线段树合并/分裂

算法简介

一种快速合并、分裂线段树(一般为权值线段树)的算法，时空间复杂度 $O(m\log n)$。

算法思想

更新、分裂线段树时动态开点，易知更新操作时空间复杂度 $O(\log n)$。

关于分裂操作，遇到空结点则返回。其余操作同线段树查询，遇到分裂区间的子区间则将原节点转移给新节点，然后返回。

所以分裂操作的时空间复杂度同线段树查询操作的时间复杂度，即 $O(\log n)$。

关于合并操作，如果遇到叶子节点或空结点就直接 return否则跑子树。

关于合并操作的时间复杂度，每次合并时间复杂度为两棵线段树的重叠部分的结点数，所以不会超过较小的那棵线段树的结点数。

所以合并操作的总时间复杂度等于空间复杂度，即 $O(m\log n)$。

代码模板

洛谷p5494

```
const int MAXN=2e5+5,MAXM=30;
struct Node{
    LL cnt;
    int ch[2];
}node[MAXN*MAXM];
struct Pool{
    int Stack[MAXN*MAXM],top,tot;
    int New(){return top?Stack[top--]:++tot;}
    void Delate(int x){
        node[x].cnt=node[x].ch[0]=node[x].ch[1]=0;
        Stack[++top]=x;
    }
}pool;
void push_up(int
k){node[k].cnt=node[node[k].ch[0]].cnt+node[node[k].ch[1]].cnt;}
void update(int &k,int lef,int rig,int pos,int v){
    if(!k)k=pool.New();
    if(lef==rig)
    return node[k].cnt+=v,void();
    int mid=lef+rig>>1;
```

```
    if(mid>=pos)
    update(node[k].ch[0],lef,mid,pos,v);
    else
    update(node[k].ch[1],mid+1,rig,pos,v);
    push_up(k);
}
void Merge(int &k1,int k2,int lef,int rig){
    if(!k1||!k2) return k1|=k2,void();
    if(lef==rig) return node[k1].cnt+=node[k2].cnt,pool.Delate(k2);
    int mid=lef+rig>>1;
    Merge(node[k1].ch[0],node[k2].ch[0],lef,mid);
    Merge(node[k1].ch[1],node[k2].ch[1],mid+1,rig);
    pool.Delate(k2);
    push_up(k1);
}
void split(int &k1,int &k2,int lef,int rig,int L,int R){
    if(!k1)return;
    if(L<=lef&&rig<=R){
        k2=k1;
        k1=0;
        return;
    }
    k2=pool.New();
    int mid=lef+rig>>1;
    if(mid>=L)
    split(node[k1].ch[0],node[k2].ch[0],lef,mid,L,R);
    if(mid<R)
    split(node[k1].ch[1],node[k2].ch[1],mid+1,rig,L,R);
    push_up(k1);push_up(k2);
}
LL Count(int k,int lef,int rig,int L,int R){
    if(!k)return 0;
    if(L<=lef&&rig<=R)
    return node[k].cnt;
    int mid=lef+rig>>1;
    if(mid>=R)
    return Count(node[k].ch[0],lef,mid,L,R);
    else if(mid<L)
    return Count(node[k].ch[1],mid+1,rig,L,R);
    else
    return
Count(node[k].ch[0],lef,mid,L,R)+Count(node[k].ch[1],mid+1,rig,L,R);
}
int Rank(int k,int lef,int rig,LL rk){
    if(node[k].cnt<rk)
    return -1;
    int mid=lef+rig>>1;
    if(lef==rig)return mid;
    if(rk<=node[node[k].ch[0]].cnt)
    return Rank(node[k].ch[0],lef,mid,rk);
```

```
    else
    return Rank(node[k].ch[1],mid+1,rig,rk-node[node[k].ch[0]].cnt);
}
int root[MAXN],root_cnt=1;
int main()
{
    int n=read_int(),m=read_int(),v,p,x,y,opt;
    _rep(i,1,n){
        v=read_int();
        if(v)
        update(root[1],1,n,i,v);
    }
    while(m--){
        opt=read_int(),p=read_int();
        switch(opt){
            case 0:
                x=read_int(),y=read_int();
                split(root[p],root[++root_cnt],1,n,x,y);
                break;
            case 1:
                x=read_int();
                Merge(root[p],root[x],1,n);
                break;
            case 2:
                x=read_int(),y=read_int();
                update(root[p],1,n,y,x);
                break;
            case 3:
                x=read_int(),y=read_int();
                enter(Count(root[p],1,n,x,y));
                break;
            case 4:
                x=read_int();
                enter(Rank(root[p],1,n,x));
                break;
        }
    }
    return 0;
}
```

