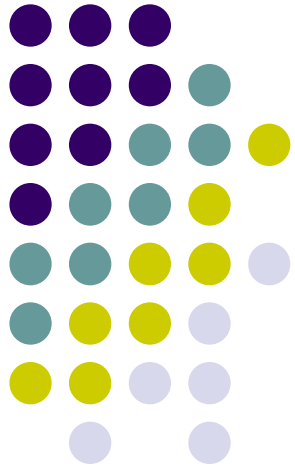
数据结构习题 第 5 章

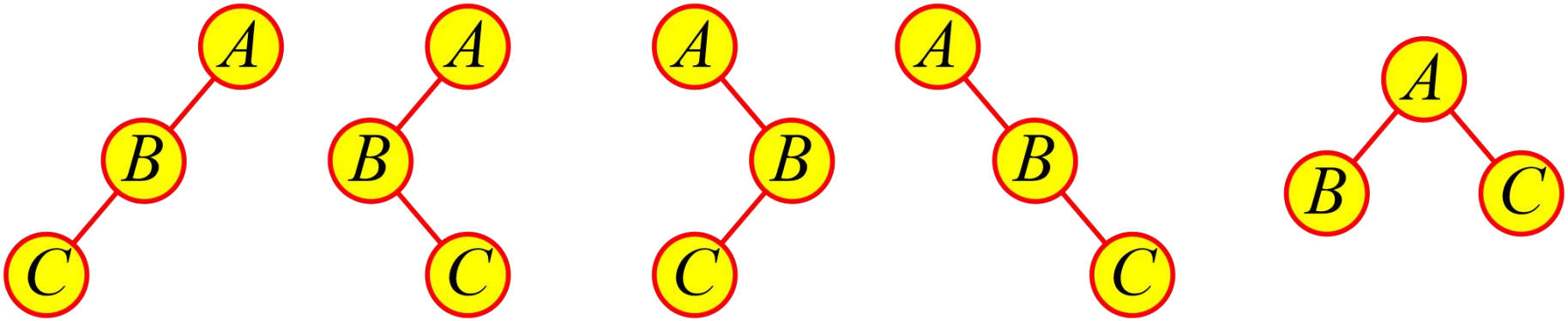


第1题

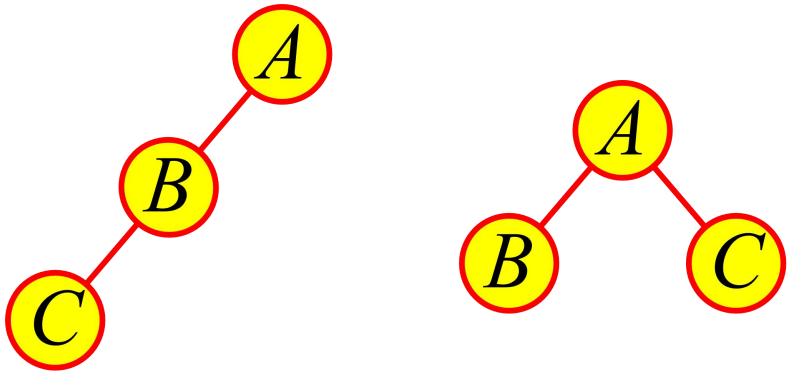
- 由三个结点*A*，*B*和*C*可以构成多少棵不同的树？可以构成多少棵不同的二叉树？

参考答案

- 二叉树有5种形态：6*5=30棵不同的树



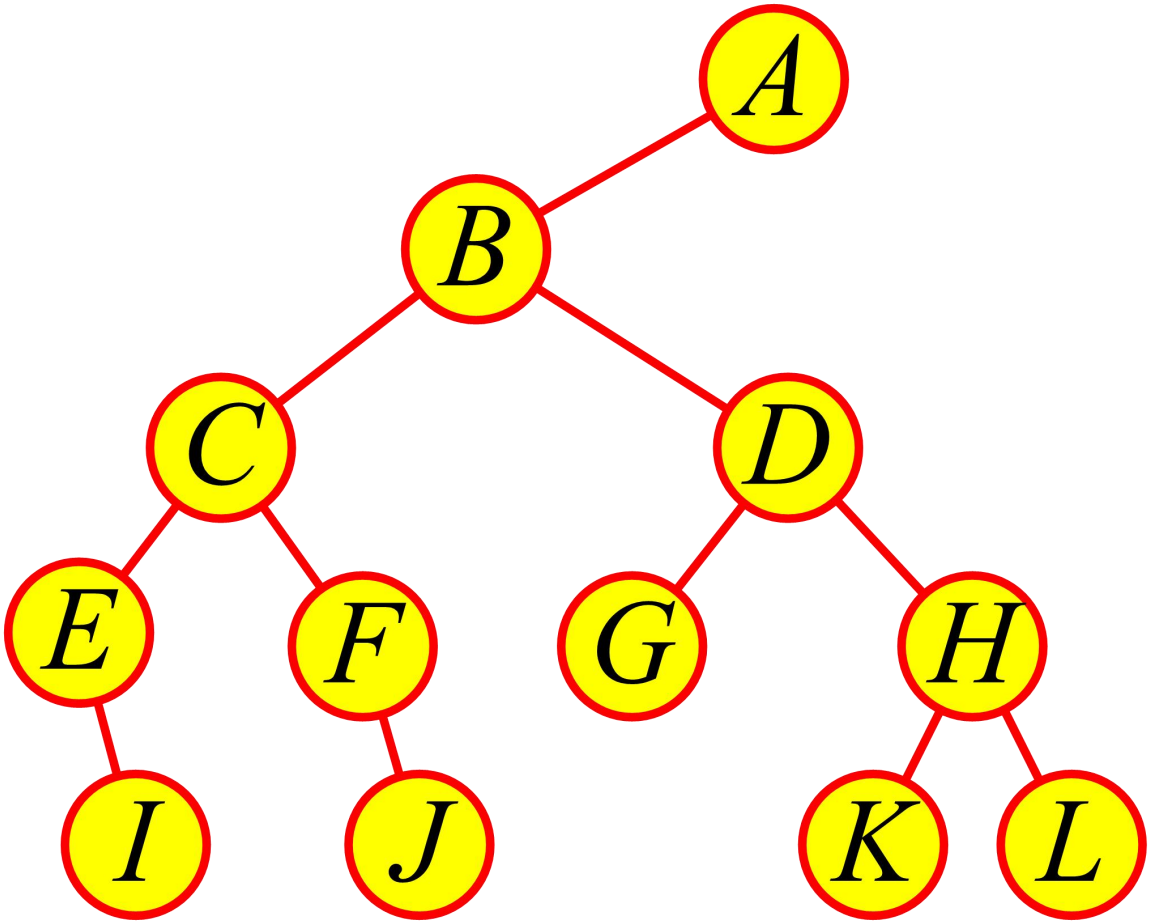
- 树有2种形态：6+3=9棵不同的树



第2题

- 判断以下命题是否为真？若真，请证明之；否则，举出反例。
  一棵二叉树形的所有的叶结点，在先根次序、中根次序和后根次序下的排列都按相同的相对位置出现。

分析



先根： A B C E I F J D G H K L

中根： E I C F J B G D K H L A

后根： I E J F C G K L H D B A

参考答案

使用数学归纳法进行证明。令h为二叉树高度；

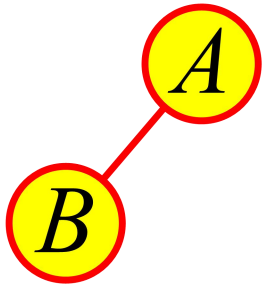
- h=0时，命题成立
- 假设 h = k 时命题成立，往证h=k+1时命题也成立。当 h = k+1 时，对任意两个叶结点l_1,l_2，有三种情况：
  - l_1，l_2 都在根的左子树中：根据归纳假设，显然。
  - l_1，l_2 都在根的右子树当中：同理。
  - l_1，l_2 不在根的同一个子树当中：根据三种遍历方式，左子树中节点都在右子树中结点之前出现，因此，在三种遍历中l_1均在l_2之前。

