数据结构和算法
（Python描述）

郭 炜

微信公众号

微博：http://weibo.com/guoweiofpku

学会程序和算法，走遍天下都不怕!

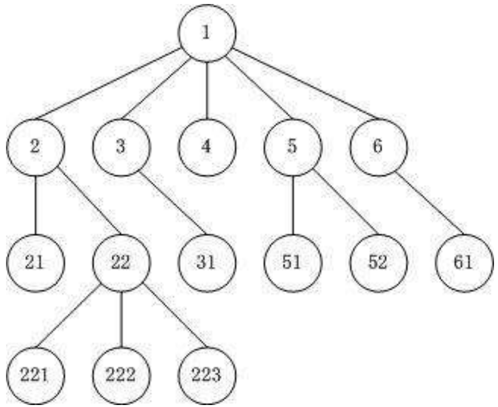
讲义照片均为郭炜拍摄

树、森林和并查集

树

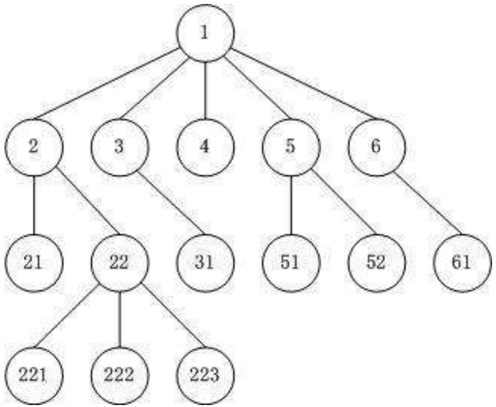
树的概念

- 每个结点可以有任意多棵不相交的子树
- 子树有序，从左到右依次是子树1,子树2......
- 二叉树的结点在只有一棵子树的情况下，要区分是左子树还是右子树。树的结点在只有一棵子树的情况下，都算其是第1棵子树 **(所以二叉树不是树)**
- 支持广度优先遍历、前序遍历(先处理根结点，再依次处理各个子树）和后序遍历（先依次处理各个子树，再处理根结点），中序遍历无明确定义



树的性质

- 结点度数最多为K的树，第i层最多K^i个结点(i从0开始）。
- 结点度数最多为K的树，高为h时最多有$(K^{h+1}-1)/(k-1)$ 个结点。
- n个结点的K度完全树，高度h是$\log_k(n)$向下取整
- n个结点的树有n-1条边



树的实现

- 直观表示法：每个结点有一个变量存放数据，加上一个可变长列表存放所有子结点指针
- 在不支持可变长列表的语言中，就要想别的办法，比如用二叉树来表示一棵树的儿子-兄弟表示法
- 父亲表示法：把所有结点编号，在每个结点内记录其父结点编号（用于并查集）

树的实现

直观表示法：

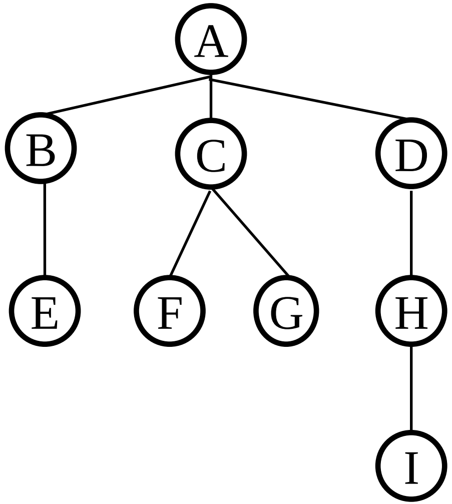
```
class Tree:
    def __init__(self,data, *subtrees): #参数个数可变的函数
        #参数subtrees是个元组，其中每个元素都是一个Tree对象
        self.data = data
        self.subtrees = list(subtrees) #self.subtrees是子树列表
    def addSubTree(self,tree): #tree是一个Tree对象
        self.subtrees.append(tree)
    def preorderTraversal(self,op):
        op(self)
        for t in self.subtrees:
            t.preorderTraversal(op)
    def postorderTraversal(self,op):
        for t in self.subtrees:
            t.postorderTraversal(op)
        op(self)
```

树的实现

```
def printTree(self,level = 0): #输出树的层次结构
    print("\t" * level + str(self.data))
    for t in self.subtrees:
        t.printTree(level+1)
```

