K题题解报告：倍增法LCA

> 大致题意：无向生成树中求任意两个节点的最短路径。

一般来说，求任意两个节点的最短路径最常用的办法是 弗洛伊德算法，但是考虑到本题数据量非常大，同时具有生成树的特点（只有n-1条路且互通，即不会有环）。

利用这一点，我们可以任取某节点为根节点来构建生成树，每个节点的深度即为他们与根节点之间的距离，而任意两个节点之间的距离可以借助根节点由该公式表示（假如ai,aj的最小公共祖先是b）:

> dist[a-b]=dist[a]+dist[b]-2*dist[b]//应该很好理解吧。

于是，这个问题就转化成了多次求最小公共祖先的问题。

同时，求每个节点的深度本身就属于lca算法中的一部分。

求lca最常用的两种方法是 **tarjan算法**和**倍增法**。在这里我采用倍增法来阐述本篇题解。

倍增法可以看作是lca朴素算法的一个优化，前者是跳跃性地向上寻找祖先，而后者则是一步一步往上爬（很耗时间）。

倍增法最核心的数组：

> fa[i][j]//这个数组的意思是i的第2^ j个祖先。

例如fa[i][0]就是i的爸爸；fa[i][1]就是i的2^ 1祖先，即他的爸爸的爸爸，爷爷；

fa[i][2]就是i的2^ 2祖先，就是他爷爷的爷爷。

注意观察上述式子，我们有：

> fa[i][j]=fa[fa[i][j-1]][j-1]//意思就是i的2^ j次方祖先就是i的2^(j-1)次方祖先的j-1次方祖先（注意参考上面例子里"爸爸的爸爸"、"爷爷的爷爷"）。

```
for(int i=1;(1<<i)<depth[pos];++i){//注意循环条件，其实就是2^i<depth[pos]
 fa[pos][i]=fa[fa[pos][i-1]][i-1];
}
```

有个这个数组以后，我们在向上查找最小公共祖先时就可以一次跳多层了，进而大大缩小搜索时间。

再详细讲点具体怎么找最小公共祖先吧。

首先先比较输入的两个节点的深度，然后让比较深的那个先往上跳，直到与另一个节点保持在了同一深度（此时仍可利用fa数组，注意要先找到比较深的节点的深度范围，依此来确定如何利用fa数组使二者达到同一深度）。

然后两个一起往上跳，跳的距离从大到小，如果跳过头了就不更新（**fa[node1][i]==fa[node2][i]**）。这样做的结果是，循环结束后两个节点的父节点都是最小公共祖先，这时候return其中任意一个就行了。

这里贴个算法步骤（解题时想到了记下来的）

【倍增法LCA】

1. 查询某两个房屋的距离时
2. 利用LCA找到两个房屋的最小公共祖先
3. 假设为c点
4. 则根据公式dist[a-b]=dist[LCA-a]+dist[LCA->b]-2*dist[LCA->c]
5. 时间复杂度O(nlogn)

> **整个算法的时间复杂度O(nmlogn)：**
>
> 其实就是倍增法LCA的时间复杂度乘以询问次数，每次询问找LCA时候是nlogn，一共m次询问，乘起来就是nmlogn。

AC代码如下：

```
#include<iostream>
#include<cstring>
const int maxn=1e6+5;

struct edge{
 int to;
 int next;
}e[maxn<<1];
int fa[maxn][25],head[maxn],depth[maxn];
int n,m,tot;

void add_edge(int u,int v);
void dfs(int pos,int father,int dep);
int lca(int node1,int node2);
int main(){
 scanf("%d%d",&n,&m);
 memset(head,-1,sizeof head);
 for(int i=1;i<n;++i){
  int u,v;
  scanf("%d%d",&u,&v);
  add_edge(u,v);
  add_edge(v,u);
 }
 dfs(1,0,0);
 for(int i=0;i<m;++i){
  int node1,node2;
  scanf("%d%d",&node1,&node2);
  printf("%d\n",depth[node1]+depth[node2]-2*depth[lca(node1,node2)]);
 }
 return 0;
}

void add_edge(int u,int v){
 e[tot].to=v;
 e[tot].next=head[u];
 head[u]=tot++;
}
```

```
void dfs(int pos,int father,int dep){
 fa[pos][0]=father;
 depth[pos]=dep;
 for(int i=1;(1<<i)<depth[pos];++i){
  fa[pos][i]=fa[fa[pos][i-1]][i-1];
 }
 for(int i=head[pos];i!=-1;i=e[i].next){
  if(e[i].to!=father)
   dfs(e[i].to,pos,dep+1);
 }
}
int lca(int node1,int node2){
 if(depth[node1]<depth[node2]){
  int temp=node1;
  node1=node2;
  node2=temp;
 }
 int i=-1,j;
 for(;(1<<(i+1))<=depth[node1];)
  ++i;
 for(j=i;j>=0;--j){
  if(depth[node1]-(1<<j)>=depth[node2])
   node1=fa[node1][j];
 }
 if(node1==node2)
  return node1;
 for(;i>=0;--i){
  if(fa[node1][i]!=fa[node2][i]){
   node1=fa[node1][i];
   node2=fa[node2][i];
  }
 }
 return fa[node1][0];
}
```