**数据结构**

2020.09.05

目录

[数据结构 2](#_Toc50207907)

[并查集 2](#_Toc50207908)

[带权并查集 2](#_Toc50207909)

[按秩合并 2](#_Toc50207910)

[路径压缩 2](#_Toc50207911)

[线段树 3](#_Toc50207912)

[树状数组 3](#_Toc50207913)

[修改与询问 3](#_Toc50207914)

[二分 3](#_Toc50207915)

[二维树状数组 4](#_Toc50207916)

[笛卡尔树 4](#_Toc50207917)

[构造 4](#_Toc50207918)

[代码 4](#_Toc50207919)

[莫队专题 5](#_Toc50207920)

[普通莫队 5](#_Toc50207921)

[带修莫队 5](#_Toc50207922)

[树上莫队 7](#_Toc50207923)

[其他 10](#_Toc50207924)

[主席树与带修主席树 11](#_Toc50207925)

[主席树 11](#_Toc50207926)

[带修主席树 12](#_Toc50207927)

[平衡树 14](#_Toc50207928)

[普通平衡树 14](#_Toc50207929)

[文艺平衡树 21](#_Toc50207930)

[可持久化平衡树 27](#_Toc50207931)

[图上数据结构 31](#_Toc50207932)

[动态树 31](#_Toc50207933)

[圆方树 38](#_Toc50207934)

[点分树 39](#_Toc50207935)

[Kd-Tree 39](#_Toc50207936)

# 数据结构

## 并查集

### 带权并查集

**// r[N]为权值**

int find(int x)

{

int p;

if(f[x] == x) return x;

p = f[x]; f[x] = find(f[x]);

r[x] = (r[x] + r[p]) % 3;

return f[x];

}

void link()

{

f[fb] = fa;

r[fb] = ((3 - r[b]) + p - 1 + r[a]) % 3;

}

### 按秩合并

if(rank[fx] < rank[fy])

{

f[fx]=fy;

rank[fy]=max(rank[fy], rank[fx]+1);

}

else

{

f[fy]=fx;

rank[fx]=max(rank[fx], rank[fy]+1);

}

### 路径压缩

int find(int x)

{

if(f[x] == x) return x;

f[x] = find(f[x]); return f[x];

}

## 线段树

## 树状数组

### 修改与询问

int lowbit(int x){ return x&(-x); }

void insert(int x,int y)

{

for(int i = x;i <= n;i += lowbit(i)) tree[i] += y;

}

int query(int x)

{

int ans=0;

for(int i = x; i; i -= lowbit(i)) ans += tree[i];

return ans;

}

### 二分

int query(int x)//查询大于等于x的第一个位置

{

int now=0;

for(int i=20;i>=0;i--)

{

if(now+(1<<i)<=n)

{

if(tree[now+(1<<i)]<x)

{

x-=tree[now+(1<<i)];

now+=(1<<i);

}

}

}

return now+1;

}

### 二维树状数组

## 笛卡尔树

将一个序列构造成满足如下条件的一棵树：

1. 父节点比子节点小（大）
2. 树的中序遍历，与原序列相同

### 构造

1. 将根、根的右子树、根的右子树的右子树……加入栈内
2. 插入新元素时，将栈内比当前元素大的元素弹出，然后将当前元素作为栈顶元素的右子树，栈中最后一个被弹出的元素为该元素的左子树。将该元素插入栈内。
3. 注意当前元素没有左子树或当前元素为根的情况。

### 代码

void build()

{

rt = 1;

stk[1] = 1;

int top = 1;

for(int i = 2; i <= n; i++)

{

while(top && p[stk[top]] > p[i])

{

l[i] = stk[top];

top--;

}

if(top) r[stk[top]] = i;

else rt = i;

stk[++top] = i;

}

}

## 莫队专题

### 普通莫队

离线处理查询问题的解决方法。

询问离线，第一关键字所在块，第二关键字的绝对位置。

代码

**//创建块**  
m=sqrt(n);bel[0]=1;  
for(int i=1;i<=n;i++)  
{  
 scanf("%d",&a[i]);  
 bel[i]=i/m+1;  
}  
**//排序**  
bool operator<(nod a,nod b)  
{  
 return bel[a.l]!=bel[b.l] ? bel[a.l]<bel[b.l] : a.r<b.r;  
}  
**//莫队**  
int nl=0,nr=0;ll res=0;  
for(int i=1;i<=q;i++)  
{  
 int l=query[i].l,r=query[i].r;  
 while(nr<r) nr++,cntr[a[nr]]++,res+=cntl[a[nr]];  
 while(nl<l) nl++,cntl[a[nl]]++,res+=cntr[a[nl]];  
 while(nr>r) res-=cntl[a[nr]],cntr[a[nr]]--,nr--;  
 while(nl>l) res-=cntr[a[nl]],cntl[a[nl]]--,nl--;  
 ans[query[i].id]+=(((ll)query[i].flag)\*res);  
}