算法练习

习题一

洛谷p4556

题意

给定一棵 n 个节点的数，m 个操作。

每个操作三个参数 x,y,z，表示给结点 x 到 y 的树链上的每个点打上一个 z 号标记。

经过所有操作后输出每个结点被打上的最多的标记的编号(满足条件的标记存在多个时输出编号最小的)，如果该结点未被标记过，输出 0。

题解 1

离散化处理标记编号，防止 MLE。

每个结点用一棵权值线段树维护该结点的所有标记状态，树上差分打标记，最后从叶子结点开始向上合并即可。

```
const int MAXN=1e5+5,MAXM=20;
struct Edge{
    int to,next;
}edge[MAXN<<1];
int head[MAXN],edge_cnt;
void Insert(int u,int v){
    edge[++edge_cnt].to=v;
    edge[edge_cnt].next=head[u];
    head[u]=edge_cnt;
}
struct LCA{
    int d[MAXN],anc[MAXN][MAXM],log2[MAXN];
    void dfs(int u,int fa,int dep){
        anc[u][0]=fa,d[u]=dep;
        for(int i=head[u];i;i=edge[i].next){
            int v=edge[i].to;
            if(v==fa)
            continue;
            dfs(v,u,dep+1);
        }
    }
    void build(int root,int n){
        log2[1]=0;
        _rep(i,2,n)
        log2[i]=log2[i>>1]+1;
        dfs(root,-1,1);
        _rep(i,1,n){//下标从1开始
            for(int j=1;(1<<j)<n;j++)
            anc[i][j]=-1;
        }
        for(int j=1;(1<<j)<n;j++){
            _rep(i,1,n){
                if(anc[i][j-1]==-1)
```

```
                continue;
                anc[i][j]=anc[anc[i][j-1]][j-1];
            }
        }
    }
    int query(int p,int q){
        if(d[p]<d[q])
        swap(p,q);
        for(int i=log2[d[p]];i>=0;i--){
            if(d[p]-(1<<i)>=d[q])
            p=anc[p][i];
        }
        if(p==q)
        return p;
        for(int i=log2[d[p]];i>=0;i--){
            if(anc[p][i]!=-1&&anc[p][i]!=anc[q][i]){
                p=anc[p][i],q=anc[q][i];
            }
        }
        return anc[p][0];
    }
}lca;
const int MAXS=MAXN*60;
int root[MAXN],tot;
struct Node{
    int max_cnt,ans;
    int ch[2];
}node[MAXS];
void push_up(int k){
    if(node[node[k].ch[0]].max_cnt>=node[node[k].ch[1]].max_cnt)
