线段树

多标记多次方

```
#include <bits/stdc++.h>
using namespace std;
#define ls i<<1
#define rs i<<1|1
#define mid (l+r)/2
typedef long long ll;
const int N=1e5+50;
const ll mod=10007;
ll st[N*4],ad[N*4],mu[N*4],sum[N*4][4];
int n,m,o,x,y,c;
void pushup(int i){
    for(int j=1;j<=3;j++){
        sum[i][j]=(sum[ls][j]+sum[rs][j])%mod;
    }
}
void build(int i,int l,int r){
    st[i]=ad[i]=0;
    mu[i]=1ll;
    if(l==r){
        for(int j=1;j<=3;j++){
            sum[i][j]=0;
        }
        return;
    }
    build(ls,l,mid);
    build(rs,mid+1,r);
    pushup(i);
}
void funSet(int i,int l,int r,ll v){
    ad[i]=0;
    mu[i]=1;
    st[i]=v;
    sum[i][3]=(v*v%mod*v%mod*(r-l+1))%mod;
    sum[i][2]=(v*v%mod*(r-l+1))%mod;
    sum[i][1]=(v*(r-l+1))%mod;
}
void funMul(int i,int l,int r,ll v){
    ad[i]=(ad[i]*v)%mod;
    mu[i]=(mu[i]*v)%mod;
    sum[i][3]=(sum[i][3]*v%mod*v%mod*v)%mod;
    sum[i][2]=(sum[i][2]*v%mod*v)%mod;
    sum[i][1]=(sum[i][1]*v)%mod;
}
void funAdd(int i,int l,int r,ll v){
    ad[i]=(ad[i]+v)%mod;
    sum[i][3]=(sum[i][3]+v*v%mod*v%mod*(r-l+1)%mod
```

```
                +3ll*sum[i][2]%mod*v%mod+3ll*sum[i][1]%mod*v%mod*v%mod)%mod;
    sum[i][2]=(sum[i][2]+v*v%mod*(r-l+1)%mod+2ll*sum[i][1]%mod*v%mod)%mod;
    sum[i][1]=(sum[i][1]+v*(r-l+1))%mod;
}
void pushdown(int i,int l,int r){
    if(st[i]){
        funSet(ls,l,mid,st[i]);
        funSet(rs,mid+1,r,st[i]);
        st[i]=0;
    }
    if(mu[i]!=1){
        funMul(ls,l,mid,mu[i]);
        funMul(rs,mid+1,r,mu[i]);
        mu[i]=1;
    }
    if(ad[i]){
        funAdd(ls,l,mid,ad[i]);
        funAdd(rs,mid+1,r,ad[i]);
        ad[i]=0;
    }
}
void update(int i,int l,int r,int ql,int qr,int o,ll v){
    if(ql<=l && qr>=r){
        v%=mod;
        if(o==1){
            funAdd(i,l,r,v);
        }else if(o==2){
            funMul(i,l,r,v);
        }else if(o==3){
            funSet(i,l,r,v);
        }
        return;
    }
    pushdown(i,l,r);
    if(ql<=mid){
        update(ls,l,mid,ql,qr,o,v);
    }
    if(qr>mid){
        update(rs,mid+1,r,ql,qr,o,v);
    }
    pushup(i);
}
ll query(int i,int l,int r,int ql,int qr,int p){
    if(ql<=l && qr>=r){
        return sum[i][p]%mod;
    }
    pushdown(i,l,r);
    ll ans=0;
    if(ql<=mid){
        ans=(ans+query(ls,l,mid,ql,qr,p))%mod;
```

```
    }
    if(qr>mid){
        ans=(ans+query(rs,mid+1,r,ql,qr,p))%mod;
    }
    return ans%mod;
}
int main(){
    while(~scanf("%d%d",&n,&m) && n+m){
        build(1,1,n);
        for(int i=0;i<m;i++){
            scanf("%d%d%d%d",&o,&x,&y,&c);
            if(o==4){
                ll ans=query(1,1,n,x,y,c);
                printf("%lld\n",ans%mod);
            }else{
                update(1,1,n,x,y,o,ll(c));
            }
        }
    }
    return 0;
}
```