第3题

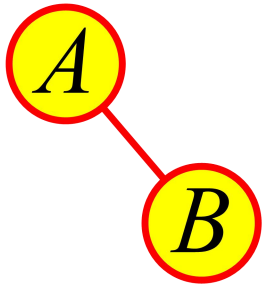
- 试找出分别满足下列条件的所有二叉树
  - 先根序列和中根序列相同；
  - 先根序列和后根序列相同；
  - 中根序列和后根序列相同；
  - 先根、中根、后根序列均相同。

分析

- 对于如下的二叉树，其先根遍历、中根遍历、后根遍历分别为：AB、BA、BA

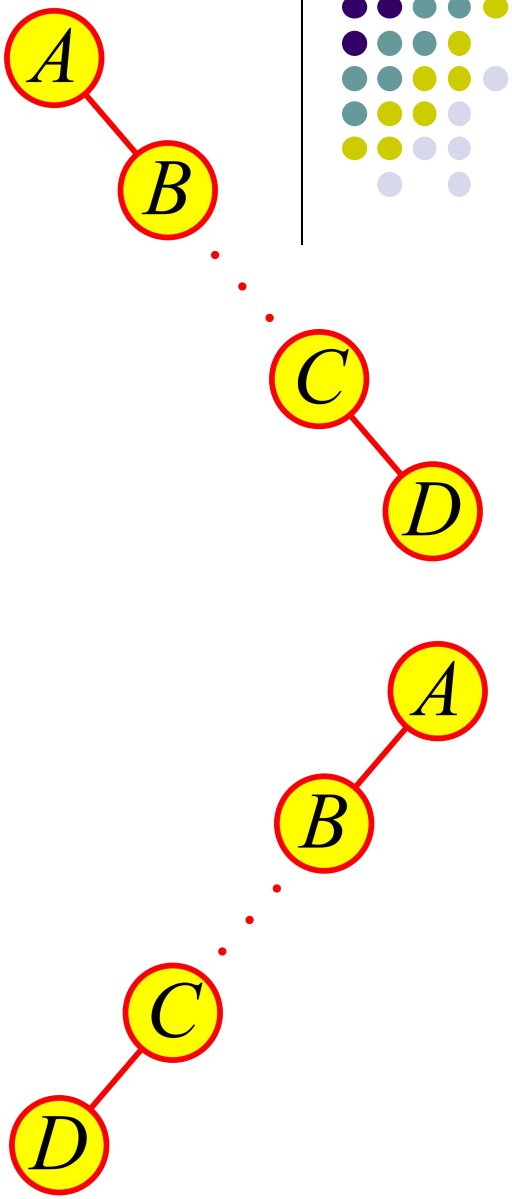


- 对于如下的二叉树，其先根遍历、中根遍历、后根遍历分别为：AB、AB、BA



参考答案

- 所有节点仅含右子节点的二叉树先根序列与中根序列相同。
- 空树或只含一个节点的二叉树的先根序列和后根序列均相同。
- 所有节点仅含左子节点的二叉树中根序列与后根序列相同。
- 空树或只含一个节点的二叉树的先根序列和后根序列均相同。

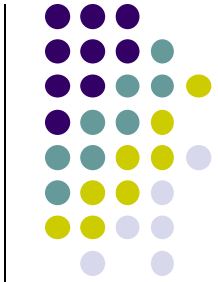


第4题

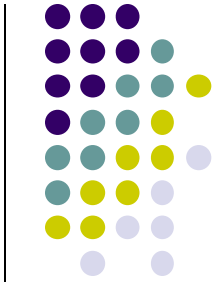
- 编写一算法，判别给定二叉树是否为完全二叉树。

分析

- 一棵二叉树是完全二叉树当且仅当其满足如下的条件：
  （1）任意节点不能有右孩子无左孩子；
  （2）任意节点不含两个子节点，按照层次序列其后的节点没有子节点。



```
bool IsComplete(BinTreeNode * t)
{
    Queue Q;
    if ( t == NULL ) return true;
    else Q.Insert(t);
    while ( true ) {
        p=Q.Delete();
        if  (p->left == NULL || p->right == NULL)  break; //
        Q.Insert(p->left);
        Q.Insert(p->right);
    }
```

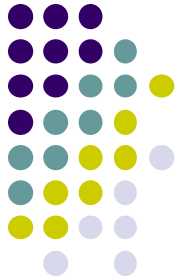


```
    if  (p->right != NULL)  return false;
    else if (p->left != NULL) Q.Insert(p->left);
    while (!Q.Empty()) { //看剩余节点是否有孩子
        p=Q.Delete();
        if ( p->left ! = NULL || p->right != NULL)
            return false;
    }
    return true;
}
```

第5题

- 编写算法求任意二叉树中一条最长的路径，并输出此路径上各结点的值。

分析



- 给定一棵树，如果左子树的高度大于右子树的高度，则最长路径的第一条边肯定是根节点的左分支，否则，第一条边是根节点的右分支。以此类推，可以找到第二条边到最后一条边。

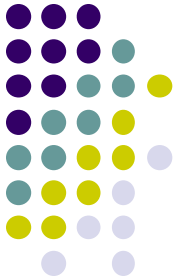
参考答案

```
int height(BinTreeNode<T>* t)
{
    if ( t==NULL ) return -1;
    return 1+max(height(t->left),height(t->right));
}
void path(BinTreeNode<T>* t)
{
    while ( t != NULL ){
        cout << t->data << " " <<endl;
        if (height(t->left)>height(t->right))  t=t->left;
        else t=t->right;
    }
}
```

第6题

- 编写算法判断两棵二叉树*T*和*T'*是否相似。两棵二叉树相似是指它们具有相同结构。

参考答案



算法Similar(t1, t2) /*判断两棵二叉树是否相似，t1、t2表示两棵树的根节点。若相似，返回值为true，否则为false*/

S1[递归出口]

IF t1=NULL AND t2=NULL **THEN RETURN** true.

IF t1=NULL OR t2=NULL **THEN RETURN** false.