输出形如:

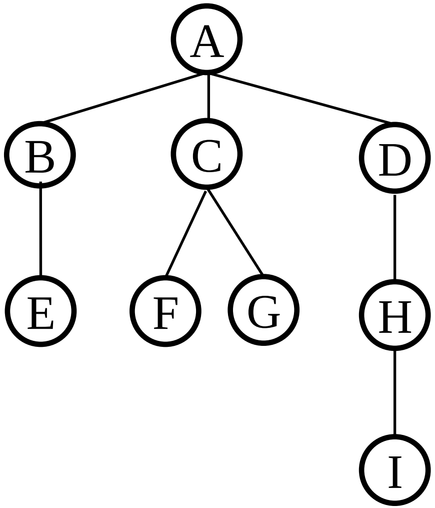
```
A
    B
        E
    C
        F
        G
    D
        H
            I
```



例题：构建树

读入：

```
A
    B
        E
    C
        F
        G
    D
        H
            I
```



构建直观表示法的树

例题：构建树

- 构建一棵直观表示法的树

```
def buildTree(level):  #读取nodesPtr指向的那一行，并建立以其为根的子树
    #该根的层次是level。建好后，令nodesPtr指向该子树的下一行
    global nodesPtr,nodes
    tree = Tree(nodes[nodesPtr][1])  #建根结点
    nodesPtr += 1  #看下一行
    while nodesPtr < len(nodes) and nodes[nodesPtr][0] == level + 1:
        tree.addSubTree(buildTree(level + 1))
    return tree
```

例题：构建树

```
nodes = []
while True:
    try:
        s = input().rstrip()
        nodes.append((len(s)-1,s.strip()))
    except:
        break
nodesPtr = 0   #表示看到nodes里的第几行
print(nodes)
tree = buildTree(0)
```

nodes内容形如：

[(0, 'A'), (1, 'B'), (2, 'E'), (1, 'C'), (2, 'F'), (2, 'G'), (1, 'D'), (2, 'H'), (3, 'I')]

元素为(缩进，数据)

树的二叉树表示法（儿子-兄弟表示法)

用二叉树表示一棵树T(二叉树形式表示的树，简称儿子兄弟树)

