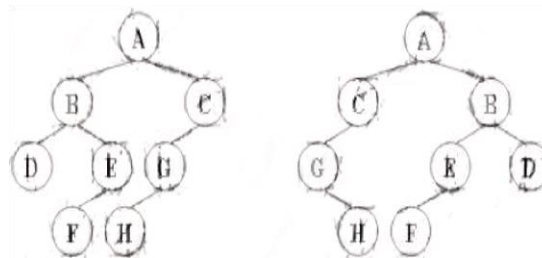
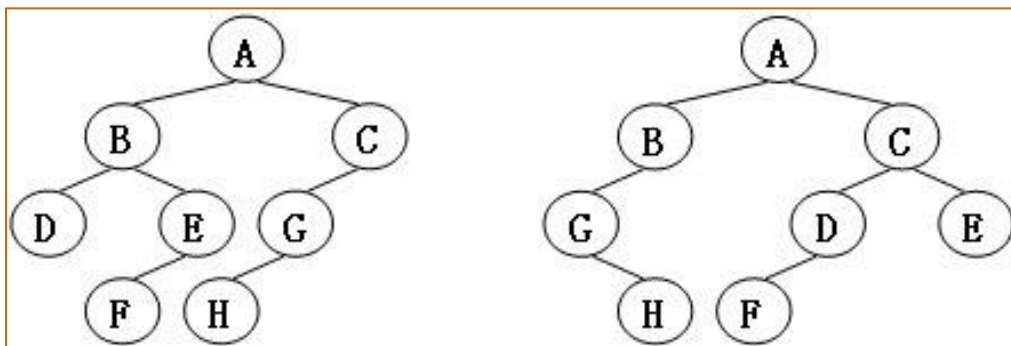
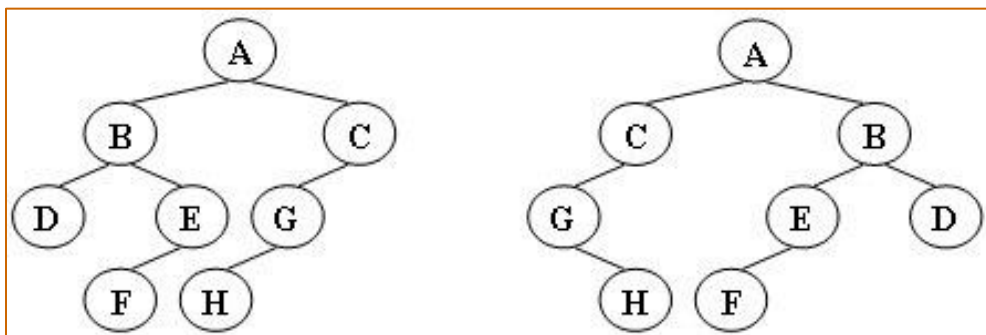


# 小白专场： 树的同构



# 题意理解

给定两棵树T1和T2。如果T1可以通过若干次左右孩子互换就变成T2，则我们称两棵树是“同构”的。  
现给定两棵树，请你判断它们是否是同构的。



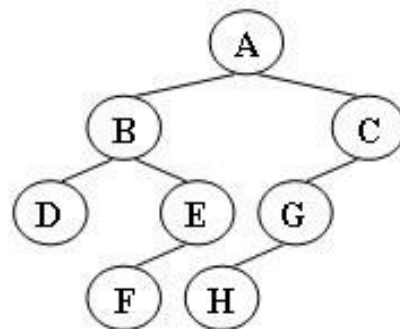
# 题意理解

输入格式：输入给出2棵二叉树的信息：

- 先在一行中给出该树的结点数，随后N行
- 第i行对应编号第i个结点，给出该结点中存储的字母、其左孩子结点的编号、右孩子结点的编号。
- 如果孩子结点为空，则在相应位置上给出“-”。

### 输入样例：

```
8
0 A 1 2
1 B 3 4
2 C 5 -
3 D - -
4 E 6 -
5 G 7 -
6 F - -
7 H - -
8
G - 4
B 7 6
F - -
A 5 1
H - -
C 0 -
D - -
E 2 -
```



# 题意理解

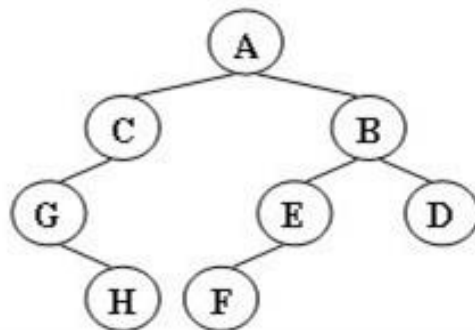
输入格式：输入给出**2**棵二叉树的信息：

- 先在一行中给出该树的结点数，随后**N**行
- 第*i*行对应编号第*i*个结点，给出该结点中存储的字母、其左孩子结点的编号、右孩子结点的编号。
- 如果孩子结点为空，则在相应位置上给出“-”。