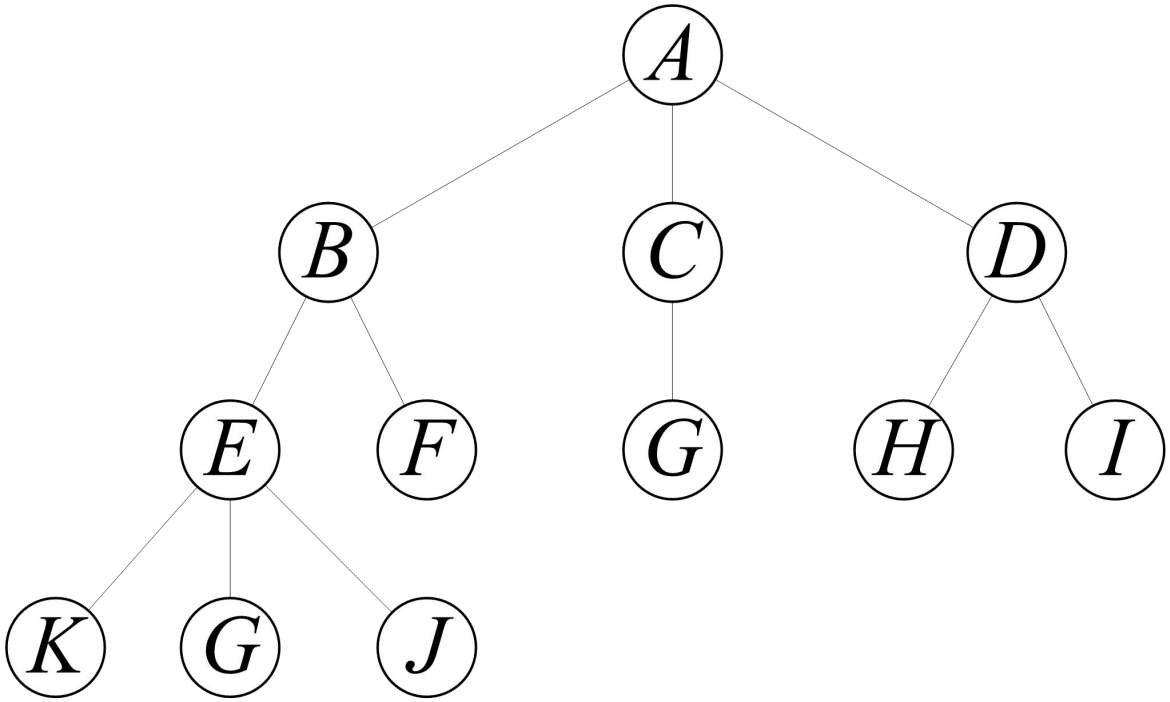
S2[递归调用]

RETURN Similar(left(t1),left(t2)) AND Similar(right(t1),right(t2)).

时间复杂度为O(min(n1, n2))

