树及树的应用

1、树的基础

由于每次读入 x 一定是 y 的父，因此用数组计数法记录每个父结点对应子结点的数量。

```
#include <bits/stdc++.h>//1-2164    子结点的数量    javacn
using namespace std;

int n,a[110];// 记录每个父有几个子

int main()
{
	cin>>n;
	int x,y;
	for(int i = 1;i < n;i++)
	{
		cin>>x>>y;
		a[x]++;
	}

	// 输出结果
	for(int i = 1;i <= n;i++)
	{
		cout<<a[i]<<" ";
	}
	return 0;
}
```

对于每个结点而言，只有 1 条边是对应父结点的，其余边都是对应子结点的

因此记录每个点对应的边的数量（除了根结点以外）

最终子结点的数量 = 边的总数 - 1

```
#include <bits/stdc++.h>//2-2165    子结点的数量（2）     javacn
using namespace std;

/*
 对于每个结点而言，只有 1 条边是对应父结点的，其余边都是对应子结点的
 因此记录每个点对应的边的数量
 最终子结点的数量 = 边的总数 - 1
*/
int n,a[110];
int main()
{
	cin>>n;
	int x,y;
	for(int i = 1;i <= n - 1;i++)
	{
		cin>>x>>y;
		a[x]++;
		a[y]++;
	}

	a[1]++;//1 是特例，没有边对应父结点
	for(int i = 1;i <= n;i++)
	{
		cout<<a[i] - 1<<" ";
	}
	return 0;
}
```

求出每个结点的孙子的数量，打擂台，求最大。

```
#include <bits/stdc++.h>//3-1775    谁的孙子最多    javacn
using namespace std;
vector<int> a[100010];
int n,x,t;
int ma,r,c;//ma 最多的孙子数，r 最多的孙子结点的编号 //c 每个结点的孙子的数量
int main()
{
	cin>>n;
	for(int i = 1;i <= n;i++)
	{
		cin>>x;// 第 i 个结点的子结点的数量
		for(int j = 1;j <= x;j++)
		{
			cin>>t;
			a[i].push_back(t);
		}
	}
	for(int i = 1;i <= n;i++)
	{
		c = 0;
		for(int j = 0;j < a[i].size();j++)
		{
			// 求孙子的总数量
			c = c + a[a[i][j]].size();
		}
		if(c > ma)
		{
			ma = c;
			r = i;
		}
	}
	cout<<r<<" "<<ma;
	return 0;
}
```

```
/*
树上结点的编号不一定是连续的
如果有多个结点的子结点都是最多的，则输出编号最大的那个
按照编号从小到大，输出这些孩子的编号
*/
```

```cpp
#include <bits/stdc++.h>//4-2188    找树根    javacn
using namespace std;
int fa[1010];// 存储每个结点的父结点编号
vector<int> a[1010];// 存储每个结点的子结点
int n,ma = 0;//ma 表示子结点最多的结点的编号

int main()
{
	cin>>n;
	int x,y;
	// 读入 n-1 个父子关系
	for(int i = 1;i <= n - 1;i++)
	{
		cin>>x>>y;
		fa[y] = x;// 维护父子关系
		// 存储每个结点的子结点
		a[x].push_back(y);
		// 打擂台求子结点最多的结点编号
		if(a[x].size()>a[ma].size()||(a[x].size()==a[ma].size()&&x>ma))
		{
			ma = x;
		}
	}
	int root = x;// 从一个存在的结点向上找根
	while(fa[root] != 0) root = fa[root];
	cout<<root<<endl;
	cout<<ma<<endl;
	// 对 ma 的子结点排序
	sort(a[ma].begin(),a[ma].end());
	for(int i = 0;i < a[ma].size();i++)
	{
		cout<<a[ma][i]<<" ";
	}
	return 0;
}
```

