]],id[x]);  
 x = fa[top[x]];  
 }  
 modify(1,1,id[x]);  
}  
  
int query\_chain(int x)  
{  
 int res = 0;  
 while(top[x]!=1)  
 {  
 res = (LL)(res + query(1,id[top[x]],id[x]))%mod;  
 x = fa[top[x]];  
 }  
 res = (LL)(res + query(1,1,id[x]))%mod;  
 return res;  
}  
  
int main()  
{  
 scanf("%d%d",&n,&q);  
 memset(h,-1,sizeof h);  
 for(int i=2;i<=n;i++)  
 {  
 cin>>fa[i];fa[i]++;  
 add(fa[i],i);  
 }  
 for(int i=0;i<q;i++)  
 {  
 int l,r,z;scanf("%d%d%d",&l,&r,&z);  
 l++,r++,z++;  
 que[i\*2] = {l-1,z,i,1};  
 que[i\*2+1] = {r,z,i,0};   
 }  
 dfs1(1,1);  
 dfs2(1,1);  
 build(1,1,n);  
 sort(que,que+q\*2);  
 int now = 0;  
 for(int i=0;i<q\*2;i++)  
 {  
 while(now<que[i].r) modify\_chain(++now);//对于每一个点都将其到根的路径中的所有点点权+1  
 if(que[i].f) ans[que[i].id].first = query\_chain(que[i].z);//遇到询问后，记录一下这是哪个询问的左端点版本还是右端点版本  
 else ans[que[i].id].second = query\_chain(que[i].z);  
 }  
 for(int i=0;i<q;i++)  
 printf("%d\n",((ans[i].second-ans[i].first)%mod+mod)%mod);  
 return 0;  
}

### 洛谷树

**题目描述**

树是一个无环、联通的无向图，由 个点和 条边构成。树上两个点之间的路径被定义为他们之间的唯一一条简单路径——显然这是一条最短路径。

现在引入一个概念——子路径。假设树上两个点 和 之间的路径是 ，那么它的子路径被定义为某一条路径 ，满足 ，其中 。显然，原路径是一条子路径，任意一个点也可以作为子路径。

我们给每条边赋予一个边权。萌萌哒的 Sugar 问小仓鼠：对于任意两个点 和 ，你能快速求出， 到 的路径上所有子路径经过的边的边权的 值的和是多少。具体地说就是，你把 到 的路径上所有子路径全部提出来，然后分别把每个子路径上经过的边的边权 在一起，最后求出得到的所有 值的和。

什么？你不知道 ？那就去百度啊！

这时候，fjzzq2002 大爷冒了粗来：窝还要你滋磁修改某条边边权的操作！

小仓鼠那么辣鸡，当然不会做这道题啦。于是他就来向你求救！

很有意思的题目，我们来从头分析。

##### 查询操作

首先不难分析，为了要求一段路径的异或值，我们采用类似前缀和的方式，预处理出来，从根节点到每个节点的边权异或值。将该数组设为sum

**这样，我们想求u,v之间路径的异或值时，只需要求sum[u] ^ sum[v] 就可以得到从u-v之间的路径异或值。**

但是，题目要求的是求**u-v路径中的所有子路径的异或值的和**。这就很麻烦了。

我们不难想到第一步转化，因为我们预处理的从根节点到任意节点的异或值。因此**u-v路径中的所有子路径的异或值的和**转化为了**u-v之间任意两点的sum的异或值的和**。接下来我们思考如何快速的算出转化后的问题。此时我们注意到异或的重要特点，**即每一位的独立性，每一位我们都可以进行独立的计算**

因此，我们想到了第二步转化，将**u-v之间任意两点的sum的异或值的和**转化为**枚举每一位，u-v任意两点之间sum的异或值该位为1的数量**

此时几乎就已经解决了，接下来我们想想如何求转化完的最后问题。

因为异或值为1一定是由一个0与一个1组成，因此最后的问题，即为求**枚举每一位，对于从u-v中的所有异或值，该位为1的数量，以及该位为0的数量，两者相乘即为该位为1的所有异或值数量**

这样就解决了。我们可以用**树剖+线段树**解决。另外其中，用到了一个小技巧，在[Can you answer these queries III](https://www.luogu.com.cn/problem/SP1716)，求区间最大子段和中用到过。我们返回的时候，需要返回时很多值，因此可以直接返回结构体。将push操作单独写出来即可。详情见代码。

另外，我需要特意强调一下免得有人跟我一样犯蠢的错误。

**这题，我们虽然操作的是边权，并且也将边权化了点权。但树中每个节点利用的值是到该点的路径异或值，因此我们需要操作的就是每个节点，不需要考虑找到LCA后，还要将id[LCA]+1**

##### 修改操作

这个就比较简单了。我们设原值为w[u]，要修改为的值为y。**枚举每一位，如果对于该位w[u]与y不相同，则在u的子树下的所有值的sum的当前位都要取反**

for(int i=0;i<10;i++)  
 if(((w[u]^y)>>i)&1)  
 modify(1,id[u],id[u]+sz[u]-1,i);

##### Ac\_code

#include<bits/stdc++.h>  
using namespace std;  
typedef long long LL;  
const int N = 3e4 + 10;  
struct Node  
{  
 int l,r,tag,num0,num1;  
}tr[N<<2][12];  
int h[N],e[N<<1],ne[N<<1],w[N<<1],idx;  
int sz[N],son[N],fa[N],dep[N];  
int top[N],id[N],nw[N],val[N],sum[N],ts;  
int n,m;  
  
void add(int a,int b,int c)  
{  
 e[idx] = b,ne[idx] = h[a],w[idx] = c,h[a] = idx++;  
}  
  
void dfs1(int u,int pa,int depth)  
{  
 sz[u] = 1,fa[u] = pa,dep[u] = depth;  
 for(int i=h[u];~i;i=ne[i])  
 {  
 int j = e[i];  
 if(j==pa) continue;  
 sum[j] = sum[u]^w[i];  
 val[j] = w[i];  
 dfs1(j,u,depth+1);  
 sz[u] += sz[j];  
 if(sz[j]>sz[son[u]]) son[u] = j;  
 }  
}  
  
void dfs2(int u,int tp)  
{  
 top[u] = tp,id[u] = ++ts;  
 nw[ts] = sum[u];  
 if(!son[u]) return ;  
 dfs2(son[u],tp);  
 for(int i=h[u];~i;i=ne[i])  
 {  
 int j = e[i];  
 if(j==fa[u]||j==son[u]) continue;  
 dfs2(j,j);  
 }  
}  
  
void push(Node &u,Node l,Node r)  
{  
 u.num0 = l.num0 + r.num0;  
 u.num1 = l.num1 + r.num1;  
}  
  
void pushup(int u,int k)  
{  
 push(tr[u][k],tr[u<<1][k],tr[u<<1|1][k]);  
}  
  
void pushdown(int u,int k)  
{  
 auto &root = tr[u][k],&left = tr[u<<1][k],&right = tr[u<<1|1][k];  
 if(root.tag)  
 {  
 swap(left.num1,left.num0);  
 left.tag ^= 1;  
 swap(right.num1,right.num0);  
 right.tag ^= 1;  
 root.tag = 0;  
 }  
}  
  
void build(int u,int l,int r,int k)  
{  
 tr[u][k] = {l,r,0,0,0};  
 if(l==r)  
 {  
 if((nw[l]>>k)&1) tr[u][k].num1 = 1;  
 else tr[u][k].num0 = 1;  
 return ;  
 }  
 int mid = l + r >> 1;  
 build(u<<1,l,mid,k),build(u<<1|1,mid+1,r,k);  
 pushup(u,k);  
}  
  
Node query(int u,int l,int r,int k)  
{  
 if(l<=tr[u][k].l&&tr[u][k].r<=r) return tr[u][k];  
 pushdown(u,k);  
 int mid = tr[u][k].l + tr[u][k].r >> 1;  
 Node res = {0,0,0,0,0};  
 if(l<=mid) push(res,res,query(u<<1,l,r,k));  
 if(r>mid) push(res,res,query(u<<1|1,l,r,k));  
 return res;  
}  
  
void modify(int u,int l,int r,int k)  
{  
 if(l<=tr[u][k].l&&tr[u][k].r<=r)   
 {  
 swap(tr[u][k].num1,tr[u][k].num0);  
 tr[u][k].tag ^= 1;  
 return ;  
 }  
 pushdown(u,k);  
 int mid = tr[u][k].l + tr[u][k].r >> 1;  
 if(l<=mid) modify(u<<1,l,r,k);  
 if(r>mid) modify(u<<1|1,l,r,k);  
 pushup(u,k);  
 return ;  
}  
  
int main()  
{  
 scanf("%d%d",&n,&m);  
 memset(h,-1,sizeof h);  
 for(int i=0;i<n-1;i++)   
 {  
 int a,b,c;scanf("%d%d%d",&a,&b,&c);  
 add(a,b,c),add(b,a,c);  
 }  
 dfs1(1,-1,1);  
 dfs2(1,1);  
 for(int i=0;i<10;i++) build(1,1,n,i);  
 while(m--)  
 {  
 int op,u,v,c;  
 scanf("%d%d%d",&op,&u,&v);  
 if(op==1)  
 {  
 LL res = 0;  
 int ub = u,vb = v;  
 for(int i=0;i<10;i++)  
 {  
 u = ub,v = vb;  
 Node t = {0,0,0,0,0};  
 while(top[u]!=top[v])  
 {  
 if(dep[top[u]]<dep[top[v]]) swap(u,v);  
 push(t,t,query(1,id[top[u]],id[u],i));  
 u = fa[top[u]];  
 }  
 if(dep[u]<dep[v]) swap(u,v);  
 push(t,t,query(1,id[v],id[u],i));  
 res += (1ll<<i)\*t.num1\*t.num0;  
 }  
 printf("%lld\n",res);  
 }  
 else   
 {  
 scanf("%d",&c);  
 if(dep[u]<dep[v]) swap(u,v);  
 for(int i=0;i<10;i++)  
 if(((c^val[u])>>i)&1)  
 modify(1,id[u],id[u]+sz[u]-1,i);  
 val[u] = c;  
 }  
 }  
 return 0;  
}

### 有趣的游戏

**题目描述**

游戏在一棵大小为 的树上进行。其中每个点都有点权，第 个点的点权为 。

每一次系统会给出一条链，小 A 可以从这条链上找出两个**点权不同**的点 ，他的得分是 。然后小 B 会从**整棵树**中选取两个**小 A 没有选过**的点，计分方式同小 A。

为了保持游戏难度，系统有时会增加一个点的权值。

当然，小 A 会尽可能使自己得分最大，他想知道这个值是多少。同时，他想知道，在自己得分最大的情况下，小 B 的最大得分是多少。

还是一样，看一看题目要求。

每一次系统会给出一条链，小 A 可以从这条链上找出两个**点权不同**的点 **x,y**，他的得分是 。然后小 B 会从**整棵树**中选取两个**小 A 没有选过**的点，计分方式同小 A。

非常容易推理出，对于A而言，其选出的最大答案是选出一条链的最大值与次大值，则对A而言最优解就是次大值

则对于B而言，**B需要从被A扣出两个点的树中，选出最大值与次大值，对B来说最优的就是这个次大值**

这个题最难的对我而言其实是，**怎么从把A选的两个点抠出来了**。

答案非常简单，用multiset就可以解决了。

直接从multiset中将A选中的删掉，再从multiset中找到最大值和次大值即可。

细节在代码中有注释。直接看。

##### Ac\_code

#include<bits/stdc++.h>  
using namespace std;  
const int N = 1e5 + 10,M = N \* 2;  
struct Node  
{  
 int l,r,fi,se;  
}tr[N<<2];  
int h[N],e[M],ne[M],w[N],idx;  
int sz[N],fa[N],son[N],dep[N];  
int top[N],id[N],nw[N],ts;  
multiset<int> s;  
int n,q;  
  
void add(int a,int b)  
{  
 e[idx] = b,ne[idx] = h[a],h[a] = idx++;  
}  
  
void dfs1(int u,int pa,int depth)  
{  
 sz[u] = 1,dep[u] = depth;  
 for(int i=h[u];~i;i=ne[i])  
 {  
 int j = e[i];  
 if(j==pa) continue;  
 fa[j] = u;  
 dfs1(j,u,depth+1);  
 sz[u] += sz[j];  
 if(sz[j]>sz[son[u]]) son[u] = j;   
 }  
}  
  
void dfs2(int u,int tp)  
{  
 top[u] = tp,id[u] = ++ts,nw[ts] = w[u];  
 if(!son[u]) return ;  
 dfs2(son[u],tp);  
 for(int i=h[u];~i;i=ne[i])  
 {  
 int j = e[i];  
 if(j==fa[u]||j==son[u]) continue;  
 dfs2(j,j);  
 }  
}  
  
void push(Node &u,Node l,Node r)  
{  
 u.fi = max(l.fi,r.fi);  
 u.se = max(l.fi==u.fi?l.se:l.fi,r.fi==u.fi?r.se:r.fi);  
}  
  
void pushup(int u)  
{  
 push(tr[u],tr[u<<1],tr[u<<1|1]);  
}  
  
void build(int u,int l,int r)  
{  
 if(l==r)  
 {  
 tr[u] = {l,l,nw[l],-1};//因为维护的是严格次小值，因此，当区间长为1时，记得初始化次小值为-1，wa死我了  
 return ;  
 }  
 tr[u] = {l,r};  
 int mid = l + r >> 1;  
 build(u<<1,l,mid),build(u<<1|1,mid+1,r);  
 pushup(u);  
}  
  
void modify(int u,int x,int k)  
{  
 if(tr[u].l==tr[u].r)  
 {  
 tr[u].fi += k;////因为维护的是严格次小值，因此，当区间长为1时，记得初始化次小值为-1，wa死我了  
 //因为这个原因，所有单点修的时候，就不要改次小值了，还是-1  
 return ;  
 }  
 int mid = tr[u].l + tr[u].r >> 1;  
 if(x<=mid) modify(u<<1,x,k);  
 else modify(u<<1|1,x,k);  
 pushup(u);  
}  
  
