可持久化线段树

主席树，学名为可持久化线段树，可以用来解决线段树存储历史状态的问题。

我们在进行单点修改后，线段树只有logn个(一条链)的节点被修改，我们可以让修改后的树与修改前的树共享节点，节省时间和空间。

在学习主席树之前，我们先引入三个前置知识：离散化、动态开点，权值线段树。

离散化

对于较大的数据范围，只要将关键点记录下来，记录下rank，就能把数据缩小到可以接受的范围，以便建立线段树或其他数据结构来解决问题。

具体步骤

（1）将所有端点加入辅助数组；
（2）按坐标从小到大排序；
（3）去重；
（4）数据离散化，用hs数组记录端点的排名而非具体数字

```
vector<int>vt;
for(int i=1;i<=n;i++){
    cin>>a[i];
    vt.push(a[i]); //加入辅助容器
}
sort(vt.begin(),vt.end());//排序
vt.erase(unique(vt.begin(),vt.end()),vt.end()); //去重
for(int i=0; i<vt.size();i++){
    hs[vt[i]]=++tot; //存储为rank
}
```

动态开点

动态开点线段树可以避免离散化。

如果权值线段树的值域较大，离散化比较麻烦，可以用动态开点的技巧。

省略了建树的步骤，而是在具体操作中加入结点。

权值线段树

线段树的叶子节点保存的是当前值的个数。

每个节点保存区间左右端点以及所在区间节点的个数。

由于值域范围通常较大，一般会配合离散化或动态开点等策略优化空间。

应用

查找一个区间的第k大的值

查询某个数的排名

查询整个数组的排序

查询前驱和后继

单点修改

```
void update(int node,int start,int end,int pos){
    if(start==end) tr[node]++;
    else{
        int mid=start+end>>1;
        if(pos<=mid) update(node<<1,start,mid,pos);
        else update(node<<1|1,mid+1,end,pos);
    }
}//tr[i]表示值为i的元素个数,pos是要查找的位置
```

查询区间中的数出现次数

```
int query(int node,int start,int end,int ql,int qr){
    if(start==ql&&end==qr) return tr[node];
    int mid=start+end>>1;
    if(qr<=mid) return query(node<<1,start,mid,ql,qr);
    else if(ql>mid) return query(node<<1|1,mid+1,end,ql,qr);
    else return query(node<<1,start,mid,ql,qr)+query(node<<1|1,mid+1,end,ql,qr);
}//对单点查询同样适用
```

查询所有数的第k大值

```
int kth(int node,int start,int end,int k){
    if(start==end) return start;
    int mid=start+end>>1;
    int s1=tr[node<<1],s2=tree[node<<1|1];
    if(k<=s2) return kth(node<<2|1,mid+1,end,k);
    else return kth(node<<1,start,mid,k-s2);
} //注意是第k大，从右边开始减，如果是第k小就减去左边
```

查询前驱(后继同)

```
int findpre(int node,int start,int end){ //找这个区间目前最大的
    if(start==end) return start; //找到直接返回
    int mid=start+end>>1;
    if(t[node<<1|1]) return findpre(node<<1|1,mid+1,end);
    return findpre(node<<1,start,mid);
}

int pre(int node,int l,int r,int pos){ //求pos的前驱
    if(r<pos){ //在最右边
        if(t[node]) return findpre(node,l,r);
        return 0;
    }
    int mid=l+r>>1,res;
    if(mid+1<pos&&t[node<<1|1]&& (res=pre(node<<1|1,mid+1,r,pos))) return res;
//在右区间寻找
    return pre(node<<1,l,mid,pos);  //在左区间寻找
}
```

主席树

学名**可持久化线段树**，以前缀和形式建立，基于动态开点的存储形式。

复杂度分析

建树O(nlogn)

询问为O(logn)

空间复杂度O(nlognlogn)。

核心思想

两棵线段树之间是可减的（每一个节点对应相减）。

存储

hjt[0]充当NULL，从1开始储存根节点

```
struct Node{
    int lc,rc,sum; //左儿子右儿子和值sum
}hjt[maxn*32] //空间一般开到32即可
int cnt,root[maxn];//内存池计数器和根节点编号
```