### 带修莫队

带修莫队显然就是增加修改操作的莫队

询问离线，第一关键字l所在块，第二关键字r所在块，第三关键字时间。

**//创建块**m=pow(n,2.0/3.0);  
for(int i=1;i<=n;i++)  
{  
 scanf("%d",&col[i]);temp[i]=col[i];  
 bel[i]=((i-1)/m)+1;  
}  
**//排序**bool operator<(nod1 a,nod1 b)  
{  
 if(bel[a.l]!=bel[b.l]) return bel[a.l]<bel[b.l];  
 if(bel[a.r]!=bel[b.r]) return bel[a.r]<bel[b.r];  
 return a.pos<b.pos;  
}  
**//更新状态  
//注意，更改时间操作的时候需要维护前一个时间是谁，方便回退状态**void update(int now)  
{  
 if(cnt[col[now]]==0) res++;  
 cnt[col[now]]++;  
}  
void erase(int now)  
{  
 cnt[col[now]]--;  
 if(cnt[col[now]]==0) res--;  
}  
void update(int now,int nl,int nr)  
{  
 if(nl<=modify[now].u && modify[now].u<=nr) erase(modify[now].u);  
 col[modify[now].u]=modify[now].now;  
 if(nl<=modify[now].u && modify[now].u<=nr) update(modify[now].u);  
}  
void erase(int now,int nl,int nr)  
{  
 if(nl<=modify[now].u && modify[now].u<=nr) erase(modify[now].u);  
 col[modify[now].u]=modify[now].pre;  
 if(nl<=modify[now].u && modify[now].u<=nr) update(modify[now].u);  
}  
**//第一步可以强行更新，防止边界出错**int nl=query[1].l,nr=query[1].r,nt=query[1].pos;  
for(int i=nl;i<=nr;i++) update(i);  
for(int i=1;i<=nt;i++) update(i,nl,nr);  
ans[query[1].id]=res;  
for(int i=2;i<=tot1;i++)  
{  
 int l=query[i].l,r=query[i].r,t=query[i].pos;  
 while(nr<r) nr++,update(nr);  
 while(l<nl) nl--,update(nl);  
 while(r<nr) erase(nr),nr--;  
 while(nl<l) erase(nl),nl++;  
 while(nt<t) nt++,update(nt,l,r);  
 while(t<nt) erase(nt,l,r),nt--;  
 ans[query[i].id]=res;  
}

### 树上莫队

考虑全序，点到点 之间只出现过一次的点，再加上点与点的，就是p到q路径上的点。

然后莫队就行了

这里 比较需要注意的是query的时候l和r到底选哪个，分情况讨论。

代码

//CF GYM 100962F   
#include<iostream>   
#include<stdio.h>   
#include<string.h>   
#include<algorithm>   
#include<math.h>   
#define maxn 200005   
using namespace std;   
typedef long long ll;   
struct nod   
{   
 int l,r,id;   
 nod(int a,int b,int c)   
 {   
 l=a;   
 r=b;   
 id=c;   
 }   
 nod(){}   
   
}query[maxn];   
int n,q,tot,bel[maxn],m;   
int head[maxn],nex[maxn],to[maxn],w[maxn];   
void add(int x,int y,int z)   
{   
 to[++tot]=y; nex[tot]=head[x]; w[tot]=z; head[x]=tot;   
}   
bool operator<(nod a,nod b)   
{   
 return bel[a.l]!=bel[b.l] ? bel[a.l]<bel[b.l] : a.r<b.r;   
}   
   
int dfn[maxn],val[maxn],in[maxn],out[maxn];   
void dfs(int now,int fa)   
{   
 in[now]=++tot;   
 dfn[tot]=now;   
 for(int i=head[now];i;i=nex[i])   
 {   
 if(to[i]!=fa)   
 {   
 val[to[i]]=min(n,w[i]);   
 dfs(to[i],now);   
 }   
 }   
 out[now]=++tot;   
 dfn[tot]=now;   
}   
int l[maxn],r[maxn];   
int cnt[maxn],sum[maxn],vis[maxn];   
void update(int now)   
{   
 if(vis[now]==0)   
 {   
 vis[now]=1;   
   
 cnt[val[now]]++;   
 if(cnt[val[now]]==1)   
 sum[bel[val[now]]]++;   
 }   
 else   
 {   
 vis[now]=0;   
   
 cnt[val[now]]--;   
 if(cnt[val[now]]==0)   
 sum[bel[val[now]]]--;   
 }   
}   
int ans[maxn];   
int getans()   
{   
 for(int i=1;i<=(n-1)/m+1;i++)   
 {   
 if(sum[i]!=(r[i]-l[i]+1))   
 {   
 for(int j=l[i];j<=r[i];j++)   
 {   
 if(!cnt[j])   
 {   
 return j-1;   
 }   
 }   
 }   
 }   
 return n;   
}   
int main()   
{   
 memset(l,0x3f,sizeof(l));   
 scanf("%d%d",&n,&q);   
   
 m=320;   
 for(int i=1;i<=n;i++)   
 {   
 bel[i]=(i-1)/m+1;   
 l[bel[i]]=min(l[bel[i]],i);   
 r[bel[i]]=max(r[bel[i]],i);   
 }   
   
 for(int i=1;i<n;i++)   
 {   
 int x,y,z;   
 scanf("%d%d%d",&x,&y,&z);   
 z++;   
 add(x,y,z);   
 add(y,x,z);   
 }   
   
 tot=0;   
 dfs(1,-1);   
   
 for(int i=1;i<=q;i++)   
 {   
 int u,v;   
 scanf("%d%d",&u,&v);   
 query[i].id=i;   
 if(in[u]>in[v]) swap(u,v);   
   