解法一：使用 vector 数组维护每个结点的子结点有哪些，求出每个结点的孙子结点的数量，打擂台，求最大。

```
#include <bits/stdc++.h>//5-1776-1    谁的孙子最多 II      javacn
using namespace std;
vector<int> a[100010];
int n,x,t;
//ma 最多的孙子数，r 最多的孙子结点的编号
//c 每个结点的孙子的数量
int ma,r,c;
int main()
{
	cin>>n;
	// 从 2 号结点开始处理
	for(int i = 2;i <= n;i++)
	{
		cin>>x;// 第 i 个结点的父结点
		a[x].push_back(i);
	}
	for(int i = 1;i <= n;i++)
	{
		c = 0;
		for(int j = 0;j < a[i].size();j++)
		{
			// 求孙子的总数量
			c = c + a[a[i][j]].size();
		}
		if(c > ma)
		{
			ma = c;
			r = i;
		}
	}
	cout<<r<<" "<<ma;
	return 0;
}
```

解法二：统计每个结点的孙子的数量，打擂台求孙子最多的结点。

```
#include <bits/stdc++.h>//5-1776-2  谁的孙子最多 II     javacn
using namespace std;

int fa[10010];
int r[10010];// 存储每个结点有几个孙子
int n,x;
int m;// 孙子最多的结点编号

int main()
{
	cin>>n;
	// 读入 i 号结点的父元素
	for(int i = 2; i <= n; i++)
	{
		cin>>x;
		fa[i] = x;
	}

	// 循环求出每个结点的孙子有几个
	for(int i = 1;i <= n;i++)
	{
		if(fa[fa[i]]!=0) r[fa[fa[i]]]++;
	}

	for(int i = 1;i <= n;i++)
	{
		if(r[i] > r[m])
		{
			m = i;
		}
	}
	cout<<m<<" "<<r[m];
	return 0;
}
```

2、树的深度和大小

解法一：深搜求解高度

```
#include <bits/stdc++.h>//1-2170 -1    树的高度    javacn
using namespace std;

int a[110][110];// 邻接矩阵存储树
int n,ma = 0;//ma 表示最大深度

// 深搜求每个结点的深度
void dfs(int x,int dep)
{
	ma = max(ma,dep);
	// 讨论 x 结点可以去哪些结点
	for(int i = 1;i <= n;i++)
	{
		// 有边表示可行
		if(a[x][i]==1)
		{
			dfs(i,dep+1);
		}
	}
}
```

```
int main()
{
	cin>>n;
	//边：父子关系
	int x,y;
	for(int i = 1;i <= n - 1;i++)
	{
		cin>>x>>y;
		a[x][y] = 1;//有向图
	}

	//从根开始深搜，默认深度为1
	dfs(1,1);

	cout<<ma;
	return 0;
}
```

解法二：求出每个结点的深度，求出最深的深度就是树的高度。

```
#include <bits/stdc++.h>//1-2170 -2    树的高度    javacn
using namespace std;
/*
1. 维护结点父子关系，求每个结点的深度
2. 打擂台求最大 */
int n,ma = 0;//ma 表示最大深度
int fa[110];// 表示每个结点的父是谁
int main()
{
	cin>>n;
	// 边：父子关系
	int x,y;
	for(int i = 1;i <= n - 1;i++)
	{
		cin>>x>>y;
		fa[y] = x;
	}
	// 求每个结点的深度
	int cnt,t;
	for(int i = 1;i <= n;i++)
	{
		cnt = 1;// 每个结点的初始深度都为 1
		t = i;// 过渡 i 的值
		// 当 t 不是根，向上找，找到根
		while(fa[t] != 0)
		{
			t = fa[t];
			cnt++;
		}
		ma = max(ma,cnt);
	}
	cout<<ma;
	return 0;
}
```