最长连续区间

```
#include <bits/stdc++.h>
using namespace std;
#define lson i<<1
#define rson i<<1|1
#define mid (l+r)/2
const int N=50050;
int lm[4*N],rm[4*N],mx[4*N];
void pushup(int i,int l,int r){
    lm[i]=lm[lson];
    rm[i]=rm[rson];
    if(lm[lson]==mid-l+1){
        lm[i]+=lm[rson];
    }
    if(rm[rson]==r-mid){
        rm[i]+=rm[lson];
    }
    mx[i]=max(max(mx[lson],mx[rson]),rm[lson]+lm[rson]);
}
void build(int i,int l,int r){
    if(l==r){
        lm[i]=rm[i]=mx[i]=1;
        return;
    }
    build(lson,l,mid);
    build(rson,mid+1,r);
    pushup(i,l,r);
```

```
}
void update(int i,int l,int r,int p,int c){
    if(l==r && l==p){
        lm[i]=rm[i]=mx[i]=c;
        return;
    }
    if(p<=mid){
        update(lson,l,mid,p,c);
    }else{
        update(rson,mid+1,r,p,c);
    }
    pushup(i,l,r);
}
int query(int i,int l,int r,int p){
    if(l==r && l==p){
        return mx[i];
    }
    if(p<=mid){
        return query(lson,l,mid,p);
    }else{
        return query(rson,mid+1,r,p);
    }
}
//查询 1 到 x 的右边区间
int queryR(int i,int l,int r,int ql,int qr){
    if(ql<=l && qr>=r){
        return rm[i];
    }
    int la=0,ra=0;
    if(ql<=mid){
        la=queryR(lson,l,mid,ql,min(mid,qr));
    }
    if(qr>mid){
        ra=queryR(rson,mid+1,r,mid+1,qr);
        if(ra==qr-mid){
            return la+ra;
        }else{
            return ra;
        }
    }else{
        return la;
    }
}
//查询 x 到 n 的左边区间
int queryL(int i,int l,int r,int ql,int qr){
    if(ql<=l && qr>=r){
        return lm[i];
    }
    int la=0,ra=0;
    if(qr>mid){
```

```
        ra=queryL(rson,mid+1,r,max(ql,mid+1),qr);
    }
    if(ql<=mid){
        la=queryL(lson,l,mid,ql,mid);
        if(la==mid-ql+1){
            return ra+la;
        }else{
            return la;
        }
    }else{
        return ra;
    }
}
int n,m,x;
char q[10];
stack<int> sta;
int main(void){
    while(~scanf("%d%d",&n,&m)){
        build(1,1,n);
        while(m--){
            scanf("%s",q);
            if(q[0]=='D'){
                scanf("%d",&x);
                sta.push(x);
                update(1,1,n,x,0);
            }else if(q[0]=='Q'){
                scanf("%d",&x);
                int ri=queryR(1,1,n,1,x);
                int le=queryL(1,1,n,x,n);
                int mm=query(1,1,n,x);
                printf("%d\n",ri+le-mm);
            }else if(q[0]=='R'){
                int las=sta.top();
                sta.pop();
                update(1,1,n,las,1);
            }
        }
    }
    return 0;
}
```

多线段树可用连续区间

```
#include <bits/stdc++.h>
using namespace std;
#define ls i<<1
#define rs i<<1|1
#define mid (l+r)/2
const int N=1e5+50;
//不同线段树的区间覆盖问题，1 表示空闲时间，ds 占用时间只会影响 ds 线段树的空闲时间
```

```
//ns 请求时间可以查询 ds 线段树，即全局空闲时间，再查询 ns 线段树，即被 ds 占用的时间
int lm[2][N*4],rm[2][N*4],mx[2][N*4],lz[2][N*4];
void pushup(int j,int i,int l,int r){
    mx[j][i]=max(max(mx[j][ls],mx[j][rs]),rm[j][ls]+lm[j][rs]);
    lm[j][i]=lm[j][ls];
    if(lm[j][i]==mid-l+1){
        lm[j][i]+=lm[j][rs];
    }
    rm[j][i]=rm[j][rs];
    if(rm[j][i]==r-mid){
        rm[j][i]+=rm[j][ls];
    }
}
void fun(int j,int i,int l,int r,int v){
    lz[j][i]=v;
    lm[j][i]=rm[j][i]=mx[j][i]=(r-l+1)*v;
}
void pushdown(int j,int i,int l,int r){
    if(lz[j][i]!=-1){
        fun(j,ls,l,mid,lz[j][i]);
        fun(j,rs,mid+1,r,lz[j][i]);
        lz[j][i]=-1;
    }
}
void build(int i,int l,int r){
    for(int j=0;j<2;j++){
        lz[j][i]=-1;
        lm[j][i]=rm[j][i]=mx[j][i]=(r-l+1);
    }
    if(l==r){
        return;
    }
    build(ls,l,mid);
    build(rs,mid+1,r);
}
void update(int j,int i,int l,int r,int ql,int qr,int v){
    if(ql<=l && qr>=r){
        fun(j,i,l,r,v);
        return;
    }
    pushdown(j,i,l,r);
    if(ql<=mid){
        update(j,ls,l,mid,ql,qr,v);
    }
    if(qr>mid){
        update(j,rs,mid+1,r,ql,qr,v);
    }
    pushup(j,i,l,r);
}
int query(int j,int i,int l,int r,int num){
```

```
    if(l==r){
        return l;
    }
    pushdown(j,i,l,r);
    if(mx[j][ls]>=num){
        return query(j,ls,l,mid,num);
    }else{
        if(rm[j][ls]+lm[j][rs]>=num){
            return mid-rm[j][ls]+1;
        }else{
            return query(j,rs,mid+1,r,num);
        }
    }
}
int T,n,m,q,r;
char s[10];
int main(){
    scanf("%d",&T);
    for(int cas=1;cas<=T;cas++){
        scanf("%d%d",&n,&m);
        printf("Case %d:\n",cas);
        build(1,1,n);
        for(int i=1;i<=m;i++){
            scanf("%s%d",s,&q);
            if(s[0]=='D'){
                if(mx[1][1]<q){
                    printf("fly with yourself\n");
                }else{
                    int ans=query(1,1,1,n,q);
                    //只减少 ds 空闲时间
                    update(1,1,1,n,ans,ans+q-1,0);
                    printf("%d,let's fly\n",ans);
                }
            }else if(s[0]=='N'){
                if(mx[1][1]<q && mx[0][1]<q){
                    printf("wait for me\n");
                }else{
                    int p;
                    if(mx[1][1]>=q){
                        //ds 空闲时间足够，直接用
                        p=query(1,1,1,n,q);
                    }else{
                        //否则，用 ns 空闲时间，会覆盖一部分的 ds 时间
                        p=query(0,1,1,n,q);
                    }
                    //减少 ds 和 ns 空闲时间
                    update(0,1,1,n,p,p+q-1,0);
                    update(1,1,1,n,p,p+q-1,0);
                    printf("%d,don't put my gezi\n",p);
                }
```

```
            }else if(s[0]=='S'){
                scanf("%d",&r);
                update(0,1,1,n,q,r,1);
                update(1,1,1,n,q,r,1);
                printf("I am the hope of chinese chengxuyuan!!\n");
            }
        }
    }
    return 0;
}
```

