## 作业五

### 数据结构的节点与结构

#### 节点如下

struct node{

int l,r;//区间[l,r]

int add;//区间的延时标记

int sum;//区间和

int mx; //区间最大值

int mn; //区间最小值

}tree[MAXN<<2];//开到4倍多的空间

#### 结构

线段树，即数组模拟二叉树，对于每一个节点R ,左子节点为 2\*R (一般写作R<<1)右子节点为 2\*R+1（一般写作R<<1|1）。

将每个区间[L,R]分解成[L,M]和[M+1,R] (其中M=(L+R)/2 这里的除法是整数除法，即对结果下取整)直到 L==R 为止。

### 建立线段树

void build(int l,int r,int index){

tree[index].l = l;

tree[index].r = r;

tree[index].add = 0;//刚开始清0

if(l == r){

scanf("%d",&tree[index].sum);

tree[index].mn = tree[index].mx = tree[index].sum;

return ;

}

int mid = (l+r)>>1;

//递归的构建左右子树

build(l,mid,index<<1);

build(mid+1,r,index<<1|1);

//建立好左右子树后，更新index节点中的内容

pushup(index);

}

void pushup(int index){

//更新区间和

tree[index].sum = tree[index<<1].sum+tree[index<<1|1].sum;

//更新区间最大值

tree[index].mx = max(tree[index<<1].mx,tree[index<<1|1].mx);

//更新区间最小值

tree[index].mn = min(tree[index<<1].mn,tree[index<<1|1].mn);

}

### 更新某一节点的值

//更新区间[l,r]的值，在其中每个数上加val,当l==r时为单点更新

//在区间index的[tree[index].l,tree[index].r]的每一个数上增加val

void updata(int l,int r,int index,int val){

//如果区间[l,r]将index的[tree[index].l,tree[index].r]包含

if(l <= tree[index].l && r >= tree[index].r){

//在原来的值的基础上加上val,因为该区间有tree[index].r-tree[index].l+1

//个数，所以区间和 以及 最值为：

tree[index].sum += (tree[index].r-tree[index].l+1)\*val;

tree[index].mn += val;

tree[index].mx += val;

tree[index].add += val;//延时标记

return ;

}

//处理之前的懒惰标记

pushdown(index);

int mid = (tree[index].l+tree[index].r)>>1;

if(l <= mid){

updata(l,r,index<<1,val);

}

if(r > mid){

updata(l,r,index<<1|1,val);

}

pushup(index);

}

//处理懒惰标记

void pushdown(int index){

//说明该区间之前更新过

//要想更新该区间下面的子区间，就要把上次更新该区间的值向下更新

if(tree[index].add > 0){

//在原来的值的基础上加上val

tree[index<<1].sum += (tree[index<<1].r-tree[index<<1].l+1)\*tree[index].add;

tree[index<<1|1].sum +=(tree[index<<1|1].r-tree[index<<1|1].l+1)\*tree[index].add;

tree[index<<1].mx += tree[index].add;

tree[index<<1|1].mx += tree[index].add;

tree[index<<1].mn += tree[index].add;

tree[index<<1|1].mn += tree[index].add;

tree[index<<1].add += tree[index].add;

tree[index<<1|1].add += tree[index].add;

tree[index].add = 0;

}

}

### 查询与求和

int query(int l,int r,int index){

if(l <= tree[index].l && r >= tree[index].r){

//return tree[index].sum;

return tree[index].mx;

//return tree[index].mn;

}

pushdown(index);

int mid = (tree[index].l+tree[index].r)>>1;

int ans = 0;

int Max = 0;

int Min = inf;

if(l <= mid){

ans += query(l,r,index<<1);

Max = max(query(l,r,index<<1),Max);

Min = min(query(l,r,index<<1),Min);

}

if(r > mid){

ans += query(l,r,index<<1|1);

Max = max(query(l,r,index<<1|1),Max);

Min = min(query(l,r,index<<1|1),Min);

}

//返回求和

//return ans;

//返回最大值

return Max;

//返回最小值

//return Min;

}