 **//这里是边权**    
 query[i].l=in[u]+1;   
 query[i].r=in[v];   
   
 **//如果是点权** //if(lca==u){query[i].l = in[u];query[i].r = in[v];}   
 //else {query[i].l = out[u];query[i].r = in[v];}   
 }   
   
 sort(query+1,query+1+q);   
   
 int nl=1,nr=0;   
 for(int i=1;i<=q;i++)   
 {   
 int l=query[i].l,r=query[i].r;   
 while(nr<r) nr++,update(dfn[nr]);   
 while(l<nl) nl--,update(dfn[nl]);   
 while(r<nr) update(dfn[nr]),nr--;   
 while(nl<l) update(dfn[nl]),nl++;   
  **//本题是边权不含lca   
 //如果是点权要加一句 if(!cnt[val[query[i].lca]]) update()**   
 ans[query[i].id]=getans();   
 }   
   
 for(int i=1;i<=q;i++)   
 {   
 printf("%d\n",ans[i]);   
 }   
}

### 其他

操作，的分块和莫队更配哦。

## 主席树与带修主席树

### 主席树

#include<iostream>

#include<cstdio>

#include<cstdlib>

#include<algorithm>

#include<cmath>

#include<cstring>

using namespace std;

const int N=2e5+10;

struct TREE{

int l,r,size;

}tree[N\*20];

int h[N],a[N],root[N];

int cnt;

void insert(int x,int &y,int v,int l,int r)

{

y=++cnt;

tree[y]=tree[x];

tree[y].size++;

if(l==r) return;

int mid=(l+r)>>1;

if(v>mid) insert(tree[x].r,tree[y].r,v,mid+1,r);

else insert(tree[x].l,tree[y].l,v,l,mid);

}

int query(int x,int y,int k,int l,int r)

{

if(l==r) return l;

int tot,mid;

tot=tree[tree[y].l].size-tree[tree[x].l].size;

mid=(l+r)>>1;

if(tot<k) return query(tree[x].r,tree[y].r,k-tot,mid+1,r);

else return query(tree[x].l,tree[y].l,k,l,mid);

}

int main()

{

int n,m,i,L,l,r,k,x;

scanf("%d%d",&n,&m);

for(i=1;i<=n;i++)

{

scanf("%d",&a[i]);

h[i]=a[i];

}

sort(h+1,h+n+1);

L=unique(h+1,h+n+1)-h-1;

for(i=1;i<=n;i++)

{

x=lower\_bound(h+1,h+L+1,a[i])-h;

insert(root[i-1],root[i],x,1,L);

}

for(i=1;i<=m;i++)

{

scanf("%d%d%d",&l,&r,&k);

printf("%d\n",h[query(root[l-1],root[r],k,1,L)]);

}

return 0;

}

### 带修主席树

代码（CQOI2011动态逆序对）

注意：空间复杂度O()

#include<iostream>

#include<cstdio>

#include<cstdlib>

#include<algorithm>

#include<cmath>

using namespace std;

const int N=1e5+10;

struct apple{ int size,l,r;}tree[N\*90];

int root[N],a[N],vist[N],p[N],h[N];

long long ans[N];

int cnt,n;

int lowbit(int x){ return x&(-x); }

void insert(int &x,int v,int l,int r) **// 线段树插入**

{

if(!x) x=++cnt;

tree[x].size++;

if(l==r) return;

int mid=(l+r)>>1;

if(v>mid) insert(tree[x].r,v,mid+1,r);

else insert(tree[x].l,v,l,mid);

}

void ins(int x) **//树状数组插入**

{

for(int i=x;i<=n;i+=lowbit(i)) insert(root[i],a[x],1,n);

}

long long query(int x,int v,int l,int r) **//线段树查询**

{

if(!x) return 0;

if(r<=v) return tree[x].size;

if(l>v) return 0;

int mid=(l+r)>>1;

return query(tree[x].l,v,l,mid)+query(tree[x].r,v,mid+1,r);

}

long long query(int x,int v) **//树状数组查询**

**{**

long long now=0;

for(int i=x;i>0;i-=lowbit(i)) now+=query(root[i],v,1,n);

return now;

}

long long solve(int x)

{ return query(n,a[x])+query(x,n)-2LL\*query(x,a[x]); }

int main()

{

int m,i;

scanf("%d%d",&n,&m);

for(i=1;i<=n;i++)

{

scanf("%d",&a[i]);

h[a[i]]=i;

}

for(i=1;i<=m;i++)

{

scanf("%d",&p[i]);

vist[h[p[i]]]=1;

}

long long now=0;

for(i=1;i<=n;i++)

if(!vist[i])

{

now+=solve(i);

ins(i);

}

for(i=m;i>0;i--)

{

now+=solve(h[p[i]]);

ans[i]=now;

ins(h[p[i]]);

}

for(i=1;i<=m;i++) printf("%lld\n",ans[i]);

return 0;

