信奥算法

长度可变的数组

vector

```cpp
#include<iostream>
#include<vector>
using namespace std;
int main(){
    vector<int> d;
    d.push_back(3);
    d.push_back(1);
    d.push_back(4);
    d.push_back(1);
    d.push_back(5);
    for(int i=0;i<d.size();i++)
        cout<<d[i]<<" ";
    cout<<endl;
    return 0;
}
```

头文件

定义

尾部增加元素

求元素个数

数组方括号操作

```
4  vector<int> d;
5  void print(){
6      for(int i=0;i<d.size();i++)
7          cout<<d[i]<<" ";
8      cout<<"size="<<d.size()<<endl;
9  }
10 int main(){
11     d.push_back(1);
12     d.push_back(2);
13     d.push_back(3);
14     print();
15     d.pop_back();
16     print();
17     d.clear();
18     print();
19     return 0;
20 }
```

预测输出结果

尾部增加元素

尾部删除元素

清空数组

预测输出结果

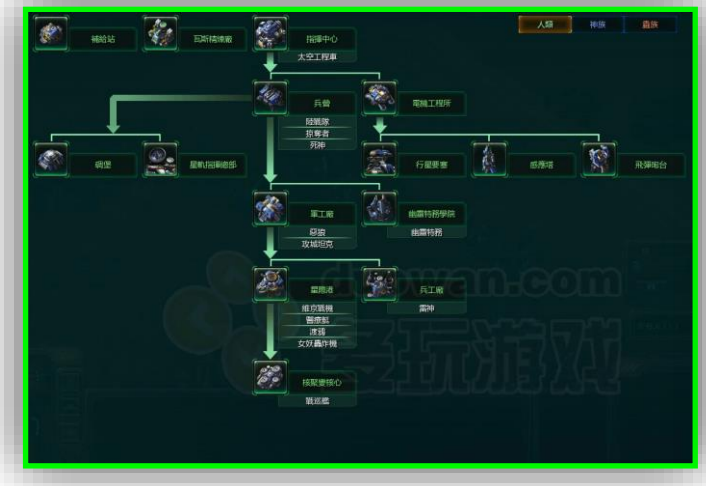
```
5   vector<int> d[5];
6   d[1].push_back(10);
7   d[1].push_back(9);
8   d[2].push_back(8);
9   d[3].push_back(7);
10  d[3].push_back(6);
11  d[3].push_back(5);
12  for(int u=0;u<5;u++){
13      for(int i=0;i<d[u].size();i++)
14          cout<<d[u][i]<<" ";
15      cout<<endl;
16  }
```

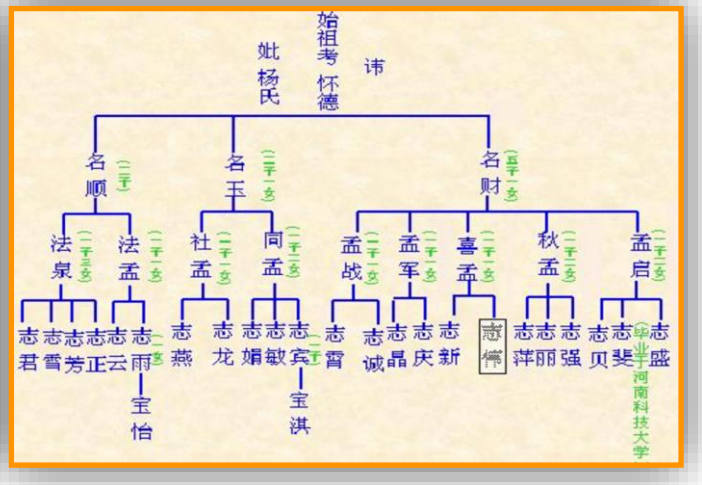
编号i	0	1	2
d[0][]			
d[1][]	10	9	
d[2][]	8		
d[3][]	7	6	5
d[4][]			