第7题

- 对于下图所示的树
  - （*a*）对其进行先根和后根遍历。
  - （*b*）给出其在自然对应下的二叉树。



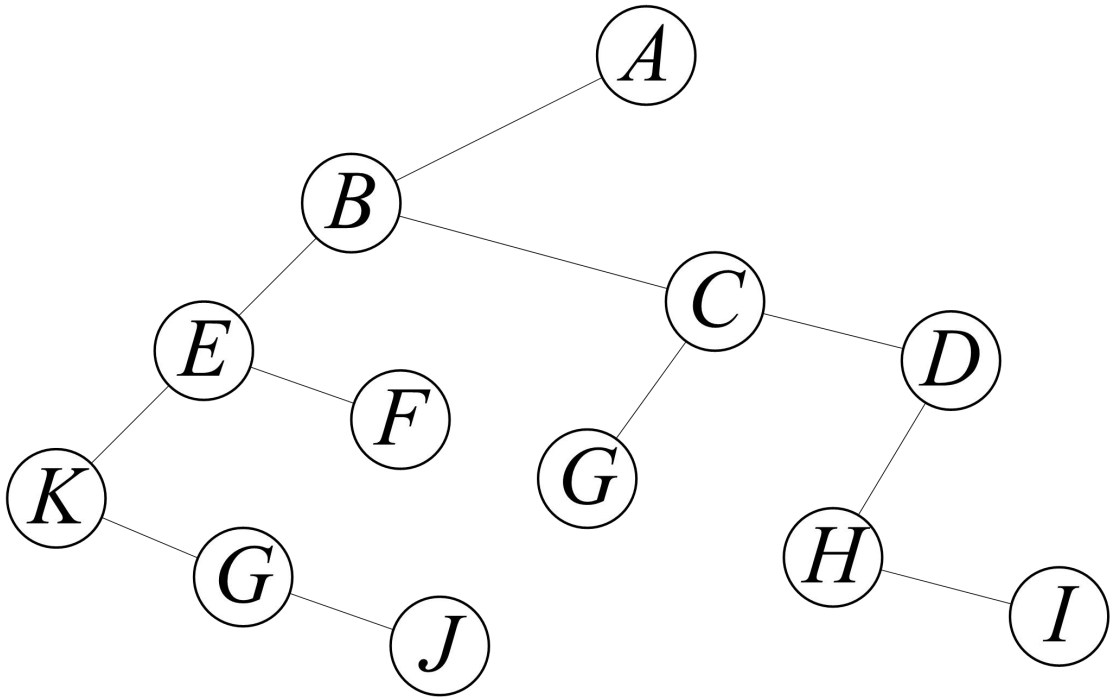
第7题参考答案

- （*a*）对其进行先根和后根遍历。

  先根遍历：ABEKGJFCGDHI

  后根遍历：KGJEFBGCHIDA

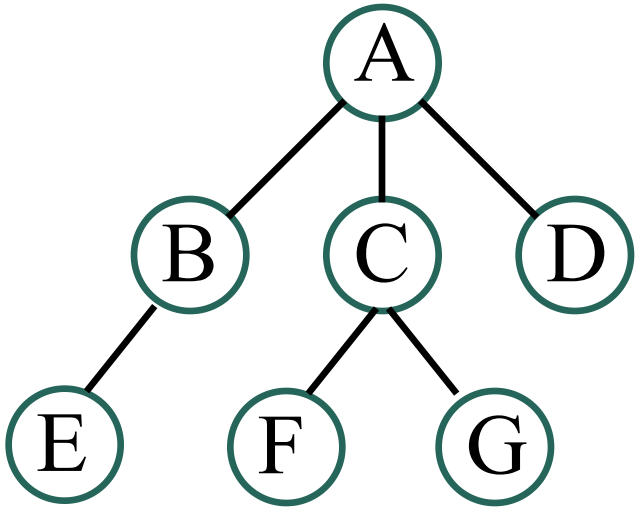
- （*b*）



第8题

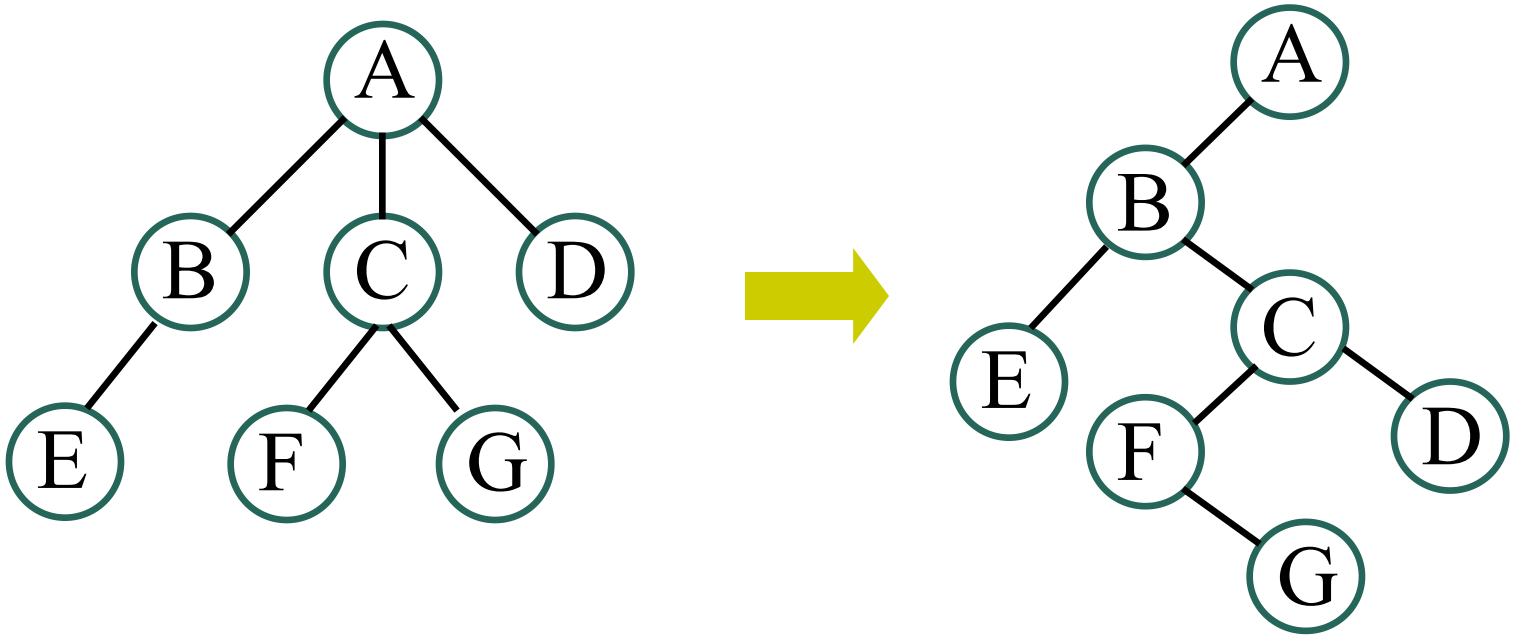
- 对以左儿子—右兄弟链接表示的树，编写计算树的深度的算法。

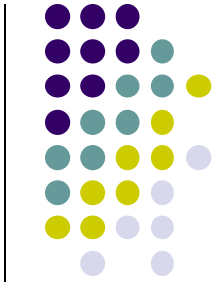
分析：解题思路



$$depth(t)=\begin{cases}-1 & t=\text{L}\\ 1+\max\{depth(p)\mid p\text{为}t\text{的子节点}\} & \text{否则}\end{cases}$$

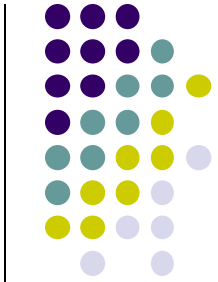
分析：解题思路





```
算法 Depth(t. d)  // 递归算法
D1[递归出口]
    IF  t = Λ THEN (d← -1. RETURN)
D2[递归调用]
    p ← FirstChild(t).
    max ← -1.  // max为各子树最大深度
    WHILE p ≠ Λ DO (
        Depth(p. dp).
        IF dp > max THEN max←dp.
        p←NextBrother(p).
    )
    d← 1 + max. RETURN.
```

```
算法 Depth2(t. d)  // 迭代算法
CREATES(NS). CREATES(RS). NS ⇐ (t, 0)
WHILE NS is nonempty DO (
    (p, flag) ⇐ NS
    IF flag = 0 THEN ( // p尚未访问
        NS ⇐ (p, 1)
        q ← FirstChild(p)
        WHILE q ≠ Λ DO (
            NS ⇐ (q, 0). q←NextBrother(q).
        )
    ) ELSE ( // p已访问
        IF p = Λ THEN RS ⇐ -1 // 递归出口
```

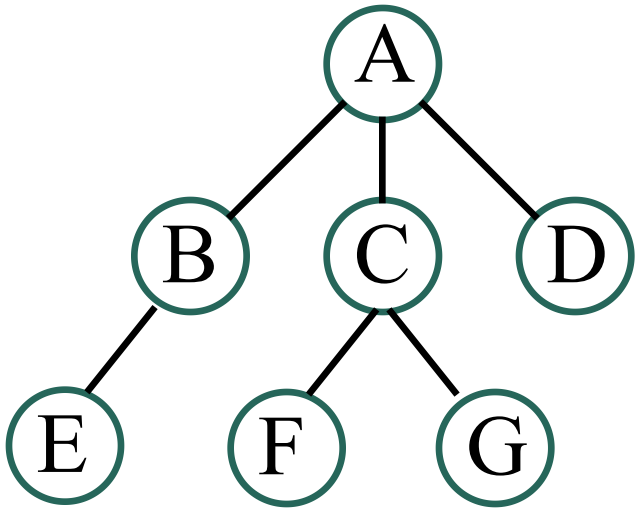


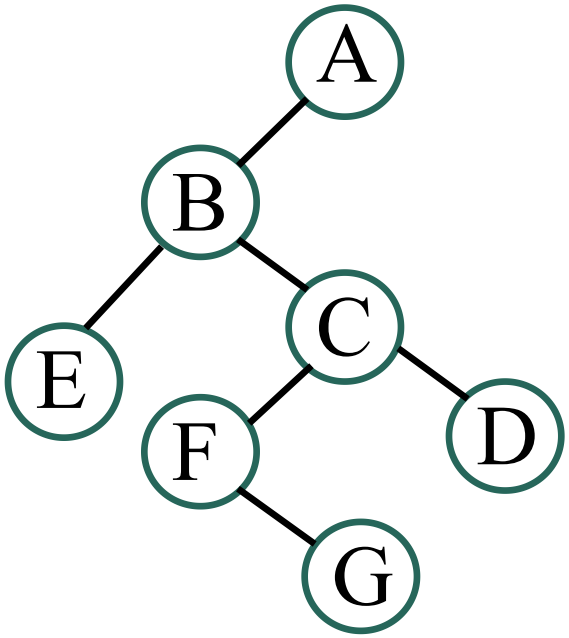
```
算法 Depth2(t. d)[续]
        ELSE (  // 结果综合
            p ← FirstChild(t). max ← -1
            WHILE p ≠ Λ DO (
                dp ⇐ RS
                IF dp > Max THEN max←dp.
                p←NextBrother(p).
            )
            RS ⇐ max+1
        )
    )
)
d ⇐ RS.
```

分析：其他解题思路

对树做层次遍历，每遍历一层树的深度+1.

