线段树

加乘线段树

```
const int maxn = 1e5+500;

int n,m,mod;

struct sgt{
  #define int long long
  #define ls rt<<1
  #define rs rt<<1|1
  int sum[maxn<<2], add[maxn<<2], mul[maxn<<2];
  void cal(int rt,int Add,int Mul,int len){
    sum[rt] = (sum[rt]*Mul%mod + len*Add%mod)%mod;
    add[rt] = (add[rt]*Mul%mod + Add)%mod;
    mul[rt] = mul[rt]*Mul%mod;
  }
  void pushup(int rt){
    sum[rt] = (sum[ls] + sum[rs])%mod;
  }
  void pushdown(int rt,int len){
    if(add[rt] != 0 || mul[rt] != 1){
      cal(ls,add[rt],mul[rt],len-len/2);
      cal(rs,add[rt],mul[rt],len/2);
      add[rt] = 0, mul[rt] = 1;
    }
  }
  void build(int l,int r,int rt){
        if(l>r) return ;
    mul[rt] = 1, add[rt] = 0;
    if(l == r){
      cin>>sum[rt];
      return ;
    }
    int mid = l+r>>1;
    build(l,mid,ls);
    build(mid+1,r,rs);
    pushup(rt);
  }
  void update(int l,int r,int rt,int L,int R,int Add,int Mul){
    if(L<=l && r<=R){
      cal(rt,Add,Mul,r-l+1);
      return ;
    }
    int mid = l+r>>1;
    pushdown(rt,r-l+1);
    if(L<=mid) update(l,mid,ls,L,R,Add,Mul);
```

```
    if(mid<R) update(mid+1,r,rs,L,R,Add,Mul);
    pushup(rt);
  }
  int query(int l,int r,int rt,int L,int R){
    if(L<=l && r<=R)
      return sum[rt];
    int mid = l+r>>1, ans = 0;
    pushdown(rt,r-l+1);
    if(L<=mid) ans = (ans + query(l,mid,ls,L,R)) % mod;
    if(mid<R) ans = (ans + query(mid+1,r,rs,L,R)) % mod;
    pushup(rt);
    return ans;
  }
}s;
```

线段树的区间最值修改 [l,r] a<-min(a,k)

```
inline void pushup(ll rt){
    tree[rt].sum = tree[ls].sum + tree[rs].sum;
    tree[rt].mx = max(tree[ls].mx,tree[rs].mx);
    if(tree[ls].mx == tree[rs].mx){
        tree[rt]._mx = max(tree[ls]._mx , tree[rs]._mx);
        tree[rt].cnt = tree[ls].cnt + tree[rs].cnt;
    }
    else{
        tree[rt]._mx = max(tree[ls]._mx , tree[rs]._mx);
        tree[rt]._mx = max(tree[rt]._mx , min(tree[ls].mx , tree[rs].mx));
        tree[rt].cnt = tree[ls].mx>tree[rs].mx?tree[ls].cnt:tree[rs].cnt;
    }
}

inline void pushdown(ll rt){
    if(tree[rt].mx!=max(tree[ls].mx,tree[rs].mx)){
        if(tree[rt].mx<tree[ls].mx&&tree[ls]._mx<=tree[rt].mx){
            tree[ls].sum -= tree[ls].cnt*(tree[ls].mx-tree[rt].mx);
            tree[ls].mx = tree[rt].mx;
        }
        if(tree[rt].mx<tree[rs].mx&&tree[rs]._mx<=tree[rt].mx){
            tree[rs].sum -= tree[rs].cnt*(tree[rs].mx-tree[rt].mx);
            tree[rs].mx = tree[rt].mx;
        }
    }
}

void update(ll rt,ll l,ll r,ll L,ll R,ll k){//改为min(a,k)
    if(tree[rt].mx<=k) return ;//k比最大值大，说明该区间不需要修改
    if(L<=l&&r<=R&&tree[rt]._mx<k){ //k介于两者之间才要修改
        tree[rt].sum -= tree[rt].cnt*(tree[rt].mx-k);
        tree[rt].mx = k;
```

```
        return ;
    }
    pushdown(rt);
    ll mid = l+r>>1;
    if(L<=mid) update(ls,l,mid,L,R,k);
    if(mid<R) update(rs,mid+1,r,L,R,k);
    pushup(rt);
}
```

维护区间和 支持区间加平方序列操作

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
#define ls rt<<1
#define rs rt<<1|1

const int maxn = 5e5+500;
const int mod = 1e9+7;

struct sgt{
  int sum2[maxn<<2], sum1[maxn<<2], c2[maxn<<2];
  int tag2[maxn<<2], tag1[maxn<<2], tagc2[maxn<<2];
  int con2[maxn<<2], con1[maxn<<2];
  void cal(int rt,int t2,int t1,int tc,int len){
    sum2[rt] = (sum2[rt] + con2[rt]*t2%mod)%mod;
    sum1[rt] = (sum1[rt] + con1[rt]*t1%mod)%mod;
    c2[rt] = (c2[rt] + len*tc%mod)%mod;

    tag2[rt] = (tag2[rt] + t2) % mod;
    tag1[rt] = (tag1[rt] + t1) % mod;
    tagc2[rt] = (tagc2[rt] + tc) % mod;
  }
  void pushdown(int rt,int len){
    if(tag1[rt] || tag2[rt] || tagc2[rt]){
      cal(ls, tag2[rt], tag1[rt], tagc2[rt], len-len/2);
      cal(rs, tag2[rt], tag1[rt], tagc2[rt], len/2);
      tag1[rt] = tag2[rt] = tagc2[rt] = 0;
    }
  }
  void pushup(int rt){
    sum1[rt] = (sum1[ls] + sum1[rs]) % mod;
    sum2[rt] = (sum2[ls] + sum2[rs]) % mod;
    c2[rt] = (c2[ls] + c2[rs]) % mod;
    con2[rt] = (con2[ls] + con2[rs]) % mod;
    con1[rt] = (con1[ls] + con1[rs]) % mod;
  }
  void build(int l,int r,int rt){
```

```
        if(l>r) return ;
    if(l == r){
      con2[rt] = l*l%mod;
      con1[rt] = l;
      sum2[rt] = sum1[rt] = c2[rt] = tag1[rt] = tag2[rt] = tagc2[rt] = 0;
      return ;
    }
    int mid = l+r>>1;
    build(l,mid,ls);
    build(mid+1,r,rs);
    pushup(rt);
  }
  void update(int l,int r,int rt,int L,int R){
    if(L<=l && r<=R){
      int be = l-L+1;
      int c = l - be;
      sum2[rt] = (sum2[rt] + con2[rt]) % mod;
      sum1[rt] = (sum1[rt] + 2*c*con1[rt]%mod)%mod;
      c2[rt] = (c2[rt] + (r-l+1)*c%mod*c%mod)%mod;
      tag2[rt] = (tag2[rt] + 1) % mod;
      tag1[rt] = (tag1[rt] + 2*c) % mod;
      tagc2[rt] = (tagc2[rt] + c*c%mod) % mod;
      return ;
    }
    int mid = l+r>>1;
    pushdown(rt,r-l+1);
    if(L<=mid) update(l,mid,ls,L,R);
    if(mid<R) update(mid+1,r,rs,L,R);
    pushup(rt);
  }
  int query(int l,int r,int rt,int L,int R){
    if(L<=l && r<=R)
      return (sum2[rt] - sum1[rt] + mod + c2[rt]) % mod;
    int mid = l+r>>1, ans = 0;
    pushdown(rt,r-l+1);
    if(L<=mid) ans = (ans + query(l,mid,ls,L,R))%mod;
    if(mid<R) ans = (ans + query(mid+1,r,rs,L,R))%mod;
    pushup(rt);
    return ans;
  }
}s;

int n,m;

signed main(){
  ios::sync_with_stdio(false),cin.tie(0),cout.tie(0);
  cin>>n>>m;
  s.build(1,n,1);
  while(m--){
```

