虚树

用途

用于处理一类特殊的树形dp问题

构建方法及应用——从一道题目说起

洛谷p2495

题目大意

给定一棵树 m 个询问，每次询问 k 个点均不与 1 号点相连的最小代价和。

暴力

这道题目的朴素做法是暴力dp 时间复杂度是 $O(nm)$ 设 $dp[now]$ 为now节点及下属节点的最小代价和，则 $dp[now]=\left\{ \begin{aligned}edge[now][fa].w,target[now]=true \\ \sum dp[v],now=1\\min(\sum dp[v],edge[now][fa].w),others\end{aligned} \right.$ 相关代码如下。

```
#include <iostream>
#include <cmath>
#include <algorithm>
#include <cstdio>
#include <string.h>
using namespace std;
int n;
int dp[250001];
int haschild[250001];
struct node
{
    int u,v,next;
    int w;
};
node edge[500001];
int first[250001];
int tot;
inline void add_edge(int x,int y,int z)
{
edge[++tot].u=x;edge[tot].v=y;edge[tot].next=first[x];first[x]=tot;edge[tot]
.w=z;
edge[++tot].u=y;edge[tot].v=x;edge[tot].next=first[y];first[y]=tot;edge[tot]
.w=z;
}
```

```
void dfs(int now,int fa)
{
    int recv;
    for (int i=first[now];i;i=edge[i].next)
        if (edge[i].v==fa)
            recv=i;

    if (haschild[now])
    {
        dp[now]=edge[recv].w;
        return;
    }
    for (int i=first[now];i;i=edge[i].next)
        {
            int v=edge[i].v;
            if (v!=fa)
            {
                dfs(v,now);
                dp[now]+=dp[v];

            }
        }
//  printf("%d %d\n",now,dp[now]);

    if (fa)
        dp[now]=min(dp[now],edge[recv].w);
}
int main()
{
    scanf("%d",&n);
    for (int i=1;i<n;i++)
    {
        int u,v,w;
        scanf("%d%d%d",&u,&v,&w);
        add_edge(u,v,w);
    }
    int m;
    scanf("%d",&m);
    for (int i=1;i<=m;i++)
    {
        memset(haschild,0,sizeof(haschild));
        memset(dp,0,sizeof(dp));
        int k;
        scanf("%d",&k);
        for (int j=1;j<=k;j++)
        {
            int t;
            scanf("%d",&t);
            haschild[t]=true;
        }
```

```
        dfs(1,0);
        printf("%d\n",dp[1]);
    }
}
```

构建虚树

事实上我们发现我们能利用到的点只和节点的数量 $\sum k$ 有关。所以我们只保留这些节点以及其LCA构建一颗树，这棵树就叫做虚树。时间复杂度为 $O(2\sum k)$ 具体构建方法如下。

对于每一个询问，把节点按dfs序排序，同时维护一个栈，表示从跟到栈顶元素的这条链。加入当前加入的节点 now 满足 $LCA(now,s.top())=s.top()$ 则直接将其入栈，假如不满足，说明 $s.top()$ 所在的子树遍历完毕，然后进行下面子树的构建。

代码

```
// luogu-judger-enable-o2
// luogu-judger-enable-o2
#include<cstdio>
#include<algorithm>
#include<cstring>
#include<vector>
#define getchar() (p1 == p2 && (p2 = (p1 = buf) + fread(buf, 1, 1 << 21,
stdin), p1 == p2) ? EOF : *p1++)
#define LL long long
char buf[(1 << 21) + 1], *p1 = buf, *p2 = buf;
using namespace std;
const int MAXN = 250001;
inline int read() {
    char c = getchar(); int x = 0, f = 1;
    while(c < '0' || c > '9') {if(c == '-') f = -1; c = getchar();}
    while(c >= '0' && c <= '9') x = x * 10 + c - '0', c = getchar();
    return x * f;
}
char obuf[1 << 24], *O=obuf;
void print(LL x) {
    if(x > 9) print(x / 10);
    *O++= x % 10 + '0';
}
int N, M;
struct Edge {
    int u, v, w, nxt;
}E[MAXN << 1];
int head[MAXN], num = 1;
inline void AddEdge(int x, int y, int z) {
    E[num] = (Edge) {x, y, z, head[x]};
    head[x] = num++;
}
vector<int> v[MAXN];
void add_edge(int x, int y) {
```

```
    v[x].push_back(y);
}
int a[MAXN], dfn[MAXN], topf[MAXN], siz[MAXN], son[MAXN], s[MAXN], top,
deep[MAXN], fa[MAXN], ID = 0;
LL mn[MAXN];
void dfs1(int x, int _fa) {
    siz[x] = 1; fa[x] = _fa;
    for(int i = head[x]; i != -1; i = E[i].nxt) {
        if(E[i].v == _fa) continue;
        deep[E[i].v] = deep[x] + 1;
        mn[E[i].v] = min(mn[x], (LL)E[i].w);
        dfs1(E[i].v, x);
        siz[x] += siz[E[i].v];
        if(siz[E[i].v] > siz[son[x]]) son[x] = E[i].v;
    }
}
void dfs2(int x, int topfa) {
    topf[x] = topfa;
    dfn[x] = ++ID;
    if(!son[x]) return ;
    dfs2(son[x], topfa);
    for(int i = head[x]; i != -1; i = E[i].nxt)
        if(!topf[E[i].v])
            dfs2(E[i].v, E[i].v);
}
int LCA(int x, int y) {
    while(topf[x] != topf[y]) {
        if(deep[topf[x]] < deep[topf[y]]) swap(x, y);
        x = fa[topf[x]];
    }
    if(deep[x] < deep[y]) swap(x, y);
    return y;
}
void insert(int x) {
    if(top == 1) {s[++top] = x; return ;}
    int lca = LCA(x, s[top]);
    if(lca == s[top]) return ;
    while(top > 1 && dfn[s[top - 1]] >= dfn[lca]) add_edge(s[top - 1],
s[top]), top--;
    if(lca != s[top]) add_edge(lca, s[top]), s[top] = lca;//
    s[++top] = x;
}
LL DP(int x) {
    if(v[x].size() == 0) return mn[x];
    LL sum = 0;
    for(int i = 0; i < v[x].size(); i++)
        sum += DP(v[x][i]);
    v[x].clear();
    return min(sum, (LL)mn[x]);
}
```

```
int comp(const int &a, const int &b) {
    return dfn[a] < dfn[b];
}
int main() {
    memset(head, -1, sizeof(head));
    //memset(mn, 0xff, sizeof(mn));
    mn[1] = 1ll << 60;
    N = read();
    for(int i = 1; i <= N - 1; i++) {
        int x = read(), y = read(), z = read();
        AddEdge(x, y, z); AddEdge(y, x, z);
    }
    deep[1] = 1;
    dfs1(1, 0);
    dfs2(1, 1);
    M = read();
    /*for(int i = 1; i <= N; i++)
        for(int j = 1; j <= N; j++)
            printf("%d %d %d\n", i, j, LCA(i, j));*/
    //for(int i = 1; i <= N; i++) printf("%d ", mn[i]); puts("");
    while(M--) {
        int K = read();
        for(int i = 1; i <= K; i++) a[i] = read();
        sort(a + 1, a + K + 1, comp);
        s[top = 1] = 1;
        for(int i = 1; i <= K; i++) insert(a[i]);
        while(top > 0)  add_edge(s[top - 1], s[top]), top--;
        print(DP(1)), *O++ = '\n';
    }
    fwrite(obuf, O-obuf, 1 , stdout);
    return 0;
}
```