树及树的应用

1、树的基础

2164. 子结点的数量

给定一棵树中的若干父结点和子结点的关系描述（结点 1 是树根），请问该树中，每个结点有多少个子结点。

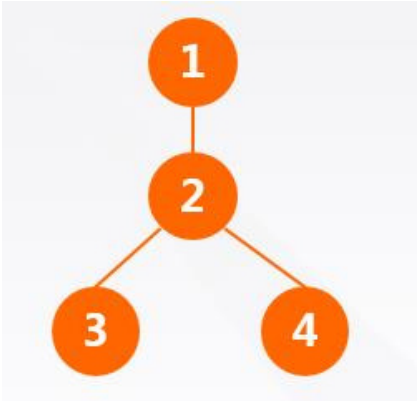
比如：读入父子关系如下，先读入父结点，再读入子结点。

1 2

2 3

2 4

根据读入，可以画出树如下。



因此每个结点的子结点的数量分别是：

1　　2　　0　　0 。

输入

第 1 行，读入一个整数 n ，表示树中结点的数量，树中的结点编号也是 1~n。（n ≤ 100）

接下来 n~1 行，每行有一对父子关系 x, y，x 表示父结点的编号，y 表示子结点的编号。

输入数据保证一定合法，能够形成一棵树，且不存在重复的父子关系的读入。

输出 n 个数，用空格隔开，表示按照编号从小到大的顺序，输出每个结点子结点的数量。

输入

4

1 2

2 3

2 4

输出

1 2 0 0

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];// 存储每个结点有几个子结点
int main()
{
	cin>>n;
	int x,y;
	for(int i = 1;i < n;i++)
	{
		cin>>x>>y;
		a[x]++;// 统计父结点出现的次数
	}

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

2165. 子结点的数量（2）

给定一棵树中的若干父结点和子结点的关系描述（结点 1 是树根），请问该树中，每个结点有多少个子结点。

比如：读入父子关系如下（请注意：本题读入的两个数 x、y ，不保证 x 是 y 的父）。

2 1

2 3

2 4

因此每个结点的子结点的数量分别是：

1　　2　　0　　0 。

输入

第 1 行，读入一个整数 n ，表示树中结点的数量，树中的结点编号也是 1~n。（n≤100）

接下来 n~1 行，每行有一对父子关系 x，y，不保证 x 是 y 的父。

输入数据保证一定合法，能够形成一棵树，且不存在重复的父子关系的读入。

输出

输出 n 个数，用空格隔开，表示按照编号从小到大的顺序，输出每个结点子结点的数量。

输入

4

2 1

2 3

2 4

输出

1 2 0 0

输入

8

3 1

1 8

1 2

2 7

6 7

5 6

4 1

输出

4 1 0 0 0 1 1 0

输入

12

8 10

5 1

1 8

5 7

7 2

10 4

7 3

4 12

12 6

9 5

12 11

输出

2 0 0 1 2 0 2 1 0 1 0 2

```cpp
#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;
}
```

1775. 谁的孙子最多

给定一棵树，其中 1 号结点是根结点，问哪一个结点的孙子结点最多，有多少个。（孙子结点，就是儿子结点的儿子结点。）

输入

第一行一个整数 N（N ≤ 10000），表示树结点的个数。

此后 N 行,第 i 行包含一个整数 C_i,表示 i 号结点儿子结点的个数,随后共 C_i 个整数，分别表示一个 i 号结点的儿子结点（结点编号在 1~N 之间）。

输出

一行两个整数，表示孙子结点最多的结点，以及其孙子结点的个数，如果有多个，输出编号最小的（本题测试数据确保有解）。

输入

5

2 2 3

1 4

0

1 5

0

输出

1 1

```cpp
#include <bits/stdc++.h>//3-1775    谁的孙子最多
using namespace std;
vector<int> a[100010];// a[i] 存储结点 i 的所有子结点编号
int n, c, x;// ma：记录最多的孙子结点数量 r：记录拥有最多孙子结点的结点编号
int ma = INT_MIN, r;
int main()
{
	cin >> n;
	for (int i = 1; i <= n; i++)	// 读入每个结点的子结点信息
	{
		cin >> c; // 结点 i 的子结点数量
		for (int j = 1; j <= c; j++)
		{
			cin >> x; // x 是 i 的一个子结点
			a[i].push_back(x);
		}
	}

	int cnt;
	for (int i = 1; i <= n; i++)  // 遍历每个结点，计算其孙子结点总数
	{
		cnt = 0;
		for (int j = 0; j < a[i].size(); j++) // 遍历 i 的每一个子结点
		{		// 累加该子结点的子结点数量（即 i 的孙子结点）
			cnt += a[a[i][j]].size();
		}
		if (cnt > ma)		// 打擂台：更新最大值和对应结点编号
		{
			ma = cnt;
			r = i;
		}
	}
	cout << r << " "<< ma;
	return 0;
}
```