```
    int op,l,r;
    cin>>op>>l>>r;
    if(op == 1)
      s.update(1,n,1,l,r);
    else
      cout<<s.query(1,n,1,l,r)<<'\n';
  }
  return 0;
}
```

势能线段树常见操作

1. 操作之后，区间mx和mi的差距会不断变小。例如，区间开根号、区间除法。到了最大值和最小值的大小相同时，可以通过区间加减法修改
2. 操作次数有限，操作完之后，可以给区间覆盖。覆盖之后，可以快速更新和查询。例如，lowbit和打蚊子

区间加 + 区间开平方

```
#include <bits/stdc++.h>
#define INF 0x3f3f3f3f
#define Mn 200010
#define Mm 2000005
#define ul u<<1
#define ur (u<<1)|1
using namespace std;
typedef long long ll;
ll a[Mn];
ll sum[Mn*4];
ll maxx[Mn*4],minn[Mn*4];
ll maxNum[Mn*4],minNum[Mn*4];
ll all[Mn*4];
ll lazy[Mn*4];
void pushUp(int u) {
    sum[u]=sum[ul]+sum[ur];
    maxx[u]=max(maxx[ul],maxx[ur]);
    minn[u]=min(minn[ul],minn[ur]);
    maxNum[u]=minNum[u]=0;
    if(maxx[u]==maxx[ul]) maxNum[u]+=maxNum[ul];
    if(maxx[u]==maxx[ur]) maxNum[u]+=maxNum[ur];
    if(minn[u]==minn[ul]) minNum[u]+=minNum[ul];
    if(minn[u]==minn[ur]) minNum[u]+=minNum[ur];
}
void pushDown(int u,int l,int r) {
    if(all[u]) {
        int mid=(l+r)>>1;
        all[ul]=all[ur]=all[u];
        sum[ul]=all[u]*(mid-l+1);
        sum[ur]=all[u]*(r-mid);
```

```
        maxx[ul]=maxx[ur]=minn[ul]=minn[ur]=all[u];
        maxNum[ul]=minNum[ul]=mid-l+1;
        maxNum[ur]=minNum[ur]=r-mid;
        lazy[ul]=lazy[ur]=0;
        all[u]=0;
    }
    if(lazy[u]) {
        int mid=(l+r)>>1;
        lazy[ul]+=lazy[u];lazy[ur]+=lazy[u];
        sum[ul]+=lazy[u]*(mid-l+1);
        sum[ur]+=lazy[u]*(r-mid);
        maxx[ul]+=lazy[u];minn[ul]+=lazy[u];
        maxx[ur]+=lazy[u];minn[ur]+=lazy[u];
        lazy[u]=0;
    }
}
void build(int l,int r,int u) {
    all[u]=lazy[u]=maxx[u]=minn[u]=sum[u]=0;
    if(l==r) {
        maxx[u]=minn[u]=sum[u]=a[l];
        maxNum[u]=minNum[u]=1;
        return ;
    }
    int mid=(l+r)>>1;
    build(l,mid,ul);
    build(mid+1,r,ur);
    pushUp(u);
}
int s,t;
void add(int l,int r,int u,ll x) {
    if(s<=l&&t>=r) {
        sum[u]+=x*(r-l+1);
        minn[u]+=x;
        maxx[u]+=x;
        lazy[u]+=x;
        return ;
    }
    pushDown(u,l,r);
    int mid=(l+r)>>1;
    if(t<=mid) add(l,mid,ul,x);
    else if(s>mid) add(mid+1,r,ur,x);
    else{add(l,mid,ul,x);add(mid+1,r,ur,x);}
    pushUp(u);
}
void Sqrt(int l,int r,int u) {
    if(s<=l&&t>=r&&(maxx[u]-minn[u]<=1)) {
        if(maxx[u]==1) return ;
        ll x=maxx[u];
        if(maxx[u]==minn[u]) {
```

```
            minn[u]=maxx[u]=floor(sqrt(x));
            // lazy[u]+=maxx[u]-x;
            lazy[u] = 0, all[u] = maxx[u];
            sum[u]=maxx[u]*(r-l+1);
            return ;
        }
//          else if(maxx[u]-minn[u]==1){
//              maxx[u]=floor(sqrt(maxx[u]));
//              minn[u]=floor(sqrt(minn[u]));
//              if(maxx[u]==minn[u]) {
//                  maxNum[u]=minNum[u]=r-l+1;
//                  lazy[u]=0;
//                  all[u]=maxx[u];
//                  sum[u]=maxx[u]*maxNum[u];
//              }
//              else if(maxx[u]-minn[u]==1) {
//                  lazy[u]+=maxx[u]-x;
//                  sum[u]=maxx[u]*maxNum[u]+minn[u]*minNum[u];
//              }
//              return ;
//          }
        pushDown(u,l,r);
        int mid=(l+r)>>1;
        if(t<=mid) Sqrt(l,mid,ul);
        else if(s>mid)  Sqrt(mid+1,r,ur);
        else{Sqrt(l,mid,ul);Sqrt(mid+1,r,ur);}
        pushUp(u);
        return;
    }
    pushDown(u,l,r);
    int mid=(l+r)>>1;
    if(t<=mid) Sqrt(l,mid,ul);
    else if(s>mid)  Sqrt(mid+1,r,ur);
    else{Sqrt(l,mid,ul);Sqrt(mid+1,r,ur);}
    pushUp(u);
}
ll query(int l,int r,int u) {
    if(s<=l&&t>=r)
        return sum[u];
    pushDown(u,l,r);
    int mid=(l+r)>>1;
    if(t<=mid) return query(l,mid,ul);
    else if(s>mid) return query(mid+1,r,ur);
    else return query(l,mid,ul)+query(mid+1,r,ur);;
}
int main() {
    int T,k;
    T = 1;
    while(T--) {
```

```
        int n,m;
        scanf("%d%d",&n,&m);
        for(int i=1;i<=n;i++)
            scanf("%lld",&a[i]);
        build(1,n,1);
        while(m--) {
            scanf("%d%d%d",&k,&s,&t);
            if(k==2) {
                ll x;scanf("%lld",&x);
                add(1,n,1,x);
            } else if(k==1){
                Sqrt(1,n,1);
            } else {
                printf("%lld\n",query(1,n,1));
            }
        }
    }
    return 0;
}
```

区间min

```
#include <bits/stdc++.h>
using namespace std;
#ifndef ONLINE_JUDGE
#define debug(x...) do { cout << "\033[33;1m " << #x << " -> "; err(x); } while (0)
void err() { cout << "\033[39;0m" << endl; }
template<template<typename...> class T, typename t, typename... A>
void err(T<t> a, A... x) { for (auto v: a) cout << v << ' '; err(x...); }
template<typename T, typename... A>
void err(T a, A... x) { cout << a << ' '; err(x...); }
#else
#define debug(...)
#endif

#define int long long
#define ls rt<<1
#define rs rt<<1|1

const int maxn = 1e6+500;

int n,a[maxn];

struct node{
  int mx, sum, se, cnt;
  int lazy;
  node(){}
};
```

```
node operator+(const node& a, const node& b){
  node tmp;
  tmp.lazy = -1;
  tmp.sum = a.sum + b.sum;
  tmp.mx = max(a.mx, b.mx);
  if(a.mx > b.mx){
    tmp.cnt = a.cnt;
    tmp.se = max(a.se, b.mx);
  }
  if(a.mx == b.mx){
    tmp.cnt = a.cnt+b.cnt;
    tmp.se = max(a.se, b.se);
  }
  if(a.mx < b.mx){
    tmp.cnt = b.cnt;
    tmp.se = max(a.mx, b.se);
  }
  return tmp;
}

struct segmenttree{
  node tr[maxn<<2];
  void pushup(int rt){
    tr[rt] = tr[ls] + tr[rs];
  }
  void pushtag(int rt,int val){
    if(tr[rt].mx > val){  //!
      tr[rt].sum += (val-tr[rt].mx)*tr[rt].cnt;
      tr[rt].mx = tr[rt].lazy = val;  //!
    }
  }
  void pushdown(int rt){
    if(tr[rt].lazy != -1){
      pushtag(ls, tr[rt].lazy);
      pushtag(rs, tr[rt].lazy);
      tr[rt].lazy = -1;
    }
  }
  void build(int l,int r,int rt){
    if(l>r) return ;
    if(l == r){
      tr[rt].mx = tr[rt].sum = a[l];
      tr[rt].se = -1;
      tr[rt].cnt = 1;
      tr[rt].lazy = -1;
      return ;
    }
    int mid = l+r>>1;
```