1) T的根就是二叉树的根R，R不会有右结点

2) R的左儿子S1，以及S1的左子树，是T的第1棵子树的二叉树表示形式

3) S1的右儿子S2，及S2的左子树，是T的第2棵子树的二叉树表示形式

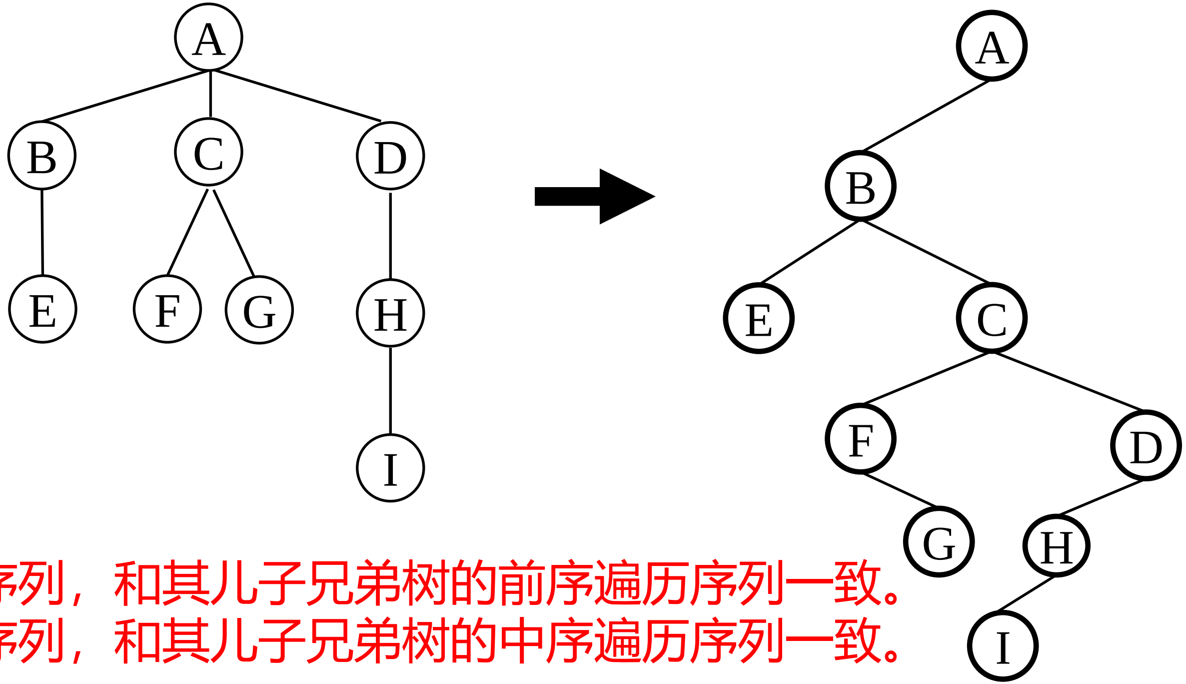
4) S2的右儿子S3，及S3的左子树，是T的第3棵子树的二叉树表示形式

....

以此类推

树的二叉树表示法（儿子-兄弟表示法)

用二叉树B表示一棵树T(二叉树形式表示的树，简称儿子兄弟树)



树的前序遍历序列，和其儿子兄弟树的前序遍历序列一致。
树的后序遍历序列，和其儿子兄弟树的中序遍历序列一致。

树的直观表示法转儿子兄弟树

```
def treeToBinaryTree(tree):
#直观表示法的树转儿子兄弟树。tree是Tree对象
    bTree = BinaryTree(tree.data) #二叉树讲义中的BinaryTree
    for i in range(len(tree.subtrees)):
        if i == 0:
            tmpTree = treeToBinaryTree(tree.subtrees[i])
            bTree.addLeft(tmpTree)
        else:
            tmpTree.addRight(treeToBinaryTree(tree.subtrees[i]))
            tmpTree = tmpTree.right
    return bTree
```

所有结点复制了一份

树的儿子兄弟表示法转直观表示法

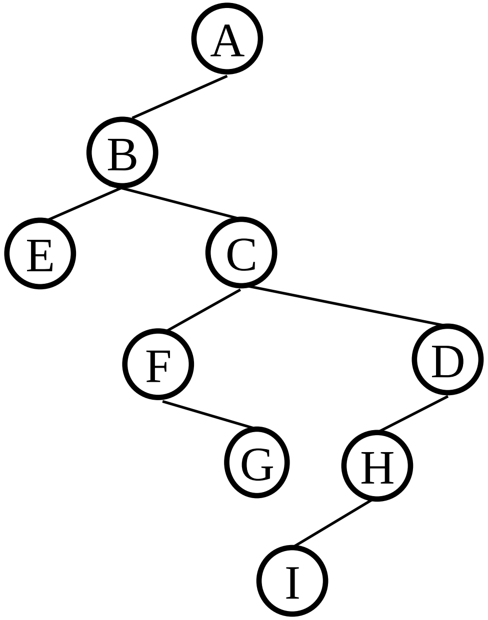
```
def binaryTreeToTree(biTree):
#儿子兄弟树转直观表示法的树转。biTree是BinaryTree对象
    tree = Tree(biTree.data)
    son = biTree.left
    if son:
        tree.addSubTree(binaryTreeToTree(son))
        while son.right:
            tree.addSubTree(binaryTreeToTree(son.right))
            son = son.right
    return tree
```

所有结点复制了一份

例题：构建儿子兄弟树

读入：

```
A
    B
        E
    C
        F
        G
    D
        H
            I
```



构建二叉树形式表示的树(留作练习)