Node query(int u,int l,int r)  
{  
 if(l<=tr[u].l&&tr[u].r<=r) return tr[u];  
 int mid = tr[u].l + tr[u].r >> 1;  
 Node res = {0,0,-1,-1};  
 if(l>mid) return query(u<<1|1,l,r);  
 else if(r<=mid) return query(u<<1,l,r);  
 else   
 {  
 auto left = query(u<<1,l,r),right = query(u<<1|1,l,r);  
 push(res,left,right);  
 return res;  
 }  
}  
  
int main()  
{  
 scanf("%d",&n);  
 memset(h,-1,sizeof h);  
 for(int i=0;i<n-1;i++)  
 {  
 int u,v;  
 scanf("%d%d",&u,&v);  
 add(u,v),add(v,u);  
 }  
 for(int i=1;i<=n;i++)  
 {  
 int x;scanf("%d",&x);  
 w[i] = x;s.insert(x);  
 }  
 scanf("%d",&q);  
 dfs1(1,-1,1);  
 dfs2(1,1);  
 build(1,1,n);  
 while(q--)  
 {  
 int op,x,y;scanf("%d%d%d",&op,&x,&y);  
 if(!op)   
 {  
 s.erase(s.lower\_bound(w[x]));//嗷，还有，单点修改后，记得将s中的对应删掉  
 //同时，给向我一样语法不好的同学，提一句，multiset里不要之间删数值，会把所有对应值删掉  
 //因此，可以先lower\_bound找到，再删找到的对应迭代器的位置。  
 w[x] += y;  
 s.insert(w[x]);//也不要忘记再加上  
 modify(1,id[x],y);  
 }  
 else   
 {  
 Node res = {0,0,-1,-1};  
 while(top[x]!=top[y])  
 {  
 if(dep[top[x]]<dep[top[y]]) swap(x,y);  
 push(res,res,query(1,id[top[x]],id[x]));  
 x = fa[top[x]];  
 }  
 if(dep[x]<dep[y]) swap(x,y);  
 push(res,res,query(1,id[y],id[x]));  
 if(res.se==-1) puts("-1");  
 else   
 {  
 s.erase(s.lower\_bound(res.fi)),s.erase(s.lower\_bound(res.se));  
 printf("%d %d\n",res.se,\*(--s.lower\_bound(\*--s.end())));//这里也是，找到最大值后，其前边的值不一定是次大值，直接二分查找  
 s.insert(res.fi),s.insert(res.se);  
 }  
 }  
 }  
 return 0;  
}

### [USACO19DEC]Milk Visits G

**题目描述**

Farmer John 计划建造 个农场，用 条道路连接，构成一棵树（也就是说，所有农场之间都互相可以到达，并且没有环）。每个农场有一头奶牛，品种为 到 之间的一个整数 。

Farmer John 的 个朋友经常前来拜访他。在朋友 拜访之时，Farmer John 会与他的朋友沿着从农场 到农场 之间的唯一路径行走（可能有 ）。除此之外，他们还可以品尝他们经过的路径上任意一头奶牛的牛奶。由于 Farmer John 的朋友们大多数也是农场主，他们对牛奶有着极强的偏好。他的每个朋友都只喝某种特定品种的奶牛的牛奶。任何 Farmer John 的朋友只有在他们访问时能喝到他们偏好的牛奶才会高兴。

请求出每个朋友在拜访过后是否会高兴。

##### 树剖+二分

**时间复杂度**

思路非常简单。

我们建立树剖后，将所有颜色的dfs序放入数组，排一下序。

对于每次询问，我们沿链上翻时，该条链的dfs序范围为[id[top[u]],id[u]]

则对于该条链，我们要询问的即为，**对应颜色有没有在这个范围内出现过，即为在对应的颜色数组中找到一个编号在重链的dfs序范围内**

这个很好求，我们直接进行二分即可。

我们来看看代码

##### Ac\_code

#include<bits/stdc++.h>  
using namespace std;  
const int N = 1e5 + 10,M = N\*2;  
int h[N],e[M],ne[M],w[N],idx;  
int sz[N],son[N],fa[N],dep[N];  
int top[N],id[N],ts;  
vector<int> col[N];  
int n,m;  
  
void add(int a,int b)  
{  
 e[idx] = b,ne[idx] = h[a],h[a] = idx++;  
}  
  
void dfs1(int u,int pa,int depth)  
{  
 sz[u] = 1,fa[u] = pa,dep[u] = depth;  
 for(int i=h[u];~i;i=ne[i])  
 {  
 int j = e[i];  
 if(j==pa) continue;  
 dfs1(j,u,depth+1);  
 sz[u] += sz[j];  
 if(sz[j]>sz[son[u]]) son[u] = j;  
 }  
}  
  
void dfs2(int u,int tp)  
{  
 top[u] = tp,id[u] = ++ts;  
 if(!son[u]) return ;  
 dfs2(son[u],tp);  
 for(int i=h[u];~i;i=ne[i])  
 {  
 int j = e[i];  
 if(j==fa[u]||j==son[u]) continue;  
 dfs2(j,j);  
 }  
}  
  
int main()  
{  
 scanf("%d%d",&n,&m);  
 memset(h,-1,sizeof h);  
 for(int i=1;i<=n;i++) scanf("%d",w+i);  
 for(int i=0;i<n-1;i++)   
 {  
 int u,v;scanf("%d%d",&u,&v);  
 add(u,v),add(v,u);  
 }  
 dfs1(1,-1,1);  
 dfs2(1,1);  
 for(int i=1;i<=n;i++) col[w[i]].push\_back(id[i]);  
 for(int i=1;i<=n;i++) sort(col[i].begin(),col[i].end());  
 for(int i=0;i<m;i++)  
 {  
 int u,v,c;scanf("%d%d%d",&u,&v,&c);  
 bool flag = 0;  
 while(top[u]!=top[v])  
 {  
 if(dep[top[u]]<dep[top[v]]) swap(u,v);  
 auto it = lower\_bound(col[c].begin(),col[c].end(),id[top[u]]);  
 if(it!=col[c].end()&&\*it<=id[u])  
 {  
 flag = 1;  
 break;  
 }  
 u = fa[top[u]];  
 }  
 if(!flag)  
 {  
 if(dep[u]<dep[v]) swap(u,v);  
 auto it = lower\_bound(col[c].begin(),col[c].end(),id[v]);  
 if(it!=col[c].end()&&\*it<=id[u]) flag = 1;  
 }  
 printf("%d",flag);  
 }  
 return 0;  
}

### 遥远的国度

**题目描述**

zcwwzdjn 在追杀 zhx ，而 zhx 逃入了一个遥远的国度。当 zcwwzdjn 准备进入遥远的国度继续追杀时，守护神 RapiD 阻拦了 zcwwzdjn 的去路，他需要 zcwwzdjn 完成任务后才能进入遥远的国度继续追杀。

问题是这样的：遥远的国度有 个城市，这些城市之间由一些路连接且这些城市构成了一颗树。这个国度有一个首都，我们可以把这个首都看做整棵树的根，但遥远的国度比较奇怪，首都是随时有可能变为另外一个城市的。遥远的国度的每个城市有一个防御值，第 个的防御值为 ，有些时候 RapiD 会使得某两个城市之间的路径上的所有城市的防御值都变为某个值。

RapiD 想知道在某个时候，如果把首都看做整棵树的根的话，那么以某个城市为根的子树的所有城市的防御值最小是多少。

由于 RapiD 无法解决这个问题，所以他拦住了 zcwwzdjn 希望他能帮忙。但 zcwwzdjn 还要追杀 zhx，所以这个重大的问题就被转交到了你的手上。

**输入格式**

第 行两个整数 ，代表城市个数和操作数。

第 行至第 行，每行两个整数 ，代表城市 和城市 之间有一条路。

第 行，有 个整数，第 个代表第 个点的初始防御值 。

第 行一个整数 ，代表初始的首都为 。

第 行至第 行，首先有一个整数 。

如果 ，接下来有一个整数 ，代表把首都修改为 ；

如果 ，接下来有三个整数 ，代表将 路径上的所有城市的防御值修改为 ；

如果 ，接下来有一个整数 ，代表询问以城市 为根的子树中的最小防御值。

**树剖+线段树+分类讨论（树剖换根）**

我们先在这里放结论

* **换根，不影响我们用从1建立的树剖跑树中路径的操作。**
* **换根，会影响利用子树的性质，因为换过根后，树中结点子树的节点dfs序可能就不连续了，但可以通过分类讨论避免掉，具体看下文。**

我们来以本题作为范本讲讲树剖换根

总共三个操作，我们需要关注的仅仅只有第三个操作。

因为根会换，因此，对于同一个节点，当根不同时，其子树也不一定相同。显然，每换一次根就跑一次树剖是不理智的。

那就到了我们需要讲述的关键了，我们可以进行分类讨论

**注意：我们这里只根据一号点进行了树剖，root是题目中的首都，在我们的树剖中，我们用的是一号点作为根**

1. 需要查询的点u即为root，则直接查询整棵树即可
2. 需要查询的点u不在1->root的路径中，那就直接查询该节点下的子树即可
3. 也是重头戏，需要查询的点u在1=>root的路径中，我们只需要删掉从u->root的路径中u的直接儿子v的子树这一部分。而这一部分是连续的，我们只需要查这一部分的两边的连续区间即可得到答案。

主要就是第3种，也很好想

**找到路径u->root上的u的直系儿子v**

就会发现root为根时,u子树覆盖不到的地方是**v及v的子树**

下边说一下，怎么找到直系儿子v

就跟树剖找LCA相同，只是最后的时候需要特判一下。

若是最后的时候，我们从一条链翻上去的时候，直接就到了u，那直接返回该条链的头即可

若是已经翻到了同一条链，则直接返回son[u]即可。

int u = root;  
while(top[u]!=top[x])  
{  
if(fa[top[u]]==x) return top[u];  
u = fa[top[u]];  
}  
return son[x];

下边直接看代码

##### Ac\_code

#include<bits/stdc++.h>  
using namespace std;  
const int N = 1e5 + 10,INF = INT\_MAX;  
struct Node  
{  
 int l,r,mi,tag;  
}tr[N<<2];  
int h[N],e[N<<1],ne[N<<1],w[N],idx;  
int sz[N],fa[N],son[N],dep[N];  
int top[N],id[N],nw[N],ts;  
int n,m,root;  
  
void add(int a,int b)  
{  
 e[idx] = b,ne[idx] = h[a],h[a] = idx++;  
}  
  
void dfs1(int u,int pa,int depth)  
{  
 sz[u] = 1,fa[u] = pa,dep[u] = depth;  
 for(int i=h[u];~i;i=ne[i])  
 {  
 int j = e[i];  
 if(j==pa) continue;  
 dfs1(j,u,depth+1);  
 sz[u] += sz[j];  
 if(sz[j]>sz[son[u]]) son[u] = j;  
 }  
}  
  
void dfs2(int u,int tp)  
{  
 top[u] = tp,id[u] = ++ts,nw[ts] = w[u];  
 if(!son[u]) return ;  
 dfs2(son[u],tp);  
 for(int i=h[u];~i;i=ne[i])  
 {  
 int j = e[i];  
 if(j==fa[u]||j==son[u]) continue;  
 dfs2(j,j);  
 }  
}  
  
void pushup(int u)  
{  
 tr[u].mi = min(tr[u<<1].mi,tr[u<<1|1].mi);  
}  
  
void pushdown(int u)  
{  
 auto &root = tr[u],&left = tr[u<<1],&right = tr[u<<1|1];  
 if(root.tag)  
 {  
 left.mi = right.mi = left.tag = right.tag = root.tag;  
 root.tag = 0;  
 }  
}  
  
void build(int u,int l,int r)  
{  
 if(l==r)  
 {  
 tr[u] = {l,r,nw[l]};  
 return ;  
 }  
 tr[u] = {l,r};  
 int mid = l + r >> 1;  
 build(u<<1,l,mid),build(u<<1|1,mid+1,r);  
 pushup(u);  
}  
  
void modify(int u,int l,int r,int x)  
{  
 if(l<=tr[u].l&&tr[u].r<=r)   
 {  
 tr[u].mi = tr[u].tag = x;  
 return ;  
 }  
 pushdown(u);  
 int mid = tr[u].l + tr[u].r >> 1;  
 if(l<=mid) modify(u<<1,l,r,x);  
 if(r>mid) modify(u<<1|1,l,r,x);  
 pushup(u);  
}  
  
int query(int u,int l,int r)  
{  
 if(l<=tr[u].l&&tr[u].r<=r) return tr[u].mi;  
 int mid = tr[u].l + tr[u].r >> 1;  
 pushdown(u);  
 int res = INF;  
 if(l<=mid) res = min(res,query(u<<1,l,r));  
 if(r>mid) res = min(res,query(u<<1|1,l,r));  
 return res;  
}  
  
int find(int x)  
{  
 if(x==root) return 1;  
 if(id[x]>=id[root]||id[x]+sz[x]-1<id[root]) return x;   
 int u = root;  
 while(top[u]!=top[x])  
 {  
 if(fa[top[u]]==x) return top[u];  
 u = fa[top[u]];  
 }  
 return son[x];  
}  
  
int main()  
{  
 scanf("%d%d",&n,&m);  
 memset(h,-1,sizeof h);  
 for(int i=0;i<n-1;i++)   
 {  
 int u,v;scanf("%d%d",&u,&v);  
 add(u,v),add(v,u);  
 }  
 for(int i=1;i<=n;i++) scanf("%d",w+i);  
 scanf("%d",&root);  
 dfs1(1,-1,1);  
 dfs2(1,1);  
 build(1,1,n);  
 while(m--)  
 {  
 int op,x,y,z;  
 scanf("%d%d",&op,&x);  
 if(op==1) root = x;  
 else if(op==2)  
 {  
 scanf("%d%d",&y,&z);  
 while(top[x]!=top[y])  
 {  
 if(dep[top[x]]<dep[top[y]]) swap(x,y);  
 modify(1,id[top[x]],id[x],z);  
 x = fa[top[x]];  
 }  
 if(dep[x]<dep[y]) swap(x,y);  
 modify(1,id[y],id[x],z);  
 }  
 else   
 {  
 int u = find(x);  
 if(u==1) printf("%d\n",query(1,1,n));  
 else if(u==x) printf("%d\n",query(1,id[u],id[u]+sz[u]-1));  
 else   
 {  
 int res = query(1,1,id[u]-1);  
 if(id[u]+sz[u]<=n) res = min(res,query(1,id[u]+sz[u],n));  
 printf("%d\n",res);  
 }  
 }  
 }  
 return 0;  
}