node[k].max_cnt=node[node[k].ch[0]].max_cnt,node[k].ans=node[node[k].ch[0]]
.ans;
    else
node[k].max_cnt=node[node[k].ch[1]].max_cnt,node[k].ans=node[node[k].ch[1]]
.ans;
}
void update(int &k,int lef,int rig,int pos,int v){
    if(!k) k=++tot;
    if(lef==rig) return node[k].max_cnt+=v,node[k].ans=pos,void();
    int mid=lef+rig>>1;
    if(pos>mid)
    update(node[k].ch[1],mid+1,rig,pos,v);
    else
    update(node[k].ch[0],lef,mid,pos,v);
    push_up(k);
}
void Merge(int &k1,int k2,int lef,int rig){
    if(!k1||!k2) return k1|=k2,void();
    if(lef==rig) return node[k1].max_cnt+=node[k2].max_cnt,void();
    int mid=lef+rig>>1;
    Merge(node[k1].ch[0],node[k2].ch[0],lef,mid);
```

```
    Merge(node[k1].ch[1],node[k2].ch[1],mid+1,rig);
    push_up(k1);
}
int X[MAXN],Y[MAXN],Z[MAXN],b[MAXN],ans[MAXN],Max_z;
void dfs(int u){
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(lca.anc[u][0]==v)
        continue;
        dfs(v);
        Merge(root[u],root[v],1,Max_z);
    }
    ans[u]=node[root[u]].max_cnt>0?node[root[u]].ans:0;
}
int main()
{
    int n=read_int(),q=read_int(),u,v,p;
    _for(i,1,n){
        u=read_int(),v=read_int();
        Insert(u,v);
        Insert(v,u);
    }
    lca.build(1,n);
    _for(i,0,q)
    X[i]=read_int(),Y[i]=read_int(),Z[i]=read_int();
    memcpy(b,Z,sizeof(Z));
    sort(b,b+q);
    Max_z=unique(b,b+q)-b;
    _for(i,0,q){
        Z[i]=lower_bound(b,b+Max_z,Z[i])-b+1;
update(root[X[i]],1,Max_z,Z[i],1);update(root[Y[i]],1,Max_z,Z[i],1);
        p=lca.query(X[i],Y[i]);
        update(root[p],1,Max_z,Z[i],-1);
        p=lca.anc[p][0];
        if(p!=-1)
        update(root[p],1,Max_z,Z[i],-1);
    }
    dfs(1);
    _rep(i,1,n)
    if(ans[i])
    enter(b[ans[i]-1]);
    else
    enter(0);
    return 0;
}
```