2188. 找树根

一棵树有 n 个结点，已知树上所有的父子结点关系，请问该树的根是几号结点，哪个结点的子结点最多，该结点有哪些子结点。

输入

第一行，有 1 个整数 n 代表结点数量（1<n ≤ 100）

接下来若干行；每行两个结点 x 和 y，表示 y 是 x 的孩子 (1 ≤ x,y ≤ 1000)

请注意：树上结点的编号不一定是连续的。

输出

第一行输出树根的编号。

第二行输出孩子最多的结点编号（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）。

第三行输出第二行求出的孩子最多的结点，有哪些孩子，按照编号从小到大，输出这些孩子的编号，用空格隔开。

输入

```
5
4 1
4 2
1 3
1 5
```

输出

```
4
4
1 2
```

输入

```
7
300 100
300 200
300 500
500 400
500 700
500 600
```

输出

```
300
500
400 600 700
```

```cpp
#include <bits/stdc++.h>//4-2188    找树根    javacn
using namespace std;
/*
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
*/
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;
	for(int i = 1;i <= n - 1;i++) // 读入 n-1 个父子关系
	{
		cin>>x>>y;
		fa[y] = x;// 维护父子关系
		a[x].push_back(y);              // 存储每个结点的子结点
		if(a[x].size()>a[ma].size()||(a[x].size()==a[ma].size()&&x>ma))
		{            // 打擂台求子结点最多的结点编号
			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;
	sort(a[ma].begin(),a[ma].end());    // 对 ma 的子结点排序
	for(int i = 0;i < a[ma].size();i++)
	{
		cout<<a[ma][i]<<" ";
	}
	return 0;
}
```

1776. 谁的孙子最多 II

给定一棵树，其中 1 号结点是根结点，问哪一个结点的孙子结点最多，有多少个。（孙子结点，就是儿子结点的儿子结点。）

输入

第一行一个整数 N（N ≤ 10000），表示树结点的个数。

此后 N-1 行，第 i 行包含一个整数 fi ，表示 i+1 号结点的父亲。

输出

一行两个整数，表示孙子结点最多的结点，以及其孙子结点的个数，如果有多个，输出编号最小的。

输入

5

1

1

2

4

输出

1 1

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

2170. 树的高度

一棵树有 n 个结点，结点编号为 1~n，其中 1 号结点为根结点，根结点的深度为 1，请问树的高度是多少。

输入

第一行是整数 n，表示结点数。（1 ≤ n ≤ 100）

后面若干行，每行两个整数 a b，表示 b 是 a 的子结点。

本题测试数据保证所有结点能构建为一棵树。

输出

求这棵树的高度。

输入

5

1 2

1 3

3 4

3 5

输出

3

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

void dfs(int x,int dep)// 深搜求每个结点的深度
{
	ma = max(ma,dep);
	for(int i = 1;i <= n;i++)	// 讨论 x 结点可以去哪些结点
	{
		if(a[x][i]==1)	// 有边表示可行
		{
			dfs(i,dep+1);
		}
	}
}
int main()
{
	cin>>n;
	int x,y;	// 边：父子关系
	for(int i = 1;i <= n - 1;i++)
	{
		cin>>x>>y;
		a[x][y] = 1;// 有向图
	}
	dfs(1,1);	// 从根开始深搜，默认深度为 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;
	int x,y;    // 边：父子关系
	for(int i = 1;i <= n - 1;i++)
	{
		cin>>x>>y;
		fa[y] = x;
	}