森林

瑞士马特洪峰

森林的概念

- 不相交的树的集合，就是森林
- 森林有序，有第1棵树、第2棵树、第3棵树之分
- 森林可以表示为树的列表，也可以表示为一棵二叉树

森林的二叉树表示法

1) 森林中第1棵树的根，就是二叉树的根S1，S1及其左子树，是森林的第1棵树的二叉树表示形式

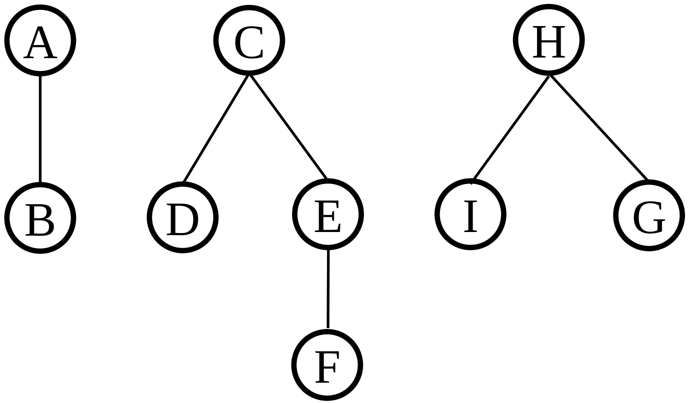
2) S1的右子节S2，以及S2的左子树，是森林的第2棵树的二叉树表示形式

3) S2的右子节S3，以及S3的左子树，是森林的第3棵树的二叉树表示形式

....