}

## 平衡树

### 普通平衡树

* 1. 操作分别为
     1. 插入 x数
     2. 删除 x 数(若有多个相同的数，因只删除一个)
     3. 查询 x 数的排名(排名定义为比当前数小的数的个数 +1 )
     4. 查询排名为 x 的数
     5. 求 x 的前驱(前驱定义为小于 x，且最大的数)
     6. 求 x的后继(后继定义为大于 x，且最小的数)
  2. Splay版本（为了防止特殊数据，添加了随机换根操作）
     1. #include<iostream> #include<stdio.h> #include<string.h> #include<algorithm> #include<set> #define maxn 100005 #define lson son[now][0] #define rson son[now][1] **using** **namespace** std; set<int> s; int n,root,tot; int son[maxn][2],fa[maxn],val[maxn],siz[maxn]; void pushup(int now) {  siz[now]=siz[lson]+siz[rson]+1; } void rotate(int x,int &k) {  int y=fa[x],z=fa[y],l,r;  l=son[y][1]==x;r=l^1;  **if**(y==k) k=x;  **else** {**if**(son[z][0]==y) son[z][0]=x;**else** son[z][1]=x;}  fa[x]=z;fa[y]=x;fa[son[x][r]]=y;  son[y][l]=son[x][r];son[x][r]=y;  pushup(y);pushup(x); } void splay(int x,int &k) {  **while**(x!=k)  {  int y=fa[x],z=fa[y];  *//cout<<"fuck "<<x<<' '<<y<<" "<<k<<endl;* **if**(y!=k)  {  **if**((son[y][0]==x)^(son[z][0]==y)) rotate(x,k);  **else** rotate(y,k);  }  rotate(x,k);  } } int cnt; void insert(int &now,int v,int last) {  *//cnt++;* **if**(!now)  {  now=++tot;  s.insert(tot);  val[tot]=v;  fa[now]=last;  splay(tot,root);  int temp=rand()%tot+1;  **if**(s.find(temp)!=s.end())  splay(temp,root);  cnt=0;  **return**;  }  **if**(v<=val[now]) insert(lson,v,now);  **else** insert(rson,v,now); } void del(int v) {  int pos=root;  **while**(val[pos]!=v) pos=son[pos][val[pos]<v]; splay(pos,root);   s.erase(pos);   int now=son[pos][1],flag;  **if**(!now)  {  flag=1;  now=son[pos][0];  **if**(!now)  {  fa[pos]=0;  val[pos]=0;  root=0;  **return**;  }  }  **else**  flag=0;  **while**(son[now][flag]) now=son[now][flag];    int t=root;splay(now,root);  son[now][flag]=son[t][flag];fa[son[t][flag]]=now;fa[now]=0;  pushup(now); } int getrank(int v) {  cnt=0;  int now=root,ans=0;  **while**(now)  {  cnt++;  **if**(v>val[now]) {ans+=siz[lson]+1;now=rson;}  **else** now=lson;  }  *//cerr<<"its "<<cnt<<endl;* **return** ans+1; } int find(int v) {  int now=root;  **while**(now)  {  **if**(siz[lson]+1==v) **return** val[now];  **else** **if**(siz[lson]+1>v) now=lson;  **else** v-=(siz[lson]+1),now=rson;  }  **return** val[now]; } int suf(int v) {  int now=root,ans;  **while**(now)  {  **if**(val[now]<v){ans=val[now];now=rson;}  **else** now=lson;  }  **return** ans; } int nex(int v) {  int now=root,ans;  **while**(now)  {  **if**(val[now]<=v) now=rson;  **else** {ans=val[now];now=lson;}  }  **return** ans; } int main() {  scanf("%d",&n);  **while**(n--)  {  int op,x;  scanf("%d%d",&op,&x);  **if**(op==1) insert(root,x,0);  **if**(op==2) del(x);  **if**(op==3) printf("%d\n",getrank(x));  **if**(op==4) printf("%d\n",find(x));  **if**(op==5) printf("%d\n",suf(x));  **if**(op==6) printf("%d\n",nex(x));  } }
  3. Treap版本（好写）
     1. #include<stdio.h>#include<string.h>#include<stdlib.h>#define maxn 100005#define lson son[now][0]#define rson son[now][1]#define min(a,b) ((a)<(b) ? (a):(b))#define max(a,b) ((a)>(b) ? (a):(b))const int inf=2e9+5;**typedef** **struct** nod{ int first,second;} nod;int root,tot,sz[maxn];int son[maxn][2],val[maxn],k[maxn];void pushup(int now){ sz[now]=sz[lson]+sz[rson]+1;}int merge(int x,int y){ **if**(!x || !y) **return** x+y; **if**(k[x]<k[y]) **return** son[x][1]=merge(son[x][1],y),pushup(x),x; **else** **return** son[y][0]=merge(x,son[y][0]),pushup(y),y;}nod split(int now,int v){ **if**(!now) { nod temp; temp.first=0; temp.second=0; **return** temp; } nod y; **if**(sz[lson]>=v) { y=split(lson,v); lson=y.second; pushup(now); y.second=now; } **else** { y=split(rson,v-sz[lson]-1); rson=y.first; pushup(now); y.first=now; } **return** y;}int find(int v){ nod x=split(root,v-1),y=split(x.second,1); int ans=y.first; root=merge(merge(x.first,ans),y.second); **return** val[ans];}int rk(int now,int v){ int minn=inf,maxx=0; **while**(now) { **if**(val[now]==v) minn=min(minn,maxx+sz[lson]+1); **if**(v>val[now]) maxx+=sz[lson]+1,now=rson; **else** now=lson; } **return** minn==inf ? maxx : minn;}void insert(int v){ int pos=rk(root,v); nod x=split(root,pos); sz[++tot]=1; val[tot]=v; k[tot]=rand(); root=merge(merge(x.first,tot),x.second);}void del(int v){ int pos=rk(root,v); nod x=split(root,pos-1),y=split(x.second,1); root=merge(x.first,y.second);}int suc(int v){ int now=root,ans=inf; **while**(now) { **if**(val[now]>v) { ans=min(ans,val[now]); now=lson; } **else** now=rson; } **return** ans;}int pre(int v){ int now=root,ans=-inf; **while**(now) { **if**(val[now]<v) { ans=max(ans,val[now]); now=rson; } **else** now=lson; } **return** ans;}int main(){ int \_; scanf("%d",&\_); **while**(\_--) { int op,x; scanf("%d%d",&op,&x); **if**(op==1) insert(x); **if**(op==2) del(x); **if**(op==3) printf("%d\n",rk(root,x)); **if**(op==4) printf("%d\n",find(x)); **if**(op==5) printf("%d\n",pre(x)); **if**(op==6) printf("%d\n",suc(x)); } **return** 0;}

