# 虚树-树上动态规划的利器

# **虚树**

## **问题引入**

在一类****树上动态规划****问题中,题目给出的询问往往包含树上的很多各节点,并保证总的点数规模小于某个值.

如果我们直接在整颗树上进行dp的话,时间复杂度与询问的次数有关,这显然是不可接受的,如果我们可以找到一种动态规划的方法,使其时间复杂度与询问中点的实际规模相关就好了.

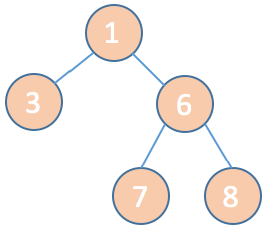
于是虚树应运而生.

## **虚树概念**

虚树即是一颗虚拟构建的一棵树,这个树只包含****关键点****以及****关键****lca的点,而其他不影响虚树结构的点和边都相当于进行了路径压缩,整颗虚树的规模不会超过关键点关键点数目的两倍.

## **举个栗子**

### **原树**



### 

### **虚树**

#### **包含关键点 1 2 3的虚树**

#### **包含关键点 1 3 7 8 的虚树**

其中6是****关键****lca节点

很显然,其他不是那么关键的点及边形成的路径我们都将他们压缩到了一条边,例如在第二个虚树中,我们相当于把1−6的路劲压缩到了边1−6中,而9号节点这种非关键点我们直接扔掉了,因为我们在dp的时候不会用到9号点.

## **虚树构建**

预处理我们对整颗树得到****dfs序列(即前序遍历)****,记为dfn[u].

我们使用一个****栈****,从栈顶到栈底的元素形成虚树的一颗树链.

当我们得到一些****询问点(关键点)****的时候,对这些点按照他们的dfn[u]值进行排序,然后从dfn值小的开始扫描,结合栈中保存的树链信息就可以将这颗虚树构建出来.

假设我们当前扫到的关键点为u,栈指针为top,栈为stk.

1.如果栈为空,或者栈中只有一个元素,那么显然应该:   
stk[++top]=u;

2.取lca=LCA(u,stk[top]),如果lca=stk[top],则说明u点应该接着stk[top]点延长当前的树链.做操作:   
stk[++top]=u;

3.如果lca≠stk[top],则说明u与stk[top]分属lca的两颗不同的子树,且包含stk[top]的这颗子树应该已经构建完成了,我们需要做的是:   
将lca的包含stk[top]子树的那部分退栈,并将这部分建边形成虚树.如果lca不在栈(树链)中,那么要把lca也加入栈中,保证虚树的结构不出现问题,随后将u加入栈中,以表延长树链.

### **代码实现**

//实现逐个将关键点插入形成一颗虚树

void insert(int u){

if(top <= 1) {stk[++top] = u;return ;}

int lca = LCA(u,stk[top]);

if(lca == stk[top]) {stk[++top] = u;return ;}

while(top > 1 && dfn[lca] <= stk[top-1]) {

addedge(stk[top-1],stk[top]);

--top;

}

if(lca != stk[top]) {

addedge(lca,stk[top]);

stk[top] = lca;

}

stk[++top] = u;

}

## 

## **虚树例题**

### **[SDOI2011]消耗战**

#### **题意**

给出n个点的一棵带有边权的树,以及q个询问.每次询问给出k个点,询问这使得这k个点与1点不连通所需切断的边的边权和最小是多少.

#### **题解**

先说说树形dp，我们设f[i]表示i节点封锁的最小开销【我们把每条向上的边直接看做该点的权值】

则f[i] = min(v[i],∑f[to])

我们知道封锁父亲效果一定不比封锁儿子差，那么每个点u的权值可以看做v[u] = min(v[u的祖先们])、

直接做肯定T，O(mn)，题目甚至直接都没有m的上限，而k的上限提醒我们只处理每次涉及到的点

如何抽出一棵树中单独的一些点呢？这就是虚树了

虚树

虚树，用来处理一棵有很多节点的树，询问只涉及其中部分节点且剩余节点的值对答案没有影响

这个时候我们只需保证树的形态不变，也就是询问点的相互位置关系不变，抽出来建一棵新的树，就是虚树

如何建树？

我们将所有点按照dfn排序，模拟递归的做法，开一个栈，表示当前正在处理以栈顶为根的子树

当我们遇到节点u时

①若u与栈顶p的lca就是p，说明u一定在p的子树内，由于是按照dfn顺序，那么接下来的节点就会在u的子树里，u入栈

②若u与栈顶p的lca不是p，那么一定在p之上，那么p的子树内的建树已经完成，不会再有里边的节点，而将处理lca为根的子树，这时候逐一出栈并建边，直至lca可以入栈的位置，lca入栈，u入栈