递归求解每个结点的父结点以及深度，在递归过程中，要注意判断，要去的点不能是该结点的父，防止死循环。

```
#include <bits/stdc++.h>//2-2171    树的高度（2）    javacn
using namespace std;

const int N = 30;
//fa：父结点 a：邻接矩阵 dep：深度
int fa[N],a[N][N],dep[N];
int n;

// 深搜求：每个结点的父结点编号、深度
void dfs(int x)
{
    // 讨论 x 点能去的点
    for(int i = 1;i <= n;i++)
    {
        // 有边 && 要去的点不是父结点
        if(a[x][i]==1&&i!=fa[x])
        {
            //i 的父是 x
            fa[i] = x;// 标记元素的父
            dep[i] = dep[x] + 1;// 求深度

            dfs(i);
        }
    }
}
```

```
int main()
{
	cin>>n;
	int x,y;
	for(int i = 1;i <= n - 1;i++)
	{
		cin>>x>>y;
		// 按无向图来建边
		a[x][y] = 1;
		a[y][x] = 1;
	}

	dep[1] = 1;// 初始化
	dfs(1);

	// 输出结果
	for(int i = 2;i <= n;i++)
	{
		cout<<i<<" "<<fa[i]<<" "<<dep[i]<<endl;
	}
	return 0;
}
```

邻接表求解：

```
#include <bits/stdc++.h>//3-2166-1    子树的大小及深度    javacn
using namespace std;
const int N = 110;
struct node{
	int to,next;
}a[N * 2];
int pre[N],k = 0;
int si[N];//si[i]：以 i 为根的子树大小
int dep[N];// 深度
int n;
void add(int u,int v)
{
	a[++k] = {v,pre[u]};
	pre[u] = k;
}
// 深搜
// 返回值：代表以 x 结点为根的子树大小
int dfs(int x,int fath)
{
	dep[x] = dep[fath] + 1;// 从父到子：更新深度
	// 初始值为 1，以 x 为根的子树大小至少有 1 个，就是自己
	si[x] = 1;
	for(int i = pre[x];i;i = a[i].next)
	{
		int to = a[i].to;//to 是 x 的子
		if(to != fath)
		{
			// 将每个子结点对应子树大小都要加到总和上
			si[x] += dfs(to,x);
		}
	}

	return si[x];
}
```

```
int main()
{
	scanf("%d",&n);
	int x,y;
	for(int i = 1;i <= n - 1;i++)
	{
		scanf("%d%d",&x,&y);
		add(x,y);
		add(y,x);
	}

	dfs(1,0);

	//输出
	for(int i = 1;i <= n;i++)
	{
		printf("%d %d\n",si[i],dep[i]);
	}
	return 0;
}
```

```
// 并查集解决树的大小和深度
#include<iostream>//3-2166-2    子树的大小及深度    zzy123
using namespace std;
const int MAX=100005;
int head[MAX],depth[MAX],size[MAX];

struct
{
	int nxt,to;
}edge[MAX];

int cnt=1;

void add_edge(int u,int v)
{
	edge[cnt].to=v;
	edge[cnt].nxt=head[u];
	head[u]=cnt++;
}

int dfs(int x,int h)
{
	depth[x]=h;
	int lu=1;
	for(int i=head[x];i;i=edge[i].nxt)
	{
		int v=edge[i].to;
		if(depth[v]) continue;
		lu+=dfs(v,h+1);
	}
	return (size[x]=lu);
}
```

```
int duru()
{
	int T,m1,m2;
	scanf("%d",&T);
	for(int i=1;i<=T-1;i++)
	{
		scanf("%d%d",&m1,&m2);
		add_edge(m1,m2);
		add_edge(m2,m1);
	}
	return T;
}

void shuchu(int T)
{

	for(int i=1;i<=T;i++)
	{
		cout<<size[i]<<" "<<depth[i]<<endl;
	}
}

int main()
{
	int num=duru();
	dfs(1,1);
	shuchu(num);
}
```