### 文艺平衡树

* 1. 集成区间赋值和翻转的Splay
     1. #include<iostream>#include<stdio.h>#include<string.h>#include<algorithm>#include<queue> #define inf 1000000000#define maxn 1000005**using** **namespace** std;int n,m,root,cnt;queue<int>q;bool tag[maxn],rev[maxn];int fa[maxn],id[maxn],a[maxn],son[maxn][2];int sum[maxn],siz[maxn],v[maxn],maxx[maxn],lx[maxn],rx[maxn];void pushup(int now){ int lson=son[now][0],rson=son[now][1]; sum[now]=sum[lson]+sum[rson]+v[now]; siz[now]=siz[lson]+siz[rson]+1; maxx[now]=max(maxx[lson],maxx[rson]); maxx[now]=max(maxx[now],lx[rson]+v[now]+rx[lson]); lx[now]=max(lx[lson],sum[lson]+v[now]+lx[rson]); rx[now]=max(rx[rson],sum[rson]+v[now]+rx[lson]);}void pushdown(int now){ int lson=son[now][0],rson=son[now][1]; **if**(tag[now]) { tag[now]=rev[now]=0; **if**(lson) tag[lson]=1,v[lson]=v[now],sum[lson]=v[now]\*siz[lson]; **if**(rson) tag[rson]=1,v[rson]=v[now],sum[rson]=v[now]\*siz[rson]; **if**(v[now]>=0) { **if**(lson) lx[lson]=rx[lson]=maxx[lson]=sum[lson]; **if**(rson) lx[rson]=rx[rson]=maxx[rson]=sum[rson]; } **else** { **if**(lson) lx[lson]=rx[lson]=0,maxx[lson]=v[now]; **if**(rson) lx[rson]=rx[rson]=0,maxx[rson]=v[now]; } } **if**(rev[now]) { rev[now]^=1;rev[lson]^=1;rev[rson]^=1; swap(lx[lson],rx[lson]);swap(lx[rson],rx[rson]); swap(son[lson][0],son[lson][1]);swap(son[rson][0],son[rson][1]); }}void rotate(int x,int &k){ int y=fa[x],z=fa[y],l,r; l=(son[y][1]==x);r=l^1; **if**(y==k) k=x; **else** son[z][son[z][1]==y]=x; fa[son[x][r]]=y;fa[y]=x;fa[x]=z; son[y][l]=son[x][r];son[x][r]=y; pushup(y);pushup(x);*//顺序千万不能错*}void splay(int x,int &k){ **while**(x!=k) { int y=fa[x],z=fa[y]; **if**(y!=k) { **if**((son[y][0]==x)^(son[z][0]==y)) rotate(x,k); **else** rotate(y,k); } rotate(x,k); }}int find(int now,int rk){ pushdown(now); int lson=son[now][0],rson=son[now][1]; **if**(siz[lson]+1==rk) **return** now; **if**(siz[lson]>=rk) **return** find(lson,rk); **return** find(rson,rk-siz[lson]-1);}void set(int now){ **if**(!now) **return**; int lson=son[now][0],rson=son[now][1]; set(lson);set(rson);q.push(now);*//一定要记住push now* fa[now]=son[now][0]=son[now][1]=0; tag[now]=rev[now]=0; **return**;}int split(int now,int tot){ int x=find(root,now),y=find(root,now+tot+1); splay(x,root);splay(y,son[x][1]); **return** son[y][0];}void query(int now,int tot){ int x=split(now,tot); printf("%d\n",sum[x]);}void modify(int now,int tot,int val){ int x=split(now,tot),y=fa[x]; tag[x]=1;v[x]=val;sum[x]=val\*siz[x]; **if**(val>=0) lx[x]=rx[x]=maxx[x]=sum[x]; **else** lx[x]=rx[x]=0,maxx[x]=val; pushup(y);pushup(fa[y]);*//pushup 一定要记清*}void reverse(int now,int tot){ int x=split(now,tot),y=fa[x]; **if**(!tag[x]) { rev[x]^=1; swap(son[x][0],son[x][1]); swap(lx[x],rx[x]); pushup(y);pushup(fa[y]); }} void erase(int now,int tot){ int x=split(now,tot),y=fa[x]; set(x);son[y][0]=0; pushup(y);pushup(fa[y]);}void build(int l,int r,int flag){ **if**(l>r) **return**; int mid=(l+r)>>1,now=id[mid],last=id[flag]; **if**(l==r) { sum[now]=a[l]; siz[now]=1; tag[now]=rev[now]=0; **if**(a[l]>0) lx[now]=rx[now]=maxx[now]=a[l]; **else** lx[now]=rx[now]=0,maxx[now]=a[l]; } **else** build(l,mid-1,mid),build(mid+1,r,mid); v[now]=a[mid];fa[now]=last;pushup(now); son[last][mid>=flag]=now;}void insert(int now,int tot){ **for**(int i=1;i<=tot;i++) scanf("%d",&a[i]); **for**(int i=1;i<=tot;i++) { **if**(!q.empty()) id[i]=q.front(),q.pop(); **else** id[i]=++cnt; } build(1,tot,0); int z=id[(1+tot)>>1]; int x=find(root,now+1),y=find(root,now+2); splay(x,root);splay(y,son[x][1]); fa[z]=y;son[y][0]=z; pushup(y);pushup(x);}void out(int now){ pushdown(now); int lson=son[now][0],rson=son[now][1]; **if**(lson) out(lson); **if**(2<=now && now<=n+1) printf("%d ",now-1); **if**(rson) out(rson);}int main(){ scanf("%d%d",&n,&m); maxx[0]=a[1]=a[n+2]=-inf; **for**(int i=1;i<=n;i++)a[i+1]=i;; **for**(int i=1;i<=n+2;i++)id[i]=i; build(1,n+2,0); root=(n+3)>>1;cnt=n+2; **while**(m--) { int l,r; scanf("%d%d",&l,&r); reverse(l,r-l+1); } out(root);}
  2. Treap（简单好写）
     1. #include<iostream>#include<stdio.h>#include<string.h>#include<algorithm>#define maxn 100005#define lson (son[now][0])#define rson (son[now][1])#define mid ((nl+nr)>>1)**using** **namespace** std;int n,m,root;int fa[maxn],son[maxn][2],sz[maxn],rev[maxn];void pushup(int now){ sz[now]=sz[lson]+sz[rson]+1;}void pushdown(int now){ **if**(rev[now]) { swap(lson,rson); rev[lson]^=1;  rev[rson]^=1; rev[now]^=1; }}void build(int nl,int nr,int pa){ **if**(nl>nr) **return**; **if**(nl==nr) { sz[nl]=1; fa[nl]=pa; son[pa][nl>pa]=nl; **return**; } build(nl,mid-1,mid); build(mid+1,nr,mid); fa[mid]=pa; son[pa][mid>pa]=mid; pushup(mid);}void rotate(int x,int &k){ int y=fa[x],z=fa[y]; int l=(son[y][1]==x),r=l^1; **if**(y==k) k=x; **else** { **if**(son[z][0]==y) son[z][0]=x; **else** son[z][1]=x; } fa[x]=z; fa[son[x][r]]=y; fa[y]=x; son[y][l]=son[x][r]; son[x][r]=y; pushup(y); pushup(x);}void splay(int x,int &k){ **while**(x!=k) { int y=fa[x],z=fa[y]; **if**(y!=k) { int \_=0; **if**((son[z][0]==y) ^\_^ (son[y][0]==x)) rotate(x,k); **else** rotate(y,k); } rotate(x,k); }}int find(int now,int v){ *// cerr<<now<<" "<<v<<endl;* pushdown(now); **if**(sz[lson]+1==v) **return** now; **if**(sz[lson]+1<v) **return** find(rson,v-(sz[lson]+1)); **return** find(lson,v);}void reverse(int l,int r){ l=find(root,l); r=find(root,r+2); splay(l,root); splay(r,son[l][1]); rev[son[r][0]]^=1;}int main(){ scanf("%d%d",&n,&m); build(1,n+2,0); root=((1+(n+2))>>1); **for**(int i=1;i<=m;i++) { int l,r; scanf("%d%d",&l,&r); reverse(l,r); } **for**(int i=2;i<=n+1;i++) printf("%d ",find(root,i)-1);}