题解 2

考虑树剖，将树上问题转换为区间问题，更新路径时只打差分打标记。

最后询问时只建一棵权值线段树，依次释放区间上每个位置的标记，维护标记前缀和、答案。

时间复杂度 $O(m\log^2 n)$空间复杂度 $O(m\log n)$但常数远小于线段树合并。

```
const int MAXN=1e5+5,MAXM=20;
struct Edge{
    int to,next;
}edge[MAXN<<1];
int head[MAXN],edge_cnt;
void Insert(int u,int v){
    edge[++edge_cnt].to=v;
    edge[edge_cnt].next=head[u];
    head[u]=edge_cnt;
}
struct lazy_tag{
    int v,next;
}lazy[MAXN*MAXM];
int head_2[MAXN],lazy_cnt;
void Insert_2(int u,int v){
    lazy[++lazy_cnt].v=v;
    lazy[lazy_cnt].next=head_2[u];
    head_2[u]=lazy_cnt;
}
int Max_cnt[MAXN<<2],Ans[MAXN<<2],lef[MAXN<<2],rig[MAXN<<2];
void build(int k,int L,int R){
    lef[k]=L,rig[k]=R;
    int M=L+R>>1;
    if(L==R)return Ans[k]=M,void();
    build(k<<1,L,M);
    build(k<<1|1,M+1,R);
}
void push_up(int k){
    if(Max_cnt[k<<1]>=Max_cnt[k<<1|1]){
        Max_cnt[k]=Max_cnt[k<<1];
        Ans[k]=Ans[k<<1];
    }
    else{
        Max_cnt[k]=Max_cnt[k<<1|1];
        Ans[k]=Ans[k<<1|1];
    }
}
void update(int k,int pos,int v){
    if(lef[k]==rig[k])
    return Max_cnt[k]+=v,void();
    int mid=lef[k]+rig[k]>>1;
    if(pos<=mid)
    update(k<<1,pos,v);
    else
    update(k<<1|1,pos,v);
    push_up(k);
```

```
}
int d[MAXN],sz[MAXN],f[MAXN],dfs_id[MAXN],inv_id[MAXN],dfs_t;
int h_son[MAXN],mson[MAXN],p[MAXN];
void dfs_1(int u,int fa,int depth){
    sz[u]=1;f[u]=fa;d[u]=depth;mson[u]=0;
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(v==fa)
        continue;
        dfs_1(v,u,depth+1);
        sz[u]+=sz[v];
        if(sz[v]>mson[u]){
            h_son[u]=v;
            mson[u]=sz[v];
        }
    }
}
void dfs_2(int u,int top){
    dfs_id[u]=++dfs_t;inv_id[dfs_t]=u;p[u]=top;
    if(mson[u])
    dfs_2(h_son[u],top);
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(v==f[u]||v==h_son[u])
        continue;
        dfs_2(v,v);
    }
}
void update_path(int u,int v,int w){
    while(p[u]!=p[v]){
        if(d[p[u]]<d[p[v]])
        swap(u,v);
        Insert_2(dfs_id[p[u]],w);
        Insert_2(dfs_id[u]+1,-w);
        u=f[p[u]];
    }
    if(d[u]>d[v])
    swap(u,v);
    Insert_2(dfs_id[u],w);
    Insert_2(dfs_id[v]+1,-w);
}
void update_node(int u){
    for(int i=head_2[u];i;i=lazy[i].next){
        if(lazy[i].v>0)
        update(1,lazy[i].v,1);
        else
        update(1,-lazy[i].v,-1);
    }
}
int X[MAXN],Y[MAXN],Z[MAXN],b[MAXN],ans[MAXN];
```

```
int main()
{
    int n=read_int(),q=read_int(),u,v,w;
    _for(i,1,n){
        u=read_int(),v=read_int();
        Insert(u,v);
        Insert(v,u);
    }
    _for(i,0,q)
    X[i]=read_int(),Y[i]=read_int(),Z[i]=read_int();
    memcpy(b,Z,sizeof(Z));
    sort(b,b+q);
    int Max_z=unique(b,b+q)-b;
    dfs_1(1,-1,0);
    dfs_2(1,1);
    build(1,1,Max_z);
    _for(i,0,q)
    update_path(X[i],Y[i],lower_bound(b,b+Max_z,Z[i])-b+1);
    _rep(i,1,n){
        update_node(i);
        ans[inv_id[i]]=Max_cnt[1]>0?b[Ans[1]-1]:0;
    }
    _rep(i,1,n)
    enter(ans[i]);
    return 0;
}
```