### Jamie and Tree

**题面翻译**

有一棵n个节点的有根树，标号为1-n，你需要维护以下三种操作

1.给定一个点v，将整颗树的根变为v

2.给定两个点u, v，将lca(u, v)所在的子树都加上x

3.给定一个点v，你需要回答以v所在的子树的权值和

## 换根树剖

这类题目不多，一般都是需要分类讨论。

题意：

支持以下操作：

1. 换根
2. 子树修改
3. LCA
4. 子树查询

我们挨个分析

**先以1为根将整棵树剖一遍**

接着处理每个操作：

1.直接换

2.我们要分类讨论，为了叙述方便，记x,y在原树中的**LCA**为**lca(x,y)**

对于任意一点**x**，有以下几种情况：

(1)**x=root**，修改整棵树

(2)**lca(x,root)!=x**，那么换根不影响子树，直接修改

(3)**lca(x,root)=x**，则我们直接找到从root到x的路径中，x的直接儿子，将直接儿子的子树除去，其余部分都是x的子树。

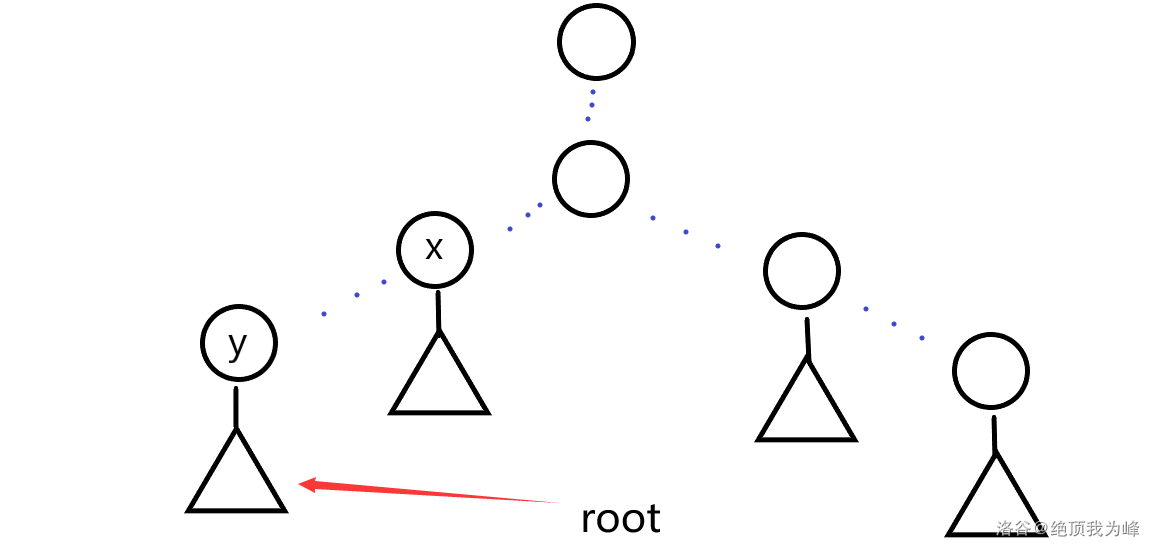
3.我们依然要分类讨论：

我们不妨默认

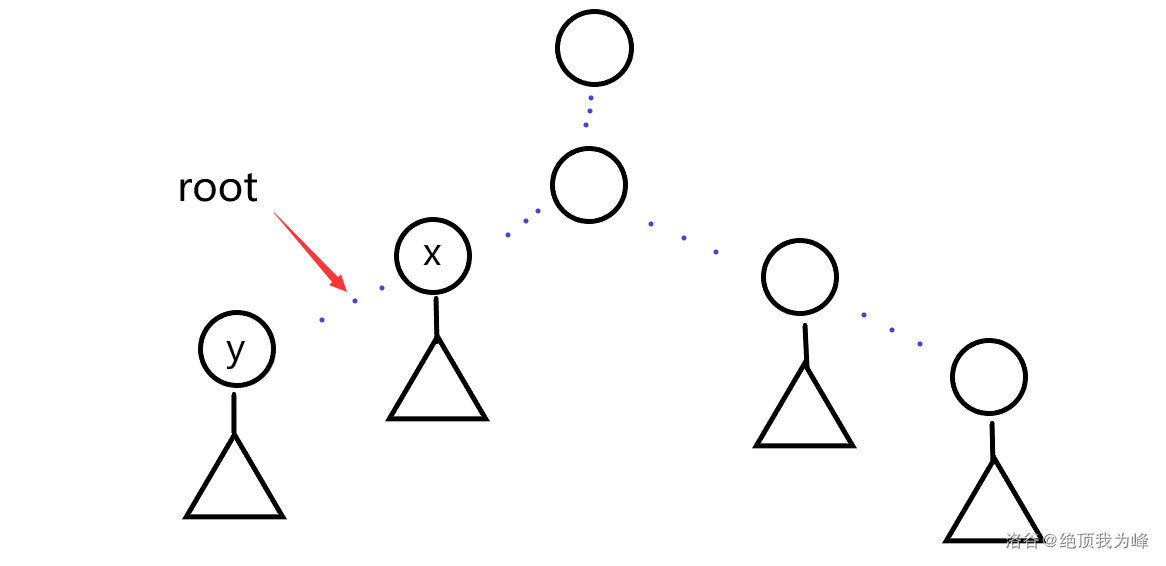
进行分类讨论：

(1)**lca(x,y)=x**

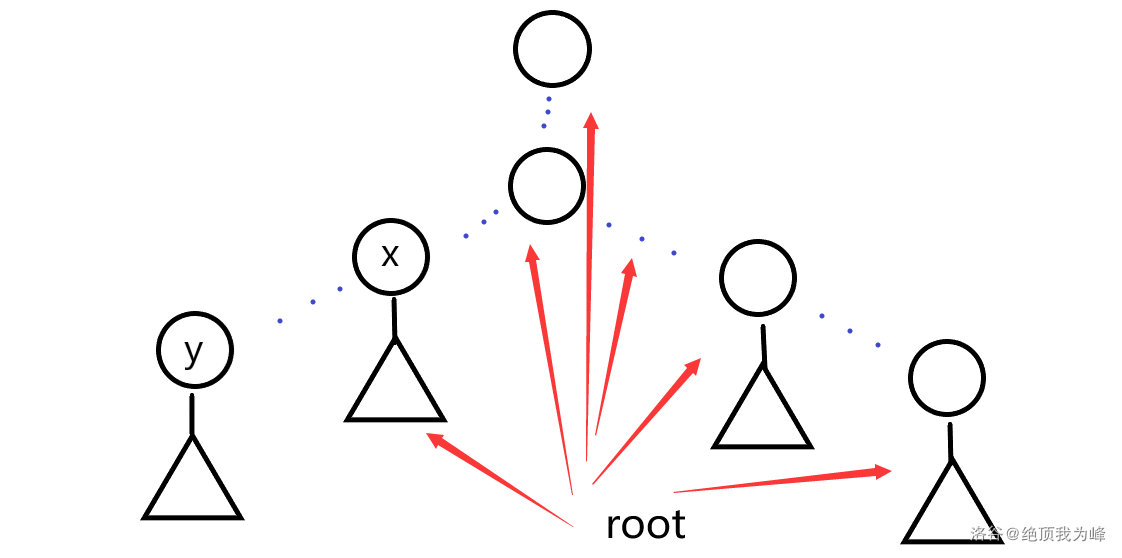
1)**root**在**y**的子树中，那么答案为**y**



2)**root**在**x**与**y**之间，那么答案为**root**

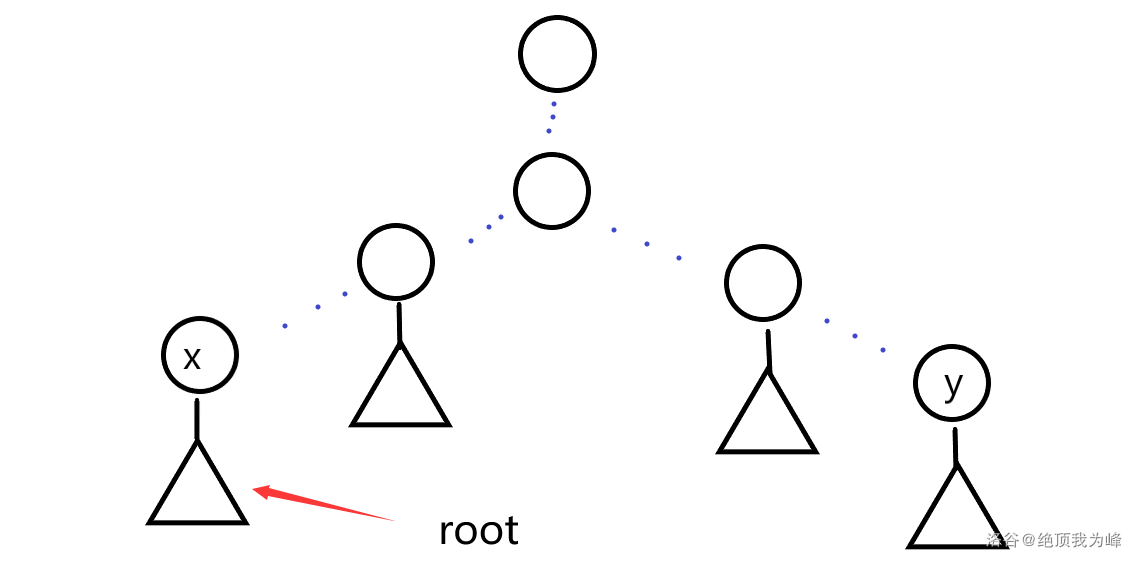


3)**root**在其他位置，那么答案为**x**

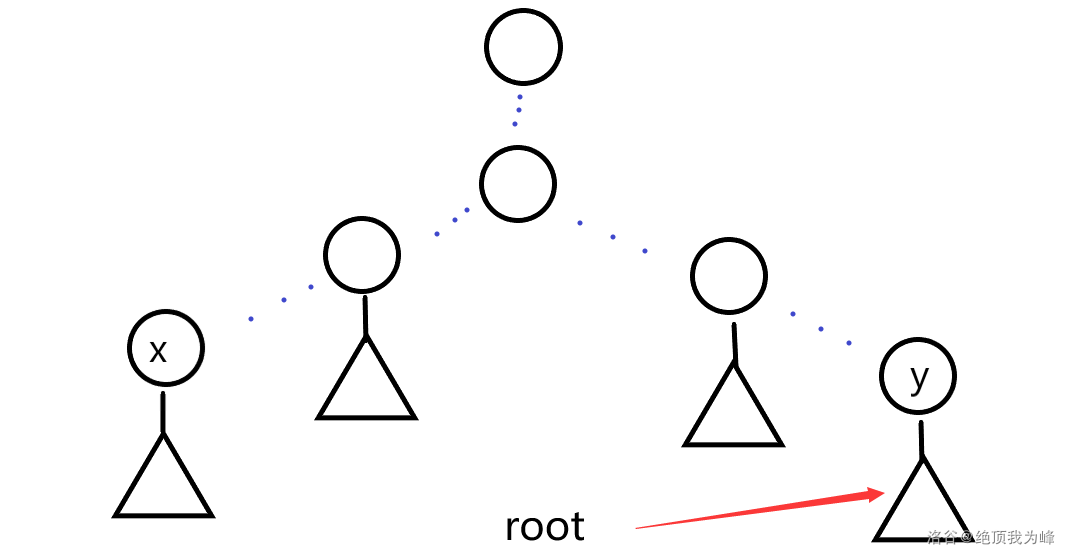


(2)**lca(x,y)!=x**

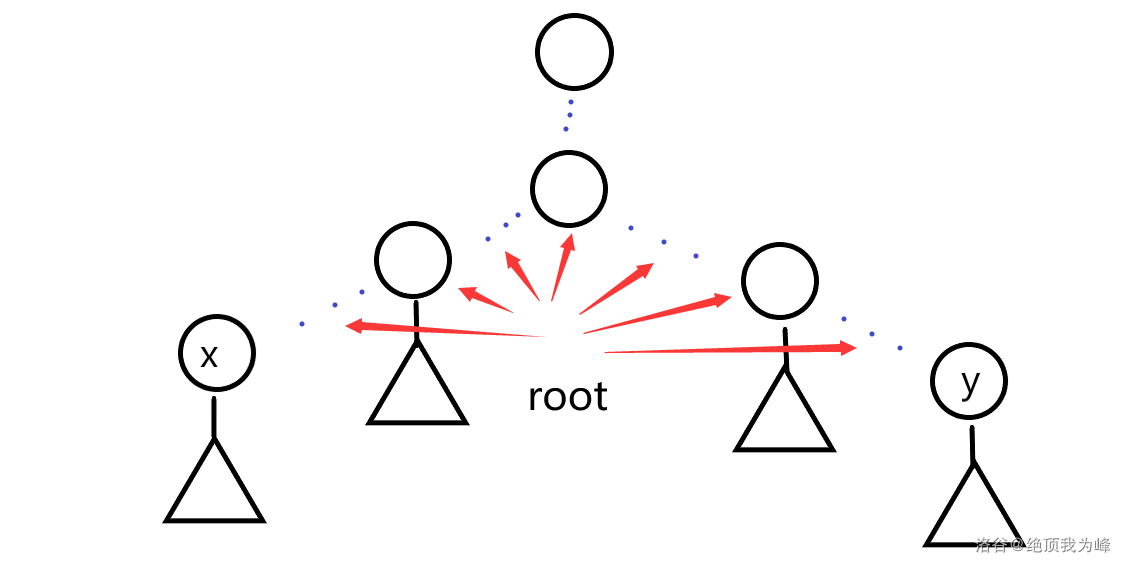
1）**root**在**x**的子树中，那么答案为**x**



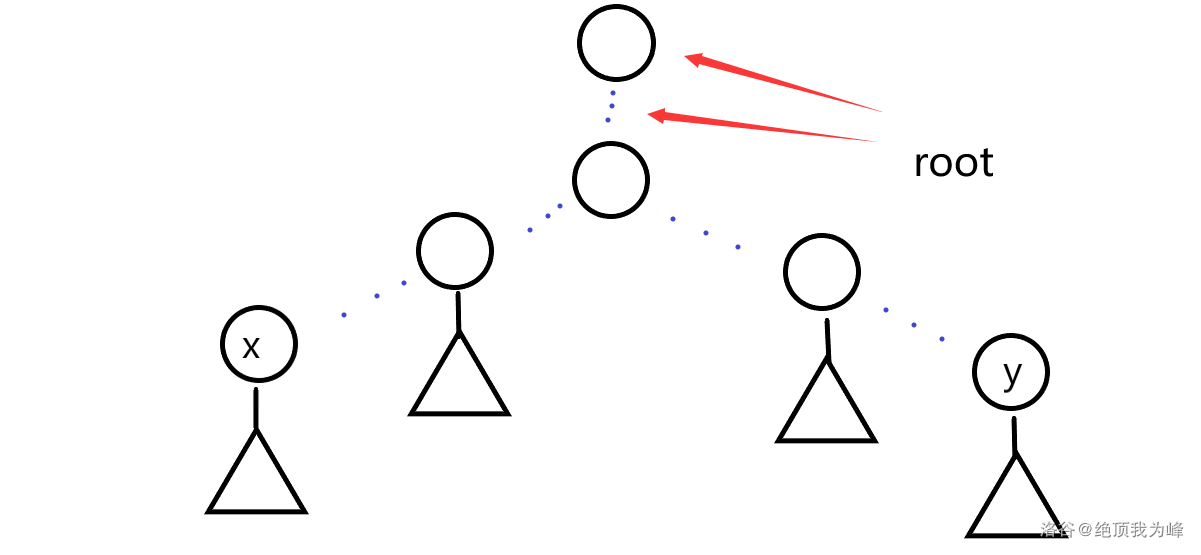
2）**root**在**y**的子树中，那么答案为**y**



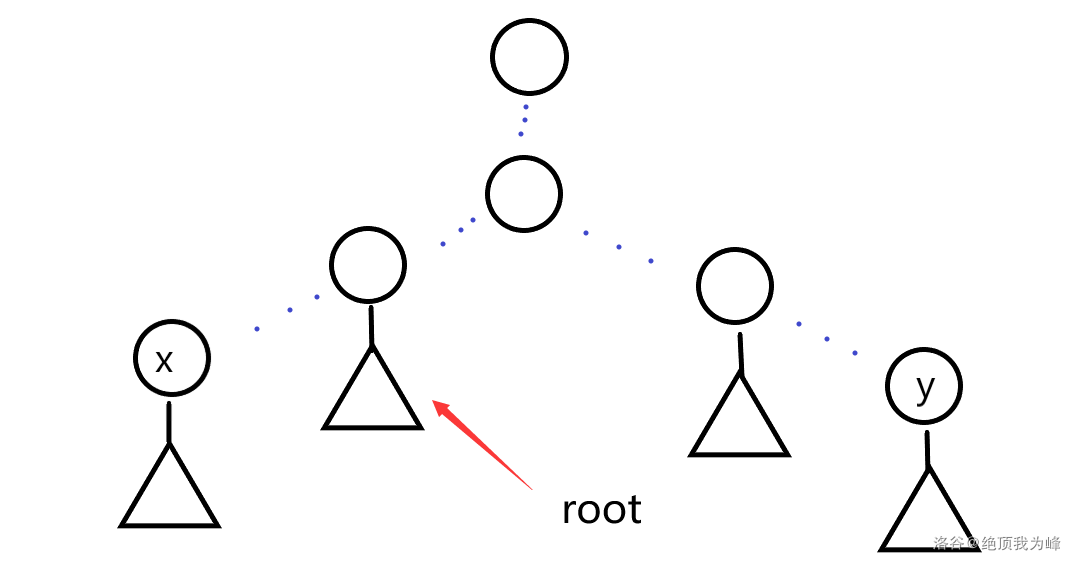
3）**root**在**x**到**y**的路径上，那么答案为**root**



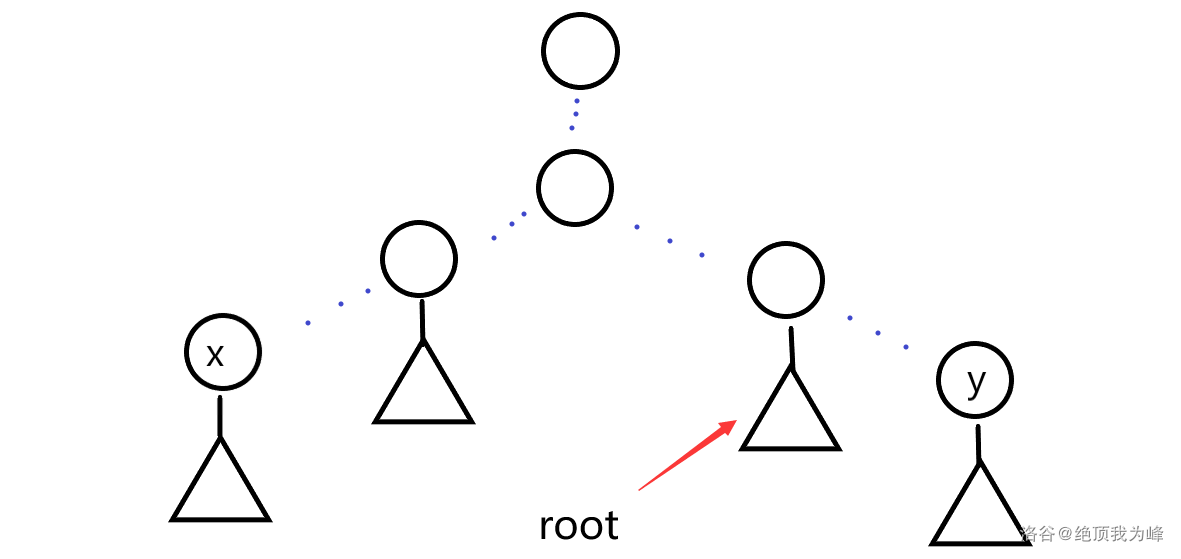
4）若**lca(x,root)=lca(y,root)**，即**root**在下图所示位置，答案为**lca(x,y)**



5）若**lca(x,y)!=lca(x,root)**，即**root**在下图位置，答案为**lca(x,root)**



6）若**lca(x,y)!=lca(y,root)**，即**root**在下图位置，答案为**lca(y,root)**



4.同2，进行分类讨论查询

##### Ac\_code

#include<bits/stdc++.h>  
using namespace std;  
typedef long long LL;  
const int N = 1e5 + 10,M = N\*2;  
const LL INF = 1e18;  
struct Node  
{  
 int l,r;   
 LL sum,tag;//可能加负数  
}tr[N<<2];  
int h[N],e[M],ne[M],w[N],idx;  
int sz[N],son[N],fa[N],dep[N];  
int top[N],id[N],nw[N],ts;  
int n,q,root;  
  
void add(int a,int b)  
{  
 e[idx] = b,ne[idx] = h[a],h[a] = idx++;  
}  
  
void dfs1(int u,int pa,int depth)  
{  
 sz[u] = 1,dep[u] = depth;  
 for(int i=h[u];~i;i=ne[i])  
 {  
 int j = e[i];  
 if(j==pa) continue;  
 fa[j] = u;  
 dfs1(j,u,depth+1);  
 sz[u] += sz[j];  
 if(sz[j]>sz[son[u]]) son[u] = j;  
 }  
}  
  
void dfs2(int u,int tp)  
{  
 top[u] = tp,id[u] = ++ts,nw[ts] = w[u];  
 if(!son[u]) return ;  
 dfs2(son[u],tp);  
 for(int i=h[u];~i;i=ne[i])  
 {  
 int j = e[i];  
 if(j==son[u]||j==fa[u]) continue;  
 dfs2(j,j);  
 }  
}  
  
void pushup(int u)  
{  
 tr[u].sum = tr[u<<1].sum + tr[u<<1|1].sum;  
}  
  
void build(int u,int l,int r)  
{  
 if(l==r)  
 {  
 tr[u] = {l,r,nw[l],0};  
 return ;  
 }  
 tr[u] = {l,r};  
 int mid = l + r >> 1;  
 build(u<<1,l,mid),build(u<<1|1,mid+1,r);  
 pushup(u);  
}  
  
void pushdown(int u)  
{  
 auto &root = tr[u],&left = tr[u<<1],&right = tr[u<<1|1];  
 if(root.tag)  
 {  
 left.tag += root.tag;  
 left.sum += root.tag\*(left.r - left.l + 1);  
 right.tag += root.tag;  
 right.sum += root.tag\*(right.r - right.l + 1);  
