1. 树状数组
2. 线段树
3. 主席树
4. 并查集
5. Splay
6. Lct
7. 莫队
8. 分块
9. 树分块及树上莫队
10. 可并堆
11. 扫描线求矩形周长的核心思想
12. KD树

树状数组：

int n;

int a[1005],c[1005]; //对应原数组和树状数组

int lowbit(int x){return x&(-x);}

void updata(int i,int k)

{ //在i位置加上k

while(i <= n){

c[i] += k;

i += lowbit(i);

}

}

int getsum(int i)

{ //求A[1 - i]的和

int res = 0;

while(i > 0){

res += c[i];

i -= lowbit(i);

}

return res;

}

线段树：

int query(int now,int l,int r,int x,int y)

{

if(x<=l&&r<=y){return c[now].sum;}

int m=(l+r)/2;

if(y<=m)return query(c[now].son[0],l,m,x,y);

else if(x>=m+1)return query(c[now].son[1],m+1,r,x,y);

else return query(c[now].son[0],l,m,x,m)+query(c[now].son[1],m+1,r,m+1,y);

}

其他注意pushdown和pushup标记即可。

主席树：

void update(int now1,int &now,int l,int r,int x,int v)

{

now=++np;

c[now]=c[now1];

if(l==r){c[now].minv=v;return;}

int m=(l+r)/2;

if(x<=m)update(c[now1].son[0],c[now].son[0],l,m,x,v);

else update(c[now1].son[1],c[now].son[1],m+1,r,x,v);

pushup(now);

}

注意版本号一定要连贯过去就行了

常规主席树带区间修改版：  
void pushdown(int now,int l,int r)

{

int m=(l+r)/2;

if(c[now].mark!=0)

{

c[++np]=c[c[now].son[0]];

c[now].son[0]=np;

c[++np]=c[c[now].son[1]];

c[now].son[1]=np;

c[c[now].son[0]].mark+=c[now].mark;

c[c[now].son[0]].sum+=(long long)c[now].mark\*(m-l+1);

c[c[now].son[1]].mark+=c[now].mark;

c[c[now].son[1]].sum+=(long long)c[now].mark\*(r-m);

c[now].mark=0;

}

}

void update(int now1,int &now,int l,int r,int x,int y,int d)

{

if(l!=r)pushdown(now1,l,r);

now=++np;

c[now]=c[now1];

if(x<=l&&r<=y)

{

c[now].mark+=d;

c[now].sum+=(long long)(r-l+1)\*d;

return;

}

int m=(l+r)/2;

if(y<=m)update(c[now1].son[0],c[now].son[0],l,m,x,y,d);

else if(x>m)update(c[now1].son[1],c[now].son[1],m+1,r,x,y,d);

else

{

update(c[now1].son[0],c[now].son[0],l,m,x,m,d);

update(c[now1].son[1],c[now].son[1],m+1,r,m+1,y,d);

}

pushup(now);

}

并查集

int find(int x)

{

if(pa[x]==x) return x;

return pa[x]=find(pa[x]);

}

Splay:

核心：

void rot(int x)//核心，旋转

{

int y=c[x].fa,z=c[y].fa;

int d=(c[y].son[0]==x);

c[y].son[d^1]=c[x].son[d];

c[c[x].son[d]].fa=y;

c[x].fa=z;

if(z)c[z].son[c[z].son[1]==y]=x;

c[y].fa=x;

c[x].son[d]=y;

pushup(y);

pushup(x);

}

void splay(int x,int anc)//核心，扭上去。

{

pushdown(x);

while(c[x].fa!=anc)

{

int y=c[x].fa,z=c[y].fa;

if(z!=anc)pushdown(z);

pushdown(y);

pushdown(x);

if(z!=anc)

{

if((c[y].son[1]==x)==(c[z].son[1]==y))rot(y);

else rot(x);

}

rot(x);

}

pushup(x);

if(anc==0)rt=x;

}

int kth(int now,int x)

{

while(now)