	int cnt,t;  // 求每个结点的深度
	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
		while(fa[t] != 0)        // 当 t 不是根，向上找，找到根
		{
			t = fa[t];
			cnt++;
		}
		ma = max(ma,cnt);
	}
	cout<<ma;
	return 0;
}
```

2171. 树的高度（2）

一棵树有 n 个结点，结点编号为 1~n，其中 1 号结点为根结点，根结点的深度为 1，请问每个结点的父结点是谁，每个结点的深度是多少。

输入

第一行是整数 n，表示结点数。（1 ≤ n ≤ 20）

后面若干行，每行两个整数 a、b，读入数据不保证结点 a 是结点 b 的父结点。

本题测试数据保证所有结点能构建为一棵树。

输出

输出有 n~1 行（根结点不需要输出），每行输出 3 个整数；

第 i 行输出：结点编号 i+1，编号为 i+1 的结点的父结点的编号，该结点的深度（三个整数之间用空格隔开）。

输入

5

2 1

4 3

1 3

3 5

输出

2 1 2

3 1 2

4 3 3

5 3 3

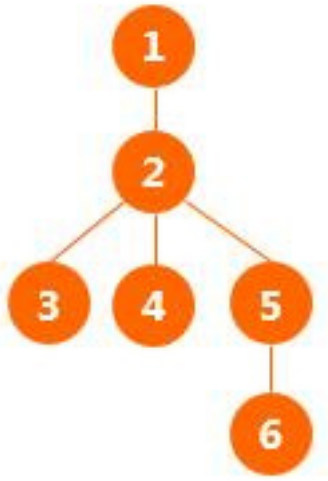
```cpp
#include <bits/stdc++.h>//2-2171 -1   树的高度（2）    javacn
using namespace std;
const int N = 30;
int fa[N], a[N][N], dep[N];// fa：父结点 a：邻接矩阵 dep：深度
int n;
void dfs(int x, int f) // 深搜求：每个结点的父结点编号、深度
{
	fa[x] = f;				// 标记元素的父
	dep[x] = dep[f] + 1;		// 求深度
	for (int i = 1; i <= n; i++)		// 讨论 x 点能去的点
	{
		if (a[x][i] == 1 && i != f)		// 有边 && 要去的点不是父结点
		{
			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;		// 按无向图来建边
		a[y][x] = 1;
	}
	dfs(1, 0);
	for (int i = 2; i <= n; i++)		// 输出结果
	{
		cout << i << " "<< fa[i] << " "<< dep[i] << endl;
	}
	return 0;
}
```

```cpp
#include <bits/stdc++.h>//2-2171-2    树的高度（2）    javacn
using namespace std;// 递归求解每个结点的父结点以及深度，
const int N = 30;//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。
int fa[N],a[N][N],dep[N];//fa：父结点 a：邻接矩阵 dep：深度
int n;
void dfs(int x)// 深搜求：每个结点的父结点编号、深度
{
	for(int i = 1;i <= n;i++)	// 讨论 x 点能去的点
	{
		if(a[x][i]==1&&i!=fa[x])		// 有边 && 要去的点不是父结点
		{
			fa[i] = x;// 标记元素的父		//i 的父是 x
			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;
		a[x][y] = 1;	// 按无向图来建边
		a[y][x] = 1;
	}
	dep[1] = 1;// 初始化
	dfs(1);
	for(int i = 2;i <= n;i++)	// 输出结果
	{
		cout<<i<<" "<<fa[i]<<" "<<dep[i]<<endl;
	}
	return 0;
}
```

2166. 子树的大小及深度

现在有一棵 n 个结点的树，结点 1 为这棵树的根，结点 1 的深度为 1，求出每棵子树的大小及每个结点的深度。

比如，有如下图所示的树：



该树中：

结点 1 对应的子树大小为 6，深度为 1。

结点 2 对应的子树大小为 5，深度为 2。

结点 3 对应的子树大小为 1，深度为 3。

结点 4 对应的子树大小为 1，深度为 3。

结点 5 对应的子树大小为 2，深度为 3。

结点 6 对应的子树大小为 1，深度为 4。

输入

输入有 n 行。

第 1 行有一个整数 n，代表结点的数量，结点的编号为 1~n。（1 ⩽ n ⩽ 100）

接下来有 n~1 行，每行有 2 个整数 x,y，表示结点 x 和 y 之间有一条边。（不保证 x 是 y 的父）

输出

输出有 n 行。

第 i 行输出 2 个整数，分别是以编号 i 为根的子树的大小，以及编号 i 对应的结点的深度。

输入

