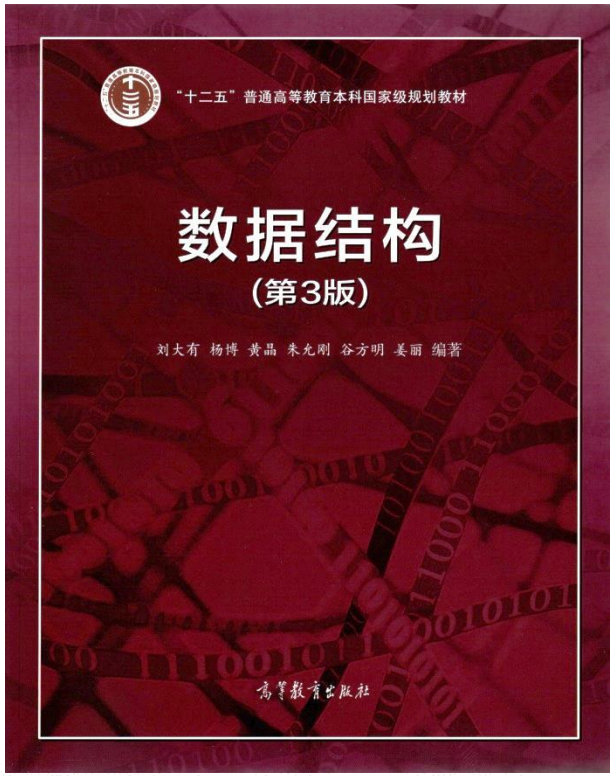


二叉树的分层遍历和路径

- 二叉树的分层遍历
- 二叉树的路径

数据之法
结构之美
算法之道

zhuyungang@jlu.edu.cn

Last updated on 2024.11

统计二叉树每层结点信息

