

线段树

前言

解决的问题

在线维护修改、查询区间最值、求和，可以扩充到二位线段树、三位线段树。

复杂度

一维线段树每次更新和查询的时间复杂度为 $O(\log\{N\})$

概念

二叉树的结构，每一个节点代表一个区间，每个子节点代表父节点区间的一半。初始版本左儿子下标是父亲下标的两倍，右儿子下标为父亲下标的两倍+1。

操作

建树

自上而下，同时更新父节点。

```
const int maxn = 100005;
int a[maxn], t[maxn<<2];           //a为原来区间 t为线段树

void Pushup(int k){                //更新函数，这里是实现最大值，同理可以变成，最小值，区间和等
    t[k] = max(t[k<<1], t[k<<1|1]);
}

//递归方式建树 build(1,1,n);
void build(int k, int l, int r){    //k为当前需要建立的结点 l为当前需要建立区间的左端点 r则为右端点
    if(l == r)                    //左端点等于右端点，即为叶子节点，直接赋值即可
        t[k] = a[l];
    else{
        int m = l + ((r-l)>>1);    //m则为中间点，左儿子的结点区间为[l,m]，右儿子的结点区间为[m+1,r]
        build(k<<1, l, m);        //递归构造左儿子结点
        build(k<<1|1, m+1, r);    //递归构造右儿子结点
        Pushup(k);                //更新父节点
    }
}
```

```
}
```

点更新

自上而下

```
//递归方式更新 updata(p,v,l,n,l);  
void updata(int p,int v,int l,int r,int k){ //p为下标 v为要加上的值 l r为结  
点区间 k为结点下标  
    if(l == r) //左端点等于右端点,即为叶子结点,直接加上v即可  
        a[k] += v,t[k] += v; //原数组和线段树数组都得到更新  
    else{  
        int m = l + ((r-l)>>1); //m则为中间点,左儿子的结点区间为[l,m],右儿子的  
结点区间为[m+1,r]  
        if(p <= m) //如果需要更新的结点在左子树区间  
            updata(p,v,l,m,k<<1);  
        else //如果需要更新的结点在右子树区间  
            updata(p,v,m+1,r,k<<1|1);  
        Pushup(k); //更新父节点的值  
    }  
}
```

区间查询

自上而下,只统计所查询区间的子区间,在某一个递归进程中查询到则该进程结束。

```
//递归方式区间查询 query(L,R,l,n,l);  
int query(int L,int R,int l,int r,int k){ //[L,R]即为要查询的区间 l r为结点  
区间 k为结点下标  
    if(L <= l && r <= R) //如果当前结点的区间真包含于要查询的区间内,则返回结点信  
息且不需要往下递归  
        return t[k];  
    else{  
        Pushdown(k); /**每次都需要更新子树的Lazy标记*/  
        int res = -INF; //返回值变量,根据具体线段树查询的什么而自定义  
        int mid = l + ((r-l)>>1); //m则为中间点,左儿子的结点区间为[l,m],右儿  
子的结点区间为[m+1,r]  
        if(L <= m) //如果左子树和需要查询的区间交集非空  
            res = max(res, query(L,R,l,m,k<<1));  
        if(R > m) //如果右子树和需要查询的区间交集非空,注意这里不是else if 因为查  
询区间可能同时和左右区间都有交集  
            res = max(res, query(L,R,m+1,r,k<<1|1));  
        return res; //返回当前结点得到的信息  
    }  
}
```