 root.tag = 0;  
 }  
}  
  
void modify(int u,int l,int r,int k)  
{  
 if(l<=tr[u].l&&tr[u].r<=r)   
 {  
 tr[u].tag += k;  
 tr[u].sum += 1ll\*k\*(tr[u].r - tr[u].l + 1);  
 return ;  
 }  
 pushdown(u);  
 int mid = tr[u].l + tr[u].r >> 1;  
 if(l<=mid) modify(u<<1,l,r,k);  
 if(r>mid) modify(u<<1|1,l,r,k);  
 pushup(u);  
}  
  
LL query(int u,int l,int r)  
{  
 if(l<=tr[u].l&&tr[u].r<=r) return tr[u].sum;  
 pushdown(u);  
 int mid = tr[u].l + tr[u].r >> 1;  
 LL res = 0;  
 if(l<=mid) res += query(u<<1,l,r);  
 if(r>mid) res += query(u<<1|1,l,r);  
 return res;  
}  
  
int lca(int x,int y)  
{  
 while(top[x]!=top[y])   
 {  
 if(dep[top[x]]<dep[top[y]]) swap(x,y);  
 x = fa[top[x]];  
 }  
 return dep[x]<dep[y]?x:y;  
}  
  
int find(int x)  
{  
 int u = root;  
 while(top[u]!=top[x])  
 {  
 if(fa[top[u]]==x) return top[u];  
 u = fa[top[u]];  
 }  
 return son[x];  
}  
  
int LCA(int x,int y)  
{  
 if(dep[x]>dep[y]) swap(x,y);  
 if(lca(x,y)==x)  
 {  
 if(id[root]>=id[y]&&id[root]<=id[y]+sz[y]-1) return y;  
 if(lca(x,root)==x) return lca(y,root);  
 return x;  
 }  
 if(id[root]>=id[x]&&id[root]<=id[x]+sz[x]-1) return x;  
 if(id[root]>=id[y]&&id[root]<=id[y]+sz[y]-1) return y;  
 if((lca(x,root)==root&&lca(x,y)==lca(y,root))||(lca(y,root)==root&&lca(x,y)==lca(x,root))) return root;  
 if(lca(root,x)==lca(y,root)) return lca(x,y);  
 if(lca(x,y)!=lca(x,root)) return lca(x,root);  
 return lca(y,root);  
}  
  
void mo2(int x,int p)  
{  
 if(root==x)   
 {  
 modify(1,1,n,p);  
 return ;  
 }  
 int res = lca(root,x);  
 if(res!=x) modify(1,id[x],id[x]+sz[x]-1,p);  
 else   
 {  
 int u = find(x);  
 modify(1,1,n,p);  
 modify(1,id[u],id[u]+sz[u]-1,-p);  
 }  
}  
  
LL q2(int x)  
{  
 if(x==root) return query(1,1,n);  
 int res = lca(root,x);  
 if(res!=x) return query(1,id[x],id[x]+sz[x]-1);  
 int u = find(x);  
 return query(1,1,n) - query(1,id[u],id[u]+sz[u]-1);  
}  
  
int main()  
{  
 scanf("%d%d",&n,&q);  
 memset(h,-1,sizeof h);  
 for(int i=1;i<=n;i++) scanf("%d",w+i);  
 for(int i=0;i<n-1;i++)  
 {  
 int u,v;scanf("%d%d",&u,&v);  
 add(u,v),add(v,u);  
 }  
 root = 1;  
 dfs1(1,-1,1);  
 dfs2(1,1);  
 build(1,1,n);  
 while(q--)  
 {  
 int op,x,y,c;  
 scanf("%d%d",&op,&x);  
 if(op==1) root = x;  
 else if(op==2)  
 {  
 scanf("%d%d",&y,&c);  
 mo2(LCA(x,y),c);  
 }  
 else printf("%lld\n",q2(x));  
 }  
 return 0;  
}

## 动态开点线段树

### [SDOI2014]旅行

**题目描述**

S国有N个城市，编号从1到N。城市间用N-1条双向道路连接，满足从一个城市出发可以到达其它所有城市。每个城市信仰不同的宗教，如飞天面条神教、隐形独角兽教、绝地教都是常见的信仰。

为了方便，我们用不同的正整数代表各种宗教， S国的居民常常旅行。旅行时他们总会走最短路，并且为了避免麻烦，只在信仰和他们相同的城市留宿。当然旅程的终点也是信仰与他相同的城市。S国政府为每个城市标定了不同的旅行评级，旅行者们常会记下途中（包括起点和终点）留宿过的城市的评级总和或最大值。

在S国的历史上常会发生以下几种事件：

“CC x c“：城市x的居民全体改信了c教；

“CW x w“：城市x的评级调整为w;