{

pushdown(now);

int t=1;

if(c[now].son[0])t+=c[c[now].son[0]].v.sz;

if(t==x)break;

if(t<x)x-=t,now=c[now].son[1];

else now=c[now].son[0];

}

//splay(now,0);因为这里每次查找都是在makeab里面所以不需要

return now;

}

int makeab(int x,int y)//构造区间a,b,并返回作为该区间的父节点它的左节点代表该区间

{

if(x==1)

{

y=kth(rt,y+1);

splay(y,0);

return y;

}

x=kth(rt,x-1);

y=kth(rt,y+1);

splay(x,0);

splay(y,x);

return y;

}

询问：abc为结构体名称

abc query(int x,int y)

{

int k=makeab(x,y);

k=c[k].son[0];

return c[k].v;

}

翻转

void rev(int x,int y)

{

int k=makeab(x,y);

k=c[k].son[0];

c[k].mark^=1;

swap(c[k].son[0],c[k].son[1]);

pushup(k);

splay(k,0);

}

插入：

void ins(int x,int y)

{

int k=makeab(x+1,x);//使左儿子为空

int k2=build(y);

c[k].son[0]=k2;

c[k2].fa=k;

splay(k2,0);

}

删除：

void del(int x,int y)

{

int k=makeab(x,y);

c[c[k].son[0]].fa=0;

c[k].son[0]=0;

splay(k,0);

}

Lct:

//lct

//普通lct

struct mydata

{

int son[2];

int pa;

int w;

int sum;

int rev;

};

struct lct

{

mydata c[maxn];

int rt,np;

int isrt(int x){return c[c[x].pa].son[0]!=x&&c[c[x].pa].son[1]!=x;}

void pushup(int now)

{

c[now].sum=w;

int lc=c[now].son[0],rc=c[now].son[1];

if(lc)c[now].sum+=c[lc].sum;

if(rc)c[now].sum+=c[rc].sum;

}

void pushdown(int now)

{

int lc=c[now].son[0],rc=c[now].son[1];

if(c[now].rev)

{

if(lc)

{

swap(c[lc].son[0],c[lc].son[1]);

c[lc].rev^=1;

}

if(rc)

{

swap(c[rc].son[0],c[rc].son[1]);

c[rc].rev^=1;

}

c[now].rev=0;

}

}

void rot(int x)

{

int y=c[x].pa,z=c[y].pa;

int d=(c[y].son[0]==x);

pushdown(y);

pushdown(x);

c[y].son[d^1]=c[x].son[d];

c[c[x].son[d]].pa=y;

c[x].pa=z;

if(!isroot(y))c[z].son[c[z].son[1]==y]=x;

c[y].pa=x;

c[x].son[d]=y;

pushup(y);

pushup(x);

}

void splay(int x)

{

pushdown(x);

while(!isrt(x))

{

int y=c[x].pa;

int z=c[y].pa;

if(!isrt(y))

{

if((c[z].son[1]==y)==(c[y].son[1]==x))rot(y);

else rot(x);

}

rot(x);

}

}

int acc(int x)//核心操作,

{

int y=0;

for(y=0;x;y=x,x=c[x].pa)

{

splay(x);

c[x].son[1]=y;

pushup(x);

}

return y;

}

void mroot(int x)//换根，很多询问通过把链上一个端点移到根上，另一个点acc完成

{

acc(x);

splay(x);

swap(c[x].son[0],c[x].son[1]);

c[x].rev^=1;

}

void link(int x,int y)//连接

{

mroot(x);

c[x].pa=y;

}

void cut(int x,int y)//切断

{

mroot(x);

acc(y);

splay(y);

c[c[y].son[0]].pa=0;

c[y].son[0]=0;

pushup(y);

}

}

//维护子树信息的lct

//pushup时更新x的总信息为：x实儿子的总信息+x虚儿子的信息+x本身的信息。

struct mydata

{

int son[2];

int tsz;//总信息（子树大小）

int sz;//虚儿子信息（子树除了链以外的大小）

int rev;

int pa;

};

struct lct//其他与之前相同

{

void pushup(int now)