```
6
1 2
5 2
2 3
4 2
5 6
```

输出

```
6 1
5 2
1 3
1 3
2 3
1 4
```

```cpp
#include <bits/stdc++.h>//3-2166-1  解法一：vector<int>数组存储树
using namespace std;
const int N = 110;
vector<int> a[N];
int dep[N], si[N];//dep:深度 si:每个结点对应的子树的大小
int n;
int dfs(int x, int f) //深搜
{
	si[x] = 1; //1 表示是自己
	dep[x] = dep[f] + 1; //深度
	for (int i = 0; i < a[x].size(); i++)	//讨论 x 可以去的点
	{
		if (a[x][i] != f)		//x 能去的点为 a[x][i]
		{					//x 去的点不能是自己的父
			si[x] = si[x] + dfs(a[x][i], x);
		}
	}
	return si[x];
}
int main() {
	cin >> n;
	int x, y;
	for (int i = 1; i <= n - 1; i++)
	{
		cin >> x >> y;
		a[x].push_back(y);
		a[y].push_back(x);
	}
	dfs(1, 0);	//深搜求每个结点的深度以及每个结点对应的子树大小
	for (int i = 1; i <= n; i++)	//输出每个结点对应的子树大小和深度
	{
		cout << si[i] << " " << dep[i] << endl;
	}
	return 0;
}
```

```cpp
#include <bits/stdc++.h>//3-2166-2  解法二：邻接表存储树
using namespace std;
const int N = 110;
int n, si[N], de[N];//si：每棵树子树的大小 //de：深度
//ne：存储该子结点的上个节点
int tot, fi[N], to[N * 2], ne[N * 2];//to：存储某个结点的子结点
void link(int x, int y) {//x 指向 y 的边
	to[++tot] = y;
	ne[tot] = fi[x];
	fi[x] = tot;
}
void dfs(int x, int fa) {// 求每棵树的子树的大小
	si[x] = 1;// 每个结点的子树大小 = 其儿子的子树大小 + 1
	de[x] = de[fa] + 1;
	for (int i = fi[x]; i != 0; i = ne[i]) {
		if (to[i] != fa) {// 如果 i 号点的子是 i 号点的父，说明是倒过来的关系
			dfs(to[i], x);   // 之前已经讨论过，不需要再讨论
			si[x] += si[to[i]];
		}
	}
}
int main() {
	cin >> n;
	int x, y;
	for (int i = 1; i < n; i++) {
		cin >> x >> y;
		link(x, y);
		link(y, x);
	}
	dfs(1, 0); // 求根 1 的子树数量
	for (int i = 1; i <= n; i++) {
		cout << si[i] <<" " << de[i] << endl;
	}
	return 0;
}
```

邻接表求解：

```
#include <bits/stdc++.h>//3-2166-3    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>//3-2166-4    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);
}
```

2206. 树的宽高及两点的距离

给定一棵树的边的关系，结点 1 为该树的根，请问该树的宽度（同一层最多的结点数）、高度（根结点的高度为 1 ），以及树中两个结点 u 和 v 之间的最短距离是多少？

比如：下图所示的树，深度为 5，宽度为 3，结点 7 到结点 9 的最短距离为 5。

输入

第 1 行输入一个整数 n（n ≤ 1000）；

接下来 n~1 行，每行有 2 个整数 x 和 y，表示结点 x 和 y 之间有一条边（1 ≤ x,y ≤ n）。

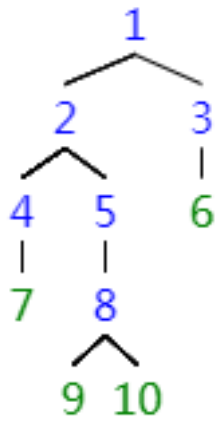
最后一行有 2 个整数 u 和 v ，表示求 u 和 v 之间最短距离。

输出

第 1 行输出该树的高度；

第 2 行输出该树的宽度；

第 3 行输出该树从结点 u 到结点 v 之间最短距离；



输入

10

2 1

1 3

2 4

5 2

3 6

7 4

8 5

9 8

10 8

7 9

输出

5

3

5

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;
}
```

2199. Milk Visits

Farmer John 计划建造 N 个农场，用 N-1 条道路连接，构成一棵树（也就是说，所有农场之间都可以到达，并且没有环）。每个农场有一头奶牛，品种为更赛牛或荷斯坦牛之一。

Farmer John 的 M 个朋友经常前来拜访他。在朋友 i 拜访之时，Farmer John 会与他的朋友沿着从农场 A_i 到农场 B_i 之间的唯一路径行走（可能有 $A_i = B_i$）。除此之外，他们还可以品尝他们经过的路径上任意一头奶牛的牛奶。由于 Farmer John 的朋友们大多数也是农场主，他们对牛奶有着极强的偏好。他的有些朋友只喝更赛牛的牛奶，其余的只喝荷斯坦牛的牛奶。 Farmer John 的朋友只有在他们访问时能喝到他们偏好的牛奶才会高兴。