插入

并不改变原来的树，而是新开根节点，并向下开辟

```
void insert(inr cur,int pre.int pos,int l,int r){
    if(l==r){ //找到这个点，当前版本点值+1
        t[cur].v=t[pre].v+1;
        return;
    }
    int mid=l+r>>1;
    if(pos<=mid){
        t[cur].lc=++e;
        t[cur].rc=t[pre].rc;
        insert(t[cur].lc,t[pre].lc,pos,l,mid);
    }else{
        t[cur].rc=++e;
        t[cur].lc=t[pre].lc;
        insert(t[cur].rc,t[pre].rc,pos,mid+1,r);
    }
    t[cur].v=t[t[cur].lc].v+t[t[cur].rc].v; //pushup
}
```

询问操作

本质上与权值线段树相同，只需要在区间上作差。

```
int query(int l,int r,int x,int y,int k){
    if (l == r){
        return l;
    }
    int mid=(l+r)/2;
    int sum=tr[tr[y].l].sum-tr[tr[x].l].sum;
    if(sum >= k){
        return query(l,mid,tr[x].l,tr[y].l,k);
    }
    else{
        return query(mid+1,r,tr[x].r,tr[y].r,k-sum);
    }
}
```

区间第K小

luoguP3834 【模板】可持久化线段树 2（主席树）

```
//#pragma GCC optimize("Ofast", "inline", "-ffast-math")
//#pragma GCC target("avx,sse2,sse3,sse4,mmx")
#include<bits/stdc++.h>
#define inf 0x3f3f3f3f
typedef long long ll;
using namespace std;
const int N=2e5+7;
const int mod=1e9+7;
int read(){ int x=0,f=1;char ch=getchar();while(ch<'0'||ch>'9'){if(ch=='-')
f=f*-1;ch=getchar();}while(ch>='0'&&ch<='9'){x=x*10+ch-'0';ch=getchar();}return
x*f;}

int n,m,cnt=0,rt[N],a[N]; //a[i]为原始序列,rt[i]为第i版本主席树

struct node{
    int l,r,sum;  //树的左右端点和元素个数
}tr[N<<5];

vector<int>vt;  //辅助容器

int getid(int x){ //返回每个数的rank
    return lower_bound(vt.begin(),vt.end(),x)-vt.begin()+1;
}

void update(int &x,int y,int l,int r,int pos){
    tr[++cnt]=tr[y];

    tr[cnt].sum++;  //区域元素+1
    x=cnt; //将原来的树地址指向当前树
    if(l==r) return; //叶子结点返回
    int mid=((l+r)>>1);
    if(pos<=mid) update(tr[x].l,tr[y].l,l,mid,pos); //左子树插入
    else update(tr[x].r,tr[y].r,mid+1,r,pos); //右子树插入
}

int query(int x,int y,int l,int r,int k){ //查询[l,r]区间第k大
    if(l==r) return l;
    int mid=((l+r)>>1);
    int sum=tr[tr[y].l].sum-tr[tr[x].l].sum; //左子树相减，其差值为两版本左边相差多少个
数
    if(k<=sum) return query(tr[x].l,tr[y].l,l,mid,k); //结果大于等于k，询问左子树
    else return query(tr[x].r,tr[y].r,mid+1,r,k-sum); //否则找右子树第k-sum小的数
}

signed main(){
    n=read();m=read();
    for(int i=1;i<=n;i++){
        a[i]=read();
        vt.push_back(a[i]); //辅助容器用于离散化
    }
    sort(vt.begin(),vt.end()); //排序
    vt.erase(unique(vt.begin(),vt.end()),vt.end()); //去重
    for(int i=1;i<=n;i++){ //每次插入都从根节点开始建一棵新树
```

```
        update(rt[i],rt[i-1],1,n,getid(a[i]));
    }
    for(int i=1,x,y,k;i<=m;i++){ //第y和x-1棵树做差，就能查出[x,y]区间第k小
        x=read();y=read();k=read();
        printf("%d\n",vt[query(rt[x-1],rt[y],1,n,k)-1]);
    }
    return 0;
}
```