区间更新

使用lazy_tag[]更新时只下放到最高一层的更新区间的子区间。查询时再更细致地下放。

```
void Pushdown(int k){ //更新子树的lazy值, 这里是RMQ的函数, 要实现区间和等则需要修改函数内容
    if(lazy[k]){ //如果有lazy标记
        lazy[k<<1] += lazy[k]; //更新左子树的lazy值
        lazy[k<<1|1] += lazy[k]; //更新右子树的lazy值
        t[k<<1] += lazy[k]; //左子树的最值加上lazy值
        t[k<<1|1] += lazy[k]; //右子树的最值加上lazy值
        lazy[k] = 0; //lazy值归0
    }
}

//递归更新区间 updata(L,R,v,l,n,1);
void updata(int L,int R,int v,int l,int r,int k){ // [L,R] 即为要更新的区间 [l,r]
    r为结点区间 k为结点下标
    if(L <= l && r <= R){ //如果当前结点的区间真包含于要更新的区间内
        lazy[k] += v; //懒惰标记
        t[k] += v; //最大值加上v之后, 此区间的最大值也肯定是加v
    }
    else{
        Pushdown(k); //重难点, 查询lazy标记, 更新子树
        int m = l + ((r-l)>>1);
        if(L <= m) //如果左子树和需要更新的区间交集非空
            update(L,R,v,l,m,k<<1);
        if(m < R) //如果右子树和需要更新的区间交集非空
            update(L,R,v,m+1,r,k<<1|1);
        Pushup(k); //更新父节点
    }
}
```

例题

The Trip On Abandoned Railway

```
#include<cstdio>
#include<cstring>
#define INF 1000000007
using namespace std;
int t,n,m,d,p[100005],T[500005];
long long tag[500005][3];
void push_down(int k)
{
    tag[k<<1][0]+=tag[k][0];
    tag[k<<1][1]+=tag[k][1];
}
```

```
tag[k<<1][2]+=tag[k][2];
tag[k<<1|1][0]+=tag[k][0];
tag[k<<1|1][1]+=tag[k][1];
tag[k<<1|1][2]+=tag[k][2];
tag[k][0]=tag[k][1]=tag[k][2]=0;
}
void update(int L,int R,int l,int r,int k,int startup,int quota)
{
    if(l>=L&& r<=R)
    {
        tag[k][0]++;
        tag[k][1]+=startup;
        tag[k][2]+=quota;
    }
    else
    {
        push_down(k);
        int mid=(l+r)/2;
        if(L<=mid)
            update(L,R,l,mid,k<<1,startup,quota);
        if(R>mid)
            update(L,R,mid+1,r,k<<1|1,startup,quota);
    }
}
int query(int k,int target,int l,int r)
{
    if(l==r)
    {
        int ans=(T[k]+tag[k][2]%INF+((tag[k][0]*l-tag[k][1])*d)%INF)%INF;
        T[k]=tag[k][0]=tag[k][1]=tag[k][2]=0;
        return ans;
    }
    push_down(k);
    int mid=(l+r)>>1;
    if(target<=mid)
        return query(k<<1,target,l,mid);
    return query(k<<1|1,target,mid+1,r);
}
void build(int now,int l,int r)
{
    if(l==r)
        T[now]=p[l];
    else
    {
        int mid=(l+r)>>1;
        build(now<<1,l,mid);
        build(now<<1|1,mid+1,r);
    }
}
int main()
{
```

```
scanf("%d",&t);
while(t--)
{
    memset(T,0,sizeof(T));
    memset(tag,0,sizeof(tag));
    scanf("%d%d%d",&n,&m,&d);
    for(int i=1;i<=n;i++)
        scanf("%d",&p[i]);
    build(1,1,n);
    for(int i=1,op,x,y;i<=m;i++)
    {
        scanf("%d",&op);
        if(op==1)
        {
            scanf("%d%d",&x,&y);
            update(x,n,1,n,1,x,y);
        }
        else
        {
            scanf("%d",&x);
            printf("%d\n",query(1,x,1,n));
        }
    }
}
return 0;
}
```

From: <https://wiki.cvbbacm.com/> - CVBB ACM Team

Permanent link: https://wiki.cvbbacm.com/doku.php?id=2020-2021:teams:no_morning_training:shaco:%E7%9F%A5%E8%AF%86%E7%82%B9:%E6%95%B0%E6%8D%AE%E7%BB%93%E6%9E%84:%E7%BA%BF%E6%AE%B5%E6%A0%91&rev=1590574470

Last update: 2020/05/27 18:14