```
    build(l,mid,ls);
    build(mid+1,r,rs);
    pushup(rt);
  }
  void update(int l,int r,int rt,int L,int R,int val){
    if(tr[rt].mx <= val) return ;
    if(L<=l && r<=R && tr[rt].se < val){  //!
      pushtag(rt,val);
      return ;
    }
    int mid = l+r>>1;
    pushdown(rt);
    if(L<=mid) update(l,mid,ls,L,R,val);
    if(mid<R) update(mid+1,r,rs,L,R,val);
    pushup(rt);
  }
  int querysum(int l,int r,int rt,int L,int R){
    if(L<=l && r<=R)
      return tr[rt].sum;
    int mid = l+r>>1, ans = 0;
    pushdown(rt);
    if(L<=mid) ans += querysum(l,mid,ls,L,R);
    if(mid<R) ans += querysum(mid+1,r,rs,L,R);
    pushup(rt);
    return ans;
  }
  int querymx(int l,int r,int rt,int L,int R){
    if(L<=l && r<=R)
      return tr[rt].mx;
    int mid = l+r>>1, r1 = -1, r2 = -1;
    pushdown(rt);
    if(L<=mid) r1 = querymx(l,mid,ls,L,R);
    if(mid<R) r2 = querymx(mid+1,r,rs,L,R);
    pushup(rt);
    return max(r1,r2);
  }
}tree;

int m;

void solve(){
  cin>>n>>m;
  for(int i=1;i<=n;i++)
    cin>>a[i];
  tree.build(1,n,1);
  while(m--){
    int op,x,y,t;
    cin>>op>>x>>y;
    if(op == 0){
```

```
      cin>>t;
      tree.update(1,n,1,x,y,t);
    }
    if(op == 1){
      cout<<tree.querymx(1,n,1,x,y)<<'\n';
    }
    if(op == 2){
      cout<<tree.querysum(1,n,1,x,y)<<'\n';
    }
  }
}

signed main(){
  ios::sync_with_stdio(false), cin.tie(0), cout.tie(0);
  int t;
  cin>>t;
  while(t--){
    solve();
  }
  return 0;
}
```

权值线段树

pre：先求出[1,x-1]区间内数的个数，再求出排名为个数的数

post：先求出[1,x]区间内数的个数，再求出排名为个数+1的数

```
#include <bits/stdc++.h>
using namespace std;
#define ls rt<<1
#define rs rt<<1|1
const int maxn=1e5+10;

int n,a[maxn],b[maxn],tree[maxn<<2],op[maxn],tot;

int getid(int x){return lower_bound(b+1,b+tot+1,x)-b;}

void pushup(int rt) {tree[rt] = tree[ls] + tree[rs];}

void add(int l,int r,int rt,int k,int pos){
    if(l==r){
        tree[rt]+=k;
        return ;
    }
    int mid = l+r>>1;
    if(pos<=mid) add(l,mid,ls,k,pos);
    else add(mid+1,r,rs,k,pos);
    pushup(rt);
```

```
}

int Rank(int l,int r,int rt,int pos){
    int mid=l+r>>1;
    if(l==r) return 1;
    if(pos<=mid) return Rank(l,mid,ls,pos);
    else return tree[ls]+Rank(mid+1,r,rs,pos);
}

int Kth(int l,int r,int rt,int pos){
    if(l==r) return l;
    int mid = l+r>>1;
    if(pos<=tree[ls]) return Kth(l,mid,ls,pos);
    else return Kth(mid+1,r,rs,pos-tree[ls]);
}

int main(){
    scanf("%d",&n);
    for(int i=1;i<=n;i++){
        scanf("%d%d",&op[i],&a[i]);
        if(op[i]!=4) b[++tot]=a[i];
    }
    sort(b+1,b+tot+1);
    tot = unique(b+1,b+tot+1)-b-1;
    for(int i=1;i<=n;i++){
        if(op[i]==1) add(1,tot,1,1,getid(a[i]));
        else if(op[i]==2) add(1,tot,1,-1,getid(a[i]));
        else if(op[i]==3) printf("%d\n",Rank(1,tot,1,getid(a[i])));
        else if(op[i]==4) printf("%d\n",b[Kth(1,tot,1,a[i])]);
        else if(op[i]==5) printf("%d\n",b[Kth(1,tot,1, Rank(1,tot,1,getid(a[i]))-1 )]);
\\pre
        else printf("%d\n",b[Kth(1,tot,1, Rank(1,tot,1,getid(a[i])+1) )]);  \\post
    }
    return 0;
}
```

线段树合并

```
struct sgt{
    /*
        sum 为最大权值
        tre 为最大权值靠左的位置
    */
    int rot[maxn];
    int tre[maxn<<6], sum[maxn<<6], sl[maxn<<6], sr[maxn<<6], cnt;//需要开大一点，不然RE
    void pushup(int rt){
        if(sl[rt] == 0){
            sum[rt] = sum[sr[rt]];
```

```
            tre[rt] = tre[sr[rt]];
            return ;
        }
        if(sr[rt] == 0){
            sum[rt] = sum[sl[rt]];
            tre[rt] = tre[sl[rt]];
            return ;
        }
        if(sum[sl[rt]]>=sum[sr[rt]]){
            sum[rt] = sum[sl[rt]];
            tre[rt] = tre[sl[rt]];
        }
        else{
            sum[rt] = sum[sr[rt]];
            tre[rt] = tre[sr[rt]];
        }
    }
    void change(int &rt,int l,int r,int pos,int val){
        if(rt == 0) rt = ++cnt;
        if(l == r){
            sum[rt] += val;
            tre[rt] = pos;
            return ;
        }
        int mid = l+r>>1;
        if(pos<=mid) change(sl[rt],l,mid,pos,val);
        else change(sr[rt],mid+1,r,pos,val);
        pushup(rt);
    }
    int merge(int a,int b,int l,int r){
        if(a==0||b==0)
            return a+b;
        if(l==r){
            sum[a]+=sum[b];
            return a;
        }
        int mid = l+r>>1;
        sl[a] = merge(sl[a],sl[b],l,mid);
        sr[a] = merge(sr[a],sr[b],mid+1,r);
        pushup(a);
        return a;
    }
}s;
```

主席树

可持久化权值线段树

```
#include <bits/stdc++.h>
using namespace std;

const int maxn = 2e5+10;
//主席树
struct node{int l,r,sum;}tree[maxn<<5];
int tot,rt[maxn];
//离散化
int a[maxn],b[maxn],len,n,m;

int getid(int x){
    return lower_bound(b+1,b+len+1,x)-b;
}

int build(int l,int r){
    int root = ++tot;
    if(l==r) return root;
    int mid = l+r>>1;
    tree[root].l = build(l,mid);//构建左子树
    tree[root].r = build(mid+1,r);//构建右子树
    return root;
}

int update(int u,int l,int r,int k){
    int pos = ++tot;
    tree[pos].l=tree[u].l,tree[pos].r=tree[u].r,tree[pos].sum=tree[u].sum+1;
    if(l==r) return pos;
    int mid = l+r>>1;
    if(k<=mid) tree[pos].l = update(tree[u].l,l,mid,k);//更新左子树
    else tree[pos].r = update(tree[u].r,mid+1,r,k);//更新右子树
    return pos;
}

int query(int u,int v,int l,int r,int k){
    int mid = l+r>>1,x = tree[tree[v].l].sum-tree[tree[u].l].sum;
    if(l==r) return l;
    if(k<=x) return query(tree[u].l,tree[v].l,l,mid,k);//查询左子树
    else return query(tree[u].r,tree[v].r,mid+1,r,k-x);//查询右子树
}