或者可以这么想，我们维护的栈实际上就是从根出发的一条链，由于按照dfs序，所以这条链按照一个方向延伸，当不能延伸的时候，前方已经没有了新的节点，链往回缩，缩的同时就把这些点给连边了，当缩到一个位置往另一边又有可以延伸的节点时，链继续延伸，最后所有点都到达，链往回缩回根。至此，所有的边都建好了

#### **代码**

#include<cstdio>

#include<cstring>

#include<algorithm>

using namespace std;

typedef long long LL;

LL inf;

int Bin[25];

struct node

{

int x,y,d,next;

}a[510000],e[510000];

int len,last[310000],elen,elast[310000];

void ins(int x,int y,int dd)

{

len++;

a[len].x=x;a[len].y=y;a[len].d=dd;

a[len].next=last[x];last[x]=len;

}

void ins2(int x,int y)

{

elen++;

e[elen].x=x;e[elen].y=y;

e[elen].next=elast[x];elast[x]=elen;

}

int cnt,pos[310000];

LL v[310000];

int dep[310000],f[25][310000];

void dfs(int x,int fa)

{

pos[x]=++cnt;

f[0][x]=fa;

for(int i=1;Bin[i]<=dep[x];i++)f[i][x]=f[i-1][f[i-1][x]];

for(int k=last[x];k;k=a[k].next)

{

int y=a[k].y;

if(y!=fa)

{

if(v[x]<a[k].d)v[y]=v[x];

else v[y]=a[k].d;

dep[y]=dep[x]+1;

dfs(y,x);

}

}

}

int LCA(int x,int y)

{

if(dep[x]<dep[y])swap(x,y);

for(int i=20;i>=0;i--)

if(dep[x]-dep[y]>=Bin[i])x=f[i][x];

if(x==y)return x;

for(int i=20;i>=0;i--)

if(dep[x]>=Bin[i]&&f[i][x]!=f[i][y]){x=f[i][x];y=f[i][y];}

return f[0][x];

}

bool cmp(int n1,int n2)

{

if(pos[n1]<pos[n2])return true;

return false;

}

int k,p[310000];

int top,sta[310000];//sta表示在已经建完毕的虚树上,以最后一个插入的点为端点的DFS链

void virtual\_tree()

{

sort(p+1,p+k+1,cmp);

//按dfs序小到大排序，对于任意点，dfs序大于该点值的点要么处于它的子树中，要么在另一棵同级的子树

int tp=1;//如果p[i]是处于p[tp]的子树里，只要p[tp]断了，p[i]同样断，so无视

for(int i=2;i<=k;i++)

if(LCA(p[i],p[tp])!=p[tp])p[++tp]=p[i];

k=tp;

top=0;sta[++top]=1;

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

{

int lca=LCA(p[i],sta[top]);

while(1){//不停往上跳令dep[lca]<dep[sta[top-1]]

if(dep[lca]>=dep[sta[top-1]]){//找到

if(lca!=sta[top])ins2(lca,sta[top]);//下面的连向lca，然后没用了

top--;

if(lca!=sta[top])sta[++top]=lca;

break;

}

ins2(sta[top-1],sta[top]);top--;//往上跳时顺便将下面的连在一起

}

if(p[i]!=sta[top])sta[++top]=p[i];

}

top--;

while(top>0){ins2(sta[top],sta[top+1]);top--;}

}

LL dp[310000];

void treeDP(int x)

{

LL ans=0;dp[x]=v[x];

for(int k=elast[x];k;k=e[k].next)

{

int y=e[k].y;

treeDP(y);

ans+=dp[y];

}

elast[x]=0;

if(ans!=0)dp[x]=min(dp[x],ans);

}

int main()

{

//freopen("repair.in","r",stdin);

//freopen("repair.out","w",stdout);

inf=1;for(int i=1;i<=60;i++)inf\*=2;

Bin[0]=1;for(int i=1;i<=25;i++)Bin[i]=Bin[i-1]\*2;

int n,x,y,dd;

scanf("%d",&n);

len=0;memset(last,0,sizeof(last));

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

{

scanf("%d%d%d",&x,&y,&dd);

ins(x,y,dd);ins(y,x,dd);

}

cnt=0;dep[1]=1;v[1]=inf;dfs(1,0);

int m;

scanf("%d",&m);

while(m--)

{

scanf("%d",&k);

for(int i=1;i<=k;i++)scanf("%d",&p[i]);

elen=0;

virtual\_tree();

treeDP(1);

printf("%lld\n",dp[1]);

}

return 0;

}