“QS x y“：一位旅行者从城市x出发，到城市y，并记下了途中留宿过的城市的评级总和；

“QM x y“：一位旅行者从城市x出发，到城市y，并记下了途中留宿过的城市的评级最大值。

由于年代久远，旅行者记下的数字已经遗失了，但记录开始之前每座城市的信仰与评级，还有事件记录本身是完好的。请根据这些信息，还原旅行者记下的数字。 为了方便，我们认为事件之间的间隔足够长，以致在任意一次旅行中，所有城市的评级和信仰保持不变。

老规矩，先来分析分析题目需求

我们总共需要满足以下四个要求

1. 将城市x的居民的信仰改为c
2. 将城市x的评分全部改为w
3. 统计对于旅行者，其从x至y的路径中所有留宿过的城市的评级总和
4. 统计对于旅行者，其从x至y的路径中所有留宿过的城市的评级最大值

不难发现，对于每一种信仰而言，我们想要统计**对于某一种具体的信仰而言，从x至y的拥有该信仰的城市的评级综合与评级最大值**

那最方便的方法，当然是对于每一种信仰直接开一颗线段树就行了....吗？

我们仔细观察后可以发现，我们极限情况，会开颗线段树，每一颗线段树都是的范围，那不用想了，必炸。

我们对于该问题的优化，即为**动态开点线段树**。如果不会，推荐去学习完再来写一下。

那，接下来，这题目就挺板子的了。

细节直接看代码。

##### Ac\_code

#include<bits/stdc++.h>  
using namespace std;  
const int N = 1e5 + 10;  
struct Node  
{  
 int l,r,sum,mx;   
}tr[N\*20];//开2\*Q\*log(N)的大小  
int h[N],ne[N<<1],e[N<<1],val[N],C[N],idx;  
int fa[N],son[N],dep[N],sz[N];  
int top[N],id[N],tid[N],ts;  
int root[N];//root[N]记录每棵树的根节点编号  
int n,q,cnt;//cnt是动态开点的编号  
  
void add(int a,int b)  
{  
 e[idx] = b,ne[idx] = h[a],h[a] = idx++;  
}  
  
void dfs1(int u,int pa,int depth)  
{  
 sz[u] = 1,dep[u] = depth;  
 for(int i=h[u];~i;i=ne[i])  
 {  
 int j = e[i];  
 if(j==pa) continue;  
 fa[j] = u;  
 dfs1(j,u,depth+1);  
 sz[u] += sz[j];  
 if(sz[j]>sz[son[u]]) son[u] = j;  
 }  
}  
  
void dfs2(int u,int tp)  
{  
 top[u] = tp,id[u] = ++ts,tid[ts] = u;  
 if(!son[u]) return ;  
 dfs2(son[u],tp);  
 for(int i=h[u];~i;i=ne[i])  
 {  
 int j = e[i];  
 if(j==fa[u]||j==son[u]) continue;  
 dfs2(j,j);  
 }  
}  
  
void pushup(int u)  
{  
 tr[u].sum = tr[tr[u].l].sum + tr[tr[u].r].sum;  
 tr[u].mx = max(tr[tr[u].l].mx,tr[tr[u].r].mx);  
}  
  
void modify(int &u,int x,int k,int l,int r)  
{  
 if(!u)//如果没有这个点，那就把这个点开出来，推荐不要搞得很复合，把这个功能独立出来  
 {  
 u = ++cnt;  
 tr[u] = {0};  
 if(l==r) tr[u].sum = tr[u].mx = val[tid[l]];//如果是叶节点，那就记录一下评级  
 }  
 if(l==r) //找到对应的位置  
 {  
 tr[u].sum = tr[u].mx = k;  
 return ;  
 }  
 int mid = l + r >> 1;  
 if(x<=mid) modify(tr[u].l,x,k,l,mid);  
 else modify(tr[u].r,x,k,mid+1,r);  
 pushup(u);  
}  
  
int querymx(int &u,int ql,int qr,int l,int r)  
{  
 if(!u) return 0;//若此树中没有该区间，则直接返回0  
 if(ql<=l&&r<=qr) return tr[u].mx;  
 int mid = l + r >> 1;  
 int res = 0;  
 if(ql<=mid) res = max(res,querymx(tr[u].l,ql,qr,l,mid));  
 if(qr>mid) res = max(res,querymx(tr[u].r,ql,qr,mid+1,r));  
 pushup(u);  
 return res;  
}  
  
int querysum(int &u,int ql,int qr,int l,int r)  
{  
 if(!u) return 0;//若此树中没有该区间，则直接返回0  
 if(ql<=l&&r<=qr) return tr[u].sum;  
 int mid = l + r >> 1;  
 int res = 0;  
 if(ql<=mid) res += querysum(tr[u].l,ql,qr,l,mid);  
 if(qr>mid) res += querysum(tr[u].r,ql,qr,mid+1,r);  
 pushup(u);  
 return res;  
}  
  
int main()  
{  
 scanf("%d%d",&n,&q);  
 memset(h,-1,sizeof h);   
 for(int i=1;i<=n;i++) scanf("%d%d",val+i,C+i);  
 for(int i=0;i<n-1;i++)  
 {  
 int x,y;scanf("%d%d",&x,&y);  
 add(x,y),add(y,x);  
 }  
 dfs1(1,-1,1);  
 dfs2(1,1);  
 for(int i=1;i<=n;i++)  
 modify(root[C[i]],id[i],val[i],1,n);  
 while(q--)  
 {  
 char op[3];  
 int x,y;  
 scanf("%s%d%d",op,&x,&y);  
 if(!strcmp(op,"CC"))  
 {  
 modify(root[C[x]],id[x],0,1,n);//将x所在的原树中的该点删除  
 modify(root[y],id[x],val[x],1,n);//将再将x插入到新的信仰y的线段树中  
 C[x] = y;//记得更改x的信仰  
 }  
 else if(!strcmp(op,"CW"))  
 {  
 modify(root[C[x]],id[x],y,1,n);//将x的所在的线段树中，x的评级改变为y  
 val[x] = y;//记得改变评级  
 }  
 else if(!strcmp(op,"QS"))//下边的查询操作就跟普通线段树相同了。  
 {  
 int res = 0,c = C[x];  
 while(top[x]!=top[y])   
 {  
 if(dep[top[x]]<dep[top[y]]) swap(x,y);  
 res += querysum(root[c],id[top[x]],id[x],1,n);  
 x = fa[top[x]];  
 }  
 if(dep[x]<dep[y]) swap(x,y);  
 res += querysum(root[c],id[y],id[x],1,n);  
 printf("%d\n",res);  
 }  
 else  
 {  
 int res = 0,c = C[x];  
 while(top[x]!=top[y])  
 {  
 if(dep[top[x]]<dep[top[y]]) swap(x,y);  
 res = max(res,querymx(root[c],id[top[x]],id[x],1,n));  
 x = fa[top[x]];  
 }  
 if(dep[x]<dep[y]) swap(x,y);  
 res = max(res,querymx(root[c],id[y],id[x],1,n));  
 printf("%d\n",res);  
 }  
 }  
 return 0;  
}

# 树状数组

int tr[N]  
void add(int x,int c)  
{  
 for(int i=x;i<=n;i+=lowbit(i)) tr[i]+=c;  
}  
  
int sum(int x)  
{  
 int res = 0;  
 for(int i=x;i;i-=lowbit(i)) res+=tr[i];  
 return res;  
}

### 单点修，求n阶前缀和

应用即为，我们可以求**区间加，区间求n-1阶前缀和**

二阶就不说了。

三阶的式子为

$$c\_x=\sum\_{i=1}^{x}b\_i \\=\sum\_{i=1}^{x}\sum\_{j=1}^{i}a\_j \\=\sum\_{i=1}^{x}\sum\_{j=1}^{i}\sum\_{k=1}^{j}d\_k \\=\sum\_{k=1}{x}\frac{(x+2-k)(x+1-k)}{2}d\_k \\=\frac{(x+2)(x+1)}{2}\sum\_{k=1}^{x}d\_k - \frac{2x+3}{2}\sum\_{k=1}^{x}d\_k\*k+\frac{1}{2}\sum\_{k=1}^{x}d\_k\*k^2$$

ll tr[3][N\*2];  
void add(int x,int c)  
{  
 ll t = x;  
 while(x<=2\*n+1)  
 {  
 tr[0][x] += c;  
 tr[1][x] += c\*t;  
 tr[2][x] += c\*t\*t;  
 x += x & -x;  
 }  
}  
  
ll sum(int x)  
{  
 ll res = 0,t = x;  
 while(x>0)  
 {  
 ll tmp = 0;  
 tmp += tr[0][x]\*(t+2)\*(t+1);  
 tmp -= tr[1][x]\*(t\*2+3);  
 tmp += tr[2][x];  
 res += tmp/2;  
 x -= x & -x;  
 }  
 return res;  
}

# Kruskal重构树

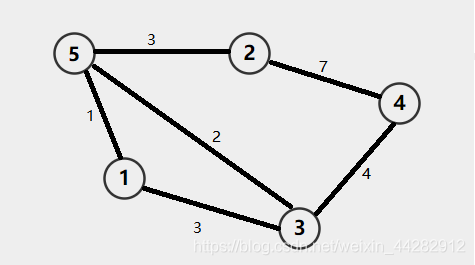
### 实现过程

这点并不难说

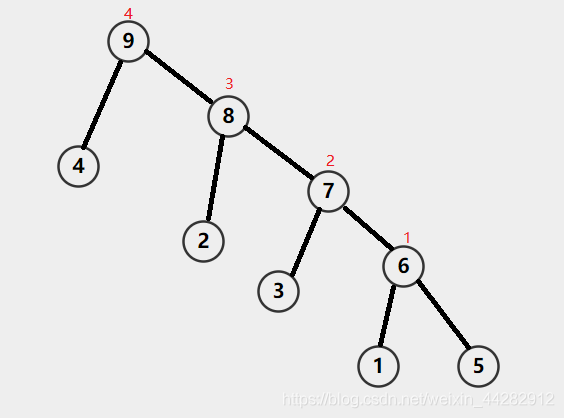
* 将边按照从大至小(从小到大)的顺序排序
* 接下来实现Kruskal算法，枚举每条边，若该边两个端点不在一个集合中，则**建立一个新点，点权为该边的边权**。
* 将边的两个端点与该点放到一个集合中，并且将从新点向两个端点的父节点分别连一条边

至此，Kruskal重构树已经完成架构。

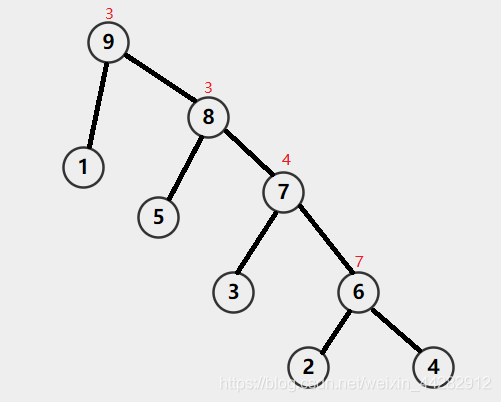
我们来看看图，来更直观的感受一下实现过程



当我们将边权从小到大排序建树时



当我们将边权从大到小排序建树时



到这里，应该算是比较直观了，我们来看看代码实现

其中有一些注意事项

* **Kruskal重构树的点数是原图的二倍点数**
* **在Kruskal重构树中，所有原点都为叶节点**
* **并查集数组，记得初始化二倍原点点数**
* **当然，如果原图不连通，我们建出来的是一个森林**

const int N = 2e4 + 10,M = 5e4 + 10;//记得点数要开到原图点数的二倍  
struct node  
{  
 int u,v,w;  
 bool operator<(const node& W) const   
 {  
 return w>W.w;  
 }  
}edges[M];//边的结构体  
int h[N],e[N],ne[N],w[N],idx;//链式向前星用来存数  
int p[N];//并查集数组  
  
void add(int a,int b)//加边数组  
{  
 e[idx]=b,ne[idx]=h[a],h[a]=idx++;  
}  
  
int find(int x)//并查集  
{  
 if(p[x]!=x) p[x]=find(p[x]);  
 return p[x];  
}  
//建树过程  
for(int i=1;i<=2\*n;i++) h[i] = -1,p[i] = i;//头结点与并查集数组的初始化  
for(int i=0;i<m;i++)//将边存起来  
{  
 int u,v,c;scanf("%d%d%d",&u,&v,&c);  
 edges[i] = {u,v,c};  
}  
sort(edges,edges+m);//将边排序  
int cnt = n;  
for(int i=0;i<m;i++)  
{  
 int pa = find(edges[i].u),pb = find(edges[i].v),c = edges[i].w;  
 if(pa!=pb)  
 {  
 w[++cnt] = c;//记得将边权存在新点上  
 p[pa] = p[pb] = cnt;//将边的两个端点与新点放到同个集合中  
 add(cnt,pa),add(cnt,pb);//从新点向两个端点的父节点连边  
 if(cnt==n\*2-1) break;//多组样例的时候就别break了\*\*\*  
 }  
}

### 性质

接下来这部分才是最重要的，我们来看看Kruskal重构树有什么性质

* 若原图不连通，那么建出来的 *Kruskal* 重构树就是一个森林。
* 如果一开始按照边权升序排序，那么建出的 *Kruskal* 重构树就是一个大根堆，反之就是小根堆。
* 若一开始按照边权升序排序，那么 lca(u,v) 的权值代表了原图中 **u** 到 **v** 路径上最大边权的最小值。反之就是最小边权的最大值。
* **Kruskal** 重构树中的叶子结点必定是原图中的节点，其余的节点都是原图的一条边。
* *Kruskal* 重构树建好以后会比原图多出 n-1个节点（如果原图联通的话）

基本上，在注意事项中，我们都有提到过

我们主要证明一下第三条，**若一开始按照边权升序排序，那么 lca(u,v) 的权值代表了原图中 u 到 v 路径上最大边权的最小值。反之就是最小边权的最大值。**