习题二

洛谷p3224

题意

给定 n 个点，所有点的点权恰好为 $1 \sim n$ 的排列，接下来两种操作。

1. 询问与某点 x 联通的点集中的第 k 大元素的编号，若不存在输出 -1。
2. 连接点 u、v。

题解

每个点维护一棵权值线段树，并查集维护连通分量。

对操作 1 类似名次树操作，对操作 2 使用线段树合并即可，时空间复杂度 $O(n\log n)$。

```
const int MAXN=1e5+5,MAXS=MAXN*40;
int root[MAXN],tot;
struct Node{
```

```
    int sz;
    int ch[2];
}node[MAXS];
void push_up(int
k){node[k].sz=node[node[k].ch[0]].sz+node[node[k].ch[1]].sz;}
void update(int &k,int lef,int rig,int pos){
    k=++tot;
    node[k].sz++;
    if(lef==rig) return;
    int mid=lef+rig>>1;
    if(pos>mid)
    update(node[k].ch[1],mid+1,rig,pos);
    else
    update(node[k].ch[0],lef,mid,pos);
}
void Merge(int &k1,int k2,int lef,int rig){
    if(!k1||!k2) return k1|=k2,void();
    int mid=lef+rig>>1;
    Merge(node[k1].ch[0],node[k2].ch[0],lef,mid);
    Merge(node[k1].ch[1],node[k2].ch[1],mid+1,rig);
    push_up(k1);
}
int query(int k,int lef,int rig,int rk){
    if(rk>node[k].sz)return 0;
    int mid=lef+rig>>1;
    if(lef==rig)return mid;
    if(rk<=node[node[k].ch[0]].sz)
    return query(node[k].ch[0],lef,mid,rk);
    else
    return query(node[k].ch[1],mid+1,rig,rk-node[node[k].ch[0]].sz);
}
int kth[MAXN],Rank[MAXN],p[MAXN],n;
int Find(int x){return x==p[x]?x:p[x]=Find(p[x]);}
void Merge(int u,int v){
    int x=Find(u),y=Find(v);
    if(x!=y){
        p[y]=x;
        Merge(root[x],root[y],1,n);
    }
}
int main()
{
    n=read_int();
    int m=read_int(),u,v;
    Rank[0]=-1;
    _rep(i,1,n){
        kth[i]=read_int(),p[i]=i,Rank[kth[i]]=i;
        update(root[i],1,n,kth[i]);
    }
    while(m--){
```

```
        u=read_int(),v=read_int();
        Merge(u,v);
    }
    int q=read_int();
    char opt;
    while(q--){
        opt=get_char(),u=read_int(),v=read_int();
        if(opt=='Q')
        enter(Rank[query(root[Find(u)],1,n,v)]);
        else
        Merge(u,v);
    }
    return 0;
}
```

习题三

洛谷p1600

题意

一棵 n 个结点的树，树上有 m 个玩家，第 i 个玩家的起点为 s_i，终点为 t_i，每个玩家每秒移动一条边。

在每个结点放置一名观察者，第 i 个结点的观察者会在第 w_i 秒观察到达该点的玩家。

玩家到终点后立即消失，但可以在该时刻被观察者观测。

问每个观察者观察到的玩家数量。

题解 1

先将树转化为一棵有根树，考虑玩家对的结点 u 的观察者的贡献。

可以把玩家的运动分解为 $s\to \text{LCA}(s,t)\to t$。

考虑 $s\to \text{LCA}(s,t)$，只当且仅当 $u\in \{s\to \text{LCA}(s,t)\}$ 且 $w_u=d_s-d_u$ 才产生贡献。

考虑 $\text{LCA}(s,t)\to t$，只当且仅当 $u\in \{\text{LCA}(s,t)\to t\}$ 且 $\text{dist}(s,t)-w_u=d_t-d_u$ 才产生贡献。

分离变量，得 $w_u+d_u=d_s$ 和 $w_u-d_u=\text{dist}(s,t)-d_t$，可以考虑每个结点用两个桶维护可以产生贡献的 d_s 和 $\text{dist}(s,t)-d_t$ 的数量。

然后考虑怎么处理 $u\in \{s\to \text{LCA}(s,t)\}$ 和 $u\in \{\text{LCA}(s,t)\to t\}$ 这两个限制条件。