//int k = upper_bound(b+1,b+sz+1,h)-b-1;  不需要提前离散，直接查询
int query(int u,int v,int l,int r,int k){ //求[l,r]小于等于k的数目
    int mid = l+r>>1,x = tree[tree[v].ls].sum-tree[tree[u].ls].sum;
    if(l==r) return tree[v].sum-tree[u].sum;
    if(k<=mid) return query(tree[u].ls,tree[v].ls,l,mid,k);
    else return x+query(tree[u].rs,tree[v].rs,mid+1,r,k);
}
```

```
//int k = lower_bound(dic+1,dic+1+len,lo)-dic;
inline int query(int lrot,int rrot,int l,int r,int pos)
{   //[l,r]区间内大于等于k的数量
    if(l==r) return p[rrot].sum-p[lrot].sum;
    int mid=l+r>>1;
    int sum=0;
    if(pos<=mid) sum=query(p[lrot].l,p[rrot].l,l,mid,pos)+p[p[rrot].r].sum-
p[p[lrot].r].sum;
    else sum=query(p[lrot].r,p[rrot].r,mid+1,r,pos);
    return sum;
}

int main(){
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;i++) scanf("%d",&a[i]),b[i]=a[i];
    sort(b+1,b+n+1);
    len = unique(b+1,b+n+1)-b-1;
    rt[0] = build(1,len);
    for(int i=1;i<=n;i++) rt[i] = update(rt[i-1],1,len,getid(a[i]));
    while(m--){
        int l,r,k;
        scanf("%d%d%d",&l,&r,&k);
        printf("%d\n",b[query(rt[l-1],rt[r],1,len,k)]);
    }
    return 0;
}
```

树上路径第K大

每个点建立一颗权值线段树，**维护根节点到当前点的路径的权值**。两点之前的路径权值，就是 $sum[u]+sum[v]-2\times sum[lca(u,v)]$

```
void dfs(int now,int par){  //rt[root] = ++tot
    for(auto to:g[now]){
        if(to.first == par) continue;
        dfn[to.first] = ++tim;
        rt[dfn[to.first]] = update(rt[dfn[now]],1,maxm,to.second);
        dfs(to.first,now);
    }
}

int query(int x,int y,int f,int l,int r,int k){
    int mid = l+r>>1;
    int sum = tr[tr[x].ls].val+tr[tr[y].ls].val-2*tr[tr[f].ls].val;
    if(l == r)
        return l;
    if(sum < k) return query(tr[x].rs,tr[y].rs,tr[f].rs,mid+1,r,k-sum);
    else return query(tr[x].ls,tr[y].ls,tr[f].ls,l,mid,k);
```

```
}
```

可持久化区间和线段树

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long

const ll maxn = 1e5+100;

ll n,m,tot,rt[maxn];
struct node{ll ls,rs,sum,lazy;}tree[maxn<<5];

ll build(ll l,ll r){
    ll now = tot++;
    tree[now].lazy=0;
    if(l==r){
        scanf("%lld",&tree[now].sum);
        return now;
    }
    ll mid = l+r>>1;
    tree[now].ls = build(l,mid);
    tree[now].rs = build(mid+1,r);
    tree[now].sum = tree[tree[now].ls].sum + tree[tree[now].rs].sum;
    return now;
}

inline void pushup(ll rt,ll len){
    tree[rt].sum = tree[tree[rt].ls].sum + tree[tree[rt].rs].sum
        + tree[tree[rt].ls].lazy*(len-len/2) + tree[tree[rt].rs].lazy*(len/2);
}

ll update(ll rt,ll l,ll r,ll L,ll R,ll add){
    ll now = tot++;
    tree[now].lazy = tree[rt].lazy;
    if(L<=l&&r<=R){
        tree[now].lazy = tree[rt].lazy + add;
        tree[now].sum = tree[rt].sum;
        tree[now].ls = tree[rt].ls;
        tree[now].rs = tree[rt].rs;
        return now;
    }
    ll mid = l+r>>1;
    if(L<=mid) tree[now].ls = update(tree[rt].ls,l,mid,L,R,add);
    else tree[now].ls = tree[rt].ls;
    if(mid<R) tree[now].rs = update(tree[rt].rs,mid+1,r,L,R,add);
    else tree[now].rs = tree[rt].rs;
```

```
    pushup(now,r-l+1);
    return now;
}

ll query(ll rt,ll l,ll r,ll L,ll R,ll add){
    if(L<=l&&r<=R){
        return tree[rt].sum + (tree[rt].lazy+add)*(r-l+1); //?
    }
    ll mid = l+r>>1,ans=0;
    if(L<=mid) ans+= query(tree[rt].ls,l,mid,L,R,add+tree[rt].lazy);
    if(mid<R) ans+= query(tree[rt].rs,mid+1,r,L,R,add+tree[rt].lazy);
    return ans;
}

int main(){
    int flag=0;
    while(scanf("%lld%lld",&n,&m)!=EOF){
        if(flag) printf("\n");
        flag++;
        int now = 0;
        tot=0;
        rt[0] = build(1,n);
        while(m--){
            char op[10];
            ll l,r,k;
            scanf("%s",op);
            if(op[0]=='Q'){
                scanf("%lld%lld",&l,&r);
                printf("%lld\n",query(rt[now],1,n,l,r,0));
            }else if(op[0]=='H'){
                scanf("%lld%lld%lld",&l,&r,&k);
                printf("%lld\n",query(rt[k],1,n,l,r,0));
            }else if(op[0]=='C'){
                scanf("%lld%lld%lld",&l,&r,&k);
                now++;
                rt[now] = update(rt[now-1],1,n,l,r,k);
            }else{
                scanf("%lld",&now); //回到过去
                tot = rt[now+1];
            }
        }
    }
    return 0;
}
```

可持久化trie

在当前节点更新子节点。当前bit 0 和 1 的个数保存在子节点中。rt[0]需要初始化0

```cpp
#include <bits/stdc++.h>
using namespace std;

const int maxn = 4e6+500;
int s[maxn],n,rt[maxn],cnt[maxn*40],ch[maxn*40][2],tot,m;

void insert(int u,int v,int bit,int val){
    if(bit<0) return ;
    int x = (val>>bit)&1;
    ch[v][!x] = ch[u][!x];
    ch[v][x] = ++tot;
    cnt[ch[v][x]] = cnt[ch[u][x]]+1;
    insert(ch[u][x],ch[v][x],bit-1,val);
}

int query(int l,int r,int bit,int val){
    if(bit<0) return 0;
    int now = (val>>bit)&1;
    if(cnt[ch[r][!now]]-cnt[ch[l][!now]]>0)
        return (1<<bit) + query(ch[l][!now],ch[r][!now],bit-1,val);
    else
        return query(ch[l][now],ch[r][now],bit-1,val);
}

int main(){
    ios::sync_with_stdio(false),cin.tie(0),cout.tie(0);
    cin>>n>>m;
    int last = 0;
    rt[0] = ++tot;
    insert(0,rt[0],25,0);
    for(int i=1;i<=n;i++){
        int tmp;
        cin>>tmp;
        last = tmp^last;
        rt[i] = ++tot;
        insert(rt[i-1],rt[i],25,last);
    }
    while(m--){
        char op;
        int l,r,x;
        cin>>op;
        if(op == 'A'){
            n++;
            rt[n] = ++tot;
            cin>>x;
            last = last^x;
            insert(rt[n-1],rt[n],25,last);
        }
        else{
```

```
            cin>>l>>r>>x;
            l--, r--;
            int val = last^x;
            if(l == 0)
                cout<<query(0,rt[r],25,val)<<'\n';
            else
                cout<<query(rt[l-1],rt[r],25,val)<<'\n';
        }
    }
}
```