深搜求树的宽度和高度，以及每个结点的深度。

求出 uv 两点的最近的公共祖先，求出两点距离公共祖先的距离，就可求两点之间的距离。

```
#include<bits/stdc++.h>//4-2206    树的宽高及两点的距离    javacn
using namespace std;

const int N = 1010;
//width：每层的结点的数量
int fa[N],dep[N],width[N];
int a[N][N];// 邻接矩阵
bool vis[N];// 标记哪些点访问过
int maxd,maxw;
int n;

// 求每个结点的父元素，深度，每一层的结点数量
void dfs(int x,int f)
{
	dep[x] = dep[f] + 1;
	width[dep[x]]++;
	fa[x] = f;// 存储每个元素的父
	maxd = max(maxd,dep[x]);// 求最大深度
	maxw = max(maxw,width[dep[x]]);// 求最大宽度

	for(int i = 1;i <= n;i++)
	{
		if(a[x][i] && i != f)
		{
			dfs(i,x);
		}
	}
}
```

```
int main()
{
	cin>>n;
	int x,y;
	for(int i = 1;i <= n - 1;i++)
	{
		cin>>x>>y;
		a[x][y] = 1;
		a[y][x] = 1;
	}
	cin>>x>>y;

	dfs(1,0);// 深搜每个结点
	cout<<maxd<<endl<<maxw<<endl;

	// 求 x 和 y 的公共祖先
	// 从其中一个点向根爬树
	int u = x,v = y,r;//r 表示最近的公共祖先的编号
	vis[x] = true;
	while(fa[x] != 0)
	{
		x = fa[x];
		vis[x] = true;
	}

	// 从 y 向上求最近的公共祖先
	r = y;
	while(vis[r] != true)
	{
		r = fa[r];
	}

	cout<<dep[u]-dep[r]+(dep[v]-dep[r]);
    return 0;
}
```

如果 x 和 y 之间有一条边，我们判断一下，x 和 y 两个点是否是同一种牛奶，如果是的话，将 x 和 y 合并到同一个集合，由于只有 2 种牛奶，因此到最后只会有 2 种集合。

询问：从 t1 到 t2 是否能喝到自己喜欢的牛奶 c，只需要判断 2 种情况：

（1）t1 如果和 t2 不在一个集合，一定能喝到，因为从 t1 到 t2 一定会经过不同种类的牛奶；

（2）t1 和 t2 在一个集合，说明这两个点的牛奶一样，那么判断其中一个点的牛奶是否是自己喜欢的牛奶；

其余的情况表示喝不到。

```
#include<bits/stdc++.h>//5-2199    Milk Visits    javacn
using namespace std;

const int N = 1e5 + 10;
int fa[N];
char s[N];
int n,m;

int find(int x)
{
	if(x == fa[x]) return x;
	else return fa[x] = find(fa[x]);
}

void merge(int x,int y)
{
	int fx = find(x);
	int fy = find(y);
	if(fx != fy)
	{
		fa[fx] = fy;
	}
}
```

```
int main()
{
	scanf("%d%d",&n,&m);
	scanf("%s",s+1);// 下标从 1 开始

	// 初始化
	for(int i =1;i <= n;i++) fa[i] = i;
	// 读入 n-1 对关系
	int x,y;
	for(int i = 1;i <= n - 1;i++)
	{
		scanf("%d%d",&x,&y);
		if(s[x] == s[y]) merge(x,y);
	}

	//m 次询问
	char c[2];
	while(m--)
	{
		scanf("%d%d%s",&x,&y,c);
		if(find(x)!=find(y) || c[0] == s[x]) printf("%d",1);
		else printf("%d",0);
	}
	return 0;
}
```

3、树的共同祖先

解法一：先标记 x 到根的每个点，在从 y 向上找到第一个被标记的点

```
#include <bits/stdc++.h>//1-2167-1  树的公共祖先（LCA）  javacn
using namespace std;
const int N = 110;
int fa[N];// 父子关系
bool vis[N];// 标记哪些点访问过
int n;
int main()
{
	cin>>n;
	// 将每个元素的父先标记为自己，认为这是一个不可能的值
	for(int i = 1;i <= n;i++) fa[i] = i;
	int x,y,a,b;
	cin>>x>>y;// 求 xy 的最近公共祖先
	for(int i = 1;i <= n - 1;i++)
	{
		cin>>a>>b;
		fa[a] = b;//a 的父是 b
	}
	vis[x] = true;// 标记出发点	// 从 x 出发，将 x 到根的所有点，标记
	while(fa[x] != x)
	{
		x = fa[x];
		vis[x] = true;
	}
	// 从 y 点向上，找到第 1 个被标记的点，就是最近的公共祖先
	while(!vis[y])
	{
		y = fa[y];
	}
	cout<<y;
	return 0;
}
```