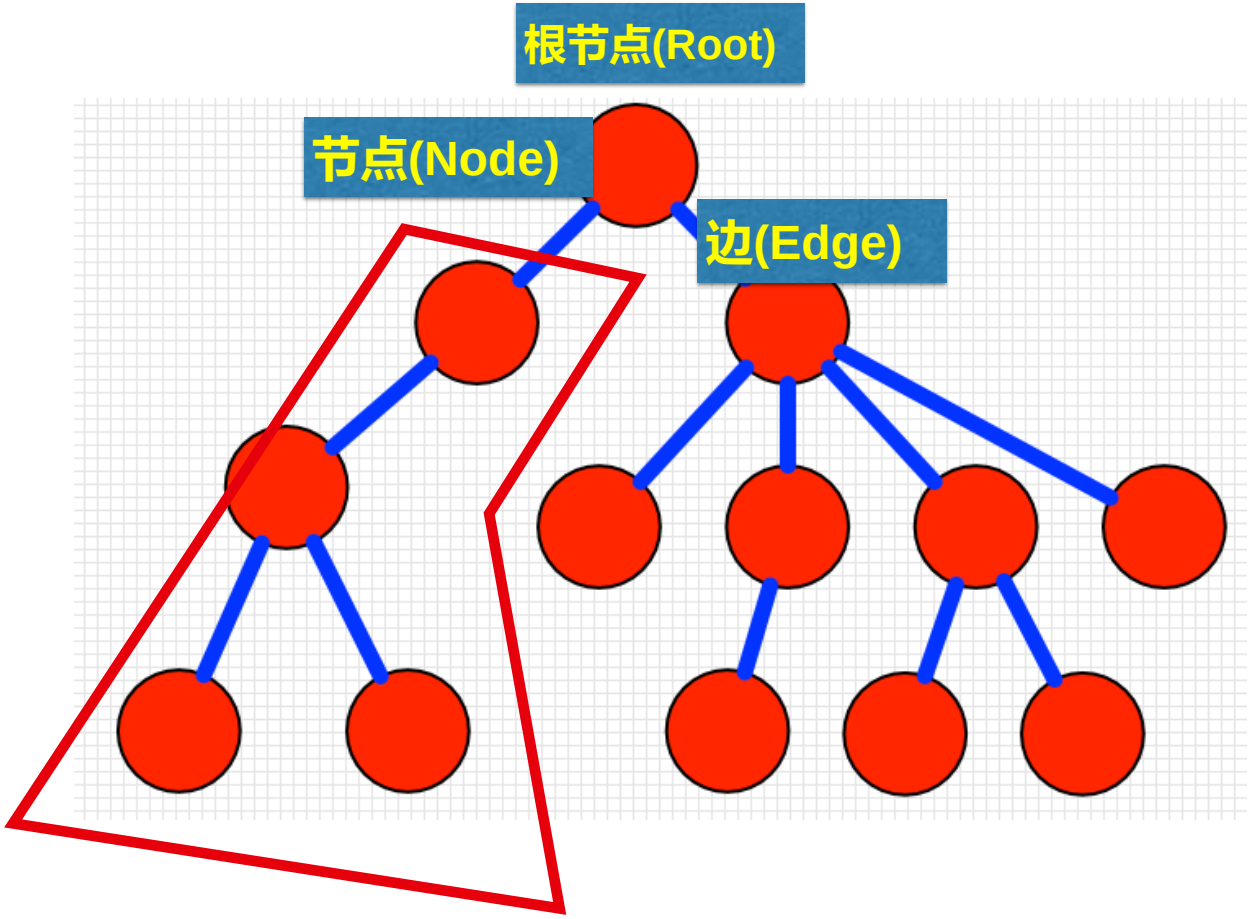
树的遍历和存储

Tree

树的起源：分支结构




始祖
考
怀德
讳
妣
杨氏
名顺
名王
名财
法泉
法孟
社孟
同孟
孟战
孟军
喜孟
秋孟
孟启
志君
志雪
志芳
志正
志云
志雨
宝怡
志燕
志龙
志娟
志敏
志宾
宝洪
志霄
志诚
志晶
志庆
志新
志萍
志丽
志强
志贝
志斐
志盛
毕业于河南科技大学