### 可持久化平衡树

* 1. **和原本平衡树不同的一点是，每一次的任何操作都是基于某一个历史版本，同时生成一个新的版本。（操作3, 4, 5, 6即保持原版本无变化）**
  2. 只能用Treap了
     1. #include<iostream>#include<stdio.h>#include<string.h>#include<stdlib.h>#include<algorithm>#define maxn 500005\*60#define lson son[now][0]#define rson son[now][1]**using** **namespace** std;const int inf=2147483647;**typedef** pair<int,int> pr;int root[maxn],tot,sz[maxn],cnt;int son[maxn][2],val[maxn],k[maxn];void pushup(int now){ sz[now]=sz[lson]+sz[rson]+1;}int copy(int now){ tot++; sz[tot]=sz[now]; son[tot][0]=son[now][0]; son[tot][1]=son[now][1]; val[tot]=val[now]; k[tot]=k[now]; **return** tot;}int merge(int x,int y){ **if**(!x || !y) **return** x+y; **if**(k[x]<k[y]) **return** son[x][1]=merge(son[x][1],y),pushup(x),x; **else** **return** son[y][0]=merge(x,son[y][0]),pushup(y),y;}pr split(int now,int v){ **if**(!now) **return** pr(0,0); pr y; **if**(sz[lson]>=v) { int x=copy(now); y=split(lson,v); son[x][0]=y.second; pushup(x); y.second=x; } **else** { int x=copy(now); y=split(rson,v-sz[lson]-1); son[x][1]=y.first; pushup(x); y.first=x; } **return** y;}int find(int ver,int v,int p){ pr x=split(root[ver],v-1),y=split(x.second,1); int ans=y.first; **if**(p==0) root[ver]=merge(merge(x.first,ans),y.second); **else** root[++cnt]=merge(merge(x.first,ans),y.second); **return** val[ans];}int rk(int now,int v){ int minn=inf,maxx=0; **while**(now) { **if**(val[now]==v) minn=min(minn,maxx+sz[lson]+1); **if**(v>val[now]) maxx+=sz[lson]+1,now=rson; **else** now=lson; } **return** minn==inf ? maxx+1 : minn;}int insrk(int now,int v){ int minn=inf,maxx=0; **while**(now) { **if**(val[now]==v) minn=min(minn,maxx+sz[lson]+1); **if**(v>val[now]) maxx+=sz[lson]+1,now=rson; **else** now=lson; } **return** minn==inf ? maxx : minn;}void insert(int ver,int v){ int pos=insrk(root[ver],v); pr x=split(root[ver],pos); sz[++tot]=1; val[tot]=v; k[tot]=rand(); root[++cnt]=merge(merge(x.first,tot),x.second);}void del(int ver,int v){ int pos=rk(root[ver],v); **if**(find(ver,pos,0)!=v) { root[++cnt]=root[ver]; **return**; } pr x=split(root[ver],pos-1),y=split(x.second,1); root[++cnt]=merge(x.first,y.second);}int suc(int ver,int v){ int now=root[ver],ans=inf; **while**(now) { **if**(val[now]>v) { ans=min(ans,val[now]); now=lson; } **else** now=rson; } cnt++; root[cnt]=root[ver]; **return** ans;}int pre(int ver,int v){ int now=root[ver],ans=-inf; **while**(now) { **if**(val[now]<v) { ans=max(ans,val[now]); now=rson; } **else** now=lson; } cnt++; root[cnt]=root[ver]; **return** ans;}int main(){ int \_; scanf("%d",&\_); **while**(\_--) { int ver,op,x; scanf("%d%d%d",&ver,&op,&x); **if**(op==1) insert(ver,x); **if**(op==2) del(ver,x); **if**(op==3) { printf("%d\n",rk(root[ver],x)); root[++cnt]=root[ver]; } **if**(op==4) printf("%d\n",find(ver,x,1)); **if**(op==5) printf("%d\n",pre(ver,x)); **if**(op==6) printf("%d\n",suc(ver,x)); }}