以此类推

森林的遍历

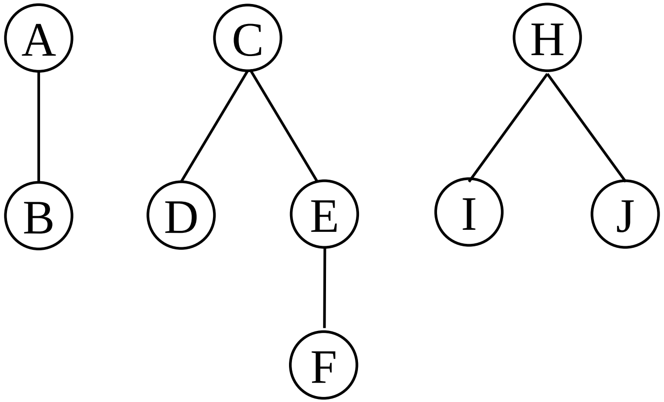


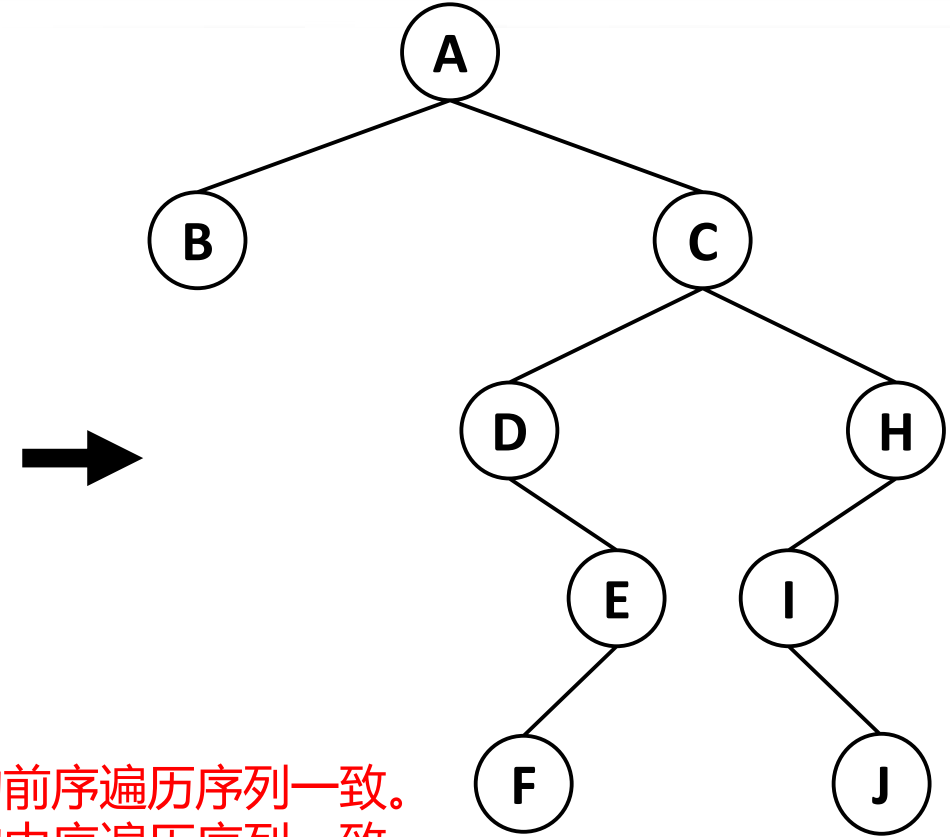
广度优先遍历: ACH BDEIG F

前序遍历：AB CDEF HIG
后续遍历：BA DFEC IGH

无中序遍历

森林的二叉树表示法





森林的前序遍历序列，和其儿子兄弟树的前序遍历序列一致。
森林的后序遍历序列，和其儿子兄弟树的中序遍历序列一致。

森林转二叉树

```
def woodsToBinaryTree(woods):
        #woods是个列表，每个元素都是一棵二叉树形式的树

        biTree = woods[0]
        p = biTree
        for i in range(1,len(woods)):
                p.addRight(woods[i])
                p = p.right
        return biTree
#biTree和woods共用结点,执行完后woods的元素不再是原儿子兄弟树
```

二叉树转森林

```
def binaryTreeToWoods(tree):
#tree是以二叉树形式表示的森林
        p = tree
        q = p.right
        p.right = None
        woods = [p]
        if q:
                woods += binaryTreeToWoods(q)
        return woods
```

woods是兄弟-儿子树的列表,woods和tree共用结点
执行完后tree的元素不再原儿子兄弟树

并查集

并查集的
原理和实现

美国黄石公园

Disjoint-Set 并查集

N 个不同的元素分布在若干个互不相交集合中，需要多次进行以下3个操作:

1. 合并a,b两个元素所在的集合 Merge(a,b)
2. 查询一个元素在哪个集合
3. 查询两个元素是否属于同一集合 Query(a,b)

并查集操作示例

Operation	**Disjoint sets**					
初始状态	{a}	{b}	{c}	{d}	{e}	{f}
Merge(a,b)	{a,b}		{c}	{d}	{e}	{f}
Query(a,c)	False					
Query(a,b)	True					
Merge(b,e)	{a,b,e}		{c}	{d}		{f}
Merge(c,f)	{a,b,e}		{c,f}	{d}		
Query(a,e)	True					
Query(c,b)	False					
Merge(b,f)	{a,b,c,e,f}			{d}		
Query(a,e)	True					
Query(d,e)	False					

简单算法

- 给集合编号

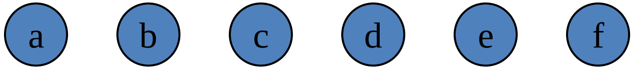
Op \ *Element*	{a}	{b}	{c}	{d}	{e}	{f}
	1	2	3	4	5	6
Merge(a,b)	1	1	3	4	5	6
Merge(b,e)	1	1	3	4	1	6
Merge(c,f)	1	1	3	4	1	3
Merge(b,f)	1	1	1	4	1	1

Query(a,e)

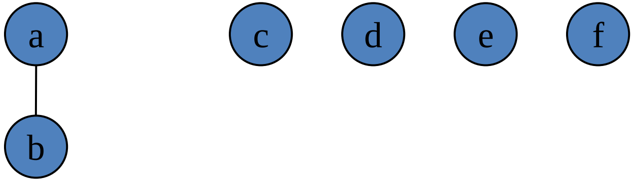
- Query – O(1); Merge – O(N)

用树表示集合的算法

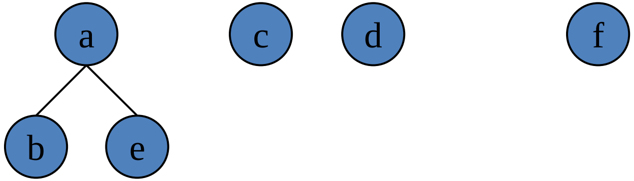
- 开始:



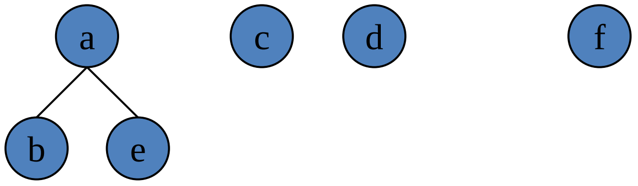
- *Merge(a,b)*



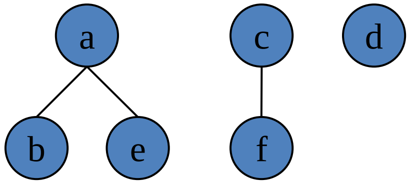
- *Merge(b,e)*



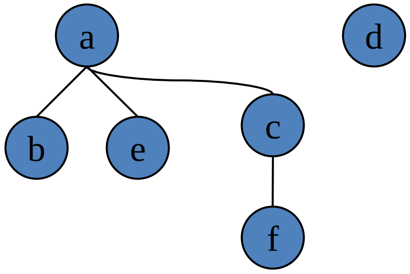
用树表示集合的算法



- *Merge(c,f)*

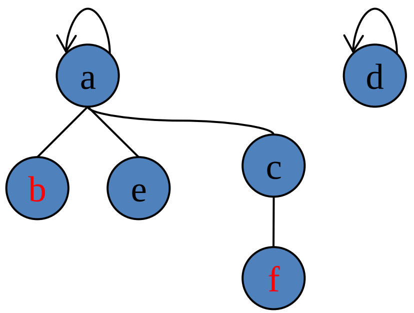


- *Merge(b,f)*



用树表示集合的算法

- 设置数组 parent, parent[i] = j 表示元素 i 的父亲是 j
- parent[i] = i 表示 i 是一棵树的根结点
- 若开始每个元素自成一个集合，则 对任何 i ， 都有 parent[i] = i



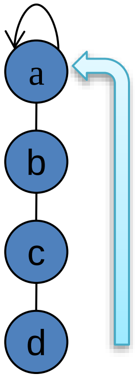
用树表示集合的算法

- 基本操作 GetRoot (a) 求a的树根

```
def GetRoot(a):
    if parent[a] == a:
        return a
    return GetRoot(parent[a])
```

用树表示集合的算法

- *Query(a,b)*
  - 比较b和f所在树的根结点是否相同
- *Merge(a,b)*
  - 将b的树根的父亲，设置为a的树根
- *缺点:*
  - 树的深度失控则查树根可能太慢！



O(N)!

改进方法一

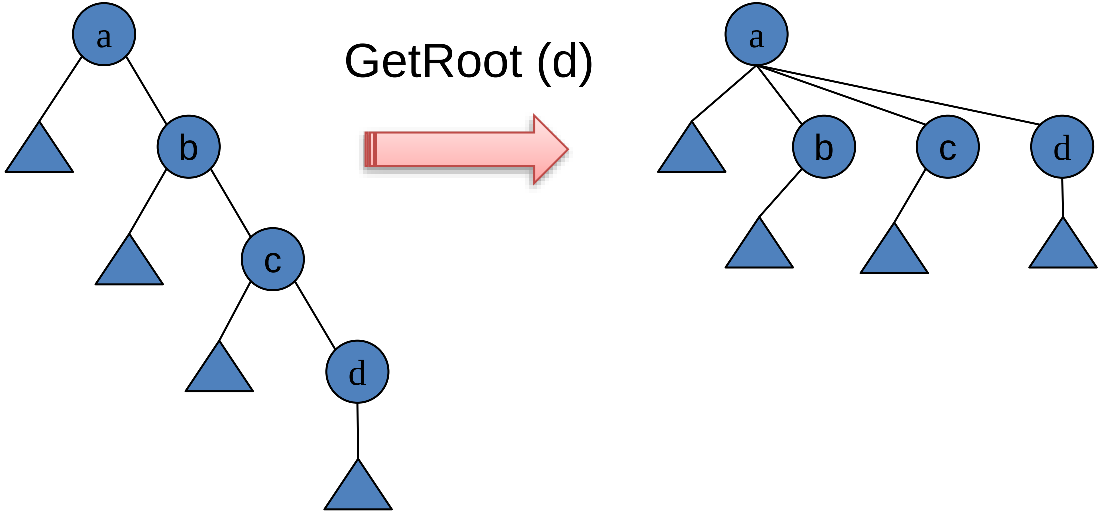
- 合并两棵树时，把深度浅的树直接挂在另一棵树的根结点下面
- 合并两棵同深度的树时，必然导致深度增加，不够理想