→ **树（Tree）是描述分支的抽象数据结构**

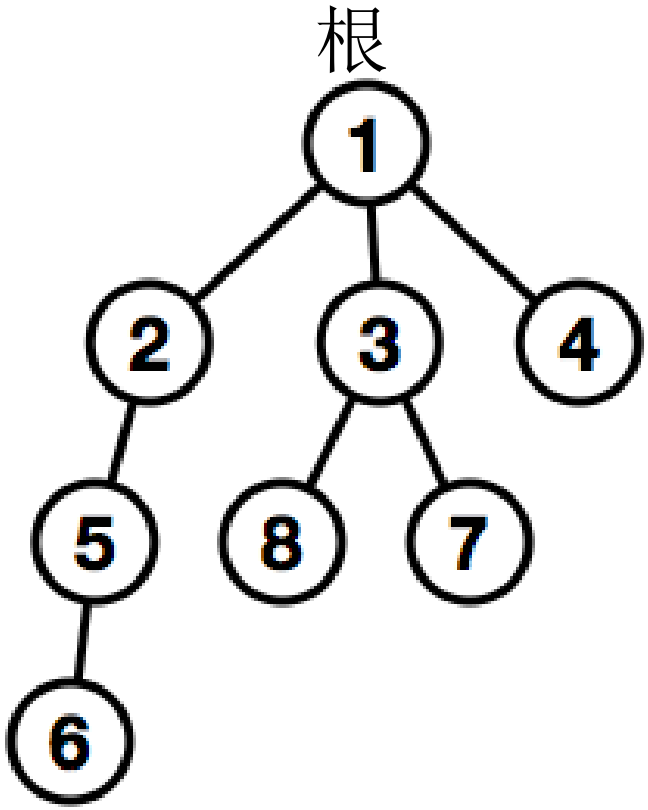
从**任意节点向下**，也是一棵树（子树）

→ **树的三大基本特征：**连通、无环、边数=节点数-1

树的3种输入方式

方式1

输入所有边的端点



描述所有边的2个端点

n=

8	
1	2
2	5
5	6
1	3
3	8
3	7
1	4

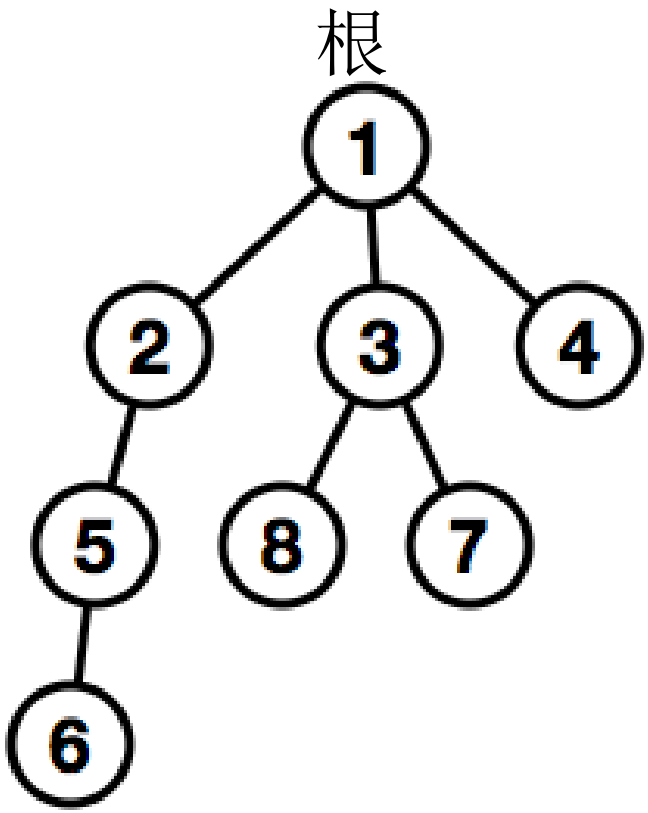
8	
1	2
1	3
1	4
2	5
3	7
3	8
5	6

输入顺序可以打乱

边数=节点数-1

方式2

输入父节点



描述每个节点的父节点是谁

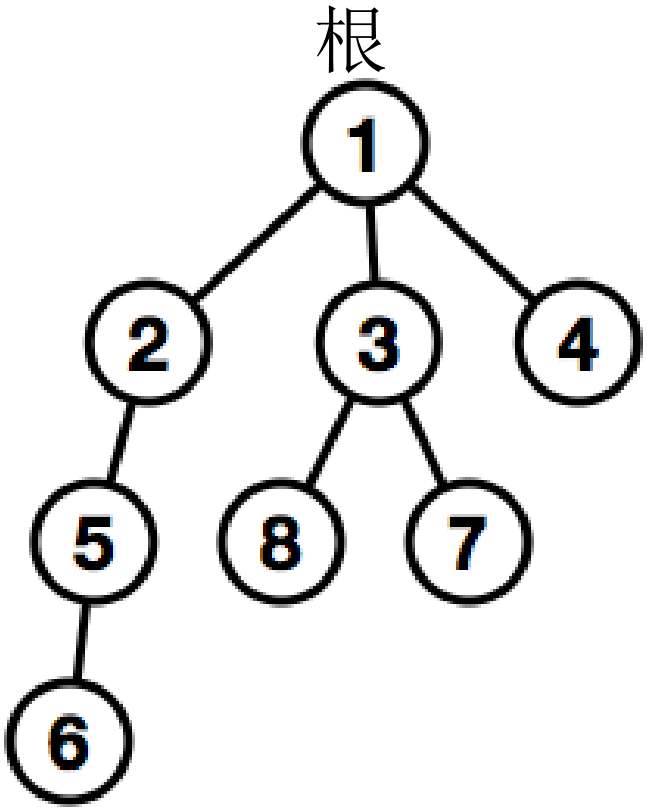
n=

8	1没有父节点
1	2的父节点是1
1	3的父节点是1
1	4的父节点是1
2	5的父节点是2
5	6的父节点是5
3	7的父节点是3
3	8的父节点是3

输入顺序不可以打乱

方式3

输入儿子列表



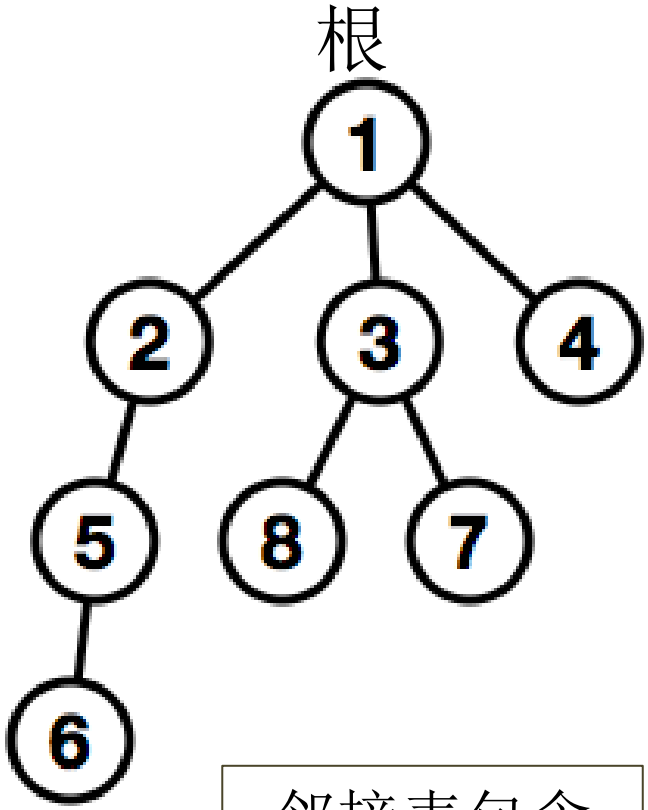
描述每个节点的儿子节点是谁

