

ACM Template

Zeng Xiaocan

September 11, 2019

Contents

1	线段树	3
1.1	多标记多次方	3
1.2	最长连续区间	5
1.3	多线段树可用连续区间	7
1.4	二分线段树查询	10

1 线段树

1.1 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

```

```

        +311*sum[i][2]%mod*v%mod+311*sum[i][1]%mod*v%mod*v%mod)%mod;
sum[i][2]=(sum[i][2]+v*v%mod*(r-l+1)%mod+211*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;
}

```

1.2 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;
}

```

1.3 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;
}

```

1.4 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;
}

```

2 树套树

2.1 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;
}

```

2.2 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;
}
```