

# 静态点分治

## 算法简介

一种利用分治进行树上路径统计的算法，算法时间复杂度与维护经过根结点路径相关信息所需的时间复杂度有关，一般为 $O(n \log n)$

## 算法思想

所有路径分为两种，一种是过根结点的，一种是完全在根结点的某棵子树中的。因此可以考虑分治算法，选取一个点将无根树转换为有根树，然后递归处理每个棵以根节点的儿子为根的子树。如果选取树的重心作为根结点，则每棵子树的结点个数不超过 $\frac{n}{2}$ 可以保证递归深度不超过 $\log n$

在这个基础上如果能在 $O(\log n)$ 时间维护经过根结点路径相关信息，则算法总时间复杂度为 $O(n \log n)$

## 代码实现

重心的寻找可以利用树形dp处理出所有结点的 $\text{sz}$ 所有结点的最大子树

$\text{mson}[\text{u}] = \max(\text{mson}[\text{son}_1], \text{mson}[\text{son}_2], \dots, \text{mson}[\text{son}_k])$ 不断更新 $\text{mson}$ 最小的结点，最后便可以得到重心，时间复杂度 $O(n)$

分治过程需要注意标记已经访问的结点，同时更新子树大小 $\text{tot\_sz}$ 具体实现见模板

## 代码模板

```
int sz[MAXN], mson[MAXN], tot_sz, root, root_sz;
bool vis[MAXN];
void query(int u); // 自定义
void find_root(int u, int fa) {
    sz[u] = 1; mson[u] = 0;
    for (int i = head[u]; i; i = edge[i].next) {
        int v = edge[i].to;
        if (vis[v] || v == fa) continue;
        find_root(v, u);
        sz[u] += sz[v];
        mson[u] = max(mson[u], sz[v]);
    }
    mson[u] = max(mson[u], tot_sz - sz[u]);
    if (mson[u] < root_sz) {
        root = u;
        root_sz = mson[u];
    }
}
```

```
    }  
}  
void solve(int u){  
    vis[u]=true;query(u);  
    for(int i=head[u];i;i=edge[i].next){  
        int v=edge[i].to;  
        if(vis[v])  
            continue;  
        tot_sz=sz[v];root_sz=inf;  
        find_root(v,u);  
        solve(root);  
    }  
}
```

## 代码练习

### 习题1

#### 洛谷p3806

题目大意是说给定一棵 $n$ 个结点的正权树，多次查询，每次查询是否存在长度为 $k$ 的路径

对根结点，先考虑暴力法，用树形dp求出子树上各节点到根结点的距离，然后两两枚举，看看有没有两个结点距离和为 $k$ 。这样每层递归的时间复杂度是 $O(n^2)$ 显然会T