主席树带修改

对于一般kth的问题，主席树求的是前缀和，这样就可以通过查分左右端点版本的主席树来求出；新增修改操作，只需要把前缀和交给树状数组维护，主席树维护位置即可。（树套树）

luoguP2617 Dynamic Rankings

支持两种操作：

- `Q l r k` 表示查询下标在区间 $[l,r]$ 中的第 k 小的数
- `C x y` 表示将a_x改为 y

```
#include<bits/stdc++.h>
using namespace std;
const int N=1e5+7;

struct Segment_Tree{int v,ls,rs;}tr[N*400];
struct Op{bool b;int l,r,k;int pos,t;}q[N]; //离线处理
int n,m,a[N],rt[N],ft[N<<1],len,tot,tmp[2][20],cnt[2];
char op;

#define mid ((l+r)>>1)
#define lson tr[p].ls,l,mid
#define rson tr[p].rs,mid+1,r
#define lb(x) (x&(-x))

inline void Modify(int &p,int l,int r,int pos,int val){
    if(!p) p=++tot;
    tr[p].v+=val;
    if(l==r) return;
    if(pos<=mid) Modify(lson,pos,val);
    else Modify(rson,pos,val);
}

inline void pre_Modify(int x,int val){
    int k=lower_bound(ft+1,ft+1+len,a[x])-ft;
    for(int i=x;i<=n;i+=lb(i)) Modify(rt[i],1,len,k,val); //处理出需要修改的log棵主席树
}

int Query(int l,int r,int k){
    if(l==r) return l;
    int sum=0;
    for(int i=1;i<=cnt[1];++i) sum+=tr[tr[tmp[1][i]].ls].v;
    for(int i=1;i<=cnt[0];++i) sum-=tr[tr[tmp[0][i]].ls].v;
```

```
    if(k<=sum){
        for(int i=1;i<=cnt[1];++i) tmp[1][i]=tr[tmp[1][i]].ls;
        for(int i=1;i<=cnt[0];++i) tmp[0][i]=tr[tmp[0][i]].ls;
        return Query(l,mid,k);
    }else{
        for(int i=1;i<=cnt[1];++i) tmp[1][i]=tr[tmp[1][i]].rs;
        for(int i=1;i<=cnt[0];++i) tmp[0][i]=tr[tmp[0][i]].rs;
        return Query(mid+1,r,k-sum);
    }
}

int pre_Query(int l,int r,int k){
    memset(tmp,0,sizeof(tmp));
    cnt[0]=cnt[1]=0;
    for(int i=r;i;i-=lb(i)) tmp[1][++cnt[1]]=rt[i];
    for(int i=l-1;i;i-=lb(i)) tmp[0][++cnt[0]]=rt[i];
    return Query(1,len,k);
}

signed main(){
    ios::sync_with_stdio(0);
    cin.tie(0);cout.tie(0);
    cin>>n>>m;
    for(int i=1;i<=n;i++) cin>>a[i],ft[++len]=a[i];
    for(int i=1;i<=m;i++){
        cin>>op;
        q[i].b=(op=='Q');
        if(q[i].b) cin>>q[i].l>>q[i].r>>q[i].k;
        else cin>>q[i].pos>>q[i].t,ft[++len]=q[i].t;
    }
    sort(ft+1,ft+1+len);
    len=unique(ft+1,ft+1+len)-ft-1; //离散化——排序+去重
    for(int i=1;i<=n;i++) pre_Modify(i,1);
    for(int i=1;i<=m;i++){
        if(q[i].b) cout<<ft[pre_Query(q[i].l,q[i].r,q[i].k)]<<"\n";
        else{
            pre_Modify(q[i].pos,-1);
            a[q[i].pos]=q[i].t;
            pre_Modify(q[i].pos,1);
        }
    }
    return 0;
}
```

