树上最近公共祖先

LCA（Least Common Ancestors），即最近公共祖先，是指这样一个问题：在有根树中，找出某两个结点 u 和 v 最近的公共祖先。

解决这类问题，容易想到一个朴素暴力算法，给出节点 u，v,，首先对 u 进行回溯一直到根节点，并对途中的节点加上标记。然后对 v 进行回溯，直到找到一个被标记的节点 T，此时 T 即为 u，v 的 LCA。此方法写起来很简单但时间复杂度太高，故只适合查询次数极少的时候。

本节将介绍三种高效求 LCA 的算法。

一、Tarjan 离线算法

Tarjan 算法 是一个在图中寻找强连通分量的算法。

算法的基本思想为：任选一结点开始进行深度优先搜索 dfs（若深度优先搜索结束后仍有未访问的结点，则再从中任选一点再次进行）。搜索过程中已访问的结点不再访问。搜索树的若干子树构成了图的强连通分量。应用到 LCA 问题上，tarjan 基于并查集，他通过在深搜树的同时，从查询集合中计算公共祖先。设 lca(u,v),u->v,路径中他们的 lca 就是他们所在支树的根。而深搜顺序是 t->u->t->v。u 先被访问，u 被访问结束回到 t 的时候，便将 u,t 合并为 1 个集合，那这个时候 u 的祖先就是 t，当访问到 v 的时候，他们的 lca 就是 u 的祖先 t。

【参考代码】

```
void tarjan(int u){
    vis[u]=1;
    fa[u]=u;
    for(int i=0;i<q[u].size();i++){//枚举 u 的询问
        int v=q[u][i];
        if(vis[v]) //若(u,v)的 v 访问过，则 uv 不再一个子树，LCA 就是 V 的祖先
            cnt[find(v)]++;
    }
    for(int i=0;i<g[u].size();i++){
        int v=g[u][i];
        if(vis[v])continue;
        tarjan(v);
        fa[v]=u;
    }
}
```

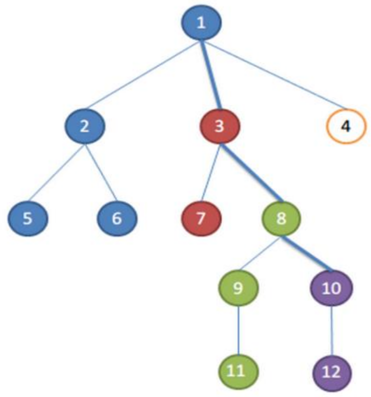
A2217. 「模板」「LCA」点的距离【参考代码】

```
#include<bits/stdc++.h>
using namespace std;
const int N=1e5+10;
struct node{
    int id, v;
};
vector<node>Q[N];
int n,q,x,y;
vector<int>G[N] ;
```

```
int fa[N],vis[N],ans[N],dis[N];
int find(int x){
    if(fa[x]==x)return x;return fa[x]=find(fa[x]);
}
void dfs(int u,int f){
    vis[u]=1;
    dis[u]=dis[f]+1;
    for(int i=0;i<Q[u].size();i++){
        int v=Q[u][i].v;
        int id=Q[u][i].id;
        if(vis[v]){
            int lca=find(v);
            ans[id]=dis[u]+dis[v]-2*dis[lca] ;
        }
    }
    for(int i=0;i<G[u].size();i++){
        int v=G[u][i];
        if(vis[v])continue;
        dfs(v,u);
        fa[v]=u;
    }
}
int main(){
    cin>>n;
    for(int i=1;i<=n;i++)fa[i]=i;
    for(int i=1;i<n;i++){
        int u,v;
        cin>>u>>v;
        G[u].push_back(v);
        G[v].push_back(u);
    }
    cin>>q;
    for(int i=1;i<=q;i++){
        cin>>x>>y;
        Q[x].push_back((node){i, y});
        Q[y].push_back((node){i, x});
    }
    dis[0]=-1;
    dfs(1,0);
    for(int i=1;i<=q;i++)
     cout<<ans[i]<<"\n";
    return 0;
}
```

二、 欧拉序 ST 表

遍历一颗树的同时记录每个节点的访问时间，得到的序列是 DFS 序。而欧拉序与 dfs 序类似，只是在记录时有所不同。每当访问完一个节点的子树，则需要返回一次该节点，然后记录第一次访问的时间和返回结束时间。如：