扫描线

面积并

```
#include <iostream>
#include <algorithm>
using namespace std;
#define ls rt<<1
#define rs rt<<1|1

const int maxn = 1e5+500;

struct seg{
    double l,r,h;
    int f;
    seg(){}
    seg(double l,double r,double h,int f):l(l),r(r),h(h),f(f){}
    bool operator<(const seg& a){return h<a.h;}
}edge[maxn];
struct node{double len;int tag;}tree[maxn<<2];
int n,tot;
double x[maxn];

inline void pushdown(int l,int r,int rt){
    if(tree[rt].tag) tree[rt].len = x[r+1]-x[l];
    else if(l==r) tree[rt].len = 0;
    else tree[rt].len = tree[ls].len + tree[rs].len;
}

void update(int l,int r,int rt,int L,int R,int k){
    if(L<=l&&r<=R){
        tree[rt].tag += k;
        pushdown(l,r,rt);
        return ;
    }
    int mid = l+r>>1;
    if(L<=mid) update(l,mid,ls,L,R,k);
```

```
    if(mid<R) update(mid+1,r,rs,L,R,k);
    pushdown(l,r,rt);
}

int main(){
    int _=1;
    while(scanf("%d",&n)!=EOF&&n){
        tot=0;
        for(int i=0;i<maxn<<2;i++)
            tree[i].len = tree[i].tag = 0;  //初始化
        for(int i=1;i<=n;i++){
            double x1,x2,y1,y2;
            scanf("%lf%lf%lf%lf",&x1,&y1,&x2,&y2);
            x[++tot] = x1 , edge[tot] = seg(x1,x2,y1,1);
            x[++tot] = x2 , edge[tot] = seg(x1,x2,y2,-1);
        }
        sort(x+1,x+tot+1);
        sort(edge+1,edge+tot+1);       //对扫描线排序
        int cnt = unique(x+1,x+tot+1)-x-1;  //离散化
        double res = 0;
        for(int i=1;i<=tot;i++){
            int l = lower_bound(x+1,x+cnt+1,edge[i].l)-x;
            int r = lower_bound(x+1,x+cnt+1,edge[i].r)-x-1;
            update(1,cnt,1,l,r,edge[i].f);
            res += tree[1].len*(edge[i+1].h-edge[i].h);//更新答案
        }
        printf("Test case #%d\nTotal explored area: %.2lf \n\n",_++,res);
    }
    return 0;
}
```

周长并

纵边总长度= $\sum$(2×被截得的线段条数×扫过的高度)

横边总长度= $\sum$|上次截得的总长-现在截得的总长|

```
#include <iostream>
#include <cstdio>
#include <algorithm>
#define lson (x << 1)
#define rson (x << 1 | 1)
using namespace std;
const int MAXN = 2e4;
int n, X[MAXN << 1];

struct ScanLine {
  int l, r, h, mark;
  bool operator<(const ScanLine& a){
```

```
        if(h == a.h)
            return mark > a.mark;
        return h < a.h;
    }
} 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;
}

int main() {
  scanf("%d", &n);
    int x1,y1,x2,y2,pre=0;
  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;
  printf("%d", res);
  return 0;
}
```

树链剖分

套线段树（维护子树与树上两点路径）

```
#include <bits/stdc++.h>
using namespace std;
#define ls rt<<1
#define rs rt<<1|1

const int maxn = 1e5+100;

int to[maxn*2],nex[maxn*2],tot,beg[maxn];   //链式前向星
int n,m,r,mod;
int dep[maxn],top[maxn],son[maxn],siz[maxn],fa[maxn],w[maxn],wt[maxn],id[maxn],cnt;
int sum[maxn<<2],lazy[maxn<<2];

inline void add(int x,int y){
    to[++tot] = y;
    nex[tot] = beg[x];
    beg[x] = tot;
}

void dfs1(int now,int f,int deep){
    dep[now] = deep;
    fa[now] = f;
    siz[now] = 1;
    son[now] = 0;
    int maxsize = -1;
    for(int i=beg[now];i;i=nex[i]){
        int nx = to[i];
        if(nx==f) continue;
        dfs1(nx,now,deep+1);
        siz[now]+=siz[nx];
        if(siz[nx]>maxsize)
            maxsize = siz[nx],son[now]=nx;
    }
}

void dfs2(int now,int topf){
    top[now] = topf;
    id[now] = ++cnt;
    wt[cnt] = w[now];
    if(!son[now]) return ;
    dfs2(son[now],topf);
    for(int i=beg[now];i;i=nex[i]){
        int nx = to[i];
        if(nx==fa[now]||nx==son[now]) continue;
        dfs2(nx,nx);
    }
}
```

```
inline void pushup(int rt){
    sum[rt] = (sum[ls]+sum[rs])%mod;
}

inline void pushdown(int rt,int len){
    if(lazy[rt]){
        lazy[ls] = (lazy[ls]+lazy[rt])%mod;
        lazy[rs] = (lazy[rs]+lazy[rt])%mod;
        sum[ls] = (sum[ls]+lazy[rt]*(len-len/2))%mod;
        sum[rs] = (sum[rs]+lazy[rt]*(len/2))%mod;
        lazy[rt] = 0;
    }
}

void build(int l,int r,int rt){
    if(l==r){
        sum[rt] = wt[l]%mod;
        return ;
    }
    int mid = l+r>>1;
    build(l,mid,ls);
    build(mid+1,r,rs);
    pushup(rt);
}

void update(int l,int r,int rt,int L,int R,int k){
    if(L<=l&&r<=R){
        sum[rt] = (sum[rt]+k*(r-l+1))%mod;
        lazy[rt] = (lazy[rt]+k)%mod;
        return ;
    }
    int mid = l+r>>1;
    pushdown(rt,r-l+1);
    if(L<=mid) update(l,mid,ls,L,R,k);
    if(mid<R) update(mid+1,r,rs,L,R,k);
    pushup(rt);
}

int query(int l,int r,int rt,int L,int R){
    if(L<=l&&r<=R) return sum[rt];
    int mid = l+r>>1 , ans=0;
    pushdown(rt,r-l+1);
    if(L<=mid) ans = (ans+query(l,mid,ls,L,R))%mod;
    if(mid<R) ans = (ans+query(mid+1,r,rs,L,R))%mod;
    pushup(rt);
    return ans;
}

void updaterange(int x,int y,int z){
```

```
    while(top[x]!=top[y]){
        if(dep[top[x]]<dep[top[y]]) swap(x,y);
        update(1,n,1,id[top[x]],id[x],z);
        x = fa[top[x]];
    }
    if(dep[x]>dep[y]) swap(x,y);
    update(1,n,1,id[x],id[y],z);
}

int queryrange(int x,int y){
    int res=0;
    while(top[x]!=top[y]){
        if(dep[top[x]]<dep[top[y]]) swap(x,y);
        res= (res+query(1,n,1,id[top[x]],id[x]))%mod;
        x = fa[top[x]];
    }
    if(dep[x]>dep[y]) swap(x,y);
    res=(res+query(1,n,1,id[x],id[y]))%mod;
    return res;
}

void updatetree(int x,int y){   //链是连续的，所以可以这样更新
    update(1,n,1,id[x],id[x]+siz[x]-1,y);
}

int querytree(int x){
    return query(1,n,1,id[x],id[x]+siz[x]-1);
}

int main(){
    scanf("%d%d%d%d",&n,&m,&r,&mod);
    for(int i=1;i<=n;i++) scanf("%d",&w[i]);
    for(int i=1,u,v;i<=n-1;i++){
        scanf("%d%d",&u,&v);
        add(u,v) , add(v,u);
    }
    dfs1(r,0,1);     //更新 深度，重儿子，父亲
    dfs2(r,r);       //将 链映射到区间上 和 链的顶点
    build(1,n,1);    //线段树 维护 树链
    while(m--){
        int op,x,y,z;
        scanf("%d%d",&op,&x);
        if(op==1){
            scanf("%d%d",&y,&z);z%=mod;
            updaterange(x,y,z);
        }else if(op==2){
            scanf("%d",&y);
            printf("%d\n",queryrange(x,y));
        }else if(op==3){
```

```
            scanf("%d",&y);y%=mod;
            updatetree(x,y);
        }else{
            printf("%d\n",querytree(x));
        }
    }
    return 0;
}
```

LCA

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 5e5+100;

int tot,beg[maxn];
struct node{int to,nex;}edge[maxn*2];
int n,m,r;

int dep[maxn],fa[maxn],siz[maxn],son[maxn],top[maxn];

inline void add(int x,int y){
    tot++;
    edge[tot].to = y;
    edge[tot].nex = beg[x];
    beg[x] = tot;
}