二分线段树查询

```
#include <bits/stdc++.h>
using namespace std;
#define ls i<<1
#define rs i<<1|1
#define mid (l+r)/2
const int N=5e4+50;
int sum[N*4],lz[N*4];
void pushup(int i){
    sum[i]=sum[ls]+sum[rs];
}
void build(int i,int l,int r){
    lz[i]=-1;
    if(l==r){
        sum[i]=1;
        return;
    }
    build(ls,l,mid);
    build(rs,mid+1,r);
    pushup(i);
}
void fun(int i,int l,int r,int v){
    lz[i]=v;
    sum[i]=(r-l+1)*v;
}
void pushdown(int i,int l,int r){
    if(lz[i]!=-1){
        fun(ls,l,mid,lz[i]);
        fun(rs,mid+1,r,lz[i]);
        lz[i]=-1;
    }
}
void update(int i,int l,int r,int ql,int qr,int v){
    if(ql<=l && qr>=r){
        fun(i,l,r,v);
        return;
    }
    pushdown(i,l,r);
```

```
    if(ql<=mid){
        update(ls,l,mid,ql,qr,v);
    }
    if(qr>mid){
        update(rs,mid+1,r,ql,qr,v);
    }
    pushup(i);
}
int query(int i,int l,int r,int ql,int qr){
    if(ql<=l && qr>=r){
        return sum[i];
    }
    pushdown(i,l,r);
    int ans=0;
    if(ql<=mid){
        ans+=query(ls,l,mid,ql,qr);
    }
    if(qr>mid){
        ans+=query(rs,mid+1,r,ql,qr);
    }
    return ans;
}
int T,n,m,o,l,r;
//二分从 l 找到第一个 sum 为 1(空花瓶) 的位置
int findL(int l){
    int L=l,R=n;
    int ans=0;
    while(L<=R){
        int md=(L+R)/2;
        if(query(1,1,n,l,md)>=1){
            ans=md;
            R=md-1;
        }else{
            L=md+1;
        }
    }
    return ans;
}
//二分从 l 找到第一个 r 使得 sum(l...r)=num(从 l 放下连续 num 朵花)
int findR(int l,int num){
    int L=l,R=n;
    int ans=0;
    while(L<=R){
        int md=(L+R)/2;
        if(query(1,1,n,l,md)>=num){
            ans=md;
            R=md-1;
        }else{
            L=md+1;
        }
```

```
    }
    //不够放，从右边二分找到最后一个空花瓶
    if(!ans){
        int LL=n,RR=1;
        while(LL>=RR){
            int md=(LL+RR)/2;
            if(query(1,1,n,md,n)>=1){
                ans=md;
                RR=md+1;
            }else{
                LL=md-1;
            }
        }
    }
    return ans;
}
int main(){
    scanf("%d",&T);
    while(T--){
        scanf("%d%d",&n,&m);
        build(1,1,n);
        for(int i=1;i<=m;i++){
            scanf("%d%d%d",&o,&l,&r);
            if(o==1){
                l++;
                int L=findL(l);
                if(L==0){
                    printf("Can not put any one.\n");
                    continue;
                }
                int R=findR(L,r);
                printf("%d %d\n",L-1,R-1);
                update(1,1,n,L,R,0);
            }else if(o==2){
                l++;
                r++;
                printf("%d\n",r-l+1-query(1,1,n,l,r));
                update(1,1,n,l,r,1);
            }
        }
        printf("\n");
    }
    return 0;
}
```