{

if(!now)return;

c[now].tsz=c[now].sz+1;

int lc=c[now].son[0],rc=c[now].son[1];

if(lc)c[now].tsz+=c[lc].tsz;

if(rc)c[now].tsz+=c[rc].tsz;

}

int acc(int x)

{

int y=0;

for(y=0;x;y=x,x=c[x].pa)

{

splay(x);

c[x].sz+=c[c[x].son[1]].tsz-c[y].tsz;

c[x].son[1]=y;

pushup(x);

}

return y;

}

void link(int x,int y)

{

mroot(x);

mroot(y);

c[x].pa=y;

c[y].sz+=c[x].tsz;

pushup(y);

}

}

//lct 处理边权，把边弄成一个点，搞成点权。

莫队：

q[i].id=(q[i].l-1)/sn+1;//先比左边块编号，在比右边位置

q[i].i=i;

for(int i=1;i<=m;i++)

{

if(q[i].l==q[i].r)//l==r算答案

continue;

for(;q[i].l<l;l--)upd(tmp,l-1,1);//移动更新

for(;q[i].r>r;r++)upd(tmp,r+1,1);

for(;q[i].r<r;r--)upd(tmp,r,-1);

for(;q[i].l>l;l++)upd(tmp,l,-1);

//更新答案

}

分块：

int sn,tn;//sn为块长，tn为块数。

sn=sqrt(n+0.5);

tn=n/sn;

int c[205][205],f[205][40005];

//c[i][ii]表示从i块到ii块，f[i][ii]表示1到i

void ready()//预处理框架

{

for(int i=1;i<=tn;i++)

for(int ii=i;ii<=tn;ii++)

for(int i3=sn\*(ii-1)+1;i3<=sn\*ii;i3++)

for(int i=1;i<=tn;i++)

for(int ii=sn\*(i-1)+1;ii<=sn\*i;ii++)

}

int query(int l,int r)

{

int wl=(l-1)/sn+1,wr=(r-1)/sn+1;//求出块编号

int md=0,mi=0;

if(wr-wl>1)//处理位于不同块

{

//处理连续的部分

for(int i=l;i<=wl\*sn;i++)//暴力枚举左边部分

for(int i=(wr-1)\*sn+1;i<=r;i++)//暴力枚举右边部分

return mi;

}

else return query2(l,r);//暴力处理位于同一块

}

可并堆：

struct mydata

{

int son[2];

int w;

};

struct myheap

{

int rt[105],np;//这个结构体里有很多个堆

mydata c[maxn];

int merge(int a,int b)//合并

{

if(a==0||b==0)return a+b;

if(c[a].w>c[b].w)swap(a,b);

c[a].son[1]=merge(c[a].son[1],b);

swap(c[a].son[1],c[a].son[0]);

return a;

}

int push(int a,int x)//插入

{

c[++np].w=x;

return merge(a,np);

}

int pop(int a)//弹出

{

return merge(c[a].son[0],c[a].son[1]);

}

int top(int a)//求top

{

return c[a].w;

}

}q;

扫描线：

void pushup(int now,int l,int r)

{

if(c[now].sum)

{

c[now].len=r-l;

}

else

{

if(r==l+1)c[now].len=0;

else c[now].len=c[c[now].son[0]].len+c[c[now].son[1]].len;

}

}

void modify(int now,int l,int r,int x,int y,int d)

{

if(x<=l&&r<=y)

{

c[now].sum+=d;

pushup(now,l,r);

return;

}

int m=(l+r)/2;

if(y<=m)modify(c[now].son[0],l,m,x,y,d);

else if(x>=m)modify(c[now].son[1],m,r,x,y,d);

else

{

modify(c[now].son[0],l,m,x,m,d);

modify(c[now].son[1],m,r,m,y,d);

}

pushup(now,l,r);

}

核心就是遇到线就区间加，然后判断区间是否大于1，大于1就给len赋区间长的值，否则赋区间0.

