李超线段树

算法简介

一个支持动态加入折线段和查询某个区间内所有折线段最高/最低点的数据结构。

算法模板

全局修改单点查询

洛谷p4254

题意

支持以下两种操作：

1. 加入直线 $y=kx+b$
2. 询问当前所有直线在 x_0 处的最大值

题解

考虑线段树维护坐标轴，每个区间维护一条直线。当一个区间加入一条新直线时，若该区间原来没有直线，则直接用新直线覆盖。

否则，考虑新旧直线的优势长度，这里一条直线的优势长度定义该直线作为区间更优解的长度。

通过比较区间中点的取值可以得到优势长度更大的直线，然后将该区间用优势长度更大的直线的线段覆盖。

另外，优势长度更大的直线一定是整个左区间或整个右区间的更优解，而在另一个区间新旧直线可能各自占用一部分长度的更优解。

于是需要将被取代的直线下放到两个线段自占用一部分长度的更优解的区间来更新答案。

对于查询操作，类似懒标记永久化的线段树，递归过程中不停更新最优解即可。时间复杂度 $O(q\log n)$

```
const int MAXN=5e4+5,MAXM=1e5+5;
double slope[MAXM],b[MAXM];
int tag[MAXN<<2];
double caly(int v,int x){
    return slope[v]*(x-1)+b[v];
}
void update(int k,int lef,int rig,int v){
    if(!tag[k]){
        tag[k]=v;
        return;
    }
```

```
    int mid=lef+rig>>1;
    double y1=caly(v,mid),y2=caly(tag[k],mid);
    if(lef==rig){
        tag[k]=y1>y2?v:tag[k];
        return;
    }
    if(y1>y2){
        if(slope[v]>slope[tag[k]])
        update(k<<1,lef,mid,tag[k]);
        else if(slope[v]<slope[tag[k]])
        update(k<<1|1,mid+1,rig,tag[k]);
        tag[k]=v;
    }
    else{
        if(slope[v]<slope[tag[k]])
        update(k<<1,lef,mid,v);
        else if(slope[v]>slope[tag[k]])
        update(k<<1|1,mid+1,rig,v);
    }
}
double query(int k,int lef,int rig,int pos){
    if(lef==rig)
    return caly(tag[k],pos);
    int mid=lef+rig>>1;
    if(mid>=pos)
    return max(query(k<<1,lef,mid,pos),caly(tag[k],pos));
    else
    return max(query(k<<1|1,mid+1,rig,pos),caly(tag[k],pos));
}
char opt[100];
int main()
{
    int n=5e4,q=read_int(),m=0;
    while(q--){
        scanf("%s",opt);
        if(opt[0]=='P'){
            ++m;
            scanf("%lf%lf",&b[m],&slope[m]);
            update(1,1,n,m);
        }
        else
        enter((int)query(1,1,n,read_int())/100);
    }
    return 0;
}
```