扫描线见 part.pdf

可持久化普通线段树

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N=1e5+50;
int n,m,tr[N],tim,l,r,t;
char q[5];
ll a[N],v;
struct HJT{
#define mid (l+r)/2
    int tot,ls[N*50],rs[N*50];
    ll sum[N*50],lz[N*50];
    void init(){
        tot=0;
    }
    //普通线段树的操作是 update 递归到下一层之前先把 lz 标记下放
    //而标记永久化是不下放，而是等 pushup 回溯再加上当前层的 lz
    void pushup(int i,int l,int r){
        sum[i]=sum[ls[i]]+sum[rs[i]]+lz[i]*(r-l+1);
    }
    void build(int &rt,int l,int r){
        rt=++tot;
        ls[rt]=rs[rt]=lz[rt]=0;
        if(l==r){
            sum[rt]=a[l];
            return;
        }
        build(ls[rt],l,mid);
        build(rs[rt],mid+1,r);
        pushup(rt,l,r);
    }
    void update(int &rt,int pre,int l,int r,int ql,int qr,ll v){
        rt=++tot;
        ls[rt]=ls[pre];
        rs[rt]=rs[pre];
        lz[rt]=lz[pre];
        sum[rt]=sum[pre];
        if(ql<=l && qr>=r){
            lz[rt]+=v;
            sum[rt]+=v*(r-l+1);
            return;
        }
        if(ql<=mid){
            update(ls[rt],ls[pre],l,mid,ql,qr,v);
        }
        if(qr>mid){
            update(rs[rt],rs[pre],mid+1,r,ql,qr,v);
        }
```

```
        pushup(rt,l,r);
    }
    ll query(int rt,int l,int r,int ql,int qr,ll add){
        if(ql<=l && qr>=r){
            //标记永久化
            return sum[rt]+(r-l+1)*add;
        }
        ll ans=0;
        if(ql<=mid){
            ans+=query(ls[rt],l,mid,ql,qr,add+lz[rt]);
        }
        if(qr>mid){
            ans+=query(rs[rt],mid+1,r,ql,qr,add+lz[rt]);
        }
        return ans;
    }
}ac;
int main(){
    // freopen("in.txt","r",stdin);
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;i++){
        scanf("%lld",&a[i]);
    }
    ac.init();
    tim=0;
    ac.build(tr[0],1,n);
    while(m--){
        scanf("%s",q);
        if(q[0]=='C'){
            scanf("%d%d%lld",&l,&r,&v);
            tim++;
            ac.update(tr[tim],tr[tim-1],1,n,l,r,v);
        }else if(q[0]=='H'){
            scanf("%d%d%d",&l,&r,&t);
            printf("%lld\n",ac.query(tr[t],1,n,l,r,0));
        }else if(q[0]=='Q'){
            scanf("%d%d",&l,&r);
            printf("%lld\n",ac.query(tr[tim],1,n,l,r,0));
        }else if(q[0]=='B'){
            scanf("%d",&tim);
        }
    }
    return 0;
}
```

线段树维护区间状态转移矩阵

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

```
#define rs i<<1|1
#define mid (l+r)/2
const int N=2e5+50;
const int INF=0x3f3f3f3f;
char s[N];
//矩阵 a[i][j] 表示状态 i 转移到状态 j 的最小花费
//0 初始状态
//1 2
//2 20
//3 201
//4 2017
struct Mat{
    int a[5][5];
    Mat operator+(const Mat& rhs)const{
       Mat ans;
       for(int i=0;i<5;i++){
           for(int j=0;j<5;j++){
               ans.a[i][j]=INF;
               for(int k=0;k<5;k++){
                   ans.a[i][j]=min(ans.a[i][j],a[i][k]+rhs.a[k][j]);
               }
           }
       }
       return ans;
    }
}t[N*4];
void pushup(int i){
    t[i]=t[ls]+t[rs];
}
void build(int i,int l,int r){
    if(l==r){
        //叶子节点表示单个字符的状态
        for(int j=0;j<5;j++){
            for(int k=0;k<5;k++){
                t[i].a[j][k]=(j==k)?0:INF;
            }
        }
        if(s[l]=='2'){
            t[i].a[0][1]=0;
            t[i].a[0][0]=1;
        }else if(s[l]=='0'){
            t[i].a[1][2]=0;
            t[i].a[1][1]=1;
        }else if(s[l]=='1'){
            t[i].a[2][3]=0;
            t[i].a[2][2]=1;
        }else if(s[l]=='7'){
            t[i].a[3][4]=0;
            t[i].a[3][3]=1;
        }else if(s[l]=='6'){
```