请求出每个朋友在拜访过后是否会高兴。

输入：输入的第一行包含两个整数 N 和 M。

第二行包含一个长为 N 的字符串。如果第 i 个农场中的奶牛是更赛牛，则字符串中第 i 个字符为 G，如果第 i 个农场中的奶牛是荷斯坦牛则为 H。

以下 N-1 行，每行包含两个不同的整数 X 和 Y（$1 \leqslant X, Y \leqslant N$），表示农场 X 与 Y 之间有一条道路。

以下 M 行，每行包含整数 A_i，B_i，以及一个字符 C_i。A_i 和 B_i 表示朋友 i 拜访时的路径的端点，C_i 是 G 或 H 之一，表示第 i 个朋友喜欢更赛牛的牛奶或是荷斯坦牛的牛奶。

输出：输出一个长为 M 的二进制字符串。如果第 i 个朋友会感到高兴，则字符串的第 i 个字符为 1，否则为 0。

输入

5 5

HHGHG

1 2

2 3

2 4

1 5

1 4 H

1 4 G

1 3 G

1 3 H

5 5 H

输出

10110

样例 1 解释：在这里，从农场 1 到农场 4 的路径包括农场 1、2 和 4。所有这些农场里都是荷斯坦牛，第一个朋友会感到满意，而第二个朋友不会。

数据范围：测试点 2~5 满足 $N \leqslant 10^3$，$M \leqslant 2 \cdot 10^3$。

对于 100% 的数据，$1 \leqslant N \leqslant 10^5$，$1 \leqslant M \leqslant 10^5$。

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。

```cpp
#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
	for(int i =1;i <= n;i++) fa[i] = i; // 初始化
	int x,y;    // 读入 n-1 对关系
	for(int i = 1;i <= n - 1;i++) {
		scanf("%d%d",&x,&y);
		if(s[x] == s[y]) merge(x,y);
	}
	char c[2];         //m 次询问
	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

2167. 树的公共祖先（LCA）

给定一棵树和两个不同的结点，求出他们最近的公共祖先父结点。

已知该树有 n 个结点，标号 1..n 。

输入

第 1 行输入一个整数 n，代表结点数量（n ≤ 100）；

第 2 行输入两个整数 x,y，表示需要计算的结点；

以下 n-1 行，每行两个整数 a 和 b，表示 a 的父结点是 b。

输出

输出 x 与 y 的最近公共祖先 root。

输入

9

9 7

2 1

3 2

4 2

5 3

8 5

9 5

6 4

7 4

输出

2

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;
}
```

```cpp
#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;
void dfs(int x)// 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
{
	if(fa[x] == x) dep[x] = 1;	// 递归的出口是根，根的默认深度为 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
	}
	dfs(x);	// 分别求 x、y 到根路径中每个结点的深度
	dfs(y);
	while(x != y)	// 从 x、y 分别向上移动，直到遇到第一个公共祖先
	{
		if(dep[x] > dep[y]) x = fa[x];
		else y = fa[y];
	}
	cout<<x;
	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。

2168. 树的公共祖先 (LCA)(2)

给定一棵树和两个不同的结点，求出他们最近的公共祖先父结点。

已知该树有 n 个节点，标号 1 … n ，标号为 1 的结点是该树的根。

输入

第 1 行输入一个整数 n，代表结点数量（n ≤ 100）；

第 2 行输入两个整数 x，y，表示需要计算的结点；

以下 n~1 行,每行两个整数 a 和 b ,结点 a 和 b 是父子关系,但不保证 a 是 b 的父;

输出

x 与 y 的最近公共祖先。

输入

9

9 7

2 1

3 2

4 2

5 3

8 5

9 5

6 4

7 4

输出

2

（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;
}
```

2200. 树的公共祖先 (LCA)(3)