关键：将队列中的结点结构变为(结点，该结点的层数i) 。





算法Depth(*t*. *d*)

//思想：对树做层次遍历，每遍历一层树的深度+1.

IF *t* = Λ **THEN** (*d* ← -1. **RETURN**) //判断t是否为Λ

CREATEQ(Q). Q ⇐(*t*,0) . //创建辅助队列, 根结点入队

WHILE NOT(IsEmpty(Q)) **DO** (//利用队列Q遍历第d层结点

(*p*, *d*) ⇐ Q.

p← FirstChild(*p*)

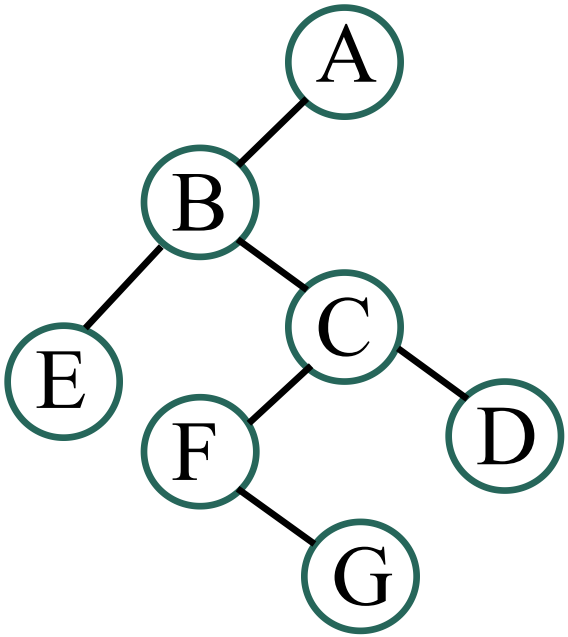
WHILE *p* ≠ Λ **DO** (

Q ⇐ (*p*, *d*+1)

p ← NextBrother(*p*) .

)

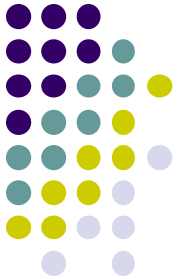
)



第9题

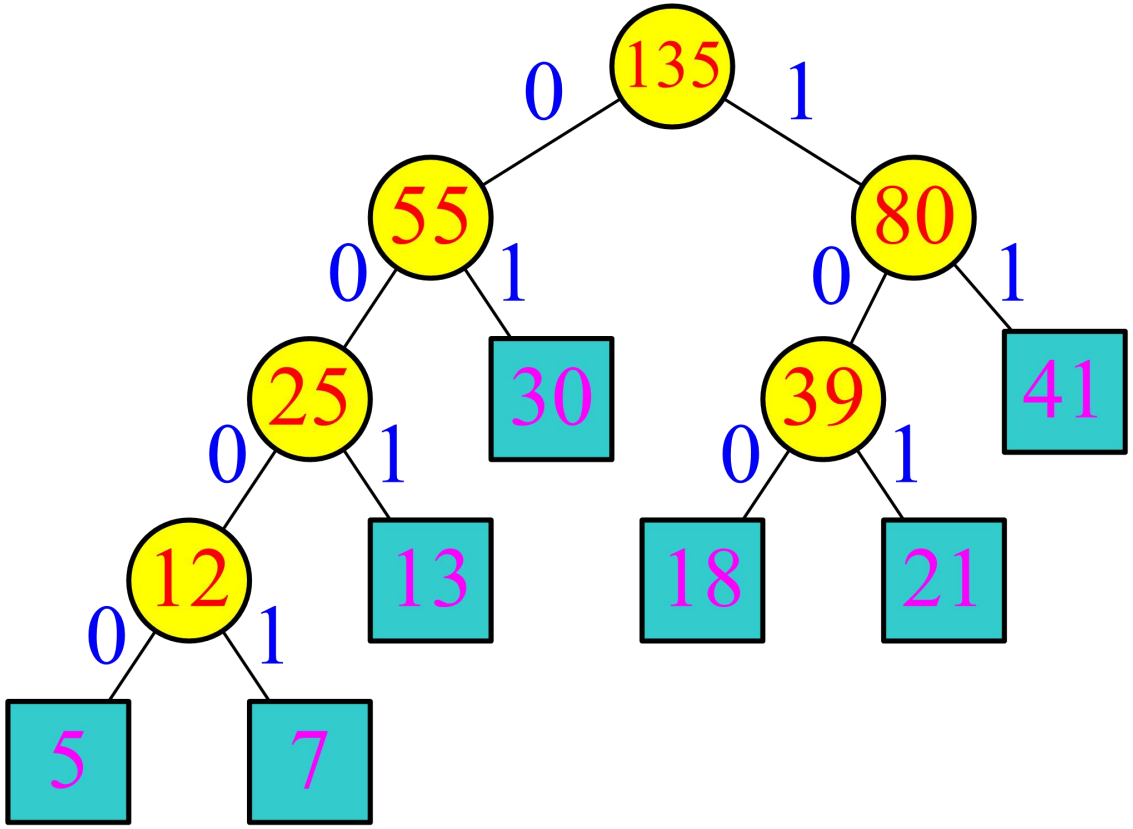
- 构造权值为{5, 13, 21, 7, 18, 30, 41}的哈夫曼树。

第9题分析



- 首先，在森林中取权值最小的两个根结点s和n，合成一棵二叉树，新生成的结点T1，作为这两个结点的父结点，T1的权值是两个子结点的权值之和；
- 对新的森林重复上一步操作，直至森林中只有唯一的根结点时，终止操作。