```
            t[i].a[3][3]=1;
            t[i].a[4][4]=1;
        }
        return;
    }
    build(ls,l,mid);
    build(rs,mid+1,r);
    pushup(i);
}
void print(Mat a){
    for(int j=0;j<5;j++){
        for(int k=0;k<5;k++){
            printf("%d ",a.a[j][k]);
        }
        printf("\n");
    }
}
Mat query(int i,int l,int r,int ql,int qr){
    if(ql<=l && qr>=r){
        return t[i];
    }
    //换个写法，不用写零元
    if(qr<=mid){
        return query(ls,l,mid,ql,qr);
    }else if(ql>mid){
        return query(rs,mid+1,r,ql,qr);
    }else{
        return query(ls,l,mid,ql,qr)+query(rs,mid+1,r,ql,qr);
    }
}
int n,q,l,r;
int main(){
//    freopen("in.txt","r",stdin);
    scanf("%d%d",&n,&q);
    scanf("%s",s+1);
    build(1,1,n);
    while(q--){
        scanf("%d%d",&l,&r);
        Mat ans=query(1,1,n,l,r);
        printf("%d\n",ans.a[0][4]==INF?-1:ans.a[0][4]);
    }
    return 0;
}
```

区间加斐波那契数列

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

```
#define mid (l+r)/2
typedef long long ll;
const ll mod=1e9+9;
const int N=3e5+50;
int n,m,o,l,r;
ll a[N];
ll sum[N*4];
//维护区间前两个位置的系数
int la[N*4],lb[N*4];
ll f[N];
//计算 f[k]
ll calc(ll a,ll b,int k){
    if(k==1){
        return a;
    }else if(k==2){
        return b;
    }else{
        return (a*f[k-2]%mod+b*f[k-1]%mod)%mod;
    }
}
//计算 f[i] 前缀和
ll get(ll a,ll b,int k){
    if(k==1){
        return a;
    }else if(k==2){
        return (a+b)%mod;
    }else{
        //f[i] 前缀和等于 h[k+2]-h[2]
        return (calc(a,b,k+2)-b+mod)%mod;
    }
}
void pushup(int i){
    sum[i]=(sum[ls]+sum[rs])%mod;
}
void build(int i,int l,int r){
    la[i]=0;
    lb[i]=0;
    if(l==r){
        sum[i]=a[l];
        return;
    }
    build(ls,l,mid);
    build(rs,mid+1,r);
    pushup(i);
}
void pushdown(int i,int l,int r){
    if(la[i]){
        //左儿子的前两项等同于父节点前两项
        la[ls]=(la[ls]+la[i])%mod;
        lb[ls]=(lb[ls]+lb[i])%mod;
```

```
        sum[ls]=(sum[ls]+get(la[i],lb[i],mid-l+1))%mod;
        //右儿子不等，计算右儿子的前两项
        ll ta=calc(la[i],lb[i],mid-l+1+1);
        ll tb=calc(la[i],lb[i],mid-l+1+2);
        la[rs]=(la[rs]+ta)%mod;
        lb[rs]=(lb[rs]+tb)%mod;
        sum[rs]=(sum[rs]+get(ta,tb,r-mid))%mod;
        la[i]=0;
        lb[i]=0;
    }
}
void update(int i,int l,int r,int ql,int qr){
    if(ql<=l && qr>=r){
        //线段树节点区间 (不是更新区间)f 序列前两项
        ll ta=f[l-ql+1];
        ll tb=f[l-ql+2];
        la[i]=(la[i]+ta)%mod;
        lb[i]=(lb[i]+tb)%mod;
        sum[i]=(sum[i]+get(ta,tb,r-l+1))%mod;
        return;
    }
    pushdown(i,l,r);
    if(ql<=mid){
        update(ls,l,mid,ql,qr);
    }
    if(qr>mid){
        update(rs,mid+1,r,ql,qr);
    }
    pushup(i);
}
ll query(int i,int l,int r,int ql,int qr){
    if(ql<=l && qr>=r){
        return sum[i]%mod;
    }
    ll ans=0;
    pushdown(i,l,r);
    if(ql<=mid){
        ans=(ans+query(ls,l,mid,ql,qr))%mod;
    }
    if(qr>mid){
        ans=(ans+query(rs,mid+1,r,ql,qr))%mod;
    }
    return ans%mod;
}
int main(void){
    // freopen("in.txt","r",stdin);
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;i++){
        scanf("%lld",&a[i]);
    }
```