n=

```
8
3 2 3 4
1 5
2 8 7
0
1 6
0
0
0
```

1有3个儿子
分别是2 3 4

树的储存方式



邻接表	对于每个节点 储存它的所有邻居
	`vector<int> to[N];`

编号i	0	1	2
to[0][]			
to[1][]	2	3	4
to[2][]	1	5	
to[3][]	1	8	7
to[4][]	1		
to[5][]	2	6	
to[6][]	5		
to[7][]	3		
to[8][]	3		

邻接表包含 父节点吗?
黄色有几格?

邻接表加边

邻接表	对于每个节点 储存它的所有邻居
	`vector<int> to[N];`

```
6 void add(int u,int v){
7     to[u].push_back(v);
8     to[v].push_back(u);
9 }
```

u的邻居数组里增加v
v的邻居数组里增加u

u,v之间连上 一条**双向边**
原题里是无向边 实际储存为**双向边**

邻接表输出

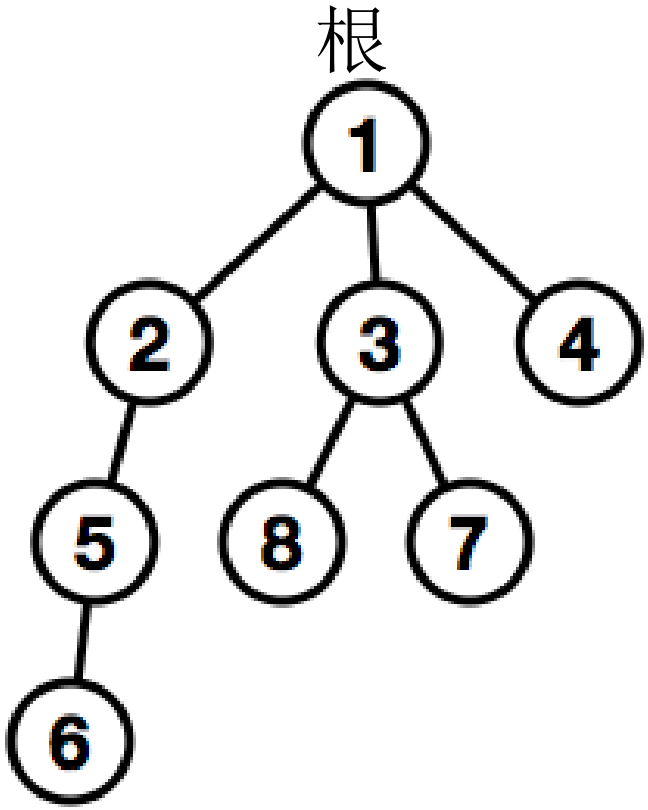
邻接表	对于每个节点 储存它的所有邻居
	vector<int> to[N];

```
for(int u=1;u<=n;u++,cout<<endl)
    for(int i=0;i<to[u].size();i++)
        cout<<to[u][i]<<" ";
```

es[u].size()代表
节点u的邻居个数

es[u][i]代表
节点u的i号邻居

树的3种输入方式



描述一棵树的结构
要表达清楚
节点和边的连接关系

描述所有边的2个端点

描述每个节点的父节点是谁

描述每个节点的儿子节点是谁

输入所有边的端点

方式1

描述所有边的2个端点

```
 6 void add(int u,int v){
 7     to[u].push_back(v);
 8     to[v].push_back(u);
 9 }
10 void input(){
11     cin>>n;
12     for(int i=1;i<=n-1;i++){
13         int u,v;
14         cin>>u>>v;
15         add(u,v);
16     }
17 }
```