我们，证明若按照升序排列，则LCA代表的是原图中u到v的路径的最大边权的最小值。

首先，我们简单证明一下第二条，按照升序排列，父节点均比其子节点大，则为一个大根堆。

若按照升序排列，则从叶节点向上走时，我们都是尽量沿着边权小的值走的，因此深度较小的LCA则为u->v中边权最大的值。同理我们可以证明降序排列。

### 应用

说了这么多，但其实应用才是我们最关注的问题。

#### **u->v的路径中，最大边权的最小值/最小边权的最大值**

这个用法就是我们性质的最直接运用，这里跟两道例题

##### 货车运输

[原题链接](https://www.luogu.com.cn/problem/P1967)

看到题目，要我们求的就是，**最小边权的最大值**了，直接求一下LCA即可。

也给我们一个提示，遇到这种**最大值最小**或者**最小值最大**这种类似的语句，可以不急着想二分，还可以想想 *Kruskal* 重构树。

至于求LCA，可以用树剖，也可以用倍增法，看你的喜欢。

一般因为树剖用的空间更小，一般我喜欢用树剖。但是！！倍增也可以的，不要因为不会树剖就不写了，倍增也是有倍增才能干的事。

解法

* 将边权降序排列，建出Kruskal重构树，可能是森林，注意标记一下
* 接下里就求LCA即可

当然，因为这是Kruskal重构树的讲解，但我们也简单讲一下另一个写法。

* 我们建立原图的最大生成树(用Kruskal的时候，将边权降序排列)
* 接下来对建立出来的最大生成树用倍增法求LCA时，我们额外预处理一个数组mmin,它维护的是倍增的路径中的边权最小值
* 接下来求LCA的时候，就直接用倍增法，顺便维护一下路径中的边权最小值即可

两个代码，我都会贴一下

**Kruskal重构树**

#include<bits/stdc++.h>  
using namespace std;  
const int N = 6e4 + 10;  
struct Edge  
{  
 int u,v,w;  
 bool operator<(const Edge& W)const  
 {  
 return w>W.w;  
 }  
}edges[N];  
int h[N],e[N],ne[N],w[N],idx;  
int p[N];  
bool st[N];  
int sz[N],fa[N],son[N],dep[N];  
int top[N];  
int n,m;  
  
void add(int a,int b)  
{  
 e[idx] = b,ne[idx] = h[a],h[a] = idx++;  
}  
  
int find(int x)  
{  
 if(x!=p[x]) p[x] = find(p[x]);  
 return p[x];  
}  
  
void dfs1(int u,int pa,int depth)  
{  
 st[u] = 1,sz[u] = 1,fa[u] = pa,dep[u] = depth;  
 for(int i=h[u];~i;i=ne[i])  
 {  
 int j = e[i];  
 if(j==pa) continue;  
 dfs1(j,u,depth+1);  
 sz[u] += sz[j];  
 if(sz[j]>sz[son[u]]) son[u] = j;  
 }  
}  
  
void dfs2(int u,int tp)  
{  
 top[u] = tp;  
 if(!son[u]) return ;  
 dfs2(son[u],tp);  
 for(int i=h[u];~i;i=ne[i])  
 {  
 int j = e[i];  
 if(j==son[u]||j==fa[u]) continue;  
 dfs2(j,j);  
 }  
}  
  
int LCA(int u,int v)  
{  
 while(top[u]!=top[v])  
 {  
 if(dep[top[u]]<dep[top[v]]) swap(u,v);  
 u = fa[top[u]];  
 }  
 return dep[u]<dep[v]?u:v;  
}  
  
int main()  
{  
 scanf("%d%d",&n,&m);  
 memset(h,-1,sizeof h);  
 for(int i=1;i<=m;i++)  
 {  
 int u,v,c;  
 scanf("%d%d%d",&u,&v,&c);  
 edges[i] = {u,v,c};  
 }  
 sort(edges+1,edges+m+1);  
 for(int i=1;i<=n;i++) p[i] = i;  
 for(int i=1;i<=m;i++)  
 {  
 int u = edges[i].u,v = edges[i].v,c = edges[i].w;  
 int pa = find(u),pb = find(v);  
 if(pa!=pb)  
 {  
 w[++n] = c;  
 p[n] = p[pa] = p[pb] = n;  
 add(n,pa);  
 add(n,pb);  
 }  
 }  
 for(int i=1;i<=n;i++)  
 if(!st[i]){  
 int f = find(i);  
 dfs1(f,-1,1),dfs2(f,f);  
 }  
 int q;scanf("%d",&q);  
 while(q--)  
 {  
 int u,v;scanf("%d%d",&u,&v);  
 int pa = find(u),pb = find(v);  
 if(pa!=pb) puts("-1");  
 else printf("%d\n",w[LCA(u,v)]);  
 }  
 return 0;  
}

**最大生成树+倍增**

#include<bits/stdc++.h>  
using namespace std;  
const int N = 1e4 + 10,M = 2\*N;  
struct node  
{  
 int a,b,w;  
 bool used;  
 bool operator<(const node &W) const{  
 return w>W.w;  
 }  
}edges[50010];  
int h[N],e[M],ne[M],w[M],idx;  
int d[N],fa[N][16],dmin[N][16],p[N];  
bool st[N];  
int n,m;  
  
template < typename T >  
inline void read(T &x)  
{  
 x = 0; bool f = 0; char ch = getchar();  
 while(!isdigit(ch)){f ^= !(ch ^ 45);ch=getchar();}  
 while(isdigit(ch)) x= (x<<1)+(x<<3)+(ch&15),ch=getchar();  
 x = f ? -x : x;  
}  
  
void add(int a,int b,int c){  
 e[idx]=b,ne[idx]=h[a],w[idx]=c,h[a]=idx++;  
}  
  
int find(int x){  
 if(p[x]!=x) p[x]=find(p[x]);  
 return p[x];  
}  
  
void bfs(int root)  
{  
 d[root]=1;  
 queue<int> q;  
 q.push(root);  
 while(q.size())  
 {  
 int t = q.front();  
 q.pop();  
 for(int i=h[t];~i;i=ne[i])  
 {  
 int j = e[i];  
 if(d[j]>d[t]+1)  
 {  
 q.push(j);  
 d[j]=d[t]+1;  
 fa[j][0]=t;  
 dmin[j][0]=w[i];  
 for(int k=1;k<=15;k++)  
 {  
 int anc = fa[j][k-1];  
 fa[j][k]=fa[anc][k-1];  
 dmin[j][k]=min(dmin[j][k-1],dmin[anc][k-1]);  
 }  
 }  
 }  
 }  
}  
  
int lca(int a,int b)  
{  
 int ans = 0x3f3f3f3f;  
 if(d[a]<d[b]) swap(a,b);  
 for(int i=15;i>=0;i--)  
 if(d[fa[a][i]]>=d[b])  
 {  
 ans = min(dmin[a][i],ans);  
 a=fa[a][i];  
 }  
 if(a==b) return ans;  
 for(int i=15;i>=0;i--)  
 if(fa[a][i]!=fa[b][i])  
 {  
 ans = min(ans,dmin[a][i]);  
 ans = min(ans,dmin[b][i]);  
 a=fa[a][i];  
 b=fa[b][i];  
 }  
 ans = min(min(dmin[a][0],dmin[b][0]),ans);  
 return ans;  
}  
  
int main()  
{  
 read(n),read(m);  
 for(int i=0;i<m;i++)  
 {  
 int a,b,c;  
 read(a),read(b),read(c);  
 edges[i]={a,b,c,0};  
 }  
 sort(edges,edges+m);  
 for(int i=1;i<=n;i++)  
 {  
 h[i]=-1;  
 p[i]=i;  
 }  
 for(int i=0;i<m;i++)  
 {  
 int a = edges[i].a,b = edges[i].b;  
 int pa=find(a),pb=find(b);  
 if(pa!=pb)  
 {  
 p[pa]=pb;  
 edges[i].used=1;  
 }  
 }  
 int cnt = 0;  
 for(int i=0;i<m;i++)  
 if(edges[i].used){  
 add(edges[i].a,edges[i].b,edges[i].w);  
 add(edges[i].b,edges[i].a,edges[i].w);  
 }  
 memset(d,0x3f,sizeof d);  
 memset(dmin,0x3f,sizeof dmin);  
 d[0]=0;  
 for(int i=1;i<=n;i++){  
 int x = find(i);  
 if(!st[x]){  
 st[x]=1;  
 bfs(x);  
 }  
 }  
 int q;  
 read(q);  
 while(q--)  
 {  
 int x,y;  
 read(x),read(y);  
 int pa = find(x),pb = find(y);  
 if(pa!=pb) puts("-1");  
 else printf("%d\n",lca(x,y));  
 }  
  
 return 0;  
}

#### 使得一堆点联通/断开的所有边，最大边权最小值/最小边权最大值

其中断开与联通略有不同，但后面的最大边权最小值或者最小边权最大值只与建树时的排序有关。

##### 牛客练习赛62——水灾（断开）

[原题链接](https://ac.nowcoder.com/acm/contest/5205/E)

几乎与货车运输一模一样，我们依旧要求最小边权的最大值

但是略有不同的是，这次我们要知道的是**将k个点断开使得其互相不连通需要去掉的所有边集的最大边权的最小值是多少**

我们先考虑将两个点断开的答案是什么？显而易见，即为两者之间的最小边权最大值。那我们建降序Kruskal重构树，然后求LCA就好。

那k个点我们就求个k个点的LCA，就开开心心的结束了....嘛？

遗憾的是，这样写样例都过不去。

为什么呢？

因为这样的话，有些点之间断开他们，需要断开的最小边权最大值，比我们直接求出的总的LCA的权值要大。

这样我们就会漏掉一些点。

如果我们不想漏，那就需要一个结论。

**结论：想要按顺序求出点集中任意两点的LCA，只需要将点集按照字典序排好，相邻两个求LCA即可。**

这样我们就可以把每一个LCA都考虑到。

#include<bits/stdc++.h>  
using namespace std;  
const int N = 1e6 + 10,M = 5e5 + 10;  
struct Edge  
{  
 int u,v,w;  
 bool operator<(const Edge &W) const  
 {  
 return w>W.w;  
 }  
}edges[M];  
int h[N],e[N],ne[N],w[N],idx;  
int sz[N],dep[N],fa[N],son[N];  
int top[N],p[N],id[N],d[N],ts;  
int n,m,q;  
  
void add(int a,int b)  
{  
 e[idx] = b,ne[idx] = h[a],h[a] = idx++;  
}  
  
int find(int x)  
{  
 if(p[x]!=x) p[x] = find(p[x]);  
 return p[x];  
}  
  
void dfs1(int u,int pa,int depth)  
{  
 sz[u] = 1,fa[u] = pa,dep[u] = depth;  
 for(int i=h[u];~i;i=ne[i])  
 {  
 int j = e[i];  
 dfs1(j,u,depth+1);  
 sz[u] += sz[j];  
 if(sz[j]>sz[son[u]]) son[u] = j;  
 }  
}  
  
void dfs2(int u,int tp)  
{  
 top[u] = tp,id[u] = ++ts;  
 if(!son[u]) return ;  
 dfs2(son[u],tp);  
 for(int i=h[u];~i;i=ne[i])  
 {  
 int j = e[i];  
 if(j==son[u]) continue;  
 dfs2(j,j);  
 }  
}  
  
int lca(int u,int v)  
{  
 while(top[u]!=top[v])  
 {  
 if(dep[top[u]]<dep[top[v]]) swap(u,v);  
 u = fa[top[u]];  
 }  
 return dep[u]<dep[v]?u:v;  
}  
  
bool cmp(int i,int j)  
{  
 return id[i]<id[j];  
}  
  
int main()  
{  
 scanf("%d%d%d",&n,&m,&q);  
 memset(h,-1,sizeof h);   
 for(int i=1;i<=m;i++)  
 {  
 int u,v,c;scanf("%d%d%d",&u,&v,&c);  
 edges[i] = {u,v,c};  
 }  
 sort(edges+1,edges+m+1);  
 for(int i=1;i<=2\*n;i++) p[i] = i;  
 int cnt = n;  
 for(int i=1;i<=m;i++)  
 {  
 int u = edges[i].u,v = edges[i].v,c = edges[i].w;  
 int pa = find(u),pb = find(v);  
 if(pa!=pb)  
 {  
 w[++cnt] = c;  
 p[pa] = p[pb] = cnt;  
 add(cnt,pa),add(cnt,pb);  
 if(cnt==n\*2-1) break;   
 }  
 }  
 dfs1(cnt,-1,1),dfs2(cnt,cnt);  
 int ans = 0;  
 while(q--)  
 {  
 int k;scanf("%d",&k);  
 int now = 0;  
 for(int i=0;i<k;i++) scanf("%d",d+i),d[i]^=ans;  
 sort(d,d+k,cmp);  
 ans = 0;  
 for(int i=1;i<k;i++)  
 ans = max(ans,w[lca(d[i-1],d[i])]);  
 printf("%d\n",ans);  
 }  
 return 0;  
}

##### E. Qpwoeirut and Vertices（联通）

[E. Qpwoeirut and Vertices](https://codeforces.com/contest/1706/problem/E)

分析

分析起来，倒是不算太麻烦。

简单说明一下题意：

**n个点，m条边的无向图。其中边按照其给的顺序赋边权，即为第一条给出的边，其边权为1，第二条给出的边，其边权为2，等。**

**给我们q次询问，每次询问给一个范围[l,r]，要求其中的点全部连通需要走的最大边权最小值。**

一看到，无向图，最大边权最小值。直接Kruskal重构树。

首先，要求最大边权最小值，那一定升序建树。

接下来，我们考虑使得一堆点互相之间可以互相走到。

**结论：求无向图中选择一个边集使得一堆点互相之间可以走到的所有边集最大边权的最小值，则只需要求这一堆点升序建完Kruskal重构树的LCA，其权值即为答案**

对于这个的理解可以从下边两个角度。

1. 我们知道，Kruskal重构树中除叶节点外的其余节点，都可以表示在限制内我们可以走到任意叶节点。则找到所有点的LCA，代表所有点都可以去其他点能经过的最大边权最小值。
2. 当一个点其想到达其余点的时候所经过的最大边权最小值，就是其到每个点的路径中的最大边权最小值的最小值（因为可以绕路）。同理我们就可以得出对于一堆点其两两之间互达也就是求这一堆点的LCA。