//dep, fa, siz, son
void dfs1(int now,int f,int deep){
    dep[now] = deep;
    fa[now] = f;
    siz[now] = 1;
    son[now] = 0;
    int maxsize = -1;
    for(int i=beg[now];i;i=edge[i].nex){
        int to = edge[i].to;
        if(to==f) continue;
        dfs1(to,now,deep+1);
        siz[now]+=siz[to];
        if(maxsize<siz[to])
            maxsize = siz[to] , son[now] = to;
    }
}

void dfs2(int now,int topf){
    top[now] = topf;
    if(!son[now]) return ;  //!
```

```
    dfs2(son[now],topf);
    for(int i = beg[now];i;i = edge[i].nex){
        int to = edge[i].to;
        if(to==fa[now]||to==son[now]) continue;
        dfs2(to,to);
    }
}

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]];
    }
    if(dep[x]<dep[y]) swap(x,y);
    return y;
}

int main(){
    scanf("%d%d%d",&n,&m,&r);
    for(int i=1,u,v;i<=n-1;i++){
        scanf("%d%d",&u,&v);
        add(u,v),add(v,u);
    }
    dfs1(r,0,1);
    dfs2(r,r);  //!
    while(m--){
        int a,b;
        scanf("%d%d",&a,&b);
        printf("%d\n",lca(a,b));
    }
    return 0;
}
```

ST表

```
int mi[maxn][30];

for(int j=1;(1<<j)<=n;++j)
    for(int i=1;i+(1<<(j-1))<=n;++i)
        mi[i][j]=min(mi[i][j-1],mi[i+(1<<(j-1))][j-1]);


int rmqmi(int l,int r) {
    int k=__lg(r-l+1);
    return min(mi[l][k],mi[r-(1<<k)+1][k]);
}
```

ST表求树上两点距离

```
#include <bits/stdc++.h>
using namespace std;

const int maxn = 4e4+500;

int n,m,tot,beg[maxn],dep[maxn],fa[maxn],dis[maxn];
struct node{int to,val,nex;}edge[maxn*2];
int st[maxn][21];

inline void add(int u,int v,int val){
    edge[++tot].to = v;
    edge[tot].val = val;
    edge[tot].nex = beg[u];
    beg[u] = tot;
}

void dfs(int now,int f,int deep){
    fa[now] = f , dep[now] = deep;
    for(int i=beg[now];i;i = edge[i].nex){
        int nx = edge[i].to , val = edge[i].val;
        if(nx==f) continue;
        dis[nx] = dis[now]+val;
        dfs(nx,now,deep+1);
    }
}

void init(){
    for(int j=0;(1<<j)<=n;j++)
        for(int i=1;i<=n;i++)
            st[i][j] = j==0?fa[i]:-1;
    for(int j=1;(1<<j)<=n;j++)
        for(int i=1;i<=n;i++)
```

```
            if(st[i][j-1]!=-1)
                st[i][j] = st[st[i][j-1]][j-1];
}

int lca(int a,int b){
    if(dep[a]<dep[b]) swap(a,b);
    int i,j;
    for(i=0;(1<<i)<=dep[a];i++) ;
    i--;
    for(j=i;j>=0;j--)
        if(dep[a]-(1<<j)>=dep[b])    //不可能跳到0
            a = st[a][j];
    if(a==b) return a;
    for(j=i;j>=0;j--)
        if(st[a][j]!=-1&&st[a][j]!=st[b][j])
            a = st[a][j], b = st[b][j];
    return fa[a];
}

int get_fa(int x,int k) //k级祖先，不存在的时候为0
{
    int t=dep[x]-k;
    for(int i=20;i>=0;i--)//倍增向上跳跃，求祖先
        if(dep[st[x][i]]>t)x=st[x][i];
    return st[x][0];
}

int main(){
    int _;
    scanf("%d",&_);
    while(_--){
        tot=0;
        scanf("%d%d",&n,&m);
        for(int i=0;i<maxn;i++) beg[i]=0;
        for(int i=1,u,v,val;i<=n-1;i++){
            scanf("%d%d%d",&u,&v,&val);
            add(u,v,val) , add(v,u,val);
        }
        dis[1] = 0;
        dfs(1,0,1);
        init();
        while(m--){
            int a,b;
            scanf("%d%d",&a,&b);
            printf("%d\n",dis[a]+dis[b]-2*dis[lca(a,b)]);
        }
    }
    return 0;
}
```

点分治

1. 找到当前树的重心 getrt
2. 以当前根节点更新答案（同时需要去除非法答案）cal
3. 找到子树的重心继续分治 divide

```
#include <bits/stdc++.h>
using namespace std;

const int maxn = 1e4+500;

struct node{int to,nex,val;}edge[maxn*2];
int n,beg[maxn],tot,m;
int vis[maxn], siz[maxn], maxsiz[maxn], root, Size;
int ans, dis[maxn], sz;

inline void add(int u,int v,int val){
    edge[++tot].to = v;
    edge[tot].nex = beg[u];
    edge[tot].val = val;
    beg[u] = tot;
}

void getrt(int now,int par){
    siz[now] = 1, maxsiz[now] = 0;
    for(int i=beg[now];i;i=edge[i].nex){
        int to = edge[i].to;
        if(vis[to]||to==par) continue;
        getrt(to,now);
        siz[now] += siz[to];
        maxsiz[now] = max(maxsiz[now], siz[to]);
    }
    maxsiz[now] = max(maxsiz[now], Size-siz[now]);
    if(maxsiz[now]<maxsiz[root]) root = now;
}

void getdis(int rt,int par,int len){
    dis[sz++] = len;
    for(int i=beg[rt];i;i = edge[i].nex){
        int to = edge[i].to;
        if(vis[to]||to==par) continue;
        getdis(to,rt,len+edge[i].val);
    }
}

int cal(int rt,int len){
    sz = 0;
    getdis(rt,0,len);
    sort(dis,dis+sz);
```

```
    int l = 0, r = sz-1, res = 0;
    while(l<=r) {
        if(dis[l]+dis[r]<=m) {res += r-l++;}
        else r--;
    }
    return res;
}

void divide(int rt){
    ans += cal(rt,0);
    vis[rt] = true;
    for(int i=beg[rt];i;i = edge[i].nex){
        int to = edge[i].to;
        if(vis[to]) continue;
        ans -= cal(to, edge[i].val);
        root = 0, maxsiz[0] = Size = siz[to];
        getrt(to,root);
        divide(root);
    }
}

int main(){
    while(scanf("%d%d",&n,&m)!=EOF&&n&&m){
        for(int i=0;i<=n;i++) vis[i] = false;
        tot = 0;
        for(int i=0;i<maxn;i++) beg[i] = 0;
        for(int i=1,u,v,val;i<n;i++){
            scanf("%d%d%d",&u,&v,&val);
            u++,v++;
            add(u,v,val), add(v,u,val);
        }
        maxsiz[0] = Size = n, root = 0;
        getrt(1,0);
        ans = 0;
        divide(root);
        printf("%d\n",ans);
    }
    return 0;
}
```

最小瓶颈路

```
int prime(){
    int res = 0;
    memset(maxcost,0,sizeof(maxcost));
    for(int i=1;i<=n;i++){
        vis[i]=0 , d[i] = INF , pre[i] = i;
    }
```

```
    d[s] = 0;                               //预先选中s
    for(int i=0;i<n;i++){
        int mx = INF , index = -1;
        for(int j=1;j<=n;j++){
            if(!vis[j]&&d[j]<mx)
                mx = d[index=j];        //选中d最小的点
        }
        if(index==-1) break;
        for(int j=1;j<=n;j++)     //j->index = j->fa->index
            if(vis[j])
                maxcost[index][j] = maxcost[j][index] =
                    max(maxcost[pre[index]][j],mx);
        res += mx;
        vis[index] = 1;
        for(int j=1;j<=n;j++){          //用选中的index更新其他节点
            if(!vis[j]&&g[index][j]<d[j]){
                d[j] = g[index][j];
                pre[j] = index;
            }
        }
    }
    return res;
}
```

并查集