n=

```
8
1 2
2 5
5 6
1 3
3 8
3 7
1 4
```

输入父节点

方式2

p[u]表示节点u的父节点

```
6  void add(int u,int v){
7      to[u].push_back(v);
8      to[v].push_back(u);
9  }
10 void input(){
11     cin>>n;
12     for(int u=2;u<=n;u++){
13         cin>>p[u];
14         add(u,p[u]);
15     }
16 }
```

n= 8

1

1

1

2

5

3

3

输入儿子列表

方式3

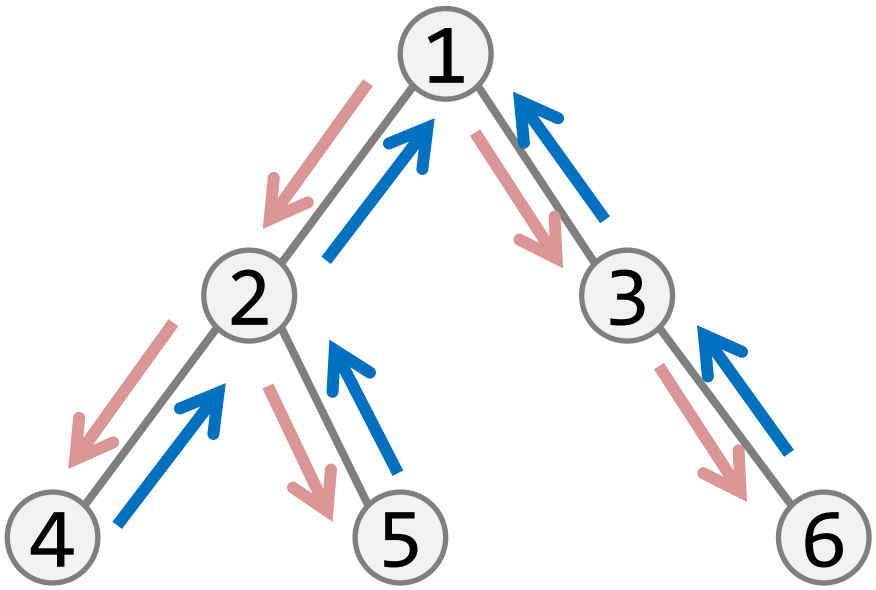
描述每个节点的儿子节点是谁

```
10 void input(){
11     cin>>n;
12     int c,v;
13     for(int u=1;u<=n;u++){
14         cin>>c;
15         for(int i=0;i<c;i++){
16             cin>>v;
17             add(u,v);
18         }
19     }
20 }
```

n=

```
8
3 2 3 4
1 5
2 8 7
0
1 6
0
0
0
```

树的遍历	访问所有节点

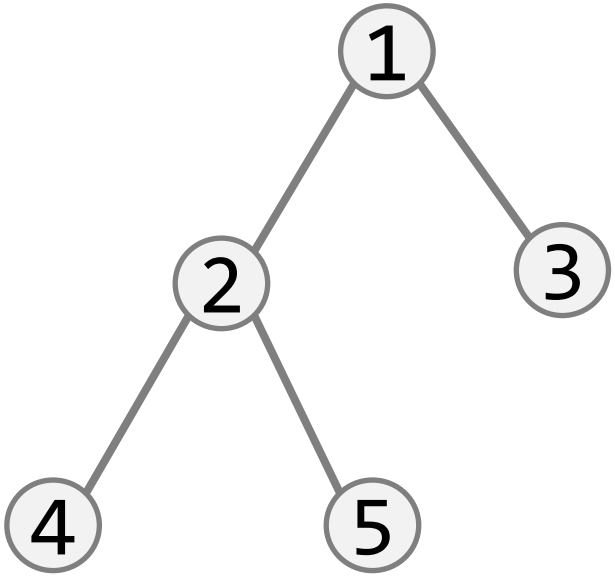


深度优先搜索DFS

请同学写出一种DFS访问顺序

124536

父节点p	p[u]表示节点u的父节点



计算顺序
从上往下

p[u]=	u=1	u=2	u=3	u=4	u=5
	0	1	?	?	?