但我们不能每次给一堆点，我们都去求这些点的LCA，那时间复杂度我们不能接受。

我们就要用另外一个结论。

**一堆点的LCA，即为其DFS序中最小的点与最大的点的LCA**

如果这些点是乱序的，我们还是没办法。我们还是只能每次就枚举区间内每个点求LCA。

但是这些点是一个连续的区间，那我们就可以用st表的时间求出来这个区间的dfs序最大与最小的点。

则，这道题就结束了。

AC\_code

#include<bits/stdc++.h>  
#define ios ios::sync\_with\_stdio(false); cin.tie(0), cout.tie(0)  
using namespace std;  
  
const int N = 2e5 + 10,M = 2e5 + 10;  
int h[N],e[N],ne[N],w[N],idx;  
int sz[N],son[N],dep[N],fa[N];  
int dfn[N],top[N],ts;  
int p[N];  
int st1[20][N>>1],st2[20][N>>1],Log[N>>1];  
int n,m,q;  
  
void add(int a,int b)  
{  
 e[idx] = b,ne[idx] = h[a],h[a] = idx++;  
}  
  
int find(int x)  
{  
 if(p[x]!=x) p[x] = find(p[x]);  
 return p[x];  
}  
  
void dfs1(int u)  
{  
 sz[u] = 1;  
 for(int i=h[u];~i;i=ne[i])  
 {  
 int j = e[i];  
 dep[j] = dep[u] + 1;fa[j] = u;  
 dfs1(j);  
 sz[u] += sz[j];  
 if(sz[j]>sz[son[u]]) son[u] = j;  
 }  
}  
  
void dfs2(int u,int tp)  
{  
 top[u] = tp;dfn[u] = ++ts;  
 if(!son[u]) return ;  
 dfs2(son[u],tp);  
 for(int i=h[u];~i;i=ne[i])  
 {  
 int j = e[i];  
 if(j==son[u]) continue;  
 dfs2(j,j);  
 }  
}  
  
int query1(int l,int r)  
{  
 int k = Log[r-l+1];  
 return dfn[st1[k][l]]>dfn[st1[k][r-(1<<k)+1]]?st1[k][l]:st1[k][r-(1<<k)+1];  
}  
  
int query2(int l,int r)  
{  
 int k = Log[r-l+1];  
 return dfn[st2[k][l]]<dfn[st2[k][r-(1<<k)+1]]?st2[k][l]:st2[k][r-(1<<k)+1];  
}  
  
int lca(int u,int v)  
{  
 // int cnt = 0;  
 while(top[u]!=top[v])  
 {  
 if(dep[top[u]]<dep[top[v]]) swap(u,v);  
 u = fa[top[u]];  
 // cnt++;if(cnt>10000000) exit(0);  
 }  
 return dep[u]<dep[v]?u:v;  
}  
  
void solve()  
{  
 cin>>n>>m>>q;  
 for(int i=1;i<=2\*n;i++) h[i] = -1,p[i] = i,w[i] = 0,son[i] = 0;  
 ts = idx = 0;  
 int cnt = n;  
 for(int i=0;i<m;i++)   
 {  
 int u,v;cin>>u>>v;  
 int pa = find(u),pb = find(v),c = i + 1;  
 if(pa!=pb)  
 {  
 w[++cnt] = c;  
 p[pa] = p[pb] = cnt;  
 add(cnt,pa),add(cnt,pb);  
 }  
 }  
 fa[2\*n-1] = 0,dep[2\*n-1] = 0;  
 dfs1(2\*n-1);  
 dfs2(2\*n-1,2\*n-1);  
 for(int j=0;j<20;j++)  
 for(int i=1;i+(1<<j)-1<=n;i++)  
 {  
 if(!j)   
 {  
 st1[j][i] = i;  
 st2[j][i] = i;  
 }  
 else   
 {  
 st1[j][i] = dfn[st1[j-1][i]]>dfn[st1[j-1][i+(1<<j-1)]]?st1[j-1][i]:st1[j-1][i+(1<<j-1)];  
 st2[j][i] = dfn[st2[j-1][i]]<dfn[st2[j-1][i+(1<<j-1)]]?st2[j-1][i]:st2[j-1][i+(1<<j-1)];  
 }  
 }  
 while(q--)  
 {  
 int u,v;cin>>u>>v;  
 cout<<w[lca(query1(u,v),query2(u,v))]<<" ";  
 }  
 cout<<"\n";  
}  
  
int main()  
{  
 ios;  
 int T;cin>>T;  
 Log[0] = -1;  
 for(int i=1;i<(N>>1);i++) Log[i] = ((i&(i-1))==0)?(Log[i-1]+1):Log[i-1];  
 while(T--)  
 {  
 solve();  
 }  
 return 0;  
}

#### **从 u 出发只经过边权不超过 （不低于于）x 的边能到达的节点**

根据性质，可以发现，只需要找到边权升序（降序）的 *Kruskal* 重构树中找到深度最小的，点权不超过 （不低于）x 的节点，那么这个节点的子树即为所求。

**因为若能到达该节点，则可以到达该节点的所有叶节点**

找这个点一般用**树上倍增**，这就是我说的有些题目只有倍增可做，因为倍增可以准确的**找到这个点**，而树剖并不可以。

这里有三道例题，难度逐渐递增

##### 46届ICPC上海H——Life is a Game

[原题链接](https://ac.nowcoder.com/acm/contest/24872/H)

这是这届的铜牌题，也是这种应用中比较简单的一种体现。

简单分析题目后，我们发现，我们要求的就是

**从给定的起点出发，我们想过一条边，需要我们的能力值超过边权，每到达一个点后，我们现在的所拥有的能力值为初始值+该节点下的子树的所有权值。**

**我们最后归纳一下，即为求从u出发，只经过不超过自身能力值的边所能达到的最高节点，答案即为初始值+该点下的所有节点能给的能力值**

解法很简单

* 先将边权升序排列，接下来建立Kruskal重构树。
* 初始倍增数组fa[N][20],并且计算出，树中每个节点下面子树的能给予的能力值的和（只有初始的n个点有值）
* 对于所有的询问，我们只需要从给定的起点出发不断递归到答案即可。

需要强调一下的事情是

**我们对我们的哨兵加一个特判，我们的哨兵是0号点，而零号点的权值为0，若不加特判，则哨兵也符合我们的条件，则会死循环**

**因此，当我们判断的是不超过的时候，我们应该判断一下倍增过去的点一定是大于0的**

看看代码实现

#include<bits/stdc++.h>  
using namespace std;  
const int N = 2e5 + 10,M = 1e5 + 10;  
struct Edge  
{  
 int u,v,w;  
 bool operator<(const Edge& W)const  
 {  
 return w<W.w;  
 }  
}edges[M];  
int h[N],e[N],ne[N],w[N],idx;  
int fa[N][20],sum[N],a[N],p[N];  
int n,m,q;  
  
void add(int a,int b)  
{  
 e[idx] = b,ne[idx] = h[a],h[a] = idx++;  
}  
  
int find(int x)  
{  
 if(p[x]!=x) p[x] = find(p[x]);  
 return p[x];  
}  
  
void dfs(int u)  
{  
 if(u<=n) sum[u] = a[u];  
 for(int i=h[u];~i;i=ne[i])  
 {  
 int j = e[i];  
 fa[j][0] = u;  
 for(int k=1;k<=19;k++) fa[j][k] = fa[fa[j][k-1]][k-1];  
 dfs(j);  
 sum[u] += sum[j];  
 }  
}  
  
int get(int u,int x,int res)  
{  
 if(!fa[u][0]) return x + res;  
 if(w[fa[u][0]]>x+res) return x+res;  
 for(int i=19;i>=0;i--)  
 if(fa[u][i]&&x+res>=w[fa[u][i]])  
 u = fa[u][i];  
 return get(u,x,sum[u]);  
}  
  
int main()  
{  
 scanf("%d%d%d",&n,&m,&q);  
 for(int i=1;i<=2\*n;i++) h[i] = -1,p[i] = i;  
 for(int i=1;i<=n;i++) scanf("%d",a+i);  
 for(int i=0;i<m;i++)  
 {  
 int u,v,c;scanf("%d%d%d",&u,&v,&c);  
 edges[i] = {u,v,c};  
 }  
 sort(edges,edges+m);  
 int cnt = n;  
 for(int i=0;i<m;i++)  
 {  
 int pa = find(edges[i].u),pb = find(edges[i].v),c = edges[i].w;  
 if(pa!=pb)  
 {  
 w[++cnt] = c;  
 p[pa] = p[pb] = cnt;  
 add(cnt,pa),add(cnt,pb);  
 if(cnt==n\*2-1) break;  
 }  
 }  
 dfs(2\*n-1);  
 while(q--)  
 {  
 int x,k;scanf("%d%d",&x,&k);  
 printf("%d\n",get(x,k,a[x]));  
 }  
 return 0;  
}

##### P4768 [NOI2018] 归程

[原题链接](https://www.luogu.com.cn/problem/P4768)

这题就要比上一题复杂的一些，但思路依旧是极其相似，我们来模仿上一题进行归纳。

我们要求的就是

**我们要求的是只经过海拔不低于x的边，能到达的所有点中，到达1号点的最短路径**

因此，我们的步骤如下

* 首先先从1号点跑出到其余所有点的最短路
* 将边权降序排列，建立Kruskal重构树
* 接下来，从重构树的根节点，跑一个树形DP，跑出**树中所有点到1号点的最短路**。并且预处理出来倍增数组
* 最后，对于每次询问，我们从v跑出答案即可

我们，来看看代码

#include<bits/stdc++.h>  
using namespace std;  
typedef pair<int,int> PII;  
const int N = 4e5 + 10,M = N<<1,INF = 0x3f3f3f3f;  
struct Edge  
{  
 int u,v,w;  
 bool operator<(const Edge& W)const  
 {  
 return w>W.w;  
 }  
}edges[N];  
int h[N],e[M],ne[M],w[M],idx;  
int f[N],fa[N][20],p[N];  
bool st[N];  
int n,m,q,k,s;  
  
void add(int a,int b,int c)  
{  
 e[idx] = b,ne[idx] = h[a],w[idx] = c,h[a] = idx++;  
}  
  
void add(int a,int b)  
{  
 e[idx] = b,ne[idx] = h[a],h[a] = idx++;  
}  
  
int find(int x)  
{  
 if(p[x]!=x) p[x] = find(p[x]);  
 return p[x];  
}  
  
void dijkstra()  
{  
 priority\_queue<PII,vector<PII>,greater<PII>> q;  
 q.push({0,1});  
 f[1] = 0;  
 while(q.size())  
 {  
 auto t = q.top();  
 q.pop();  
 int ver = t.second;  
 if(st[ver]) continue;  
 st[ver] = 1;  
 for(int i=h[ver];~i;i=ne[i])  
 {  
 int j = e[i];  
 if(f[j]>f[ver]+w[i])  
 {  
 f[j] = f[ver] + w[i];  
 q.push({f[j],j});  
 }  
 }  
 }   
}  
  
void dfs(int u)  
{  
 for(int i=h[u];~i;i=ne[i])  
 {  
 int j = e[i];  
 fa[j][0] = u;  
 for(int k=1;k<=19;k++) fa[j][k] = fa[fa[j][k-1]][k-1];  
 dfs(j);  
 f[u] = min(f[u],f[j]);  
 }  
}  
  
int get(int u,int p)  
{  
 for(int i=19;i>=0;i--)  
 if(w[fa[u][i]]>p)  
 u = fa[u][i];  
 return u;  
}  
  
int main()  
{  
 int T;scanf("%d",&T);  
 while(T--)  
 {  
 scanf("%d%d",&n,&m);  
 idx = 0;  
 for(int i=0;i<=2\*n;i++)  
 {  
 f[i] = INF;  
 h[i] = -1;  
 p[i] = i;  
 st[i] = 0;  
 }  
 for(int i=0;i<m;i++)  
 {  
 int u,v,l,a;  
 scanf("%d%d%d%d",&u,&v,&l,&a);  
 add(u,v,l),add(v,u,l);  
 edges[i] = {u,v,a};  
 }  
 dijkstra();  
 sort(edges,edges+m);  
 for(int i=0;i<=2\*n;i++) h[i] = -1,w[i] = 0;  
 idx = 0;  
 int cnt = n;  
 for(int i=0;i<m;i++)  
 {  
 int u = edges[i].u,v = edges[i].v,c = edges[i].w;  
 int pa = find(u),pb = find(v);  
 if(pa!=pb)  
 {  
 w[++cnt] = c;  
 p[pa] = p[pb] = cnt;  
 add(cnt,pa),add(cnt,pb);  
 if(cnt==2\*n-1) break;  
 }  
 }  
 dfs(find(1));  
 scanf("%d%d%d",&q,&k,&s);  
 int lastans = 0;  
 while(q--)  
 {  
 int v0,p0;scanf("%d%d",&v0,&p0);  
 int v = (v0+lastans\*k-1)%n+1,p = (p0+k\*lastans)%(s+1);  
 printf("%d\n",lastans = f[get(v,p)]);  
 }  
 }  
 return 0;  
}

##### P7834 [ONTAK2010] Peaks 加强版

[原题链接](https://www.luogu.com.cn/problem/P7834)