```
struct DSU {
    std::vector<int> f, siz;
    DSU(int n) : f(n), siz(n, 1) { std::iota(f.begin(), f.end(), 0); }
    int leader(int x) {
        while (x != f[x]) x = f[x] = f[f[x]];
        return x;
    }
    bool same(int x, int y) { return leader(x) == leader(y); }
    bool merge(int x, int y) {
        x = leader(x);
        y = leader(y);
        if (x == y) return false;
        siz[x] += siz[y];
        f[y] = x;
        return true;
    }
    int size(int x) { return siz[leader(x)]; }
};
```

科技

查询区间区间两数之差绝对值最小值

```
const int N=4e5+7;
int mi,a[N],v[N];vector<int>num[N],g[maxn];
#define ls rt<<1
#define rs rt<<1|1
void build(int rt,int l,int r){
    num[rt].clear();
    v[rt]=inf;
    for(int i=l;i<=r;++i) num[rt].push_back(a[i]);
    if(l==r) return;
    sort(num[rt].begin(),num[rt].end());
    int mid=l+r>>1;
    build(ls,l,mid),build(rs,mid+1,r);
}
void upd(int rt,int l,int r,int pos,int x){
    if(l>pos) return;
    if(r<=pos){
        auto it=upper_bound(num[rt].begin(),num[rt].end(),x);
        if(it!=num[rt].end()) v[rt]=min(v[rt],*it-x);
        if(it!=num[rt].begin()) --it,v[rt]=min(v[rt],x-*it);
        if(mi<=v[rt]) return;
        if(l>=r){mi=min(mi,v[rt]);return;}
    }
    int mid=(l+r)>>1;
    upd(rs,mid+1,r,pos,x),upd(ls,l,mid,pos,x);
    v[rt]=min({v[rt],v[ls],v[rs]});
    mi=min(mi,v[rt]);
}
int qry(int rt,int l,int r,int ql,int qr){
    if(r<ql || l>qr) return inf;
    if(ql<=l && r<=qr) return v[rt];
    int mid=(l+r)>>1;
    return min(qry(ls,l,mid,ql,qr),qry(rs,mid+1,r,ql,qr));
}
int main(){
  scanf("%d",&n);
  for(int i=1;i<=n;++i) scanf("%d",&a[i]);
  build(1,1,n);
  scanf("%d",&m);
  for(int i=1,l,r;i<=m;++i){
    scanf("%d%d",&l,&r);
    le[i]=l; g[r].push_back(i);
  }
  for(int i=2;i<=n;++i){
    mi=inf;
    upd(1,1,n,i-1,a[i]);
    for(auto &j:g[i]) ans[j]=qry(1,1,n,le[j],i);
  }
  for(int i=1;i<=m;++i)printf("%d\n",ans[i]);
  return 0;
```

```
}
```

平衡树

区间翻转

```
#define ls (a[u].l)
#define rs (a[u].r)
const int maxn=200010;
int n,m,x,y,z,l,r,tot,root;
struct treap{int l,r,v,rnd,size,rot;}a[maxn];
void newnode(int val){a[++tot]=(treap){0,0,val,rand(),1,0};}
void update(int u){a[u].size=a[ls].size+a[rs].size+1;}
void rotate(int u){if(a[u].rot) a[u].rot^=1,swap(ls,rs),a[ls].rot^=1,a[rs].rot^=1;}
void split(int u,int k,int &x,int &y){
    rotate(u);
    if(!k){x=0; y=u; return;}
    if(a[u].size==k){x=u; y=0; return;}
    if(a[ls].size>=k) split(ls,k,x,ls),y=u;
    else split(rs,k-a[ls].size-1,rs,y),x=u;
    update(u);
}
int merge(int x,int y){
    if(!x||!y) return x+y;
    rotate(x); rotate(y);
    if(a[x].rnd<a[y].rnd){
        a[x].r=merge(a[x].r,y); update(x); return x;
    }
    else{
        a[y].l=merge(x,a[y].l); update(y); return y;
    }
}
void put(int u){
    rotate(u);
    if(ls) put(ls);
    printf("%d ",a[u].v);
    if(rs) put(rs);
}
int main(){
    n=read(); m=read();
    for(int i=1;i<=n;i++) newnode(i),root=merge(root,tot);
    while(m--){
        l=read()-1; r=read();
        split(root,l,x,y); split(y,r-l,y,z);
        a[y].rot^=1;
        root=merge(merge(x,y),z);
    }
    put(root);
```

```
    return 0;
}
```

```
struct Splay
{

    struct Node
  {
        int father, childs[2], key, cnt, sz;
        void init() {father = childs[0] = childs[1] = key = cnt = sz = 0;}
        void init(int fa, int lc, int rc, int k, int c, int s)
    {
            father = fa;
      childs[0] = lc;
        childs[1] = rc;
            key = k;
        cnt = c;
      sz = s;
        }
    } tre[maxn];

    int tot, root;
    void init() {tot = root = 0;}

    inline bool judge(int x) {return tre[ tre[x].father ].childs[1] == x;}

    inline void update(int x)
  {
        if(x)
    {
            tre[x].sz = tre[x].cnt;
            if(tre[x].childs[0])
                tre[x].sz += tre[ tre[x].childs[0] ].sz;
            if(tre[x].childs[1])
                tre[x].sz += tre[ tre[x].childs[1] ].sz;
        }
    }

    inline void rotate(int x)
  {
        int y = tre[x].father, z = tre[y].father, k = judge(x);
        tre[y].childs[k] = tre[x].childs[!k];
      tre[ tre[x].childs[!k] ].father = y;
        tre[x].childs[!k] = y;
      tre[y].father = x;
        tre[z].childs[ tre[z].childs[1] == y ] = x;
      tre[x].father = z;
```

```
        update(y);
    }

    void splay(int x,int goal)
{
        for(int father; (father = tre[x].father) != goal; rotate(x) )
            if(tre[father].father != goal)
                rotate(judge(x) == judge(father) ? father : x);
    if(goal == 0)
      root = x;
    }

    void insert(int x)
{
        if(root == 0)
    {
            tre[++tot].init(0, 0, 0, x, 1, 1);
            root = tot;
            return ;
        }
        int now = root, father = 0;
        while(1)
    {
            if(tre[now].key == x)
      {
                tre[now].cnt ++;
                update(now), update(father);
                splay(now, 0);
                break;
            }
            father = now;
            if(x > tre[now].key)
                now = tre[now].childs[1];
            else
                now = tre[now].childs[0];

            if(now == 0)
      {
                tre[++tot].init(father, 0, 0, x, 1, 1);
                if(x > tre[father].key)
                    tre[father].childs[1] = tot;
                else
                    tre[father].childs[0] = tot;
                update(father);
                splay(tot, 0);
                break;
            }
        }
    }
```

```
  int pre()
{
      int now = tre[root].childs[0];
      while(tre[now].childs[1])
          now = tre[now].childs[1];
      return now;
  }

  int next()
{
      int now = tre[root].childs[1];
      while(tre[now].childs[0])
          now = tre[now].childs[0];
      return now;
  }

int rnk(int x)
{ /// 找x的排名
      int now = root, ans = 0;
      while(1)
  {
          if(x < tre[now].key)
              now = tre[now].childs[0];
          else
    {
              if(tre[now].childs[0])
                  ans += tre[ tre[now].childs[0] ].sz;
              if(x == tre[now].key)
      {
                  splay(now, 0);
                  return ans + 1;
              }
              ans += tre[now].cnt;
              now = tre[now].childs[1];
          }
      }
  }

  int kth(int x)
{ /// 找排名为x的数字
      int now = root;
      while(1)
  {
          if(tre[now].childs[0] && x <= tre[ tre[now].childs[0] ].sz )
              now = tre[now].childs[0];
          else
    {
              int lchild = tre[now].childs[0], sum = tre[now].cnt;
```