访问顺序：1,2,5,6,3,7,8,9,11,10,12,4

欧拉序顺序：1,2,5,2,6,2,1,3,7,3,8,9,11,9,8,10,12,10,8,3,1,4,1

欧拉序编号：1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23

观察欧拉序发现，对于点对<u,v>，他们的 LCA 在 u,v 的第一次访问序号之间，且是深度最小的点。于是得到如下算法：

1、 通过 dfs 遍历树，得到一个欧拉序。计 st[u]表示点 u 在欧拉序第一次出现的序号，计 dep[u]表示点 u 的深度。
2、 查询点对<u,v>，即查询[st[u],st[v]]区间深度最小的点

对于区间查询最小值可以用 ST 表来完成。

A2217. 「模板」「LCA」点的距离**【参考代码】**

```
#include<bits/stdc++.h>
using namespace std;
const int N=1e5+10;
int f[N*2][20],prd[N*2],dep[N*2];
int  st[N], vis[N],dis[N];
int tim=0,n,q;
vector<int>G[N];
void dfs(int x,int d){
    st[x]=++tim;prd[tim]=x;dep[tim]=d;
    //prd[i]记录欧拉序 i 对应的节点编号，dep[i]记录欧拉序 i 的深度
    vis[x]=1;dis[x]=d;//dis[x]记录节点 x 的深度
    for(int i=0;i<G[x].size();i++){
        int v=G[x][i];
        if(vis[v])continue;
        dfs(v,d+1);
        dep[++tim]=d;prd[tim]=x;
    }
}
int cal(int x,int y){
    return dep[x]<dep[y]?x:y;
```

```
}
void ST(int len){
    for(int i=1;i<=len;i++)
        f[i][0]=i;
    for(int j=1;(1<<j)<=len;j++)
        for(int i=1;i+(1<<j-1)<len;i++)
            f[i][j]=cal(f[i][j-1],f[i+(1<<j-1)][j-1]);
}
int query(int x,int y){//返回的是欧拉序
    int k=log2(y-x+1);
    return cal(f[x][k],f[y-(1<<k)+1][k]);
}
int lca(int x,int y){
    int l=st[x],r=st[y];
    if(l>r)swap(l,r);
    return prd[query(l,r)];//欧拉序转换为节点编号返回
}
int main(){
    cin>>n;
    for(int i=1;i<n;i++){
        int u,v;
        cin>>u>>v;
        G[u].push_back(v);
        G[v].push_back(u);
    }
    dfs(1,0);
    ST(tim);
    cin>>q;
    for(int i=1;i<=q;i++){
        int x,y;
        cin>>x>>y;
        int LCA=lca(x,y);
        cout<<dis[x]+dis[y]-2*dis[LCA]<<"\n";
    }

    return 0;
}
```

三、 倍增法

思想：求出每个点深度，对于求点对<u,v>的 lca，首先将更深的点向上跳成深度一致，然后两个点同时向上跳，跳到同一个点就是最近公共祖先。 但是暴力这样做效率过低，考虑通过倍增的思想来完成这一过程。

算法过程：

1） dfs 遍历树，记录每个点的深度以及父亲

2） 设 f[i][j]表示节点 i 的 2^j 祖先。

3）通过倍增同步跳的方式查询 LCA

【参考代码】

```
#include<bits/stdc++.h>
using namespace std;
const int N=1e5+10;
int f[N][20];
int vis[N],dis[N];
int n,q;
vector<int>G[N];
void dfs(int x,int d){
    vis[x]=1;dis[x]=d;
    for(int i=1;(1<<i)<=dis[x];i++)
        f[x][i]=f[f[x][i-1]][i-1];
    for(int i=0;i<G[x].size();i++){
        int v=G[x][i];
        if(vis[v])continue;
        f[v][0]=x;
        dfs(v,d+1);
    }
}
int lca(int x,int y){
    if(dis[x]<dis[y])swap(x,y);
    int k=dis[x]-dis[y];
    for(int i=0;(1<<i)<=k;i++)//让深度大的点 x 向上跳与 y 保持深度一致
        if(k&(1<<i))x=f[x][i];
    if(x==y)return x;
    for(int i=19;i>=0;i--){
        if(f[x][i]!=f[y][i])
            x=f[x][i],y=f[y][i];
    }
    return f[x][0];
}
int main(){
    cin>>n;
    for(int i=1;i<n;i++){
        int u,v;
        cin>>u>>v;
        G[u].push_back(v);
        G[v].push_back(u);
    }
    dfs(1,0);
    cin>>q;
    for(int i=1;i<=q;i++){
        int x,y;
```

```
        cin>>x>>y;
        int LCA=lca(x,y);
        cout<<dis[x]+dis[y]-2*dis[LCA]<<"\n";
    }
    return 0;
}
```