考虑树上差分，在 s 和 t 结点打上增加标记，$\text{LCA}(s,t)$ 结点打上删除标记。

对每个结点，先 dfs 子节点，然后处理增加标记，然后查询答案，最后处理删除标记。

考虑如果 $\text{LCA}(s,t)$ 满足 $w_{\text{LCA}(s,t)}+d_{\text{LCA}(s,t)}=d_s$必有 $w_{\text{LCA}(s,t)}-d_{\text{LCA}(s,t)}=\text{dist}(s,t)-d_t$导致 s 和 t 重复贡献，所以要提前处理。

```
const int MAXN=3e5+5;
int head[MAXN],edge_cnt;
struct Edge{
    int to,next;
}edge[MAXN<<1];
void Insert(int u,int v){
    edge[++edge_cnt].next=head[u];
    edge[edge_cnt].to=v;
    head[u]=edge_cnt;
}
struct LCA{
    int d[MAXN],sz[MAXN],f[MAXN];
    int h_son[MAXN],mson[MAXN],p[MAXN];
    void dfs_1(int u,int fa,int depth){
        sz[u]=1;f[u]=fa;d[u]=depth;mson[u]=0;
        for(int i=head[u];i;i=edge[i].next){
            int v=edge[i].to;
            if(v==fa)
            continue;
            dfs_1(v,u,depth+1);
            sz[u]+=sz[v];
            if(sz[v]>mson[u])
            h_son[u]=v,mson[u]=sz[v];
        }
    }
    void dfs_2(int u,int top){
        p[u]=top;
        if(mson[u])dfs_2(h_son[u],top);
        for(int i=head[u];i;i=edge[i].next){
            int v=edge[i].to;
            if(v==f[u]||v==h_son[u])
            continue;
            dfs_2(v,v);
        }
    }
    void Init(int root){dfs_1(root,-1,0);dfs_2(root,root);}
    int query_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;
    }
    int query_dis(int u,int v){return d[u]+d[v]-2*d[query_lca(u,v)];}
}lca;
int C[MAXN<<2],*c1=C,*c2=C+MAXN*3;
```

```
struct Path{
    int u,v,len;
}path[MAXN];
struct Tag{
    int p,next;
}tag[MAXN<<1];
int w[MAXN],head_s[MAXN],head_t[MAXN],head_lca[MAXN],tot,ans[MAXN];
void Insert_t(int node,int k){
    tag[++tot].next=head_t[node];
    tag[tot].p=k;
    head_t[node]=tot;
}
void Insert_lca(int node,int k){
    tag[++tot].next=head_lca[node];
    tag[tot].p=k;
    head_lca[node]=tot;
}
void dfs(int u,int fa){
    int t1=c1[w[u]+lca.d[u]],t2=c2[w[u]-lca.d[u]];
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(v==fa)
        continue;
        dfs(v,u);
    }
    c1[lca.d[u]]+=head_s[u];
    for(int i=head_t[u];i;i=tag[i].next){
        Path& p=path[tag[i].p];
        c2[p.len-lca.d[p.v]]++;
    }
    ans[u]+=c1[w[u]+lca.d[u]]-t1+c2[w[u]-lca.d[u]]-t2;
    for(int i=head_lca[u];i;i=tag[i].next){
        Path& p=path[tag[i].p];
        c1[lca.d[p.u]]--;
        c2[p.len-lca.d[p.v]]--;
    }
}
int main()
{
    int n=read_int(),m=read_int(),u,v;
    _for(i,1,n){
        u=read_int(),v=read_int();
        Insert(u,v);
        Insert(v,u);
    }
    lca.Init(1);
    _rep(i,1,n)
    w[i]=read_int();
    _rep(i,1,m){
        path[i].u=u=read_int(),path[i].v=v=read_int();
        int p=lca.query_lca(u,v);
```

```
        path[i].len=lca.d[u]+lca.d[v]-lca.d[p]*2;
        head_s[u]++;
        Insert_t(v,i);
        Insert_lca(p,i);
        if(lca.d[p]+w[p]==lca.d[u])
        ans[p]--;
    }
    dfs(1,0);
    _rep(i,1,n)
    space(ans[i]);
    return 0;
}
```