第9题答案



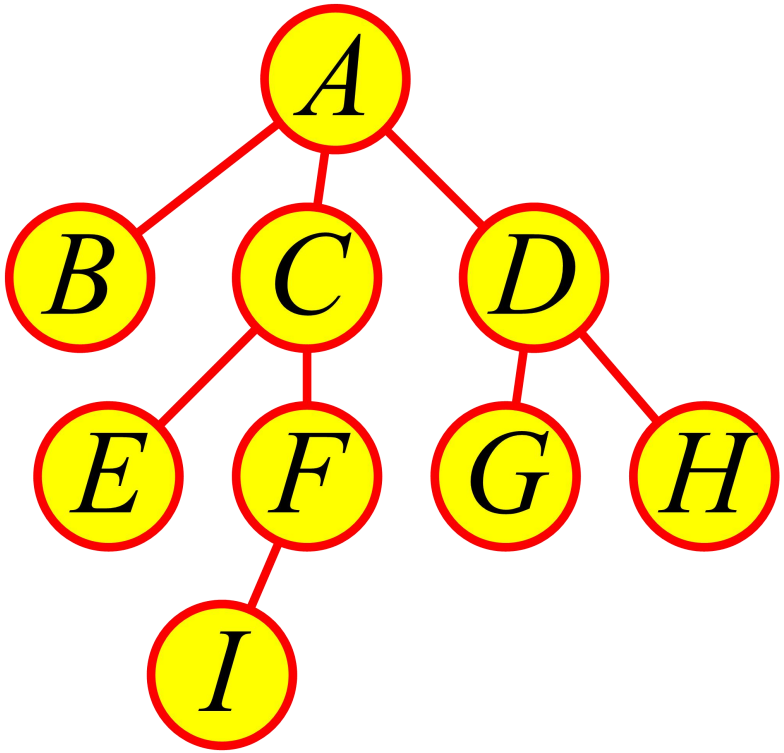
{5, 13, 21, 7, 18, 30, 41}

第10题

- 已知一棵树的层次遍历序列及相应的每个节点的次数序列，请写出构造此树的左孩子—右兄弟链接表示的算法。

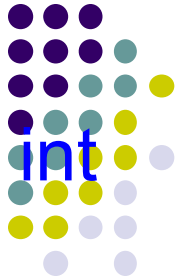
分析

A	B	C	D	E	F	G	H	I
3	0	2	2	0	1	0	0	0



参考答案

```
TreeNode * Level2Tree( char * nodes, int * degrees,
    int size )
{
    TreeNode * p = new TreeNode(nodes[0]);
    Queue q( size );
    q.QInsert(  p );
    for ( int i = 0, j = 1; i < size; j += degrees[i++] ) {
        p = q.QDelete( p );
        Children( p, nodes + j, degrees[i], q );
    }
}
```



```
void Children(TreeNode * parent, char * nodes, int
   d, Queue & q )
{
    if ( d == 0 ) { parent->child = NULL; return; }
    parent->child = new TreeNode( nodes[0] );
    TreeNode * ch = parent->child;  q.QInsert(ch);
    for ( int i = 1; i < d; i++) {
        ch->sibling = new TreeNode( nodes[i] );
        ch = ch->sibling;  q.QInsert(ch);
    }
    ch->sibling = NULL;
}
```

补充1：腾讯某年笔试题

已知一棵二叉树，如果先序遍历的节点顺序是：ADCEFGHB，中序遍历是：CDFEGHAB，则后序遍历结果为（　）

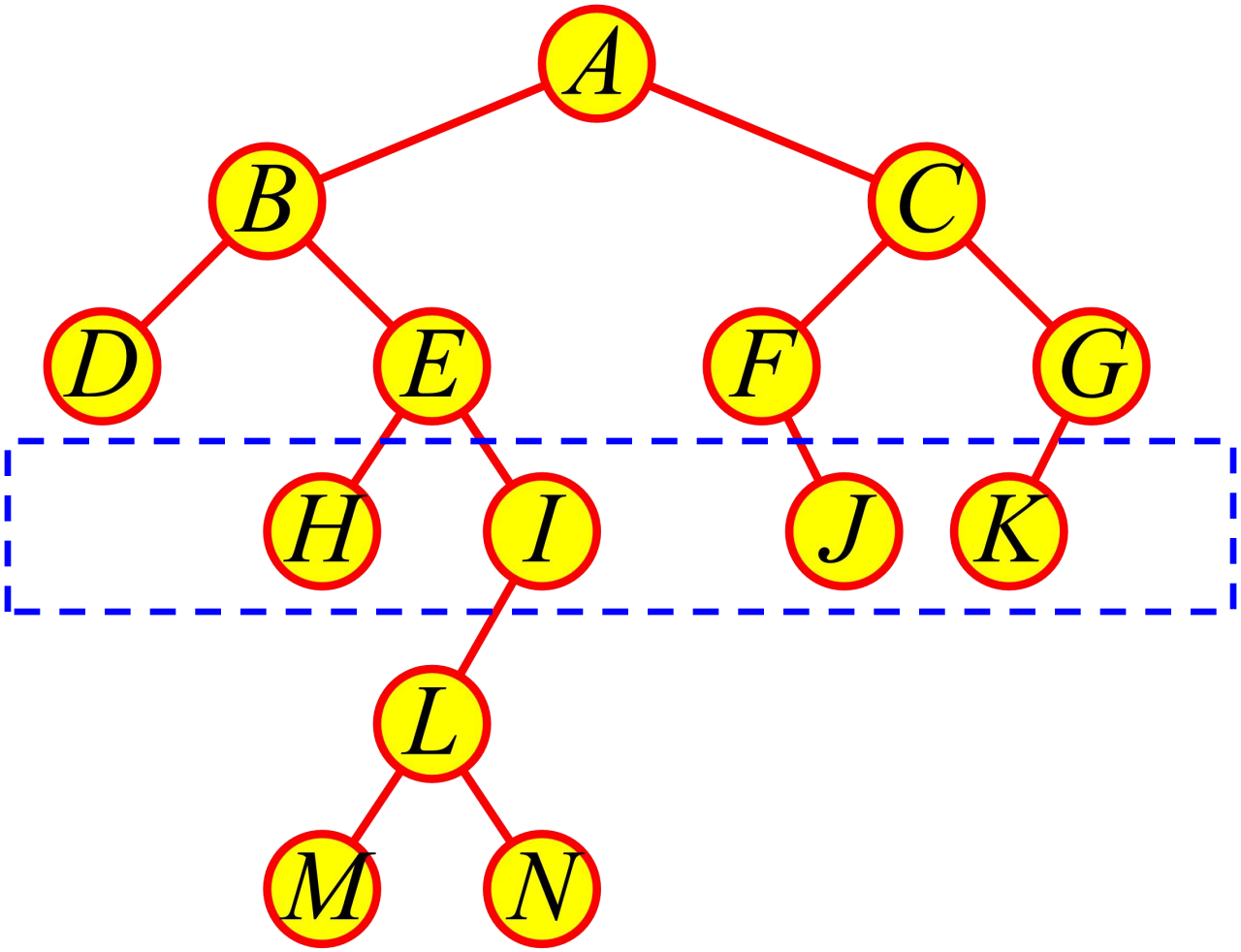
A．CFHGEBDA　　B．CDFEGHBA

C．FGHCDEBA　　D．CFHGEDBA

补充2

- 设计基于迭代的算法计算二叉树中第i层的节点数。

分析

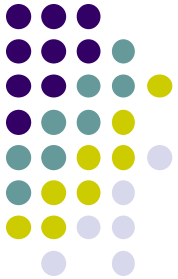


分析

- 我们使用#nodes(t, i)表示以t为根的二叉树第i层的节点数，#nodes(t, i)递归定义如下：

$$\#nodes(t,i)=\begin{cases}0 & t=\text{L}\\ 1 & t^{1}\ \text{L 且}\\ & i=0\\ \#nodes(Left(t),i-1)+ & \text{否则}\\ \#nodes(Right(t),i-1) & \end{cases}$$

分析:递归算法



算法 #nodes(t, i. n)

#n1[递归出口]

IF t = Λ **THEN** (d← 0. **RETURN**)

IF i = 0 **THEN** (d← 1. **RETURN**)

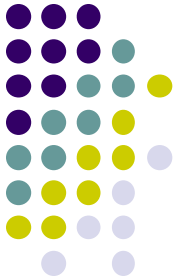
#n2[递归调用]

#nodes(Left(t), i - 1. n1).