四、 LCA 应用

应用一、树上点对距离

题目 1： 见模板题

题目 2：Q1043. 「牛客 2022 多校 DAY3-A」Ancestor

题意： 给出两棵编号 1-n 的树 A B，A B 树上每个节点均有一个权值，给出 k 个关键点的编号 $x\_1 \cdots x\_n$，问有多少种方案使得去掉恰好一个关键点使得剩余关键点在树 A 上 LCA 的权值大于树 B 上 LCA 的权值。

【解析】 首先，容易想到一种直接的做法，枚举删除每个关键点，然后算出 2 颗树剩下的关键点的 LCA，比较大小统计即可。然后就是思考如何计算多个点的 LCA？计算多点 LCA，有一个结论是，就是 dfs 序最小和最大的 LCA。

当然，对于本题可能需要多次计算多个点的 LCA，可以直接用前缀思想，先计算出 2 颗树的前缀关键点 lca 和后缀 lca。然后暴力统计即可。

【参考代码】

```
#include<bits/stdc++.h>
using namespace std;
const int N=1e5+10;
struct Qtree{
    vector<int>G[N];
    int f[N][20],dis[N],w[N],vis[N],s[N],h[N];
    Qtree(){
        memset(vis,0,sizeof(vis));
    }
    void dfs(int x,int d){
        vis[x]=1;dis[x]=d;
        for(int i=1;(1<<i)<=dis[x];i++)
            f[x][i]=f[f[x][i-1]][i-1];
        for(int i=0;i<G[x].size();i++){
            int v=G[x][i];
            if(vis[v])continue;
            f[v][0]=x;
            dfs(v,d+1);
        }
    }
    int lca(int x,int y){
        if(dis[x]<dis[y])swap(x,y);
        int k=dis[x]-dis[y];
        for(int i=0;(1<<i)<=k;i++)//让深度大的点 x 向上跳与 y 保持深度一致
```

```
            if(k&(1<<i))x=f[x][i];
        if(x==y)return x;
        for(int i=19;i>=0;i--){
            if(f[x][i]!=f[y][i])
                x=f[x][i],y=f[y][i];
        }
        return f[x][0];
    }
}t1,t2;
int n,k,a[N];
int main(){
    cin>>n>>k;
    for(int i=1;i<=k;i++)cin>>a[i];
    for(int i=1;i<=n;i++)cin>>t1.w[i];
    for(int i=2;i<=n;i++){
        int x;cin>>x;
        t1.G[i].push_back(x);
        t1.G[x].push_back(i);
    }
    for(int i=1;i<=n;i++)cin>>t2.w[i];
    for(int i=2;i<=n;i++){
        int x;cin>>x;
        t2.G[i].push_back(x);
        t2.G[x].push_back(i);
    }
    t1.dfs(1,0);t2.dfs(1,0);
    t1.s[1]=t2.s[1]=a[1];
    for(int i=2;i<=k;i++){
        t1.s[i]=t1.lca(t1.s[i-1] , a[i]);
        t2.s[i]=t2.lca(t2.s[i-1] , a[i]);
    }
    t1.h[k]= t2.h[k]=a[k];
    for(int i=k-1;i>=1;i--){
        t1.h[i]=t1.lca(t1.h[i+1] , a[i]);
        t2.h[i]=t2.lca(t2.h[i+1] , a[i]);
    }
    int ans=0;

    for(int i=1;i<=k;i++){
        int w1,w2;
        if(i==1)w1=t1.w[t1.h[2]],w2=t2.w[t2.h[2]];
        if(i==k)w1=t1.w[t1.s[k-1]],w2=t2.w[t2.s[k-1]];
        if(i>1&&i<k)w1=t1.w[t1.lca(t1.s[i-1] , t1.h[i+1])];
        if(i>1&&i<k) w2=t2.w[t2.lca(t2.s[i-1] , t2.h[i+1])];
```

```
        if(w1>w2)ans++;
    }
    cout<<ans;
    return 0;
}
```