解法二：

（1）分别求出 x、y 到根路径中每个结点的深度；

（2）如果两个结点重叠，那么得知公共祖先；

（3）如果不重叠，选择深度更大的结点，向父元素移动，重复上述过程，直到重叠。

```
#include <bits/stdc++.h>//1-2167-2  树的公共祖先（LCA）  javacn
using namespace std;

const int N = 110;
int fa[N];// 父子关系
int dep[N];// 求结点深度
int n;

// 从某个结点向上深搜一直到根，在后退时求每个结点的深度
void dfs(int x)
{
	// 递归的出口是根，根的默认深度为 1
	if(fa[x] == x) dep[x] = 1;
	else
	{
		dfs(fa[x]);
		dep[x] = dep[fa[x]] + 1;
	}
}
```

```
int main()
{
	cin>>n;
	// 将每个元素的父先标记为自己，认为这是一个不可能的值
	for(int i = 1;i <= n;i++) fa[i] = i;

	int x,y,a,b;
	cin>>x>>y;// 求 xy 的最近公共祖先
	for(int i = 1;i <= n - 1;i++)
	{
		cin>>a>>b;
		fa[a] = b;//a 的父是 b
	}

	// 分别求 x、y 到根路径中每个结点的深度
	dfs(x);
	dfs(y);

	// 从 x、y 分别向上移动，直到遇到第一个公共祖先
	while(x != y)
	{
		if(dep[x] > dep[y]) x = fa[x];
		else y = fa[y];
	}

	cout<<x;
	return 0;
}
```

（1）求出每个结点的深度；

（2）如果两个结点重叠，那么得知公共祖先；

（3）如果不重叠，选择深度更大的结点，向父元素移动，重复上述过程，直到重叠。

```
#include <bits/stdc++.h>//2-2168  树的公共祖先（LCA）(2)     javacn
using namespace std;
const int N = 110;
/*
      fa：每个元素的父元素 dep：每个元素的深度
      pre：每个点的最后一条边
      k：边的数量
*/
int fa[N],dep[N],pre[N],k;

struct node{
      int from,to,next;
}a[N*2];

int x,y,t1,t2,n;

// 加边
void add(int u,int v)
{
      k++;
      a[k].from = u;
      a[k].to = v;
      a[k].next = pre[u];
      pre[u] = k;
}

// 深搜求元素的父和深度
```

```
// 深搜求元素的父和深度
void dfs(int x,int fath)
{
	fa[x] = fath;
	dep[x] = dep[fath] + 1;

	for(int i = pre[x];i != 0;i = a[i].next)
	{
		if(a[i].to != fath)
		{
			dfs(a[i].to,x);
		}
	}
}
int main()
{
	cin>>n>>x>>y;
	// 读入关系
	for(int i = 1;i <= n - 1;i++)
	{
		cin>>t1>>t2;
		add(t1,t2);
		add(t2,t1);
	}
	dfs(1,0);// 求每个点的父及深度

	// 从较深的元素向上找
	while(x != y)
	{
		if(dep[x] > dep[y]) x = fa[x];
		else y = fa[y];
	}
	cout<<x;
	return 0;
}
```