### 输入样例：

```
8
A 1 2
B 3 4
C 5 -
D - -
E 6 -
G 7 -
F - -
H - -
8
```

```
0 G - 4
1 B 7 6
2 F - -
3 A 5 1
4 H - -
5 C 0 -
6 D - -
7 E 2 -
```



# 求解思路

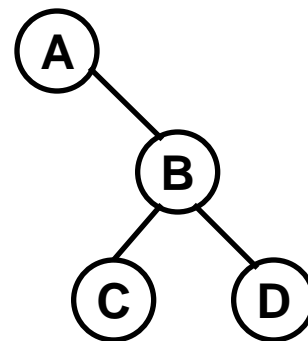
1. 二叉树表示
2. 建二叉树
3. 同构判别

# 二叉树表示

结构数组表示二叉树：静态链表

```
#define MaxTree 10
#define ElementType char
#define Tree int
#define Null -1

struct TreeNode
{
    ElementType Element;
    Tree Left;
    Tree Right;
} T1[MaxTree], T2[MaxTree];
```



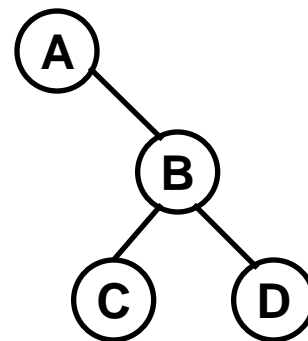
	A	B	C	D	
Left	-1	2	-1	-1	
Right	1	3	-1	-1	
	0	1	2	3	4

# 二叉树表示

## 结构数组表示二叉树

```
#define MaxTree 10  
#define ElementType char  
#define Tree int  
#define Null -1
```

```
struct TreeNode  
{  
    ElementType Element;  
    Tree Left;  
    Tree Right;  
} T1[MaxTree], T2[MaxTree];
```



	B	A		D	C
Left	4	-1		-1	-1
Right	3	0		-1	-1
	0	1	2	3	4

# 程序框架搭建

```
int main()  
{  
    建二叉树1  
    建二叉树2  
    判别是否同构并输出  
  
    return 0;  
}
```

需要设计的函数:

- 读数据建二叉树
- 二叉树同构判别

```
int main()  
{  
    Tree R1, R2;  
  
    R1 = BuildTree(T1);  
    R2 = BuildTree(T2);  
    if (Isomorphic(R1, R2)) printf("Yes\n");  
    else printf("No\n");  
  
    return 0;  
}
```



# 如何建二叉树

8	
0	A 1 2
1	B 3 4
2	C 5 -
3	D - -
4	E 6 -
5	G 7 -
6	F - -
7	H - -

```
Tree BuildTree( struct TreeNode T[] )
```

```
{
```

```
    .....
```

```
    scanf("%d\n", &N);
```

```
    if (N) {
```

```
        .....
```

```
        for (i=0; i<N; i++) {
```

```
            scanf("%c %c %c\n", &T[i].Element, &cl, &cr);
```

```
            .....
```

```
        }
```

```
        .....
```

```
        Root = ???
```

```
    }
```

```
    return Root;
```

```
}
```

T[i]中没有任何结点的  
left(cl)和right(cr)指向它。  
只有一个

# 如何建二叉树

```
Tree BuildTree( struct TreeNode T[] )
{
    .....
    scanf("%d\n", &N);
    if (N) {
        for (i=0; i<N; i++) check[i] = 0;
        for (i=0; i<N; i++) {
            scanf("%c %c %c\n", &T[i].Element, &cl, &cr);
            if (cl != '-') {
                T[i].Left = cl-'0';
                check[T[i].Left] = 1;
            }
            else T[i].Left = Null;
            ..... /*对cr的对应处理 */
        }
        for (i=0; i<N; i++)
            if (!check[i]) break;
        Root = i;
    }
    return Root;
}
```



# 如何判别两二叉树同构

```
int Isomorphic ( Tree R1, Tree R2 )
{
    if ( (R1==Null)&&(R2==Null) ) /* both empty */
        return 1;
    if ( ((R1==Null)&&(R2!=Null)) || ((R1!=Null)&&(R2==Null)) )
        return 0; /* one of them is empty */
    if ( T1[R1].Element != T2[R2].Element )
        return 0; /* roots are different */
    if ( ( T1[R1].Left == Null )&&( T2[R2].Left == Null ) )
        /* both have no left subtree */
        return Isomorphic( T1[R1].Right, T2[R2].Right );
    .....
}
```

# 如何判别两二叉树同构

```
int Isomorphic ( Tree R1, Tree R2 )
{
    .....
    if ( ((T1[R1].Left!=Null)&&(T2[R2].Left!=Null))&&
        ((T1[T1[R1].Left].Element)==(T2[T2[R2].Left].Element)) )
        /* no need to swap the left and the right */
        return ( Isomorphic( T1[R1].Left, T2[R2].Left ) &&
                  Isomorphic( T1[R1].Right, T2[R2].Right ) );

    else /* need to swap the left and the right */
        return ( Isomorphic( T1[R1].Left, T2[R2].Right) &&
                  Isomorphic( T1[R1].Right, T2[R2].Left ) );
}
```