## 图上数据结构

### 动态树

**//uoj274 温暖会指引我前行，动态维护最小生成树**

**//注意每一个点和每一条边都需要开成动态树一个点**

**//动态树上每一棵子树对应一条链**

#include<iostream>

#include<cstdio>

#include<stack>

using namespace std;

const int N = 1e5 + 10 , M = (N << 2); **//N是点数，M是需要内存池大小**

class Node{

public:

Node \*ch[2] , \*p;

int rev;**//翻转标记**

int id , len , mine;

**//id是节点编号，len是代表路径长度，mine是子树维护路径的最大的边的边权**

Node(){}

Node(int a , int Id){ **//新建一个点，代表路径长度为a，编号为Id**

ch[0] = ch[1] = p = NULL;

rev = 0;

id = Id , len = a , mine = Id;

}

} \*node[M] , nodes[M];

int st[M] , to[M] , len[M] , tem[M];

int n , m;

char s[10];

const int oo = 1e9 + 10;

stack <Node \*> S;

void init(){

for(int i = 1 ; i <= n ; ++ i)

node[i] = &nodes[i] , nodes[i] = Node(0 , 0);

}

void rev(Node \*u){

if(u){

swap(u -> ch[0] , u -> ch[1]);

u -> rev ^= 1;

}

}

void down(Node \*u){

if(u -> rev){

rev(u -> ch[0]);

rev(u -> ch[1]);

u -> rev ^= 1;

}

}

void update(Node \*u){

if(u -> id){ **//当前根节点对应一条边**

int ed = u -> id , minn = tem[u -> id];

if(u -> ch[0] && u -> ch[0] -> mine && tem[u -> ch[0] -> mine] < minn)

ed = u -> ch[0] -> mine , minn = tem[u -> ch[0] -> mine];

if(u -> ch[1] && u -> ch[1] -> mine && tem[u -> ch[1] -> mine] < minn)

ed = u -> ch[1] -> mine , minn = tem[u -> ch[1] -> mine];

u -> mine = ed;

}

else{ **//当前根节点对应一个点**

int ed = 0 , minn = oo;

if(u -> ch[0] && u -> ch[0] -> mine && tem[u -> ch[0] -> mine] < minn)

ed = u -> ch[0] -> mine , minn = tem[u -> ch[0] -> mine];

if(u -> ch[1] && u -> ch[1] -> mine && tem[u -> ch[1] -> mine] < minn)

ed = u -> ch[1] -> mine , minn = tem[u -> ch[1] -> mine];

u -> mine = ed;

}

int ret = 0;

if(u -> ch[0])

ret += u -> ch[0] -> len;

if(u -> ch[1])

ret += u -> ch[1] -> len;

if(u -> id)

ret += len[u -> id];

u -> len = ret;

}

bool is\_root(Node \*u){

return u -> p == NULL || u -> p -> ch[0] != u && u -> p -> ch[1] != u;

}

void link(Node \*f , Node \*c , bool dir){

f -> ch[dir] = c;

if(c)

c -> p = f;

update(f);

}

void rotate(Node \*u , bool dir){

Node \*p = u -> p , \*ch = u -> ch[dir];

if(is\_root(u)){

link(u , ch -> ch[1 ^ dir] , dir);

link(ch , u , 1 ^ dir);

ch -> p = p;

}

else{

link(u , ch -> ch[1 ^ dir] , dir);

link(ch , u , 1 ^ dir);

link(p , ch , u == p -> ch[1]);

}

}

void splay(Node \*u){

S.push(u);

for(Node \*p = u ; !is\_root(p) ; p = p -> p)

S.push(p -> p);

while(!S.empty()){

down(S.top());

S.pop();

}

while(!is\_root(u)){

Node \*p = u -> p;

bool dir1 = u == p -> ch[1];

if(is\_root(p)){

rotate(p , dir1);

return ;

}

Node \*g = p -> p;

bool dir2 = p == g -> ch[1];

if(dir1 ^ dir2)

rotate(p , dir1) , rotate(g , dir2);

else

rotate(g , dir2) , rotate(p , dir1);

}

}

void access(Node \*u){

Node \*la = NULL , \*per = u;

while(u){

splay(u);

link(u , la , 1);

la = u;

u = u -> p;