考虑用中途相遇法，可以将每层递归的时间复杂度优化到 $O(n)$ 。单次查询时间复杂度 $O(n \log n)$

但要注意三点

一、枚举过根结点的路径时路径两端点显然不能在同一棵子树里，所以要先求出一棵子树所有的 $\text{dist}$ 全部判断完后再进行标记

二、题目给定 $k \leq 10^7$ 所以不用标记大于 $10^7$ 的 $\text{dist}$

三、清空标记不能用 $\text{memset}$ 会T。这里开了一个 $\text{vector}$ 来记录之前的标记

另一个优化是可以离线处理查询，这样只需要分治一次，可以减小常数。

```
#include <cstdio>  
#include <cctype>  
#include <vector>  
#define _for(i,a,b) for(int i=(a);i<(b);++i)  
using namespace std;  
inline int read_int(){  
    int t=0;bool sign=false;char c=getchar();  
    while(!isdigit(c)){sign|=c=='-';c=getchar();}  
    while(isdigit(c)){t=(t<<1)+(t<<3)+(c&15);c=getchar();}  
    return sign?-t:t;  
}
```

```

const int MAXN=1e4+5,inf=1e7+5;
struct Edge{
    int to,w,next;
}edge[MAXN<<1];
int n,edge_cnt,head[MAXN];
int m,q[MAXN];
int sz[MAXN],mson[MAXN],tot_sz,root,root_sz;
int dis[MAXN];
vector<int> d,sd;
bool vis[MAXN],mark[inf],ok[MAXN];
void Insert(int u,int v,int w){
    edge[++edge_cnt].to=v;
    edge[edge_cnt].w=w;
    edge[edge_cnt].next=head[u];
    head[u]=edge_cnt;
}
void find_root(int u,int fa){
    sz[u]=1;mson[u]=0;
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(vis[v]||v==fa)
            continue;
        find_root(v,u);
        sz[u]+=sz[v];
        mson[u]=max(mson[u],sz[v]);
    }
    mson[u]=max(mson[u],tot_sz-sz[u]);
    if(mson[u]<root_sz){
        root=u;
        root_sz=mson[u];
    }
}
void dfs(int u,int fa){
    d.push_back(dis[u]);
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(vis[v]||v==fa)
            continue;
        dis[v]=dis[u]+edge[i].w;
        dfs(v,u);
    }
}
void query(int u){
    sd.clear();
    for(int i=head[u];i;i=edge[i].next){
        d.clear();
        int v=edge[i].to;
        if(vis[v])
            continue;
        dis[v]=edge[i].w;
        dfs(v,u);
    }
}

```

```
    _for(j,0,d.size())
    _for(k,0,m){
        if(q[k]>=d[j])
            ok[k]|=mark[q[k]-d[j]];
    }
    _for(j,0,d.size()){
        if(d[j]<inf){
            mark[d[j]]=true;
            sd.push_back(d[j]);
        }
    }
}
_for(i,0,sd.size())
    mark[sd[i]]=false;
}
void solve(int u){
    vis[u]=mark[0]=true;query(u);
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(vis[v])
            continue;
        tot_sz=sz[v];root_sz=inf;
        find_root(v,u);
        solve(root);
    }
}
int main()
{
    n=read_int(),m=read_int();
    int u,v,w;
    _for(i,1,n){
        u=read_int()-1,v=read_int()-1,w=read_int();
        Insert(u,v,w);
        Insert(v,u,w);
    }
    _for(i,0,m)
        q[i]=read_int();
    root_sz=inf;
    tot_sz=n;
    find_root(0,-1);
    solve(root);
    _for(i,0,m){
        if(ok[i])
            puts("AYE");
        else
            puts("NAY");
    }
    return 0;
}
```

## 习题2

### 洛谷p4149

给一棵树，每条边有权。求一条简单路径，权值和等于 $q$ 且边的数量最小。

类似习题1，开个 $\text{mark}$ 数组存一下到根结点距离为 $\text{dist}$ 的路径的最短边数 $\text{vector}$ 不仅要记录距离，还要记录深度，时间复杂度 $O(n \log n)$

```

#include <cstdio>
#include <cctype>
#include <vector>
#define _for(i,a,b) for(int i=(a);i<(b);++i)
using namespace std;
inline int read_int(){
    int t=0;bool sign=false;char c=getchar();
    while(!isdigit(c)){sign|=c=='-';c=getchar();}
    while(isdigit(c)){t=(t<<1)+(t<<3)+(c&15);c=getchar();}
    return sign?-t:t;
}
const int MAXN=2e5+5,inf=1e6+5;
struct Edge{
    int to,w,next;
}edge[MAXN<<1];
int n,edge_cnt,head[MAXN];
int sz[MAXN],mson[MAXN],tot_sz,root,root_sz;
int dis1[MAXN],dis2[MAXN],mark[inf],q,ans=MAXN;
vector<pair<int,int> > d,sd;
bool vis[MAXN];
void Insert(int u,int v,int w){
    edge[++edge_cnt].to=v;
    edge[edge_cnt].w=w;
    edge[edge_cnt].next=head[u];
    head[u]=edge_cnt;
}
void find_root(int u,int fa){
    sz[u]=1;mson[u]=0;
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(vis[v]||v==fa)
            continue;
        find_root(v,u);
        sz[u]+=sz[v];
        mson[u]=max(mson[u],sz[v]);
    }
    mson[u]=max(mson[u],tot_sz-sz[u]);
    if(mson[u]<root_sz){
        root=u;
        root_sz=mson[u];
    }
}