#nodes(Right(t), i - 1. n2).

n ← n1 + n2. ▌

参考答案



算法 #nodes2(*t*, *i*. *n*)

CREATE(*NS*). CREATE(*RS*). *NS* ⇐ (*t*, *i*, 0)

WHILE *NS* is nonempty **DO** (

(*p*, *i*, *flag*) ⇐ NS

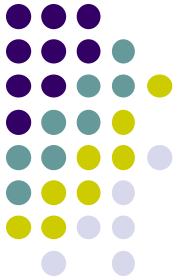
IF *flag* = 0 **THEN** (// p节点尚未访问

IF *t* = Λ **THEN** *RS* ⇐ 0. // 递归出口

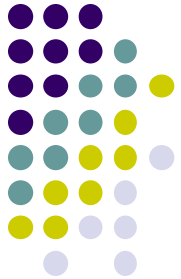
ELSE IF *i* = 0 **THEN** *RS* ⇐ 1. // 递归出口

算法 #nodes2(*t*, *i*. *n*)[续]

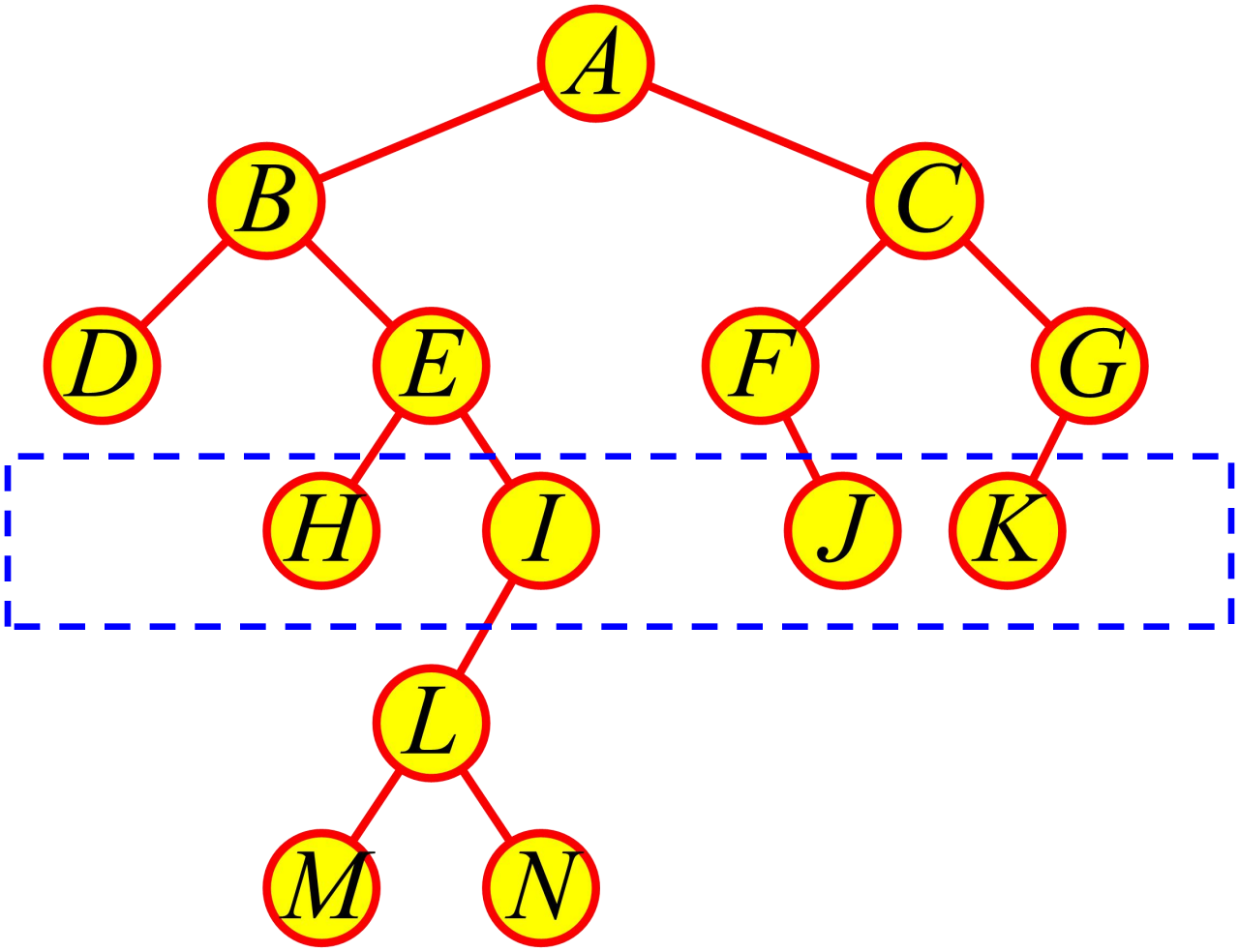
```
        ELSE ( // p节点已访问
                NS ⇐ (p, i, 1).
                NS ⇐ (Left(p), i – 1, 0). // 子节点入栈
                NS ⇐ (Right(p), i – 1, 0). //子节点入栈
        )
    )
    ELSE ( //结果出栈，综合后新结果入栈
        nr ⇐ RS. nl ⇐ RS. RS ⇐ nl + nr.
    )
)
n ⇐ RS.
```



另一种思路



- 使用层次遍历法，在每层之间加一个间隔符，这样就可以计算每层之间的节点数。


A
B
C
D
E
F
G
H
I
J
K
L
M
N

参考答案2

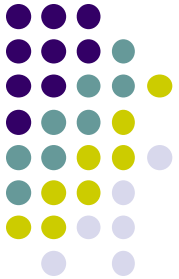
算法#nodes3(*t, i. n*)

LevelOrder1. [建空队并入队]

CREATE(*Q*).

IF $t = \Lambda$ **THEN (**$n \leftarrow 0$. **RETURN)**

$Q \Leftarrow$ #. $Q \Leftarrow t$.



算法#nodes3(*t, i. n*) [续]

LevelOrder2. [层次遍历]

WHILE *Q* is nonempty **DO (**

p ⇐ *Q* .

IF *p* = # **THEN (**

IF *i* = 0 **THEN BREAK**

ELSE (*i* ← *i* – 1. *Q* ⇐ #.)

) ELSE (

IF *Left*(*p*) ≠ Λ **THEN** *Q* ⇐ *Left*(*p*).

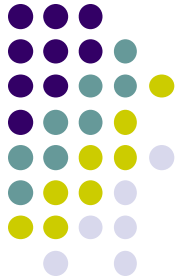
IF *Right*(*p*) ≠ Λ **THEN** *Q* ⇐ *Right*(*p*).