区间修改单点查询

洛谷p4097

题意

支持以下两种操作：

1. 加入线段 $(x_0,y_0)\to (x_1,y_1)$
2. 询问所有与 $x=x_0$ 直线相交的线段中纵坐标最大的线段编号

题解

将线段拆成 $O(\log n)$ 个线段树上的区间，然后套用全局修改算法即可，于是修改操作时间复杂度变为 $O\left(q\log^2 n\right)$

另外，对于斜率不存在的线段，可以将 $(x,y_0)\to (x,y_1)$ 转化为 $(x,\max(y_0,y_1))$

```
const int MAXN=4e4+5,MAXM=1e5+5;
const double eps=1e-8;
double slope[MAXM],b[MAXM];
int lef[MAXN<<2],rig[MAXN<<2],tag[MAXN<<2];
double caly(int v,int x){
    return slope[v]*x+b[v];
}
void build(int k,int L,int R){
    lef[k]=L,rig[k]=R;
    if(L==R)return;
    int M=L+R>>1;
    build(k<<1,L,M);
    build(k<<1|1,M+1,R);
}
void update(int k,int v){
    if(!tag[k]){
        tag[k]=v;
        return;
    }
    int mid=lef[k]+rig[k]>>1;
    double y1=caly(v,mid),y2=caly(tag[k],mid);
    if(lef[k]==rig[k]){
        tag[k]=(y1>y2+eps)?v:tag[k];
        return;
    }
    if(y1>y2+eps){
        if(slope[v]>slope[tag[k]])
        update(k<<1,tag[k]);
        else if(slope[v]<slope[tag[k]])
        update(k<<1|1,tag[k]);
        tag[k]=v;
    }
    else{
        if(slope[v]<slope[tag[k]])
        update(k<<1,v);
```

```
        else if(slope[v]>slope[tag[k]])
        update(k<<1|1,v);
    }
}
void update(int k,int L,int R,int v){
    if(L<=lef[k]&&rig[k]<=R)
    update(k,v);
    else{
        int mid=lef[k]+rig[k]>>1;
        if(mid>=L)
        update(k<<1,L,R,v);
        if(mid<R)
        update(k<<1|1,L,R,v);
    }
}
pair<double,int> get_s(pair<double,int> a,pair<double,int> b){
    if(fabs(a.first-b.first)<eps)
    return make_pair(a.first,min(a.second,b.second));
    else if(a.first>b.first)
    return a;
    else
    return b;
}
pair<double,int> query(int k,int pos){
    pair<double,int> v=make_pair(caly(tag[k],pos),tag[k]);
    if(lef[k]==rig[k])
    return v;
    int mid=lef[k]+rig[k]>>1;
    if(mid>=pos)
    return get_s(query(k<<1,pos),v);
    else
    return get_s(query(k<<1|1,pos),v);
}
const int mod1=39989,mod2=1e9+1;
int main()
{
    int n=4e4,q=read_int(),m=0,lastans=0;
    build(1,1,n);
    b[0]=-1e12;
    while(q--){
        int opt=read_int();
        if(opt==0){
            int pos=(read_int()+lastans-1)%mod1+1;
            enter(lastans=query(1,pos).second);
        }
        else{
            int x0=read_int(),y0=read_int(),x1=read_int(),y1=read_int();
            x0=(x0+lastans-1)%mod1+1;
            y0=(y0+lastans-1)%mod2+1;
            x1=(x1+lastans-1)%mod1+1;
```

```
            y1=(y1+lastans-1)%mod2+1;
            m++;
            if(x0==x1){
                slope[m]=0;
                b[m]=max(y0,y1);
            }
            else{
                slope[m]=1.0*(y1-y0)/(x1-x0);
                b[m]=y0-slope[m]*x0;
            }
            update(1,min(x0,x1),max(x0,x1),m);
        }
    }
    return 0;
}
```

区间修改区间查询

洛谷p4069

题意

给定一棵树，支持以下两种操作：

1. $w(u)=\max(w(u),a\times \text{dis}(u,s)+b)(u\in s\to t)$
2. 询问 $\max(w(u))(u\in s\to t)$

题解

考虑树剖转化为序列问题，接下来考虑处理更新操作。设 $d(u)$ 表示根节点到 u 的距离，$p=\text{LCA}(s,t)$，分两种情况：

1. $u\in p\to s$，此时有 $\text{dis}(u,s)=d(s)-d(u)$，代入得 $w(u)=-a\times d(u)+b+a\times d(s)$
2. $u\in p\to t$，此时有 $\text{dis}(u,s)=d(s)+d(u)-2d(p)$，代入得 $w(u)=a\times d(u)+b+a\times (d(s)-2d(p))$