```
    //先预处理出 1 1 开头的斐波那契数列，对于任意斐波那契数列，记录前两项，就能 O(1) 推出
    f[1]=1ll;
    f[2]=1ll;
    for(int i=3;i<=n+2;i++){
        f[i]=(f[i-1]+f[i-2])%mod;
    }
    build(1,1,n);
    while(m--){
        scanf("%d%d%d",&o,&l,&r);
        if(o==1){
            update(1,1,n,l,r);
        }else{
            printf("%lld\n",query(1,1,n,l,r));
        }
    }
    return 0;
}
```

区间加等差数列

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N=1e5+50;
ll a[N];
int n,m,o,l,r,k,d,p;
struct ST{
    //线段树，区间修改，区间查询
}ac;
int main(){
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;i++){
        scanf("%lld",&a[i]);
    }
    ac.build(1,1,n);
    while(m--){
        scanf("%d",&o);
        if(o==1){
            scanf("%d%d%d%d",&l,&r,&k,&d);
            //维护差分数组，最后查询求前缀和就是单点的累加值
            //第一项加 k
            ac.update(1,1,n,l,l,k);
            if(l<r){
                //等差数列，中间每一项相差 d
                ac.update(1,1,n,l+1,r,d);
            }
            if(r<n){
                //差分数组的区间修改，类比 a[l]++ a[r+1]--
                ac.update(1,1,n,r+1,r+1,-(k+(r-l)*d));
            }
```

```
        }else if(o==2){
            scanf("%d",&p);
            printf("%lld\n",a[p]+ac.query(1,1,n,1,p));
        }
    }
    return 0;
}
```

树套树

带修区间第 k 小

```
#include <bits/stdc++.h>
using namespace std;
const int N=2e5+50;
int n,m,nx,a[N];
//用 bit 维护权值线段树，因此查询区间 [l,r] 不再是直接查询 tr[l-1] 到 tr[r]
//因为 bit 维护的前缀和是不连续的前缀和，所以要先预处理出查询路径
int x[N],y[N];
int c1,c2;
struct Orz{
    vector<int> a;
    void init(){
        a.clear();
    }
    int siz(){
        return a.size();
    }
    void add(int x){
        a.push_back(x);
    }
    void work(){
        sort(a.begin(),a.end());
        a.erase(unique(a.begin(),a.end()),a.end());
    }
    int idx(int v){
        return lower_bound(a.begin(),a.end(),v)-a.begin()+1;
    }
    int val(int i){
        return a[i-1];
    }
}orz;
int tr[N*100];
struct HJT{
#define mid (l+r)/2
    int tot,sum[N*100],ls[N*100],rs[N*100];
    //动态开点权值线段树
    void update(int& rt,int l,int r,int v,int add){
        if(!rt){
            rt=++tot;
```

```
        }
        sum[rt]+=add;
        if(l<r){
            if(v<=mid){
                update(ls[rt],l,mid,v,add);
            }else{
                update(rs[rt],mid+1,r,v,add);
            }
        }
    }
    //查询区间第 k 大
    int query(int l,int r,int k){
        if(l>=r){
            return l;
        }
        //普通的主席树是直接前缀和做差求区间 sum
        //这里要根据 bit 预处理出来的子树路径计算 sum
        int ans=0;
        for(int i=1;i<=c1;i++){
            ans-=sum[ls[x[i]]];
        }
        for(int i=1;i<=c2;i++){
            ans+=sum[ls[y[i]]];
        }
        if(k<=ans){
            for(int i=1;i<=c1;i++){
                x[i]=ls[x[i]];
            }
            for(int i=1;i<=c2;i++){
                y[i]=ls[y[i]];
            }
            return query(l,mid,k);
        }else{
            for(int i=1;i<=c1;i++){
                x[i]=rs[x[i]];
            }
            for(int i=1;i<=c2;i++){
                y[i]=rs[y[i]];
            }
            return query(mid+1,r,k-ans);
        }
    }
}ac;
struct BIT{
    int lowbit(int x){
        return x&(-x);
    }
    //修改权值线段树的 bit 前缀和 (非连续)
    void modify(int i,int x){
        int k=orz.idx(a[i]);
```

```
        while(i<=n){
            ac.update(tr[i],1,nx,k,x);
            i+=lowbit(i);
        }
    }
    //预处理权值线段树的查询路径
    int query(int l,int r,int k){
        c1=c2=0;
        for(int i=(l-1);i;i-=lowbit(i)){
            x[++c1]=tr[i];
        }
        for(int i=r;i;i-=lowbit(i)){
            y[++c2]=tr[i];
        }
        return ac.query(1,nx,k);
    }
}bit;
struct Query{
    int o,l,r,k;
}q[N];
char op[5];
int main(){
    scanf("%d%d",&n,&m);
    orz.init();
    for(int i=1;i<=n;i++){
        scanf("%d",&a[i]);
        orz.add(a[i]);
    }
    for(int i=1;i<=m;i++){
        scanf("%s",op);
        scanf("%d%d",&q[i].l,&q[i].r);
        if(op[0]=='Q'){
            q[i].o=1;
            scanf("%d",&q[i].k);
        }else{
            orz.add(q[i].r);
        }
    }
    orz.work();
    nx=orz.siz();
    for(int i=1;i<=n;i++){
        bit.modify(i,1);
    }
    for(int i=1;i<=m;i++){
        if(q[i].o==1){
            printf("%d\n",orz.val(bit.query(q[i].l,q[i].r,q[i].k)));
        }else{
            bit.modify(q[i].l,-1);
            a[q[i].l]=q[i].r;
            bit.modify(q[i].l,1);
```

```
        }
    }
    return 0;
}
```