}

splay(per);

}

void setroot(Node \*u){

access(u);

rev(u);

}

void extract(Node \*u , Node \*v){

setroot(u);

access(v);

}

void link(Node \*u , Node \*v){ **//链接两个还未连通的节点**

setroot(u);

u -> p = v;

}

void cut(Node \*u , Node \*v){ **//切断两个已经连通的节点且在原图中相邻的节点**

extract(u , v);

v -> ch[0] = u -> p = NULL;

update(v);

}

void add(Node \*u , Node \*v , int ed){ **//u到v节点增加新的编号为ed的边**

extract(u , v);

if(is\_root(u)){

link(u , node[ed + n]);

link(node[ed + n] , v);

}

else{

if(tem[v -> mine] >= tem[ed])

return ;

int e = v -> mine;

cut(node[st[e]] , node[e + n]);

cut(node[to[e]] , node[e + n]);

link(u , node[ed + n]);

link(node[ed + n] , v);

}

}

int main(){

scanf("%d%d" , &n , &m);

init();

int id , u , v , t , l;

for(int i = 1 ; i <= m ; ++ i){

scanf("%s" , s);

if(s[0] == 'f'){ **//增加新的边**

scanf("%d%d%d%d%d" , &id , &u , &v , &t , &l);

u ++ , v ++; id ++;

len[id] = l , tem[id] = t;

st[id] = u , to[id] = v;

node[id + n] = &nodes[id + n] , nodes[id + n] = Node(len[id] , id);

add(node[u] , node[v] , id);

}

else if(s[0] == 'm'){ **//询问u到v的路径长度和**

scanf("%d%d" , &u , &v);

if(u == v){

printf("0\n");

continue;

}

u ++ , v ++;

extract(node[u] , node[v]);

if(is\_root(node[u])) **//u不能到达v**

printf("-1\n");

else

printf("%d\n", node[v] -> len);

}

else{

scanf("%d%d" , &id , &l);

id ++;

setroot(node[id + n]); **//修改编号为id的边，一定要把这条边设置为根**

node[id + n] -> len = node[id + n] -> len - len[id] + l;

len[id] = l;

}

}

return 0;

}

### 圆方树

**//建立是在点双联通分量基础上修改一点**

int low[N] , dfn[N];

bool flag[N];

void dfs(int pos , int fa){

++ num; low[pos] = dfn[pos] = num;

flag[pos] = 1;

stack[++ top] = pos;

for(int e = first[pos] ; e ; e = edge[e].next){

int to = edge[e].to;

if(to == fa) continue;

if(!flag[to]){

dfs(to , pos);

low[pos] = min(low[pos] , low[to]);

if(low[to] >= dfn[pos]){ //一个新的点双连通分量

++ n; **//新加入编号为n 的点**

do{

int now = stack[top];

**这里是fa[now]=n，即点now的父亲是n**

}while(stack[top --] != to);

**这里是fa[n]=pos，即点n的父亲是pos**

}

}

else

low[pos] = min(low[pos] , dfn[to]);

}

}

**//圆方树建立后，dfs起始点是根节点**

### 点分树

## Kd-Tree

**//时间复杂度**

#include <iostream>

#include <algorithm>

using namespace std;

typedef long long ll;

const int N = 50505;

int T;

class Node{

public:

int d[2] , mi[2] , mx[2];

int ch[2] , val;

ll sum;

} node[N];

bool operator< (const Node i1 , const Node i2){

return i1.d[T] < i2.d[T];

}

int root;

void update(int x){

int c[2] = {node[x].ch[0] , node[x].ch[1]};

for(int i = 0 ; i < 2 ; ++ i){

node[x].mi[i] = node[x].mx[i] = node[x].d[i];

for(int j = 0 ; j < 2 ; ++ j)

if(c[j]){

node[x].mi[i] = min(node[x].mi[i] , node[c[j]].mi[i]);

node[x].mx[i] = max(node[x].mx[i] , node[c[j]].mx[i]);

}

}

node[x].sum = node[c[0]].sum + node[c[1]].sum + node[x].val;

}

int build(int l , int r , int tp){

T = tp; int mid = (l + r) >> 1;

nth\_element(node + l , node + mid , node + r + 1);

node[mid].ch[0] = (l < mid ? build(l , mid - 1 , tp^1) : 0);

node[mid].ch[1] = (r > mid ? build(mid + 1 , r , tp^1) : 0);

update(mid);

return mid;

}

int a , b , c;

bool check(int x , int y){

return x \* a + y \* b < c;

}

ll Query(int x){

int tot = check(node[x].mi[0] , node[x].mi[1]) + check(node[x].mx[0] , node[x].mi[1])

+ check(node[x].mi[0] , node[x].mx[1]) + check(node[x].mx[0] , node[x].mx[1]);

if(tot == 4 || !tot) return (tot == 4 ? node[x].sum : 0);

ll res = check(node[x].d[0] , node[x].d[1]) \* node[x].val;

if(node[x].ch[0]) res += Query(node[x].ch[0]);

if(node[x].ch[1]) res += Query(node[x].ch[1]);

return res;

}

int main(){

ios::sync\_with\_stdio(0);

int n , q;

cin >> n >> q;

for(int i = 1 ; i <= n ; ++ i)

cin >> node[i].d[0] >> node[i].d[1] >> node[i].val;

root = build(1 , n , 0);

for(int i = 1 ; i <= q ; ++ i){

cin >> a >> b >> c;

cout << Query(root) << endl;

}

return 0;

}