改进方法二：路径压缩

- 查询一个结点的根时，直接将该结点到根路径上的每个结点，都直接挂在根下面。

- 经过多次查询，很可能所有树的深度都是 <= 2

改进方法二：路径压缩



改进方法二：路径压缩

- 基本操作 GetRoot (a) 求a的树根

```
def GetRoot(a):
    if parent[a] != a:
        parent[a] = GetRoot(parent[a])
    return parent[a]
```

```
def GetRoot(a): #old one

    if parent[a] == a:
        return a
    return GetRoot(parent[a])
```

改进方法二：路径压缩

```
parent = [i for i in range(N)]
def Merge(a,b):
  #把b树根挂到a树根下
      parent[GetRoot(b)] = GetRoot(a)

def Query(a,b)
   #查询a,b是否位于同一棵树
      return GetRoot(a) == GetRoot(b)
```

并查集的空间复杂度O(N),时间复杂度就是GetRoot的复杂度

并查集的时间复杂度

GetRoot的时间复杂度,是O(log(n))

并查集例题
The Suspects

新疆喀拉峻鳄鱼湾

例题1： POJ1611 The Suspects

- 有n个学生，编号0到 n-1, 以及m个团体，(0 < n <= 30000 , 0 <= m <= 500) 一个学生可以属于多个团体，也可以不属于任何团体。一个学生得病，则它所属的整个团体都会被他传染而得病。
- 开始只有0号学生得病。已知每个团体都由哪些学生构成，求最终一共多少个学生会得病。

例题1: POJ1611 The Suspects

Sample Input

100 4 //100人,4团体
2 1 2 //本团体有2人，编号 1,2
5 10 13 11 12 14
2 0 1
2 99 2
200 2 //200人,2团体
1 5 //本团体有1人，编号 5
5 1 2 3 4 5
1 0 //1人,0团体
0 0 //结束标记

Sample Output

4
1
1

三组数据，分别是:
100个人，4个团体
200个人,2个团体
1个人,0个团体

例题1： POJ1611 The Suspects

关键:

互相感染的人，应该属于同一个集合。

一个集合中任意一人得病，全集合人都得病

所有病人都是直接或间接被0号传染，因此所有病人都和0号在同一集合

开始每个人自成一个团体。a和b在同一个团体，就将a所在的集合和b所在的集合合并。
最终问0所在的集合有几个元素

例题1： POJ1611 The Suspects

```
MAX = 30000
parent = [ 0 for i in range(MAX+10) ]
total = [ 0 for i in range(MAX+10) ]
#total[GetRoot(a)]是a所在的group的人数
def GetRoot( a ):  #获取a的根，并把a的父结点改为根
       if parent[a]!= a:
              parent[a] = GetRoot(parent[a])
       return parent[a]
def Merge( a, b ):
       p1 = GetRoot(a)
       p2 = GetRoot(b)
       if p1 == p2:
              return
       total[p1] += total[p2]
       parent[p2] = p1
```

例题1： POJ1611 The Suspects

```
while True:
        n, m = list( map( int, input().split() ) )
        if n == 0 and m == 0:
                break
        for i in range(n):
                parent[i] = i
                total[i] = 1
        for i in range(m):
                lst = list( map( int, input().split() ) )
                k = lst[0]
                h = lst[1]
                for j in range( 2, k + 1 ):
                        Merge( h, lst[j] )
        print( total[GetRoot(0)] )    # 此处写parent[0]可否?
```

例题1： POJ1611 The Suspects

```
while True:
        n, m = list( map( int, input().split() ) )
        if n == 0 and m == 0:
                break
        for i in range(n):
                parent[i] = i
                total[i] = 1
        for i in range(m):
                lst = list( map( int, input().split() ) )
                k = lst[0]
                h = lst[1]
                for j in range( 2, k + 1 ):
                        Merge( h, lst[j] )
        print( total[GetRoot(0)] )    # 此处写parent[0]可否?
```

不行，因为可能还没进行过路径压缩，parent[0]的值不正确

并查集解题的套路

在GetRoot 和 Merge中维护必要的信息

注意：两棵树合并以后，parent[a]未必就是a的根，因为从a到根的路径可能还没经过压缩。GetRoot(a)后parent[a]才是a的根
GetRoot(a)后，如果又将a所在的集合并到其它集合上，那么parent[a]就又不是根了

并查集解题的套路

问有多少个集合，就是问有多少个元素的parent就是它自己