主席树+树链剖分

luoguP3313 [SDOI2014]旅行

经典题，支持以下操作：

- 树上单点修改权值
- 树上单点修改颜色
- 树上某一颜色路径和
- 树上某一颜色最大值

```
#include<bits/stdc++.h>
#define int long long
using namespace std;
const int N=1e5+7;

vector<int>G[N];
inline void addedge(int u,int v){
    G[u].push_back(v);
    G[v].push_back(u);
}

int sz[N],fa[N],son[N],dep[N],dfn[N],idfn[N],bel[N],idx;

inline void dfs1(int x){
    sz[x]=1;
    for(int i=0;i<G[x].size();i++){
        int to=G[x][i];
        if(to==fa[x]) continue;
        fa[to]=x;
        dep[to]=dep[x]+1;
        dfs1(to);
        sz[x]+=sz[to];
        if(sz[son[x]]<sz[to]) son[x]=to;
    }
}

inline void dfs2(int x,int tp){
    dfn[x]=++idx;
    idfn[idx]=x;
    bel[x]=tp;
    if(son[x]) dfs2(son[x],tp);
    for(int i=0;i<G[x].size();i++){
        int to=G[x][i];
        if(to==son[x]||to==fa[x]) continue;
        dfs2(to,to);
    }
}

#define ls tr[p].lson
#define rs tr[p].rson
#define MID int mid=l+r>>1

int rt[N],SZ;
struct node{
    int lson,rson,mx,sum;
}tr[N*24];

inline void pushup(int p){
```

```
    tr[p].sum=tr[ls].sum+tr[rs].sum;
    tr[p].mx=max(tr[ls].mx,tr[rs].mx);
}

inline void build(int &p,int l,int r,int pos,int val){
    if(!p) p=++SZ;
    if(l==r){
        tr[p].mx=tr[p].sum=val;
        return;
    }
    MID;
    if(pos<=mid) build(ls,l,mid,pos,val);
    else build(rs,mid+1,r,pos,val);
    pushup(p);
}

inline void del(int &p,int l,int r,int pos){
    if(l==r){
        tr[p].mx=tr[p].sum=p=0;
        return;
    }
    MID;
    if(pos<=mid) del(ls,l,mid,pos);
    else del(rs,mid+1,r,pos);
    pushup(p);
    if(!ls&&!rs) tr[p].mx=tr[p].sum=p=0;
}

inline int qs(int &p,int l,int r,int ql,int qr){
    if(!p) return 0;
    if(ql<=l&&r<=qr) return tr[p].sum;
    MID;int res=0;
    if(ql<=mid) res=qs(ls,l,mid,ql,qr);
    if(qr>mid) res+=qs(rs,mid+1,r,ql,qr);
    return res;
}

inline int qm(int &p,int l,int r,int ql,int qr){
    if(!p) return 0;
    if(ql<=l&&r<=qr) return tr[p].mx;
    MID;int res=0;
    if(ql<=mid) res=qm(ls,l,mid,ql,qr);
    if(qr>mid) res=max(res,qm(rs,mid+1,r,ql,qr));
    return res;
}

int n,m,x,y,res,rd,tmp,val[N],col[N];
string op;

signed main(){
    ios::sync_with_stdio(0);
    cin.tie(0);cout.tie(0);
    cin>>n>>m;
    for(int i=1;i<=n;i++) cin>>val[i]>>col[i];
    for(int i=1;i<n;i++){
        cin>>x>>y;
        addedge(x,rd=y); //随机根节点
    }
```

```
    dfs1(rd);
    dfs2(rd,rd);
    for(int i=1;i<=n;i++) build(rt[col[idfn[i]]],1,n,i,val[idfn[i]]);
    for(int i=1;i<=m;i++){
        cin>>op>>x>>y;
        if(op=="CC"){
            del(rt[col[x]],1,n,dfn[x]);
            col[x]=y;
            build(rt[col[x]],1,n,dfn[x],val[x]);
        }else if(op=="CW"){
            del(rt[col[x]],1,n,dfn[x]);
            val[x]=y;
            build(rt[col[x]],1,n,dfn[x],val[x]);
        }else if(op=="QS"){
            res=0;
            tmp=col[x];
            while(bel[x]!=bel[y]){
                if(dep[bel[x]]<dep[bel[y]]) swap(x,y);
                res+=qs(rt[tmp],1,n,dfn[bel[x]],dfn[x]);
                x=fa[bel[x]];
            }
            if(dep[x]<dep[y]) swap(x,y);
            res+=qs(rt[tmp],1,n,dfn[y],dfn[x]);
            cout<<res<<"\n";
        }else{
            res=0;
            tmp=col[x];
            while(bel[x]!=bel[y]){
                if(dep[bel[x]]<dep[bel[y]]) swap(x,y);
                res=max(res,qm(rt[tmp],1,n,dfn[bel[x]],dfn[x]));
                x=fa[bel[x]];
            }
            if(dep[x]<dep[y]) swap(x,y);
            res=max(res,qm(rt[tmp],1,n,dfn[y],dfn[x]));
            cout<<res<<"\n";
        }
    }
    return 0;
}
```

参考资料

https://www.cnblogs.com/young-children/p/11787490.html

https://www.cnblogs.com/young-children/p/11787493.html

https://blog.csdn.net/ModestCoder_/article/details/90107874

https://blog.csdn.net/a1351937368/article/details/78884465

https://www.cnblogs.com/hanruyun/p/9916299.html