给定一棵 n 个结点的树（结点标号 1…n ）以及树中结点的边，结点 s 为树的根。

有 m 次询问，请求出每次询问的两个结点 x 和 y 的最近的公共祖先结点。

输入

第 1 行输入 3 个整数 n、m、s（n ≤ 500000，m ≤ 500000，1 ≤ s ≤ n）；

接下来 n~1 行，每行两个整数 a 和 b ，结点 a 和 b 是父子关系，但不保证 a 是 b 的父，数据保证一定能构成树；

接下来 m 行，每行两个整数 x，y，表示要求出 x 和 y 结点的公共祖先。

输出

输出 m 行，每行一个整数，表示 m 次询问求出的结果。

输入

5 5 4

3 1

2 4

5 1

1 4

2 4

3 2

3 5

1 2

4 5

输出

4

4

1

4

4

```cpp
#include <bits/stdc++.h>//3-2200-1  树的公共祖先（LCA）(3)  steve_csp
using namespace std;
const int N = 5e5+10, L = 20;
int n, m, s, k;
int f[N][L];
int pre[N], dep[N], lg[N];

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

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

void dfs(int x, int fath)
{
	dep[x] = dep[fath] + 1;
	f[x][0] = fath;
	for(int i = pre[x]; i != 0; i = a[i].next)
	{
		int to = a[i].to;
		if(to != fath)
		{
			dfs(to, x);
		}
	}
}
```

```
int lca(int x, int y) {
	if(dep[x] < dep[y])  swap(x, y);
	while(dep[x] > dep[y]) {
		x = f[x][lg[dep[x]-dep[y]]];  }
	if(x == y) return x;
	for(int i = L - 1; i >= 0; i--) {
		if(f[x][i] != f[y][i]) {
			x = f[x][i];
			y = f[y][i];
		}
	}
	return f[x][0];
}
int main(){
	scanf("%d%d%d", &n, &m, &s);
	int x, y;
	for(int i = 1; i <= n - 1; i++) {
		scanf("%d%d", &x, &y);
		add(x, y);
		add(y, x);
	}
	dfs(s, 0);
	for(int j = 1; j < L; j++) {
		for(int i = 1; i <= n; i++){
			f[i][j] = f[f[i][j-1]][j-1];
		}
	}
	lg[1] = 0;
	for(int i = 2; i <= n; i++) lg[i] = lg[i/2] + 1;
	while(m--) {
		scanf("%d%d", &x, &y);
		printf("%d\n", lca(x, y));
	}
	return 0;
}
```

```cpp
#include <bits/stdc++.h> // 3-2200-2  树的公共祖先（LCA）(3)  倍增法  ku_sunny
using namespace std;
const int maxn = 500000 + 2;
int n, m, s;
int k = 0;
// head 数组就是链接表标配了吧？
// d 存的是深度（deep）
// p[i][j] 存的是 [i] 向上走 2 的 j 次方那么长的路径
int head[maxn], d[maxn], p[maxn][21];

struct node {
    int v, next;
} e[maxn * 2]; // 存树

void add(int u, int v) {
    e[k].v = v;
    e[k].next = head[u];
    head[u] = k++;
} // 加边函数

void dfs(int u, int fa) {
    d[u] = d[fa] + 1;
    p[u][0] = fa;
    for (int i = 1; (1 << i) <= d[u]; i++)
        p[u][i] = p[p[u][i - 1]][i - 1];
    for (int i = head[u]; i != -1; i = e[i].next) {
        int v = e[i].v;
        if (v != fa)
            dfs(v, u);
    }
} // 首先进行的预处理，将所有点的 deep 和 p 的初始值 dfs 出来
```

```
int lca(int a, int b) { // 非常标准的 lca 查找
    if (d[a] > d[b])
        swap(a, b); // 保证 a 是在 b 结点上方，即 a 的深度小于 b 的深度

    for (int i = 20; i >= 0; i--)
        if (d[a] <= d[b] - (1 << i))
            b = p[b][i]; // 先把 b 移到和 a 同一个深度

    if (a == b)
        return a; // 特判，如果 b 上来就和 a 一样了，那就可以直接返回答案了

    for (int i = 20; i >= 0; i--) {
        if (p[a][i] == p[b][i])
            continue;
        else
            a = p[a][i], b = p[b][i]; // A 和 B 一起上移
    }
    return p[a][0];
}
int main() {
    memset(head, -1, sizeof(head));
    int a, b;
    scanf("%d%d%d", &n, &m, &s);
    for (int i = 1; i < n; i++) {
        scanf("%d%d", &a, &b);
        add(a, b);
        add(b, a); // 无向图，要加两次
    }
    dfs(s, 0);
    for (int i = 1; i <= m; i++) {
        scanf("%d%d", &a, &b);
        printf("%d\n", lca(a, b));
    }
    return 0;
}
```