带修区间值域个数

```
#include <bits/stdc++.h>
using namespace std;
const int N=2e5+150;
int n,m,a[N],b[N];
int o,l,r,xi,yi;
int x[N],y[N];
int c1,c2;
int tr[N*150];
struct HJT{
#define mid (l+r)/2
    int tot,sum[N*150],ls[N*150],rs[N*150];
    void update(int& rt,int l,int r,int v,int add){
        //因为 b 数组有 0 值, 在查询时 0 不算
        if(!v){
            return;
        }
        if(!rt){
            rt=++tot;
        }
        sum[rt]+=add;
        if(l<r){
            if(v<=mid){
                update(ls[rt],l,mid,v,add);
            }else{
                update(rs[rt],mid+1,r,v,add);
            }
        }
    }
    //区间 [l,r] 值域在 [1,k] 的个数
    int query(int l,int r,int k){
        if(k==0){
            return 0;
        }
        if(r<=k){
            int ans=0;
            for(int i=1;i<=c1;i++){
                ans-=sum[x[i]];
            }
            for(int i=1;i<=c2;i++){
                ans+=sum[y[i]];
            }
            return ans;
        }
```

```
        if(k<=mid){
            for(int i=1;i<=c1;i++){
                x[i]=ls[x[i]];
            }
            for(int i=1;i<=c2;i++){
                y[i]=ls[y[i]];
            }
            return query(l,mid,k);
        }else{
            int ans=0;
            for(int i=1;i<=c1;i++){
                ans-=sum[ls[x[i]]];
            }
            for(int i=1;i<=c2;i++){
                ans+=sum[ls[y[i]]];
            }
            for(int i=1;i<=c1;i++){
                x[i]=rs[x[i]];
            }
            for(int i=1;i<=c2;i++){
                y[i]=rs[y[i]];
            }
            return ans+query(mid+1,r,k);
        }
    }
}ac;
struct BIT{
    int lowbit(int x){
        return x&(-x);
    }
    //修改权值线段树的 bit 前缀和 (非连续)
    void modify(int i,int x){
        int k=b[i];
        while(i<=n){
            ac.update(tr[i],1,n,k,x);
            i+=lowbit(i);
        }
    }
    //预处理权值线段树的查询路径
    int query(int l,int r,int xi,int yi){
        c1=c2=0;
        for(int i=(l-1);i;i-=lowbit(i)){
            x[++c1]=tr[i];
        }
        for(int i=r;i;i-=lowbit(i)){
            y[++c2]=tr[i];
        }
        int R=ac.query(1,n,yi);
        c1=c2=0;
        for(int i=(l-1);i;i-=lowbit(i)){
```

```
            x[++c1]=tr[i];
        }
        for(int i=r;i;i-=lowbit(i)){
            y[++c2]=tr[i];
        }
        int L=ac.query(1,n,xi-1);
        return R-L;
    }
}bit;
int main(){
//    freopen("in.txt","r",stdin);
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;i++){
        scanf("%d",&a[i]);
        b[i]=(a[i]==a[i-1])?0:a[i];
    }
    for(int i=1;i<=n;i++){
        bit.modify(i,1);
    }
    for(int i=1;i<=m;i++){
        scanf("%d%d%d",&o,&l,&r);
        if(o==1){
            bit.modify(l,-1);
            bit.modify(l+1,-1);
            a[l]=r;
            b[l]=(a[l]==a[l-1])?0:a[l];
            b[l+1]=(a[l+1]==a[l])?0:a[l+1];
            bit.modify(l,1);
            bit.modify(l+1,1);
        }else{
            scanf("%d%d",&xi,&yi);
            int ans=bit.query(l+1,r,xi,yi);
            if(a[l]>=xi && a[l]<=yi){
                ans++;
            }
            printf("%d\n",ans);
        }
    }
    return 0;
}
```