Kd树

int D,inf=2000000000,ans;//KD表示有几维，D表示目前比较哪一维

struct point

{

int d[KD];

friend bool operator <(point one,point two)

{

return one.d[D]<two.d[D];

}

}a[maxn];

struct kdtree

{

int son[maxn][2];

point p[maxn],mi[maxn],mx[maxn];//mx表示该区间的最大值，mi表示最小值

int rt,np;

int newnode(point now)

{

++np;

son[np][1]=son[np][0]=0;

p[np]=mi[np]=mx[np]=now;

return np;

}

void pushup(int now)

{

mi[now]=mx[now]=p[now];

for(int i=0;i<KD;i++)

{

if(son[now][0])

{

int t=son[now][0];

mi[now].d[i]=min(mi[now].d[i],mi[t].d[i]);

mx[now].d[i]=max(mx[now].d[i],mx[t].d[i]);

}

if(son[now][1])

{

int t=son[now][1];

mi[now].d[i]=min(mi[now].d[i],mi[t].d[i]);

mx[now].d[i]=max(mx[now].d[i],mx[t].d[i]);

}

}

}

void build(int &now,int l,int r,int k)

{

if(l>r){now=0;return;}

int m=(l+r)/2;

D=k;nth\_element(a+l,a+m,a+r+1);//找出当前位于中间的一个元素。

now=newnode(a[m]);

build(son[now][0],l,m-1,(k+1)%KD);

build(son[now][1],m+1,r,(k+1)%KD);

pushup(now);

}

void ins(int &now,point ap,int k)

{

if(now==0)

{

now=newnode(ap);

return;

}

D=k;//简单的类似搜索树的插入

if(ap<p[now])ins(son[now][0],ap,(k+1)%KD);

else ins(son[now][1],ap,(k+1)%KD);

pushup(now);

}

int dist(point x,point y)//询问k维的距离

{

int tmp=0;

for(int i=0;i<KD;i++)tmp+=abs(x.d[i]-y.d[i]);

return tmp;

}

int nearmi(point ap,int x)//贪心减枝，看那个区间离询问点的距离近。

{

if(!x)return inf;

int tmp=0;

for(int i=0;i<KD;i++)

{

//基本思路就是看在这个区间的左边和右边分别算，到时候这个肯定是要推的。

if(ap.d[i]<mi[x].d[i])tmp+=mi[x].d[i]-ap.d[i];

if(ap.d[i]>mx[x].d[i])tmp+=ap.d[i]-mx[x].d[i];

}

return tmp;

}

void qmin(int now,point ap)

{

if(!now)return;

ans=min(ans,dist(p[now],ap));

int t1=nearmi(ap,son[now][0]);

int t2=nearmi(ap,son[now][1]);//先选近的，再选远的

if(t1<=t2)

{

if(t1<ans)qmin(son[now][0],ap);//减枝

if(t2<ans)qmin(son[now][1],ap);

}

else

{

if(t2<ans)qmin(son[now][1],ap);

if(t1<ans)qmin(son[now][0],ap);

}

}

int nearmx(point ap,int x)

{

if(!x)return -inf;

int tmp=0;

for(int i=0;i<KD;i++)

{

//算最远距离，基本就是算可能的最远距离，每次贪心的取最远。

if(ap.d[i]<mi[x].d[i])tmp+=mx[x].d[i]-ap.d[i];

else if(ap.d[i]>mx[x].d[i])tmp+=ap.d[i]-mi[x].d[i];

else tmp+=max(mx[x].d[i]-ap.d[i],ap.d[i]-mi[x].d[i]);

}

return tmp;

}

void qmax(int now,point ap)

{

if(!now)return;

//单点询问基本思路算左右然后贪心的走

ans=max(ans,dist(ap,p[now]));

int t1=nearmx(ap,son[now][0]);

int t2=nearmx(ap,son[now][1]);

if(t1>=t2)

{

//注意若没有ans优就直接退掉即可。

if(t1>ans)qmax(son[now][0],ap);

if(t2>ans)qmax(son[now][1],ap);

}

else

{

if(t2>ans)qmax(son[now][1],ap);

if(t1>ans)qmax(son[now][0],ap);

}

}

}c;