4、树的直径中心重心

2172. 树的直径

树的直径指的是，树中两点之间的最长路径。现给定一棵树的数据，请问该树的直径的是多少？

比如，如图所示的树，直径应该是 3-2-5-6，路径长度为 3。

输入

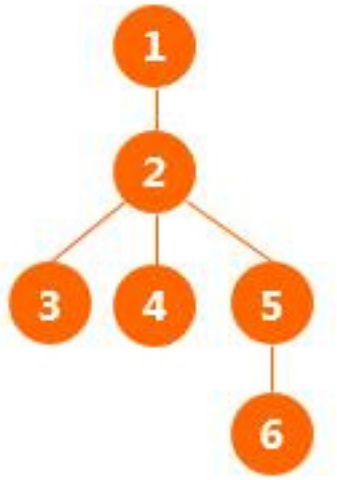
第 1 行有一个整数 n，代表结点的数量，结点的编号为 1 ~ n。（n ≤ 100）

接下来有 n-1 行，每行有 2 个整数 x,y，表示结点 x 和 y 之间有一条边。（不保证 x 是 y 的父）

输出

输出一个整数 n 代表树的直径。

输入	输出
6	3
1 2	
3 2	
5 6	
2 4	
5 2	




```
/*
1. 以任意的一个点为出发点，求离它最远（深度最深）的结点 x
2. 以 x 为根，求出离 x 最远（深度最深）的结点 y 的深度 d
3. 直径值 = d - 1，直径的 2 个断点分别是 x 和 y
*/
```

```cpp
#include <bits/stdc++.h>//1-2172-1  树的直径  解法一：邻接矩阵求解
using namespace std;
const int N = 110;
int a[N][N];
int fa[N], dep[N];
int n;
void dfs(int x, int f) {      // 深搜求每个结点的父和深度
	fa[x] = f;
	dep[x] = dep[f] + 1;
	for (int i = 1; i <= n; i++) {    // 深搜 x 结点可以去的结点
		if (a[x][i] == 1 && i != f)   dfs(i, x);
	}
}
int main() {
	cin >> n;
	int x, y;
	for (int i = 1; i <= n - 1; i++) {    // 读入边
		cin >> x >> y;
		a[x][y] = 1;
		a[y][x] = 1;
	}
	dfs(1, 0);    // 以 1 号结点为根，求所有结点的父，及所有结点的深度
	x = 1;    // 求出离 1 号结点最远的结点编号 x
	for (int i = 2; i <= n; i++) {
		if (dep[i] > dep[x])     x = i;
	}
	dfs(x, 0);    // 以 x 结点为根，求出所有结点的深度
	y = 1;    // 求离 x 结点最远的结点编号
	for (int i = 2; i <= n; i++) {
		if (dep[i] > dep[y])
			y = i;
	}
	cout << dep[y] - 1;
	return 0;
}
```

```cpp
#include <bits/stdc++.h>//1-2172-2  树的直径  解法二 ：结构体数组实现邻接表
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;
void add(int u, int v) {	// 加边
	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
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);
		}
	}
}
```

```cpp
int main()
{
	cin >> n;
	for (int i = 1; i <= n - 1; i++)	// 读入关系
	{
		cin >> x >> y;
		add(x, y);
		add(y, x);
	}

	dfs(1, 0); // 求每个点的父及深度

	x = 1;
	for (int i = 2; i <= n; i++)
	{
		if (dep[i] > dep[x])
			x = i;
	}

	dfs(x, 0);

	y = 1;
	int ans = 0; // 直径
	for (int i = 2; i <= n; i++)
	{
		if (dep[i] > dep[y])
			y = i;
	}

	// cout << x <<" " << y << endl << dep[y] - 1;
	cout << dep[y] - 1;
	return 0;
}
```

```cpp
#include <bits/stdc++.h>// 1-2172-3  树的直径  解法三：数组实现邻接表
using namespace std;
const int N = 110;
int fi[N], to[N * 2], ne[N * 2], k;
int n;
int de[N];        // de：表示每个结点的深度
void add(int u, int v)        // 加边
{
	k++;
	to[k] = v;
	ne[k] = fi[u];
	fi[u] = k;
}


void dfs(int x, int fath) // 求结点的深度
{
	de[x] = de[fath] + 1;
	//  cout << x << " " << fath <<" "<< de[x] << endl;
	for (int i = fi[x]; i != 0; i = ne[i])
	{
		if (to[i] != fath)
		{
			dfs(to[i], x);
		}
	}
}
```