接下来考虑区间查询操作，需要动态维护区间最优解，同时查询过程需要考虑当前区间覆盖线段的影响。

```
const int MAXN=1e5+5;
const LL inf=1234567891234567890;
LL slope[MAXN<<1],b[MAXN<<1];
LL a[MAXN];
LL caly(int v,int x){
    return slope[v]*a[x]+b[v];
}
namespace Tree{
    int lef[MAXN<<2],rig[MAXN<<2],tag[MAXN<<2];
```

```
LL s[MAXN<<2];
void push_up(int k){
    s[k]=min(s[k],min(s[k<<1],s[k<<1|1]));
}
void build(int k,int L,int R){
    lef[k]=L,rig[k]=R,s[k]=inf;
    if(L==R)return;
    int M=L+R>>1;
    build(k<<1,L,M);
    build(k<<1|1,M+1,R);
}
void update(int k,int v){
    s[k]=min(s[k],min(caly(v,lef[k]),caly(v,rig[k])));
    if(!tag[k]){
        tag[k]=v;
        return;
    }
    int mid=lef[k]+rig[k]>>1;
    LL y1=caly(v,mid),y2=caly(tag[k],mid);
    if(lef[k]==rig[k]){
        tag[k]=(y1<y2)?v:tag[k];
        return;
    }
    if(y1<y2){
        if(slope[v]<slope[tag[k]])
        update(k<<1,tag[k]);
        else if(slope[v]>slope[tag[k]])
        update(k<<1|1,tag[k]);
        tag[k]=v;
    }
    else{
        if(slope[v]>slope[tag[k]])
        update(k<<1,v);
        else if(slope[v]<slope[tag[k]])
        update(k<<1|1,v);
    }
    push_up(k);
}
void update(int k,int L,int R,int v){
    if(L<=lef[k]&&rig[k]<=R)
    update(k,v);
    else{
        int mid=lef[k]+rig[k]>>1;
        if(mid>=L)
        update(k<<1,L,R,v);
        if(mid<R)
        update(k<<1|1,L,R,v);
        push_up(k);
    }
}
```

```
    LL query(int k,int L,int R){
        if(L<=lef[k]&&rig[k]<=R)
        return s[k];
        LL
ans=tag[k]==0?inf:min(caly(tag[k],max(lef[k],L)),caly(tag[k],min(rig[k],R))
);
        int mid=lef[k]+rig[k]>>1;
        if(mid>=L)
        ans=min(ans,query(k<<1,L,R));
        if(mid<R)
        ans=min(ans,query(k<<1|1,L,R));
        return ans;
    }
}
struct Edge{
    int to,w,next;
}edge[MAXN<<1];
int head[MAXN],edge_cnt;
void Insert(int u,int v,int w){
    edge[++edge_cnt]=Edge{v,w,head[u]};
    head[u]=edge_cnt;
}
int d[MAXN],sz[MAXN],f[MAXN],dfn[MAXN],dfs_t;
int hson[MAXN],mson[MAXN],p[MAXN];
LL dis[MAXN];
void dfs1(int u,int fa){
    sz[u]=1,f[u]=fa,mson[u]=0;
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(v==fa)continue;
        d[v]=d[u]+1;
        dis[v]=dis[u]+edge[i].w;
        dfs1(v,u);
        sz[u]+=sz[v];
        if(sz[v]>mson[u]){
            hson[u]=v;
            mson[u]=sz[v];
        }
    }
}
void dfs2(int u,int top){
    dfn[u]=++dfs_t,p[u]=top;
    a[dfs_t]=dis[u];
    if(mson[u])
    dfs2(hson[u],top);
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(v==f[u]||v==hson[u])
        continue;
        dfs2(v,v);
    }
```

```
}
int lca(int u,int v){
    while(p[u]!=p[v]){
        if(d[p[u]]<d[p[v]])
        swap(u,v);
        u=f[p[u]];
    }
    return d[u]<d[v]?u:v;
}
void update(int u,int pu,int v){
    while(p[u]!=p[pu]){
        Tree::update(1,dfn[p[u]],dfn[u],v);
        u=f[p[u]];
    }
    Tree::update(1,dfn[pu],dfn[u],v);
}
LL query(int u,int v){
    LL ans=inf;
    while(p[u]!=p[v]){
        if(d[p[u]]<d[p[v]])
        swap(u,v);
        ans=min(ans,Tree::query(1,dfn[p[u]],dfn[u]));
        u=f[p[u]];
    }
    if(d[u]>d[v])
    swap(u,v);
    return min(ans,Tree::query(1,dfn[u],dfn[v]));
}
int main()
{
    int n=read_int(),m=read_int(),cnt=0;
    _for(i,1,n){
        int u=read_int(),v=read_int(),w=read_int();
        Insert(u,v,w);
        Insert(v,u,w);
    }
    Tree::build(1,1,n);
    dfs1(1,0);
    dfs2(1,1);
    while(m--){
        int opt=read_int(),u=read_int(),v=read_int();
        if(opt==1){
            LL a0=read_int(),b0=read_int();
            int p=lca(u,v);
            ++cnt;
            slope[cnt]=-a0,b[cnt]=a0*dis[u]+b0;
            update(u,p,cnt);
            ++cnt;
            slope[cnt]=a0,b[cnt]=a0*(dis[u]-2*dis[p])+b0;
            update(v,p,cnt);
```

```
        }
        else
        enter(query(u,v));
    }
    return 0;
}
```