```

```
}  
void dfs(int u,int fa){  
    if(dis1[u]>q)  
        return;  
    d.push_back(make_pair(dis1[u],dis2[u]));  
    for(int i=head[u];i;i=edge[i].next){  
        int v=edge[i].to;  
        if(vis[v]||v==fa)  
            continue;  
        dis1[v]=dis1[u]+edge[i].w;  
        dis2[v]=dis2[u]+1;  
        dfs(v,u);  
    }  
}  
void query(int u){  
    mark[0]=0;  
    sd.clear();  
    for(int i=head[u];i;i=edge[i].next){  
        int v=edge[i].to;  
        if(vis[v])  
            continue;  
        d.clear();  
        dis1[v]=edge[i].w,dis2[v]=1;  
        dfs(v,u);  
        _for(j,0,d.size())  
            ans=min(ans,d[j].second+mark[q-d[j].first]);  
        _for(j,0,d.size()){  
            mark[d[j].first]=min(mark[d[j].first],d[j].second);  
            sd.push_back(d[j]);  
        }  
    }  
    _for(i,0,sd.size())  
        mark[sd[i].first]=MAXN;  
}  
void solve(int u){  
    vis[u]=true;query(u);  
    for(int i=head[u];i;i=edge[i].next){  
        int v=edge[i].to;  
        if(vis[v])  
            continue;  
        tot_sz=sz[v];root_sz=MAXN;  
        find_root(v,u);  
        solve(root);  
    }  
}  
int main()  
{  
    n=read_int(),q=read_int();  
    int u,v,w;  
    _for(i,1,n){
```

```

    u=read_int(),v=read_int(),w=read_int();
    Insert(u,v,w);
    Insert(v,u,w);
}
_for(i,0,inf)
mark[i]=MAXN;
root_sz=MAXN;
tot_sz=n;
find_root(0,-1);
solve(root);
if(ans==MAXN)
puts("-1");
else
printf("%d",ans);
return 0;
}

```

### 习题3

[洛谷p4178](#)

给定一棵  $n$  个结点的正权树和一个正数  $k$  统计有多少对结点  $(a,b)$  满足  $\text{dist}(a,b) \leq k$

把中途相遇法换成树状数组或名次树即可，时间复杂度  $O(n \log^2 n)$

### 习题4

[UVA12161](#)

给定一棵  $n$  个结点的树, 每条边包含长度  $L$  和费用  $D$  ( $1 \leq D, L \leq 1000$ ) 选择一条总费用不超过  $m$  的路径, 使得路径总长度最大。

考虑单调队列, 时间复杂度  $O(n \log^2 n)$  另外本题存在  $O(n \log n)$  解法, 有兴趣的可以自己尝试

### 习题5

[洛谷P2664](#)

给定一棵  $n$  个结点的树, 树的每个节点有个颜色

定义  $s(i,j)$  为  $i$  到  $j$  的颜色数量  $\sum_i = \sum_{j=1}^n s(i,j)$  要求输出所有  $\sum_i$

这题需要计算贡献, 思路较复杂, 这里只给出代码供参考, 时间复杂度  $O(n \log n)$  另外本题存在  $O(n)$  解法, 有兴趣的可以自己尝试

```

#include <cstdio>
#include <cctype>