)

)

n ← *Size*(*Q*). ▌

变型：某年百度笔试题



给定以下二叉树：

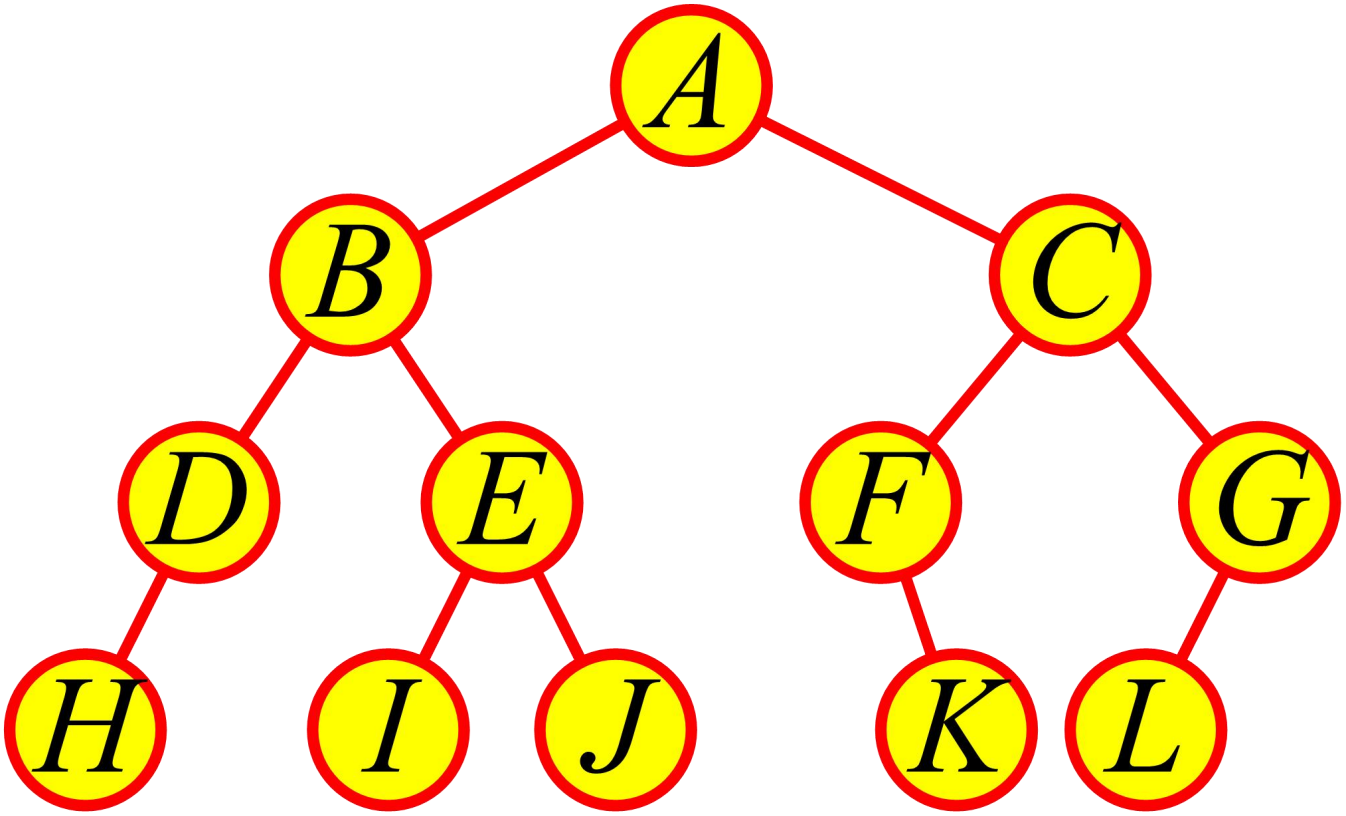
```
struct node_t
{
    node_t *left, *right;
    int value;
};
```

要求编写函数 node_t* foo(node_t *node, unsigned int m, unsigned int k)输出以 node 为根的二叉树第 m 层的第 k 个节点值。(注：level, k 均从 0 开始计数)

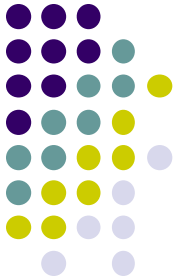
补充3：计算所某年考研题

- 给定一个以t为根的二叉树，求其中任意节点u和v的最近公共祖先。

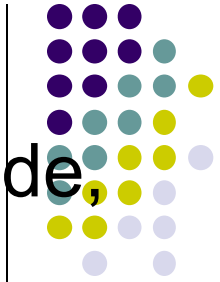
分析



参考答案

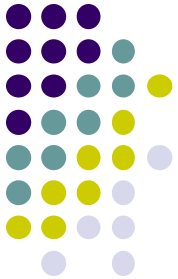


```
BinTreeNode * CommonAncestor(BinTreeNode *
   root, BinTreeNode * u, BinTreeNode * v )
{
    BinTreeNode * path1[N], *path2[N];
    int len1 = Path(root, u, path1);
    int len2 = Path(root, v, path2);
    int i, len = len1 < len2 ? len1 : len2;
    for (i = 0; i < len&&path1[i] == path2[i]; i++ );
    return path1[i-1];
}
```



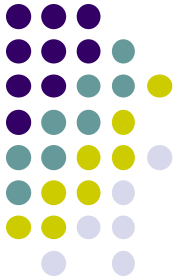
```
int Path(BinTreeNode * root, BinTreeNode * node,
    BinTreeNode ** path )
{
    if ( root == NULL ) return -1;
    if ( root == node ) return 0;
    path[0] = root;
    int len = Path(root->left, node, path+1);
    if (len != -1) return len;
    len = Path(root->right, node, path+1 );
    return len != -1 ? len : -1;
}
```

补充4：阿里巴巴某年笔试题



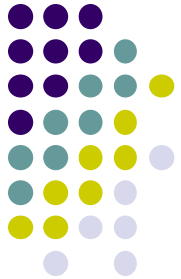
- 写一个函数，输入一个二叉树，树中每个节点存放了一个整数值，函数返回这棵二叉树中相差最大的两个节点间的差值绝对值。请注意程序效率。

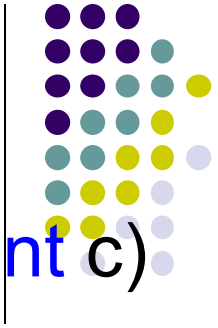
参考答案



```
void MaxMin(BTreeNode *root, int &max, int &min)
{
    if (root->left == NULL && root->right==NULL) {
        max= min = root->data;
    }
    else if (root->left == NULL) {
        MaxMin( root->right, max, min );
        if (root->data > max) max = root->data;
        else if (root->data < min) min = root->data;
    }
```

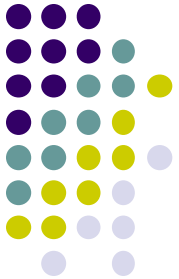
```
    else if (root->right == NULL) {
        MaxMin( root->left, max, min );
        if (root->data > max) max = root->data;
        else if (root->data < min) min = root->data;
    }
    else {
        int lmax, lmin, rmax, rmin;
        MaxMin( root->left, lmax, lmin );
        MaxMin( root->right, rmax, rmin );
        max = Max(root->data, lmax, rmax);
        min = Min(root->data, lmin, rmin);
    }
}
```





```
int Max(int a, int b, int c)
{
    if ( a > b ) {
        if ( a > c ) return a;
        else return c;
    } else {
        if ( b > c ) return b;
        else return c;
}
```

```
int Min(int a, int b, int c)
{
    if ( a < b ) {
        if ( a < c ) return a;
        else return c;
    } else {
        if ( b < c ) return b;
        else return c;
}
```



THE END