题解 2

树上差分的思路同题解 1，但这次考虑用线段树代替桶维护答案，对每个点可以直接处理贡献，然后从叶子结点向上合并线段树即可。

贴了一段别人的代码。

```
#include <bits/stdc++.h>
using namespace std;
char ch;
bool fi;
template<typename _Type>
inline void read(_Type&d) {
    d=0, fi=0, ch=getchar();
    while(!isdigit(ch) && ch!='-') ch=getchar();
    if(ch=='-') fi=1, ch=getchar();
    do d=d*10+ch-'0', ch=getchar();
    while(isdigit(ch));
    if(fi) d=~d+1;
}

const int N=300000+1;

struct DymSegTree {
    int ls[30*N],rs[30*N],sum[30*N],rt[N];
    int rb[30*N],top,tot;
    inline int newNode() {
        return top?rb[top--]:++tot;
    }
    inline void delNode(int x) {
        rb[++top]=x;
        ls[x]=rs[x]=sum[x]=0;
    }
    void modify(int&x,int l,int r,int pos,int val) {
        if(!x) x=newNode();
        if(l==r) { sum[x]+=val; return;}
```

```
        int mid=(l+r)>>1;
        if(pos<=mid) modify(ls[x],l,mid,pos,val);
        else modify(rs[x],mid+1,r,pos,val);
        sum[x]=sum[ls[x]]+sum[rs[x]];
    }
    int merge(int x,int y,int l,int r) {
        if(!x||!y) return x|y;
        int now=newNode();
        if(l==r) sum[now]=sum[x]+sum[y];
        else {
            int mid=(l+r)>>1;
            ls[now]=merge(ls[x],ls[y],l,mid);
            rs[now]=merge(rs[x],rs[y],mid+1,r);
            sum[now]=sum[ls[now]]+sum[rs[now]];
        }
        delNode(x), delNode(y);
        return now;
    }
    int query(int x,int l,int r,int pos) {
        if(l==r) return sum[x];
        int mid=(l+r)>>1;
        return (mid>=pos) ? query(ls[x],l,mid,pos)
            :query(rs[x],mid+1,r,pos);
    }
} T1, T2;

vector<int> E[N];
vector<int> G[N];
struct Player {
    int u,v,tim;
} a[N];
int n,m,w[N],ans[N],dep[N],fa[N][22];

void pre(int x,int pa) {
    dep[x]=dep[fa[x][0]=pa]+1;
    for(int i=1; (1<<i)<=dep[x]; ++i)
        fa[x][i]=fa[fa[x][i-1]][i-1];
    for(int y:E[x]) if(y!=pa) pre(y,x);
}
inline int lca(int x,int y) {
    if(dep[x]<dep[y]) x^=y^=x^=y;
    int dif=dep[x]-dep[y];
    for(int i=20; ~i; --i)
        if(dif&(1<<i)) x=fa[x][i];
    if(x==y) return x;
    for(int i=20; ~i; --i) if(fa[x][i]!=fa[y][i])
        x=fa[x][i], y=fa[y][i];
    return fa[x][0];
}
void dfs(int x,int pa) {
    for(int y:E[x]) if(y!=pa) dfs(y,x);
```

```
    for(int i:G[x]) T2.modify(T2.rt[x],1,N+N, a[i].tim-dep[a[i].v]+N,-1);
    ans[x]=T1.query(T1.rt[x],1,N+N, dep[x]+w[x])
          +T2.query(T2.rt[x],1,N+N, w[x]-dep[x]+N);
    for(int i:G[x]) T1.modify(T1.rt[x],1,N+N, dep[a[i].u],-1);
    if(pa) {
        T1.rt[pa]=T1.merge(T1.rt[pa],T1.rt[x],1,N+N);
        T2.rt[pa]=T2.merge(T2.rt[pa],T2.rt[x],1,N+N);
    }
}

int main() {
    read(n); read(m);
    for(int x,y,i=n; --i; ) {
        read(x); read(y);
        E[x].push_back(y);
        E[y].push_back(x);
    }
    pre(1,0);
    for(int i=1; i<=n; ++i) read(w[i]);
    for(int i=1; i<=m; ++i) {
        read(a[i].u); read(a[i].v);
        int LCA=lca(a[i].u,a[i].v);
        a[i].tim=dep[a[i].u]+dep[a[i].v]-2*dep[LCA];
        T1.modify(T1.rt[a[i].u],1,N+N, dep[a[i].u],1);
        T2.modify(T2.rt[a[i].v],1,N+N, a[i].tim-dep[a[i].v]+N,1);
        G[LCA].push_back(i);
    }
    dfs(1,0);
    for(int i=1; i<=n; ++i) printf("%d ",ans[i]);
    return 0;
}
```