```

```
#include <vector>
#define _for(i,a,b) for(int i=(a);i<(b);++i)
using namespace std;
typedef long long LL;
inline int read_int(){
    int t=0;bool sign=false;char c=getchar();
    while(!isdigit(c)){sign|=c=='-';c=getchar();}
    while(isdigit(c)){t=(t<<1)+(t<<3)+(c&15);c=getchar();}
    return sign?-t:t;
}
inline void write(LL x){
    register char c[21],len=0;
    if(!x)return putchar('0'),void();
    if(x<0)x=-x,putchar('-');
    while(x)c[++len]=x%10,x/=10;
    while(len)putchar(c[len--]+48);
}
inline void space(LL x){write(x),putchar(' ');}
inline void enter(LL x){write(x),putchar('\n');}
const int MAXN=1e5+5;
struct Edge{
    int to,next;
}edge[MAXN<<1];
int n,c[MAXN],edge_cnt,head[MAXN];
int sz[MAXN],mson[MAXN],tot_sz,root,root_sz;
LL sum,cnt[MAXN],col[MAXN],ans[MAXN],base,num;
bool vis[MAXN];
void Insert(int u,int v){
    edge[++edge_cnt].to=v;
    edge[edge_cnt].next=head[u];
    head[u]=edge_cnt;
}
void find_root(int u,int fa){
    sz[u]=1;mson[u]=0;
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(vis[v]||v==fa)continue;
        find_root(v,u);
        sz[u]+=sz[v];
        mson[u]=max(mson[u],sz[v]);
    }
    mson[u]=max(mson[u],tot_sz-sz[u]);
    if(mson[u]<root_sz){
        root=u;
        root_sz=mson[u];
    }
}
void dfs(int u,int fa){
    sz[u]=1,cnt[c[u]]++;
```

```
for(int i=head[u];i;i=edge[i].next){
    int v=edge[i].to;
    if(vis[v]||v==fa)
        continue;
    dfs(v,u);
    sz[u]+=sz[v];
}
if(cnt[c[u]]==1){//该颜色第一次出现时，将其子树对根结点的答案的贡献算的该结点上
    sum+=sz[u];
    col[c[u]]+=sz[u];
}
cnt[c[u]]--;
}
void change(int u,int fa,int type){
    cnt[c[u]]++;
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(vis[v]||v==fa)
            continue;
        change(v,u,type);
    }
    if(cnt[c[u]]==1){
        sum+=sz[u]*type;
        col[c[u]]+=sz[u]*type;
    }
    cnt[c[u]]--;
}
void cal(int u,int fa){
    cnt[c[u]]++;
    if(cnt[c[u]]==1){//该颜色对所有子代均产生贡献
        sum-=col[c[u]];
        num++;
    }
    ans[u]+=sum+base*num;
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(vis[v]||v==fa)
            continue;
        cal(v,u);
    }
    if(cnt[c[u]]==1){
        sum+=col[c[u]];
        num--;
    }
    cnt[c[u]]--;
}
void clear(int u,int fa){
    col[c[u]]=0;
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(vis[v]||v==fa)
```

```
        continue;
        clear(v,u);
    }
}
void query(int u){
    sum=num=0;dfs(u,-1);//得到所有子树对根的贡献
    ans[u]+=sum;
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(vis[v])
            continue;
        sum-=sz[v];col[c[u]]-=sz[v];
        cnt[c[u]]++;change(v,u,-1);cnt[c[u]]--;//消除该子树贡献
        base=sz[u]-sz[v];cal(v,u);
        sum+=sz[v];col[c[u]]+=sz[v];
        cnt[c[u]]++;change(v,u,1);cnt[c[u]]--;
    }
    clear(u,-1);
}
void solve(int u){
    vis[u]=true;query(u);
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(vis[v])
            continue;
        tot_sz=sz[v];root_sz=MAXN;
        find_root(v,u);
        solve(root);
    }
}
int main()
{
    n=read_int();
    _for(i,0,n)
    c[i]=read_int();
    int u,v,w;
    _for(i,1,n){
        u=read_int()-1,v=read_int()-1;
        Insert(u,v);
        Insert(v,u);
    }
    root_sz=MAXN;
    tot_sz=n;
    find_root(0,-1);
    solve(root);
    _for(i,0,n)
    enter(ans[i]);
    return 0;
}
```

From:  
<https://wiki.cvbbacm.com/> - CVBB ACM Team

Permanent link:  
[https://wiki.cvbbacm.com/doku.php?id=2020-2021:teams:legal\\_string:jxm2001:%E9%9D%99%E6%80%81%E7%82%B9%E5%88%86%E6%B2%BB&rev=1590071172](https://wiki.cvbbacm.com/doku.php?id=2020-2021:teams:legal_string:jxm2001:%E9%9D%99%E6%80%81%E7%82%B9%E5%88%86%E6%B2%BB&rev=1590071172) 

Last update: **2020/05/21 22:26**