带插入区间第 k 大

```
#include <bits/stdc++.h>
using namespace std;
#define mid (l+r)/2
const int N=1e5+50;
typedef long long ll;
int n,m,ns;
struct Q{
```

```
    int o,a,b;
    ll c;
}q[N];
struct Orz{
    //离散化
}orz;
//内层的普通线段树，维护外层值域区间为 [L,R] 的数在位置区间 [l,r] 的个数
struct ST{
    int tot,ls[N*200],rs[N*200];;
    ll sum[N*200],lz[N*200];
    void update(int &i,int l,int r,int ql,int qr,int v){
        if(!i){
            i=++tot;
        }
        //标记永久化
        sum[i]+=min(qr,r)-max(ql,l)+1;
        if(ql<=l && qr>=r){
            lz[i]+=v;
            return;
        }
        if(ql<=mid){
            update(ls[i],l,mid,ql,qr,v);
        }
        if(qr>mid){
            update(rs[i],mid+1,r,ql,qr,v);
        }
    }
    ll query(int i,int l,int r,int ql,int qr,int add=0){
        if(!i){
            //有上面带下来的标记，不能返回 0
            return (min(qr,r)-max(ql,l)+1)*add;
        }
        if(ql<=l && qr>=r){
            return sum[i]+add*(r-l+1);
        }
        ll ans=0;
        if(ql<=mid){
            ans+=query(ls[i],l,mid,ql,qr,add+lz[i]);
        }
        if(qr>mid){
            ans+=query(rs[i],mid+1,r,ql,qr,add+lz[i]);
        }
        return ans;
    }
}st;
//外层的权值线段树
struct VST{
#define ls i<<1
#define rs i<<1|1
    int tr[N*4];
```

```
    //当前 vst 值域范围为 [l,r], 更新值 v 对应的线段树区间 [ql,qr], 即加上 v 这个数
    void update(int i,int l,int r,int ql,int qr,int v){
        //该值域区间 [l,r] 所对应的线段树的区间 [ql,qr] 所维护的数字个数加 1
        st.update(tr[i],1,n,ql,qr,1);
        if(l==r){
            return;
        }
        if(v<=mid){
            update(ls,l,mid,ql,qr,v);
        }else{
            update(rs,mid+1,r,ql,qr,v);
        }
    }
    //查询区间 [ql,qr] 值域 [l,r] 的第 k 大!
    int query(int i,int l,int r,int ql,int qr,ll k){
        if(l==r){
            return l;
        }
        //区间 [ql,qr] 左边值域对应数的个数
        ll sum=st.query(tr[rs],1,n,ql,qr);
        if(k<=sum){
            return query(rs,mid+1,r,ql,qr,k);
        }else{
            return query(ls,l,mid,ql,qr,k-sum);
        }
    }
}vst;
int main(){
//    freopen("in.txt","r",stdin);
   scanf("%d%d",&n,&m);
    for(int i=1;i<=m;i++){
       scanf("%d%d%d%lld",&q[i].o,&q[i].a,&q[i].b,&q[i].c);
        if(q[i].o==1){
            orz.add(q[i].c);
        }
    }
    orz.work();
    ns=orz.siz();
    for(int i=1;i<=m;i++){
        if(q[i].o==1){
            q[i].c=orz.idx(q[i].c);
        }
    }
    for(int i=1;i<=m;i++){
        if(q[i].o==1){
            vst.update(1,1,ns,q[i].a,q[i].b,q[i].c);
        }else{
            printf("%lld\n",orz.val(vst.query(1,1,ns,q[i].a,q[i].b,q[i].c)));
        }
    }
```

```
    return 0;
}
```

整体二分

区间第 k 小

```
#include <bits/stdc++.h>
using namespace std;
//值域有负数的情况，用位运算除法
#define mid ((l+r)>>1)
const int N=1e6+50;
const int INF=1e9+7;
int n,m,a[N],l,r,k,ans[N];
struct Q{
    int id,l,r,k;
}q[N],L[N],R[N];
struct BIT{
    //树状数组
}bit;
//整体二分 [ql,qr] 是询问范围 [l,r] 是值域范围
void solve(int ql,int qr,int l,int r){
    if(ql>qr){
        return;
    }
    if(l==r){
        for(int i=ql;i<=qr;i++){
            //询问
            if(q[i].id){
                ans[q[i].id]=l;
            }
        }
        return;
    }
    int cl=0,cr=0;
    for(int i=ql;i<=qr;i++){
        if(!q[i].id){
            //插入
            if(q[i].k<=mid){
                L[++cl]=q[i];
                bit.add(q[i].l,1);
            }else{
                R[++cr]=q[i];
            }
        }else{
            //询问
            int t=bit.sum(q[i].r)-bit.sum(q[i].l-1);
            if(q[i].k<=t){
                L[++cl]=q[i];
            }else{
```

```
                q[i].k-=t;
                R[++cr]=q[i];
            }
        }
    }
    for(int i=1;i<=cl;i++){
        q[ql-1+i]=L[i];
        if(!L[i].id){
            bit.add(L[i].l,-1);
        }
    }
    for(int i=1;i<=cr;i++){
        q[ql-1+cl+i]=R[i];
    }
    solve(ql,ql+cl-1,l,mid);
    solve(ql+cl,qr,mid+1,r);
}
int main(){
//    freopen("in.txt","r",stdin);
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;i++){
        scanf("%d",&a[i]);
    }
    for(int i=1;i<=n;i++){
        q[i]=Q{0,i,i,a[i]};
    }
    for(int i=1;i<=m;i++){
        scanf("%d%d%d",&l,&r,&k);
        q[n+i]=Q{i,l,r,k};
    }
    solve(1,n+m,-INF,INF);
    for(int i=1;i<=m;i++){
        printf("%d\n",ans[i]);
    }
    return 0;
}
```