应用二、最小瓶颈生成树

题目：A2215

给定一个包含 n 个节点和 m 条边的图，每条边有一个权值。

你的任务是回答 k 个询问，每个询问包含两个正整数 s 和 t 表示起点和终点，要求寻找从 s 到 t 的一条路径，使得路径上权值最大的一条边权值最小。

【解析】

结论 1：要求任意两点路径最大权值最小，则 S-T 的路径每条边尽可能小，答案一定在最小生成树上

证明：设边 e 是不在最小生成树上的 s-t 的路径，设 s-t 最小生成树的路径最大权值边 e’,则 e'<e.e'比 e 更优。

根据上述结论，我们可以先求出最小生成树，再 dfs 求任意两点的答案，时间复杂度 m*logm+q*n；进一步优化：因为我们只需要求 s-t 路径的最大边权，可以用 LCA 爬山法来优化，时间复杂度：m*logm+q*log n

【参考代码】

```
#include<bits/stdc++.h>
using namespace std;
#define N 10010
#define M 100010
const int inf=1e8;
int n,m,q,cnt=0,first[M];
int f[N];
int first1[M],cnt1=0;
struct edge{
    int u,v,w,nxt;
}e[M],e1[M];

void add(int u,int v,int w){
    e1[++cnt].u=u;e1[cnt].v=v;e1[cnt].w=w;
    e1[cnt].nxt=first1[u];first1[u]=cnt;
}

bool cmp(edge a,edge b){
    return a.w<b.w;
}

int getfa(int x){
    if(f[x]==x) return x;
    else return f[x]=getfa(f[x]);
}
```

```
void merge(int a,int b){
    int dx=getfa(a);
    int dy=getfa(b);
    f[dx]=dy;
}
void kruskal(){
    int total=0;
    for(int i=1;i<=n;i++) f[i]=i;
    for(int i=1;i<=m;i++){
        if(getfa(e[i].u)!=getfa(e[i].v)){
            merge(e[i].u,e[i].v);
            add(e[i].u,e[i].v,e[i].w);
            add(e[i].v,e[i].u,e[i].w);
            total++;
        }
        if(total==n-1) break;
    }
}

int fa[N][20],dis[N][20]={},dep[N];
bool vis[N];

void dfs(int u){
    vis[u]=true;
    for(int i=1;(1<<i)<=dep[u];i++){
        fa[u][i]=fa[fa[u][i-1]][i-1];
        dis[u][i]=max(dis[fa[u][i-1]][i-1],dis[u][i-1]);
    }
    for(int i=first1[u];i;i=e1[i].nxt){
        int v=e1[i].v;
        if(!vis[v]){
            dep[v]=dep[u]+1;
            fa[v][0]=u;
            dis[v][0]=e1[i].w;
            dfs(v);
        }
    }
}

int lca(int a,int b){
    int ret=-inf;
    if(dep[a]<dep[b]) swap(a,b);
    int t=dep[a]-dep[b];
```

```
    for(int i=0;(1<<i)<=t;i++){
        if(t&(1<<i)){
            ret=max(ret,dis[a][i]);
            a=fa[a][i];
        }
    }
    if(a==b) return ret;
    for(int j=19;j>=0;j--)
        if(fa[a][j]!=fa[b][j]){
            ret=max(ret,max(dis[a][j],dis[b][j]));
            a=fa[a][j];
            b=fa[b][j];
        }
    return max(ret,max(dis[a][0],dis[b][0]));
}

int query(int u,int f){
    int t=dep[u]-dep[f],ans=-0x3f3f3f3f;
    for(int i=0;(1<<i)<=t;i++)
        if(t&(1<<i)){
            ans=max(ans,dis[u][i]);
            u=fa[u][i];
        }
    return ans;
}

int main(){
    cin>>n>>m>>q;
    for(int i=1;i<=m;i++)
        cin>>e[i].u>>e[i].v>>e[i].w;
    for(int i=1;i<=n;i++){
        e[++m].u=i;e[m].v=n+1;e[m].w=inf;
    }
    n++;
    sort(e+1,e+m+1,cmp);
    kruskal();
    dfs(n);
    while(q--){
        int u,v;
        cin>>u>>v;
        int ans=lca(u,v);
        printf("%d\n",ans==inf?-1:ans);
    }
    return 0;
```

```
}
```

小结：本节介绍了 3 个求 LCA 的算法，其中倍增法是最常用的方法，使用面最广。另外还有树链剖分的方法可以求 LCA，这个在以后的学习中会有相应的介绍。