```
            if(lchild)
                sum += tre[lchild].sz;
            if(x <= sum)
                return tre[now].key;
            x -= sum;
            now = tre[now].childs[1];
        }
    }
}

void del(int x)
{
    find(x);
    if(tre[root].cnt > 1)
    {
        tre[root].cnt --;
        update(root);
        return ;
    }
    if(!tre[root].childs[0] && !tre[root].childs[1])
    {
        tre[root].init();
        root = 0;
        return ;
    }
    if(!tre[root].childs[0])
    {
        int old_root = root;
        root = tre[root].childs[1];
    tre[root].father = 0;
    tre[old_root].init();
        return ;
    }
    if(!tre[root].childs[1])
    {
        int old_root = root;
        root = tre[root].childs[0];
    tre[root].father = 0;
    tre[old_root].init();
        return ;
    }
    int pre_node = pre(), old_root = root;
    splay(pre_node, 0);
    tre[root].childs[1] = tre[old_root].childs[1];
    tre[ tre[old_root].childs[1] ].father = root;
    tre[old_root].init();
    update(root);
}
```

```
    bool find(int x)
  {
        int now = root;
        while(1)
    {
            if(now == 0)
                return 0;
            if(x == tre[now].key)
      {
                splay(now, 0);
                return 1;
            }
            if(x > tre[now].key)
                now = tre[now].childs[1];
            else
                now = tre[now].childs[0];
        }
    }
} S;

mt19937 rnd(13331);
///切割区间
///处理从x往左 找区间大于等于某数的最近位置
///然后维护区间的值修改等操作
struct fhq_treap
{
    #define ls(x) ch[x][0]
    #define rs(x) ch[x][1]
    int sz[N],mi[N],val[N],ch[N][2],key[N];///key随机权值
    ll sum[N];
    int cnt,root;
    void init()
    { //要初始化
        for(int i=0;i<=n+1;i++) ls(i)=rs(i)=0;
        cnt=0,mi[0]=1e9+7;
    }
    inline void up(int o)
    {
        sz[o]=sz[ls(o)]+sz[rs(o)]+1;
        sum[o]=sum[ls(o)]+sum[rs(o)]+val[o];
        mi[o]=min(val[o],min(mi[ls(o)],mi[rs(o)]));
    }
    int built(int *a,int l,int r)
    { //直接构建
        if(l>r) return 0;
        int now=++cnt;
        int mid=(l+r)/2;
        sum[now]=mi[now]=val[now]=a[mid]; ///每个结点维护区间中点
        sz[now]=(r-l+1),key[now]=rnd();
```

```
    ls(now)=built(a,l,mid-1);
    rs(now)=built(a,mid+1,r);
    up(now);
    return now;
}
inline void spilt(int o,int siz,int &x,int &y)
{ ///按大小分
    if(o==0) {x=y=0; return;}
    if(sz[ls(o)]<siz) x=o,spilt(rs(o),siz-sz[ls(o)]-1,rs(o),y);
    else y=o,spilt(ls(o),siz,x,ls(o));
    up(o);
}
inline void spilt_r_min(int o,int v,int &x,int &y)
{ ///重要操作，将大于等于v的分到y树上，其余分到x树上
    if(o==0) {x=y=0;return ;}
    if(mi[rs(o)]<v || val[o]<v) x=o,spilt_r_min(rs(o),v,rs(o),y);
    else y=o,spilt_r_min(ls(o),v,x,ls(o));
    up(o);
}
inline int merge(int x,int y)
{
    if(x==0 || y==0) return x+y;
    if(key[x]>key[y])
    {
        rs(x)=merge(rs(x),y);
        up(x);
        return x;
    }
    else
    {
        ls(y)=merge(x,ls(y));
        up(y);
        return y;
    }
    return -1;
}
int getnum(int pos)
{
    int now=root;
    while(sz[ls(now)]+1!=pos)
    {
        if(pos>sz[ls(now)]+1) pos-=sz[ls(now)]+1,now=rs(now);
        else now=ls(now);
    }
    return val[now];
}
ll getsum(int x,int y)
{
    if(getnum(x)<y) return 0;
```

```
        int t1,t2,t3,t4,t5;
        spilt(root,x,t1,t2); ///t2代表x右边的
        if(mi[t1]>=y)
        {
            root=merge(t1,t2);
            return 0;
        }
        spilt_r_min(t1,y,t1,t3);  ///大于等于y的一段 t3
        ll res=sum[t3]-1ll*(y-1)*sz[t3]; ///统计移动个数  不动的 每行都是y-1 sz行

        spilt(t1,sz[t1]-1,t1,t4);///分一个给t4代表1-1  t1代表前面的
        spilt(t3,1,t3,t5);///t3代表1,t5代表中间的
        val[t4]+=val[t3]-y+1;
        mi[t4]=sum[t4]=val[t4];

        val[t3]=mi[t3]=sum[t3]=y-1;///t3表示1 实际上准备放到x的位置
        root=merge(merge(merge(merge(t1,t4),t5),t3),t2);
        return res;
    }
}treap;
int main()
{
    scanf("%d",&T);
    while(T--)
    {
        scanf("%d%d",&n,&q);
        treap.init();
        for(int i=1;i<=n;i++) scanf("%d",&b[i]);
        treap.root=treap.built(b,1,n);
        while(q--)
        {
            int op;
            scanf("%d",&op);
            if(op==1)
            {
                scanf("%d%d",&x,&y);
                printf("%lld\n",treap.getsum(x,y));
            }
            else
            {
                scanf("%d",&x);
                int gs=treap.getnum(x);
                printf("%d\n",gs);
            }
        }
        for(int i=1;i<=n;i++) printf("%d%c",treap.getnum(i),i==n?'\n':' ');
    }
}
///各种基本操作 强制在线
```

```
#define maxn 1100010
struct pair{
  int a,b;
  pair(int a_=0,int b_=0) { a=a_; b=b_; }
};
int key[maxn],wei[maxn],size[maxn],son[maxn][2];
int n,m,cnt,ans,seed=1,root,last;
int rand1() { return seed*=19260817; }
inline void pushup(int u)
  { size[u]=size[son[u][0]]+size[son[u][1]]+1; }
pair split(int u,int k){
  if(!u) return pair(0,0);
  if(key[u]<k){
    pair t=split(son[u][1],k);
    son[u][1]=t.a;
    pushup(u);
    return pair(u,t.b);
  }else{
    pair t=split(son[u][0],k);
    son[u][0]=t.b;
    pushup(u);
    return pair(t.a,u);
  }
}
int merge(int u,int v){
  if(!u||!v) return u+v;
  if(wei[u]<wei[v]){
    son[u][1]=merge(son[u][1],v);
    pushup(u);
    return u;
  }else{
    son[v][0]=merge(u,son[v][0]);
    pushup(v);
    return v;
  }
}
void insert(int k){
  key[++cnt]=k; wei[cnt]=rand1(); size[cnt]=1;
  pair t=split(root,k);
  root=merge(merge(t.a,cnt),t.b);
}
void eraser(int k){
  pair x,y;
  x=split(root,k);
  y=split(x.b,k+1);
  y.a=merge(son[y.a][0],son[y.a][1]);
  root=merge(x.a,merge(y.a,y.b));
}
int find1(int k){
```

```
  int re;
  pair t=split(root,k);
  re=size[t.a]+1;
  root=merge(t.a,t.b);
  return re;
}
int find2(int k){
  int pos=root;
  while(pos){
    if(k==size[son[pos][0]]+1) return key[pos];
    if(k<=size[son[pos][0]]) pos=son[pos][0];
    else { k-=size[son[pos][0]]+1; pos=son[pos][1]; }
  }
}
int lst(int k) { return find2(find1(k)-1); }
int nxt(int k) { return find2(find1(k+1)); }
int main(){
  n=read(); m=read();
  for(int i=1;i<=n;i++){
    int a=read();
    insert(a);
  }
  for(int i=1;i<=m;i++){
    int o=read(),x; x=read();
    if(o==1) insert(x^last);
    if(o==2) eraser(x^last);
    if(o==3) { last=find1(x^last); ans^=last; }
    if(o==4) { last=find2(x^last); ans^=last; }
    if(o==5) { last=lst(x^last); ans^=last; }
    if(o==6) { last=nxt(x^last); ans^=last; }
  }
  printf("%d\n",ans);
  return 0;
}
```