已知树的邻接表

求每个节点的父节点

利用DFS顺序

父节点p

father

从节点u继续深度优先搜索	fa表示u的前序节点

```
10 void dfs(int u,int fa){
11     p[u]=fa;
12     for(int i=0;i<to[u].size();i++){
13         int v=to[u][i];
14         if(v==fa)continue;
15         dfs(v,u);
16     }
17 }
```

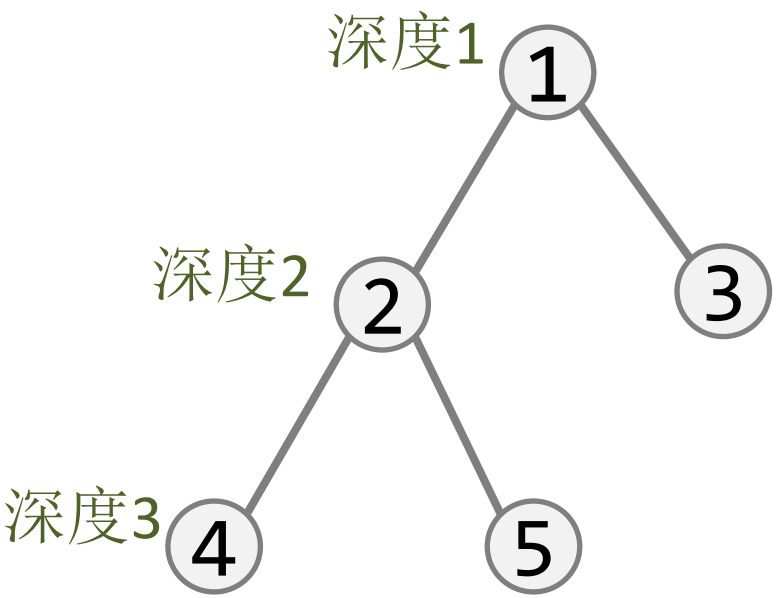
枚举u的所有邻居

用v记录u的i号邻居

如果v是u的父节点,就忽略

从节点v继续深度优先搜索

深度d	d[u]表示节点u的深度



根的深度
约定为1

计算顺序
从上往下

d[u]=

u=1	u=2	u=3	u=4	u=5
1	2	?	?	?

1号根节点作为起点到u号节点的
路径长度是d[u]-d[1]

已知树的邻接表

求每个节点的深度

利用DFS顺序

深度d

father

从节点u继续深度优先搜索	fa表示u的前序节点

```
10 void dfs(int u,int fa){
11     p[u]=fa;
12     d[u]=d[fa]+1;
13     for(int i=0;i<to[u].size();i++){
14         int v=to[u][i];
15         if(v==fa)continue;
16         dfs(v,u);
17     }
18 }
```