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

	dfs(1, 0);	// 求直径
	x = 1;
	for (int i = 2; i <= n; i++)
	{
		if (de[i] > de[x])
			x = i;
	}

	//  cout << "-----------" << endl;
	dfs(x, 0);
	y = 1;
	int ans = 0; // 直径
	for (int i = 2; i <= n; i++)
	{
		if (de[i] > de[y])
			y = i;
	}

	//  cout << x << " " << y << endl << de[y] - 1;
	cout << de[y] - 1;
	return 0;
}
```

2207. 树的中心

给定一棵树，树中有 n 个结点（结点编号为 1~n ），请求出该树的中心结点的编号。树的中心指的是，该结点离树中的其他结点，最远距离最近。

比如，按照树的中心的定义，下图中的结点 1 和结点 3 ，就是该树的中心。

输入

第 1 行包括一个整数 n（$n \leqslant 10^5$），代表树中结点的数量。

接下来 n~1 行，每行有 2 个整数 x 和 y，代表结点 x 到结点 y 之间有一条边。（不确定结点之间的父子关系）

输出

请输出树的中心结点的编号，如果有多个中心结点，请按照从小到大的顺序输出所有中心结点的编号。

输入

5

2 1

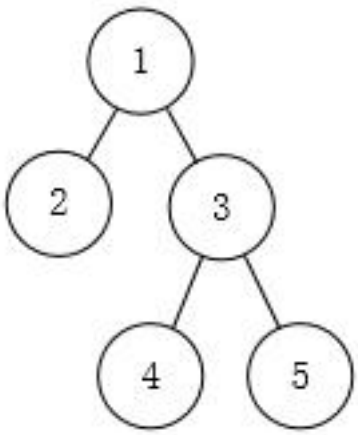
1 3

3 4

3 5

输出

1 3



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;
}
void dfs(int x,int f)// 深搜求父子关系及深度
{
	fa[x] = f;
	dep[x] = dep[f] + 1;
	for(int i = pre[x];i != 0;i = a[i].next)  // 讨论 x 能去的点
	{
		if(a[i].to != f)  // 如果去的点不是父
		{
			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
	}
	dfs(1,0);	// 从 1 号结点开始深搜，求每个结点的深度和父子关系
	x = 1;		// 求直径的一个端点 x
	for(int i = 2;i <= n;i++)
	{
		if(dep[i] > dep[x])	x = i;
	}
	dfs(x,0);	// 从 x 点出发（以 x 为根）求另一个端点
	y = 1;
	for(int i = 2;i <= n;i++)
	{
		if(dep[i] > dep[y])	y = i;
	}
	int d = dep[y];	// 通过爬树得到中心点的编号
	if(d % 2 != 0)	// 如果深度为奇数，中心点有一个
	{
		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;
}
```

2190. 树的重心

给定一颗树，树中有 n 个结点（编号 1~n ）。 请你找到树的重心，并输出树的重心的结点编号。

重心定义：重心是指树中的一个结点，如果将这个点删除后，剩余各个连通块中结点数的最大值最小，那么这个结点被称为树的重心。

如下图所示的树的重心为 1 号结点。

输入

第 1 行读入一个整数 n，代表树的结点的数量（ $1 \leqslant n \leqslant 10^5$ ）；

接下来 n~1 行，每行读入两个整数 x 和 y ，表示结点 x 和 y 之间有一条边（注意：不确定 x 和 y 的父子关系）。

输出

请输出树的重心的结点编号，如果树有多个重心，请按照编号从小到大依次输出，数字之间用空格隔开。

输入

9

1 2

1 7

2 8

2 5

4 3

1 4

3 9

4 6

输出

1

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;
}
```

2191. 树的重心（2）

给定一棵树，树中有 n 个结点（编号 1 ~ n）。 请求出，删除该重心后，剩余子树的最多结点数？

重心定义：重心是指树中的一个结点，如果将这个点删除后，剩余各个连通块中点数的最大值最小，那么这个结点被称为树的重心。

输入

第 1 行读入一个整数 n ，代表树的结点的数量（$1 \leqslant n \leqslant 10^5$）。

接下来 n~1 行，每行读入两个整数 x 和 y，表示结点 x 和 y 之间有一条边。（注意：不确定 x 和 y 的父子关系）

输出

输出一个整数，代表删除重心后，剩余子树的最多结点。

输入

9

1 2

1 7

2 8

2 5

4 3

1 4

3 9

4 6

输出

4