这题，题意说的很明白了

**求从 u 开始只经过权值 小于等于x 的边所能到达的权值第 k 大的点的权值，如果不存在输出 -1。**

我们知道，只经过不超过x的边，所能到达的所有原图中的点，即为我们在重构树中跑出符合限制的节点的子树的叶节点。

那我们需要操作的即为**对于重构树中跑出的节点的子树中的叶节点中权值第k的点的权值**

不难想到，求权值第k大，我们可以用主席树。

同时，一个子树中所有的叶节点编号，我没想让它们连续并不难办。

我们只需要对dfn与sz数组稍作改变。使dfn[u]只记录叶节点u的编号，sz[u]只记录对应节点u下的叶节点数量。

并且，我们增加两个辅助数组f,tk,其中f[u]用来记录，在子树中最小的叶节点编号，而tk则是桶排数组，其记录的是dfs序所对应的节点编号。

我们可以发现f与dfn数组可以合为一个。

因此，当我们找到符合限制的节点后，其间的叶节点的编号范围为[f[u],f[u]+sz[u]-1]

我们再利用，主席树即可求出区间第k大

嗷，千万别忘记离散化了。

我们来看看代码

#include<bits/stdc++.h>  
using namespace std;  
const int N = 1e5 + 10,M = 5e5 + 10,INF = 0x3f3f3f3f;  
struct Node  
{  
 int l,r;  
 int cnt;  
}tr[N\*4+N\*17];//主席树结构体  
struct Edge  
{  
 int u,v,w;  
 bool operator<(const Edge& W)const  
 {  
 return w<W.w;  
 }  
}edges[M];//边的结构体  
int h[N<<1],e[N<<1],ne[N<<1],w[N<<1],idx;//用来存树  
int a[N],p[N<<1];//a是原图节点的权值，p为重构树中各节点编号的并查集数组  
int fa[N<<1][20],f[N<<1],sz[N<<1],ts;//倍增数组与辅助数组  
int root[N],id,tk[N];//主席树版本数组与桶排数组  
vector<int> nums;//离散化后的数组  
int n,m,Q;  
  
void add(int a,int b)//加边  
{  
 e[idx] = b,ne[idx] = h[a],h[a] = idx++;  
}  
  
int find(int x)//并查集  
{  
 if(p[x]!=x) p[x] = find(p[x]);  
 return p[x];  
}  
  
int get(int x)//获取a的离散化后的下标  
{  
 return lower\_bound(nums.begin(),nums.end(),x) - nums.begin();  
}  
  
int get(int u,int x)//获取符合限制的节点编号  
{  
 for(int i=19;i>=0;i--)  
 if(fa[u][i]&&w[fa[u][i]]<=x)//记得为哨兵节点加一个特判  
 u = fa[u][i];  
 return u;  
}  
  
void dfs(int u)//对倍增数组的初始化，并对f，sz，tk数组初始化  
{  
 if(u<=n) f[u] = ++ts,sz[u] = 1,tk[ts] = u;  
 else f[u] = INF;  
 for(int i=h[u];~i;i=ne[i])  
 {  
 int j = e[i];  
 fa[j][0] = u;  
 for(int k=1;k<=19;k++) fa[j][k] = fa[fa[j][k-1]][k-1];  
 dfs(j);  
 f[u] = min(f[u],f[j]);  
 sz[u] += sz[j];  
 }  
}  
  
int insert(int p,int l,int r,int x)//建立不同版本的主席树  
{  
 int q = ++id;  
 tr[q] = tr[p];  
 if(l==r)  
 {  
 tr[q].cnt++;  
 return q;  
 }  
 int mid = l + r >> 1;  
 if(x<=mid) tr[q].l = insert(tr[p].l,l,mid,x);  
 else tr[q].r = insert(tr[p].r,mid+1,r,x);  
 tr[q].cnt=tr[tr[q].l].cnt+tr[tr[q].r].cnt;  
 return q;  
}  
  
int query(int q,int p,int l,int r,int k)//查询区间第k小(我们将求的第k大做个转化即可)  
{  
 if(l==r) return l;  
 int mid = l + r >> 1;  
 int cnt = tr[tr[q].l].cnt - tr[tr[p].l].cnt;  
 if(k<=cnt) return query(tr[q].l,tr[p].l,l,mid,k);  
 else return query(tr[q].r,tr[p].r,mid+1,r,k-cnt);  
}  
  
int main()  
{  
 scanf("%d%d%d",&n,&m,&Q);  
 for(int i=1;i<=2\*n;i++) p[i] = i,h[i] = -1;//初始化  
 for(int i=1;i<=n;i++) scanf("%d",a+i),nums.push\_back(a[i]);  
 sort(nums.begin(),nums.end());  
 nums.erase(unique(nums.begin(),nums.end()),nums.end());//离散化  
 for(int i=0;i<m;i++)  
 {  
 int u,v,c;scanf("%d%d%d",&u,&v,&c);  
 edges[i] = {u,v,c};  
 }  
 sort(edges,edges+m);  
 int t = n;  
 for(int i=0;i<m;i++)//建树  
 {  
 int u = edges[i].u,v = edges[i].v,c = edges[i].w;  
 int pa = find(u),pb = find(v);  
 if(pa!=pb)  
 {  
 w[++t] = c;  
 p[pa] = p[pb] = t;  
 add(t,pa),add(t,pb);  
 if(t==2\*n-1) break;  
 }  
 }  
 dfs(t);  
 for(int i=1;i<=n;i++)//建立主席树  
 root[i] = insert(root[i-1],0,nums.size()-1,get(a[tk[i]]));  
 int lastans = 0;  
 while(Q--)  
 {  
 int u0,x0,k0;scanf("%d%d%d",&u0,&x0,&k0);  
 int u = (u0^lastans)%n+1,k = (k0^lastans)%n+1,x = x0^lastans;  
 int ii = get(u,x);  
 if(sz[ii]<k) puts("-1"),lastans = 0;//判断子树中是否有k个叶节点  
 else printf("%d\n",lastans = nums[query(root[f[ii]+sz[ii]-1],root[f[ii]-1],0,nums.size()-1,sz[ii]-k+1)]);//将第k大，转化为求sz[ii]-k+1小  
 }  
}

# 堆

## 模拟堆

### 目的

1. 插入一个数x
2. 输出当前集合中的最小值
3. 删除当前集合的最小值
4. 删除第k个插入的数
5. 修改第k个插入的数，将其变为x

### 实现

struct modheap  
{  
 int h[N],ph[N],hp[N],idx=0,len=0;  
 void heap\_swap(int a,int b)  
 {  
 swap(h[a],h[b]);  
 swap(hp[a],hp[b]);  
 swap(ph[hp[a]],ph[hp[b]]);  
 }  
 void down(int x)  
 {  
 int t = x;  
 if(x\*2<=len&&h[x\*2]<h[t]) t=2\*x;  
 if(x\*2+1<=len&&h[x\*2+1]<h[t]) t=2\*x+1;  
 if(t!=x)  
 {  
 heap\_swap(t,x);  
 down(t);  
 }  
 }  
 void up(int x)  
 {  
 if(x/2&&h[x/2]>h[x])   
 {  
 heap\_swap(x/2,x);  
 up(x>>1);  
 }  
 }  
 void insert(int x)  
 {  
 len++,idx++;  
 h[len]=x;  
 ph[idx]=len,hp[len]=idx;  
 up(len);  
 }  
 int get()  
 {  
 return h[1];  
 }  
 void del\_mi()  
 {  
 heap\_swap(1,len);  
 len--;  
 down(1);  
 }  
 void del\_kth(int k)  
 {  
 int u = ph[k];  
 heap\_swap(u,len);  
 len--;  
 up(u);down(u);  
 }  
 void mod\_kth(int k,int x)  
 {  
 h[ph[k]]=x;  
 up(ph[k]);down(ph[k]);  
 }  
};

## 对顶堆

### 目的

动态维护中位数

### 实现原理

对顶堆，一个是小根堆，另一个是大根堆。假设 g 是大根堆，l 是小根堆，那么观察下图不难发现，两堆中间的元素，左边都是小于它的，且 g.top() 是小于它的最大值，右边都是大于它的，且 l .top() 是大于它的最小值。

### 实现

struct midheap  
{  
 priority\_queue<int> g;  
 priority\_queue<int,vector<int>,greater<int>> l;  
 void insert(int x)  
 {  
 if(!g.size()||x<g.top()) g.push(x);  
 else l.push(x);  
 if(g.size()>l.size()+1) l.push(g.top()),g.pop();  
 if(l.size()>g.size()+1) g.push(l.top()),l.pop();  
 }  
 //若出现偶数个，且中间两个不同的情况，则该get获得的是较大的  
 //若想得到较小的，则将>改为>=  
 int get()  
 {  
 return g.size() > l.size() ? g.top() : l.top();  
 }  
};

## 可删除堆

### 目的

可以完成删除堆内指定元素。

### 实现原理

我们建一个临时堆，如果要删除哪个元素，就把哪个元素压入临时堆，然后待此元素和正常堆的堆顶元素相同时（即两个堆顶一样），就同时pop掉。

### 实现

//可删除堆  
struct popheap  
{  
 priority\_queue<int> \_add,\_del;//其中的方向要相同，可以使大根堆也可以是小根堆  
 int size()  
 {  
 while(!\_add.empty()&&!\_del.empty()&&\_add.top()==\_del.top()) \_add.pop(),\_del.pop();   
 return \_add.size();  
 }  
 int get()  
 {  
 while(!\_add.empty()&&!\_del.empty()&&\_add.top()==\_del.top()) \_add.pop(),\_del.pop();   
 return \_add.top();  
 }  
 void add(int x)  
 {  
 \_add.push(x);  
 }  
 void del(int x)  
 {  
 \_del.push(x);  
 }  
};

# 并查集

## 可撤销并查集

其中一定记住，成功连上了我们才可以进行撤销。

struct DSU  
{  
 int fa[N];  
 int sz[N];  
 vector<pair<int&, int>>his\_sz;  
 vector<pair<int&, int>>his\_fa;  
 void init(int n) {  
 for (int i = 1; i <= n; i++)fa[i] = i, sz[i] = 1;  
 }  
 int find(int x) {  
 while (x != fa[x])x = fa[x];  
 return x;  
 }  
 bool same(int u, int v) {  
 return find(u) == find(v);  
 }  
 void merge(int u, int v) {  
 int x = find(u);  
 int y = find(v);  
 if (x == y) return;  
 if (sz[x] < sz[y]) std::swap(x, y);  
 his\_sz.push\_back({ sz[x], sz[x] });  
 sz[x] = sz[x] + sz[y];  
 his\_fa.push\_back({ fa[y],fa[y] });  
 fa[y] = x;  
 }  
   
 int histroy() {  
 return his\_fa.size();  
 }  
   
 void roll(int h) {  
 while (his\_fa.size() > h) {  
 his\_fa.back().first = his\_fa.back().second;  
 his\_fa.pop\_back();  
 his\_sz.back().first = his\_sz.back().second;  
 his\_sz.pop\_back();  
 }  
 }  
   
}dsu;

# ST表

template<typename T=int,const int N=18,typename Comp=less<T>>//Comp则是设立比较方式  
struct ST  
{  
 T ans[N][1<<N];int n,lg[1<<N],k;  
 void init(T\*a,int n\_){  
 n=n\_;  
 lg[0]=-1;for(int i=1;i<=n;i++)lg[i]=lg[i>>1]+1;  
 for(int i=1;i<=n;i++)ans[0][i]=\*++a;  
 for(int i=1;i<N;i++)  
 for(int j=1;j+(1<<i)<=n+1;j++)  
 ans[i][j]=max(ans[i-1][j],ans[i-1][j+(1<<(i-1))],Comp());  
 }  
 int query(const int l,const int r){  
 k=31-\_\_builtin\_clz(r-l+1);   
 return max(ans[k][l],ans[k][r-(1<<k)+1],Comp());  
 }  
};  
  
struct cmp  
{  
 bool operator()(const int a,const int b)  
 {  
 return a<b;  
 }  
};  
  
ST<int,17,cmp> st;

# 二维数点

**可以求得，数组区间[l,r]内，大于等于x，小于等于y的数字个数。**

## 离线

我们实现，数组区间[l,r]内，**大于等于x，小于等于y**的数字个数。

步骤是，先将区间内每一个数插入。。接下来将每一个询问插入，。

最后直接查询即可。

利用树状数组实现。

struct tp {  
 // add 添加点  
 // que 添加矩形 参数为矩形的左下角，右上角，询问的id   
 // get 传入一个数组，答案会放到数组里面  
 static const int maxnum = 1e7 + 5;//询问数据范围  
 static const int maxn = 5e5 + 5;//区间和问题的数量  
 int tree[maxnum];  
 int n = 0, m = 0;  
 struct node1  
 {  
 int x, y;  
 }v[maxn];  
 struct node2  
 {  
 int x, y, id, type;  
 } q[maxn << 2];  
 static bool cmp1(node1 &a, node1 &b) {  
 return a.x < b.x;  
 }  
 static bool cmp2(node2 &a, node2 &b) {  
 return a.x < b.x;  
 }  
 void update(int idx, int x) {  
 while(idx<=maxnum)  
 {  
 tree[idx] += x;  
 idx += idx & -idx;  
 }  
 }  
 int query(int n) {  
 int ans = 0;  
 while(n)  
 {  
 ans += tree[n];  
 n -= n & -n;   
 }  
 return ans;  
 }  
 int cnt = 0;  
 void add(int x, int y) {  
 v[++n].x = x, v[n].y = y;  
 }  
 void que(int x1, int y1, int x2, int y2, int i) {  
 q[++cnt] = { x2, y2, i, 1 };  
 q[++cnt] = { x1 - 1, y2, i, -1 };  
 q[++cnt] = { x2, y1 - 1, i, -1 };  
 q[++cnt] = { x1 - 1, y1 - 1, i, 1 };  
 m++;  
 }  
 void get(int \*ans) {  
 sort(v + 1, v + 1 + n, cmp1);  
 sort(q + 1, q + 1 + cnt, cmp2);  
 int u = 1;  
 for (int i = 1; i <= cnt; i++) {  
 while (v[u].x <= q[i].x&&u <= n) {  
 update(v[u].y, 1);  
 u++;  
 }  
 ans[q[i].id] += q[i].type \* (query(q[i].y));  
 }  
 for (int i = 1; i <= m; i++) ans[i] = max(ans[i], 0);  
 }  
};

## 在线

就像我们说的，离线的办法，我们改成主席树就可以变成在线了。