枚举u的所有邻居

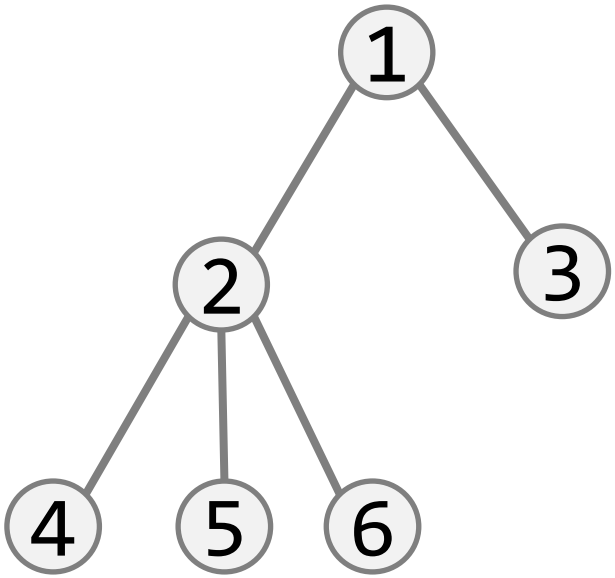
用v记录u的i号邻居

如果v是u的父节点,就忽略

从节点v继续深度优先搜索

子树大小sz

sz[u]表示以节点u为根的子树大小



计算顺序
从下往上

sz[u]=	u=1	u=2	u=3	u=4	u=5	u=6
	?	?	?	1	1	1

$$sz[u] = \sum_{v是u的子节点} sz[v]$$

已知树的邻接表

求每个节点的深度

利用DFS回溯的顺序

子树大小sz

```
10 void dfs(int u,int fa){
11     p[u]=fa;
12     d[u]=d[fa]+1;
13     sz[u]=1;
14     for(int i=0;i<to[u].size();i++){
15         int v=to[u][i];
16         if(v==fa)continue;
17         dfs(v,u);
18         sz[u]+=sz[v];
19     }
20 }
```

枚举u的所有邻居

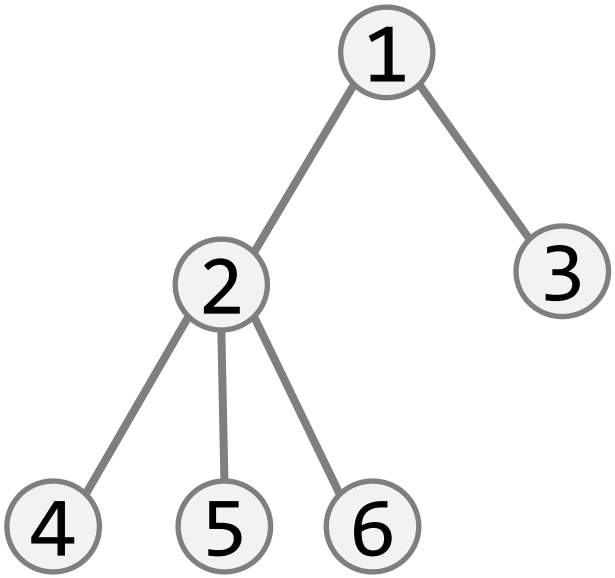
用v记录u的i号邻居

如果v是u的父节点,就忽略

从节点v继续深度优先搜索

第17和18行能否交换顺序?

高度h	h[u]表示以节点u的高度



计算顺序
从下往上

叶节点的高度
约定为1

	u=1	u=2	u=3	u=4	u=5	u=6
h[u]=	?	?	?	1	1	1

$$h[u] = \max\{h[v]\} + 1$$

v是u的子节点

已知树的邻接表

求每个节点的高度

利用DFS回溯的顺序

高度h

```
10  void dfs(int u,int fa){
11      p[u]=fa;
12      d[u]=d[fa]+1;
13      sz[u]=1;
14      h[u]=1;
15      for(int i=0;i<to[u].size();i++){
16          int v=to[u][i];
17          if(v==fa)continue;
18          dfs(v,u);
19          sz[u]+=sz[v];
20          h[u]=max(h[u],h[v]+1);
21      }
22  }
```


树的4种基本信息

p[u]代表什么?	节点u的父节点
	自顶向下
d[u]代表什么?	节点u的深度
	自顶向下
sz[u]代表什么?	以节点u为根的子树大小
	自底向上
h[u]代表什么?	以节点u为根的子树高度
	自底向上

太戈编程

	241
	1645
拓展题	861