4、树的直径中心重心

求出树的直径的两个端点 x 和 y，假设 x 是根，y 是叶子，那么从叶子 y 向上爬树，直到遇到中心点。

如果直径上结点数量是奇数，中心点只有一个；

如果直径上结点数量是偶数，中心点有两个。

```
#include <bits/stdc++.h>//2-2207    树的中心      javacn
using namespace std;
const int N = 1e5 + 10;
//fa: 父子关系 dep: 结点深度
//pre: 以每个顶点为出发点的最后一条边
int fa[N],dep[N],pre[N];
struct node{
      int from,to,next;
}a[N*2];
int n,k;//k 表示 a 数组的下标
// 加边
void add(int u,int v)
{
      k++;
//    a[k].from = u;
      a[k].to = v;
      a[k].next = pre[u];
      pre[u] = k;
}
// 深搜求父子关系及深度
```

```
// 深搜求父子关系及深度
void dfs(int x,int f)
{
	fa[x] = f;
	dep[x] = dep[f] + 1;

	// 讨论 x 能去的点
	for(int i = pre[x];i != 0;i = a[i].next)
	{
		// 如果去的点不是父
		if(a[i].to != f)
		{
			dfs(a[i].to,x);
		}
	}
}

int main()
{
	cin>>n;
	int x,y;
	for(int i = 1;i <= n - 1;i++)
	{
		cin>>x>>y;
		add(x,y);
		add(y,x);// 双向建边
	}

	// 从 1 号结点开始深搜，求每个结点的深度和父子关系
```

```
    // 从 1 号结点开始深搜，求每个结点的深度和父子关系
    dfs(1,0);
    // 求直径的一个端点 x
    x = 1;
    for(int i = 2;i <= n;i++)
    {
        if(dep[i] > dep[x])     x = i;
    }

    // 从 x 点出发（以 x 为根）求另一个端点
    dfs(x,0);
    y = 1;
    for(int i = 2;i <= n;i++)
    {
        if(dep[i] > dep[y])     y = i;
    }

    // 通过爬树得到中心点的编号
    int d = dep[y];
    // 如果深度为奇数，中心点有一个
    if(d % 2 != 0)
    {
        for(int i = 1;i <= d / 2;i++)       y = fa[y];
        cout<<y;
    }
    else
    {
        for(int i = 1;i <= d / 2 - 1;i++)       y = fa[y];
        if(y < fa[y])     cout<<y<<" "<<fa[y];
        else        cout<<fa[y]<<" "<<y;
    }
    return 0;
}
```

求出每个结点删除后的最大子树的大小，并求出该值的最小值，最后循环看哪些树删除后子树大小 == 所求的最小值，哪些结点就是重心。

```
#include <bits/stdc++.h>//3-2190  树的重心     javacn
using namespace std;
const int N = 1e5 + 10;
//si：存储每个结点对应的子树大小
//r：存储以每个结点为根对应的最大子树的大小
int fa[N],pre[N],si[N],r[N];
struct node{
	int from,to,next;
}a[N*2];
int n,k;
// 加边
void add(int u,int v)
{
	k++;
	a[k].to = v;
	a[k].next = pre[u];
	pre[u] = k;
}
// 求出每个结点对应的子树的大小和父子关系
int dfs(int x,int f)
{
	si[x] = 1;
	fa[x] = f;
	// 讨论 x 点可以去的结点
	for(int i = pre[x];i != 0;i = a[i].next)
	{
		if(a[i].to != f)
		{
			si[x] = si[x] + dfs(a[i].to,x);
		}
	}
	return si[x];
}
```

```
int main()
{
	cin>>n;
	// 读入边
	int x,y;
	for(int i = 1;i <= n - 1;i++)
	{
		cin>>x>>y;
		add(x,y);
		add(y,x);
	}
	dfs(1,0);    // 以任意一个点为根，求出每个结点对应的子树的大小
	// 求出以每个结点为根，对应的最大子树的大小      // 以及最大子树的最小值
	int mi = INT_MAX;
	for(int i = 1;i <= n;i++)
	{
		r[i] = n - si[i];
		// 循环 i 号结点对应的子树
		for(int j = pre[i];j != 0;j = a[j].next)
		{
			// 如果 a[j].to 的父是 i，也就是 a[j].to 是 i 的子树
			if(fa[a[j].to] == i)
			{
				r[i] = max(r[i],si[a[j].to]);
			}
		}
		mi = min(mi,r[i]);            //cout<<i<<" "<<r[i]<<endl;
	}
	// 判断几号结点对应的子树的最大值为 mi，说明就是重心
	for(int i = 1;i <= n;i++)
	{
		if(r[i] == mi) cout<<i<<" ";
	}
	return 0;
}
```