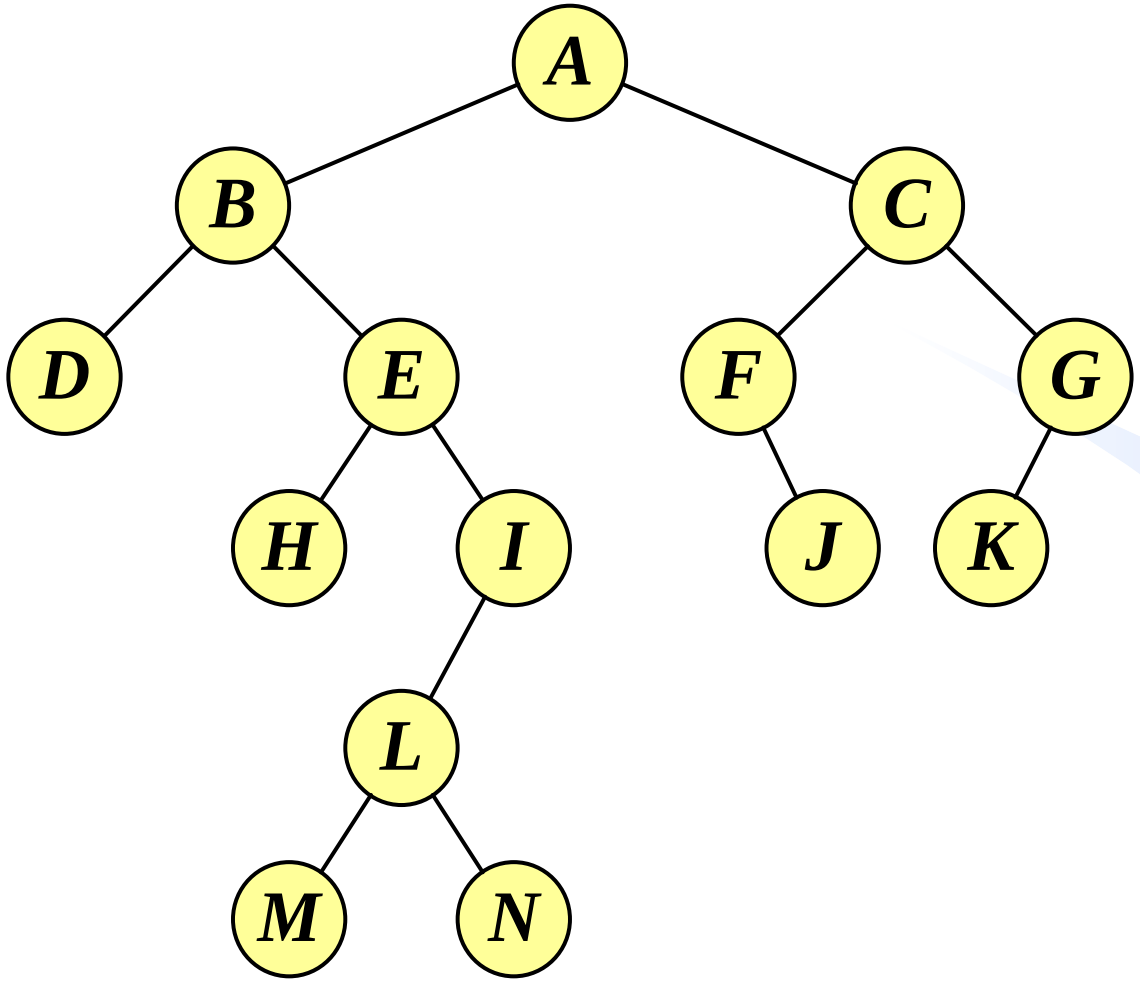
统计一棵二叉树中每层叶结点的数目【吉林大学上机考试题】。

➢需要识别每层的结束，每层结束后，统计该层信息。

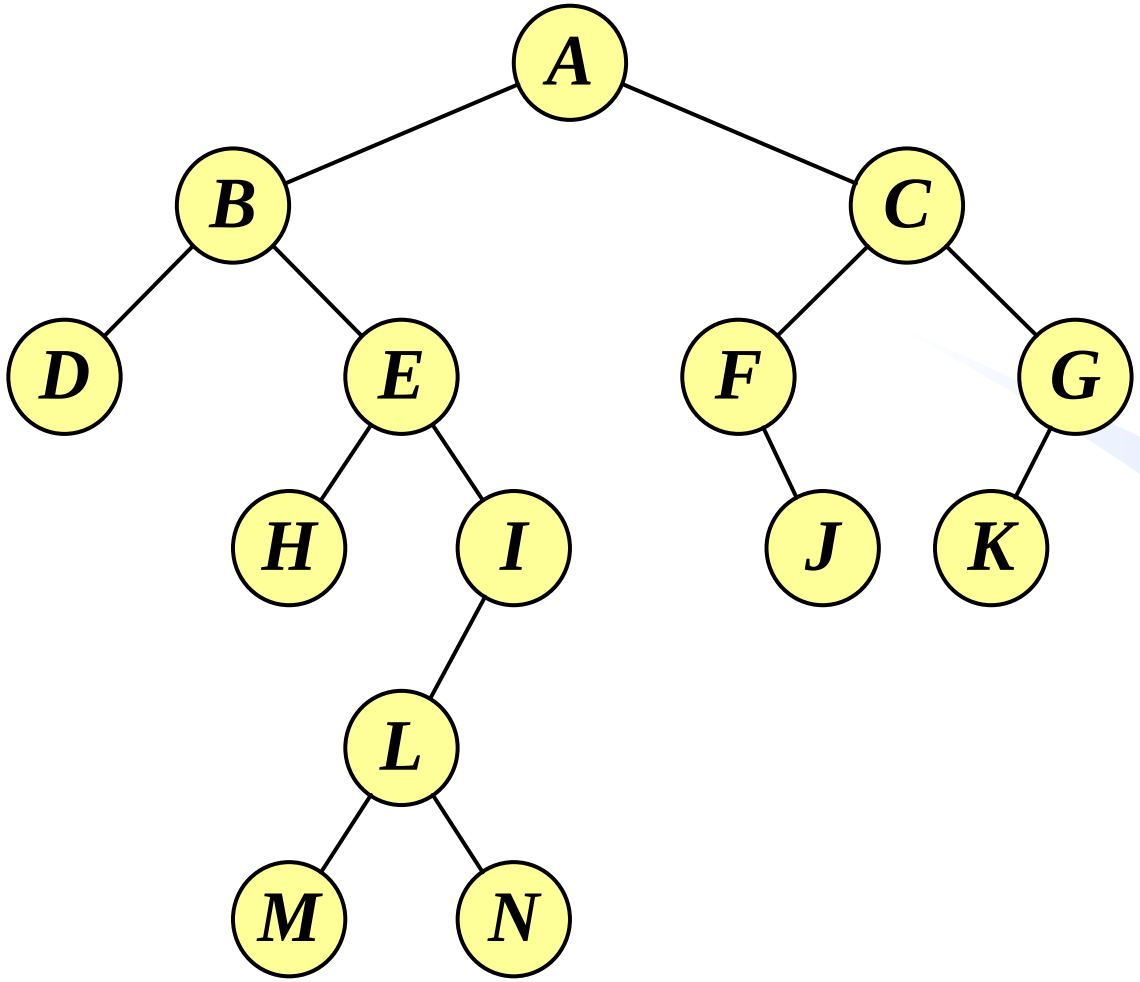
方案1:

✓可以设置一个不同于队列中其他结点的特殊结点（比如空结点NULL）来表示每层的结束。

✓遍历第0层时，先将根结点入队，再将NULL入队。在遍历过程中，当NULL出队时，表示已经遍历完本层，将NULL再入队，此时NULL即为下一层的结尾。

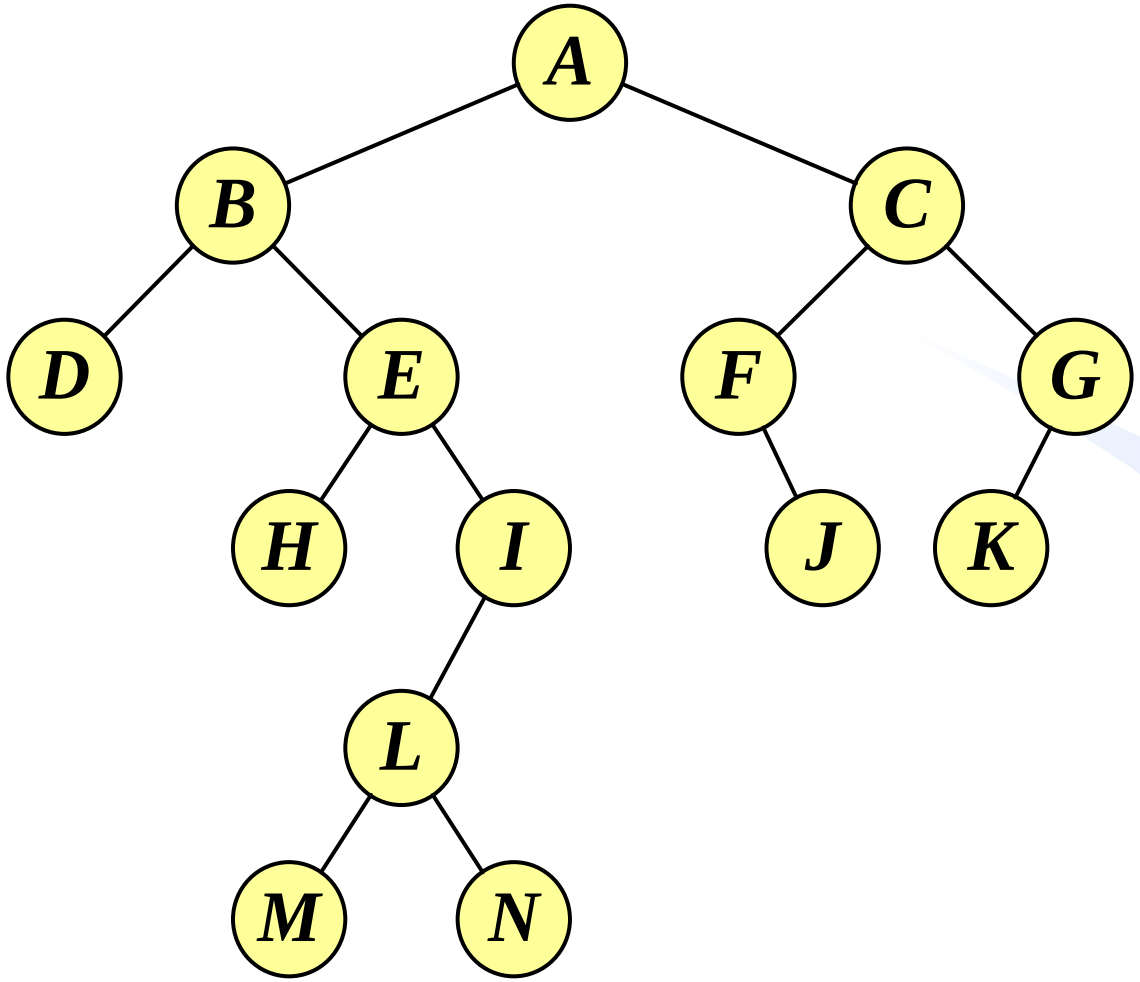


A null



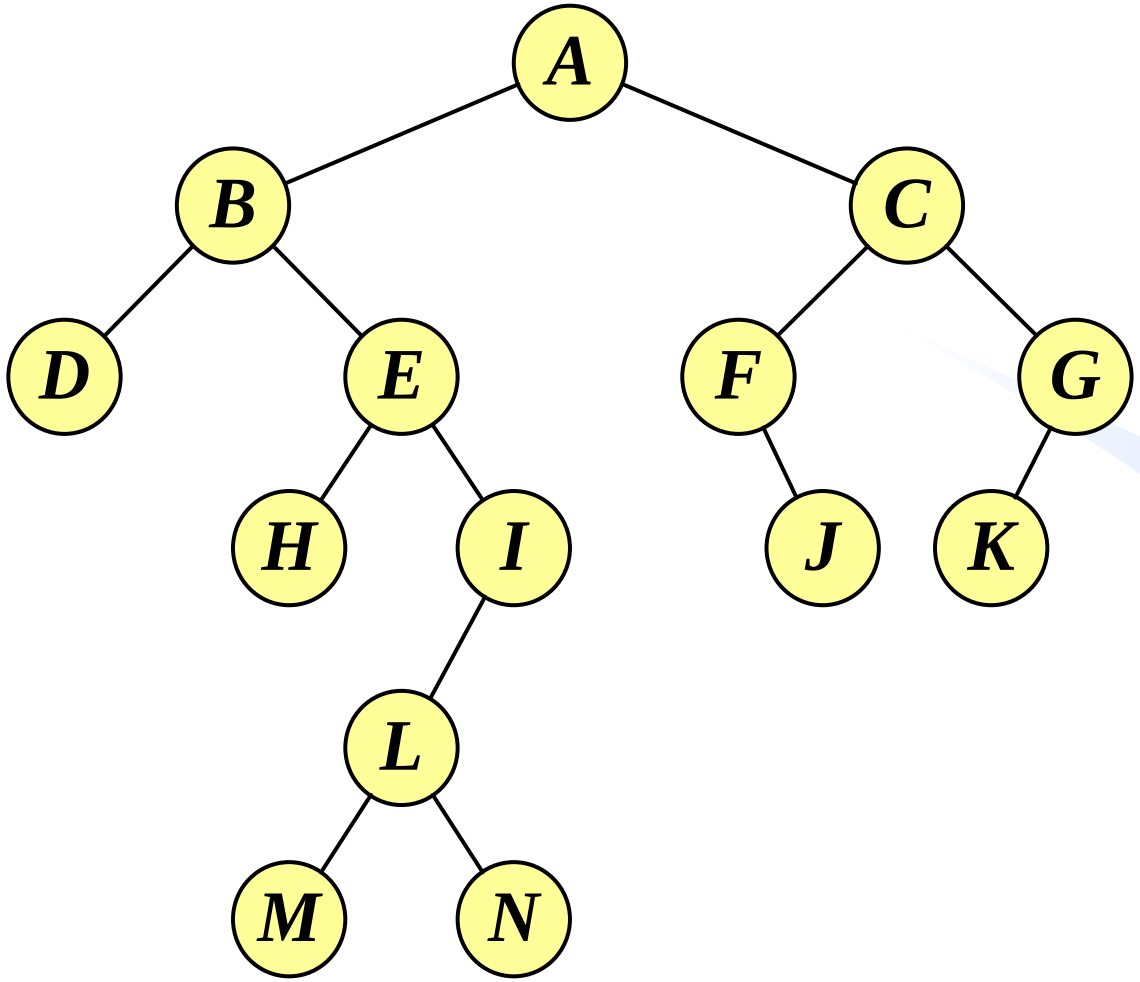
A

null B C


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

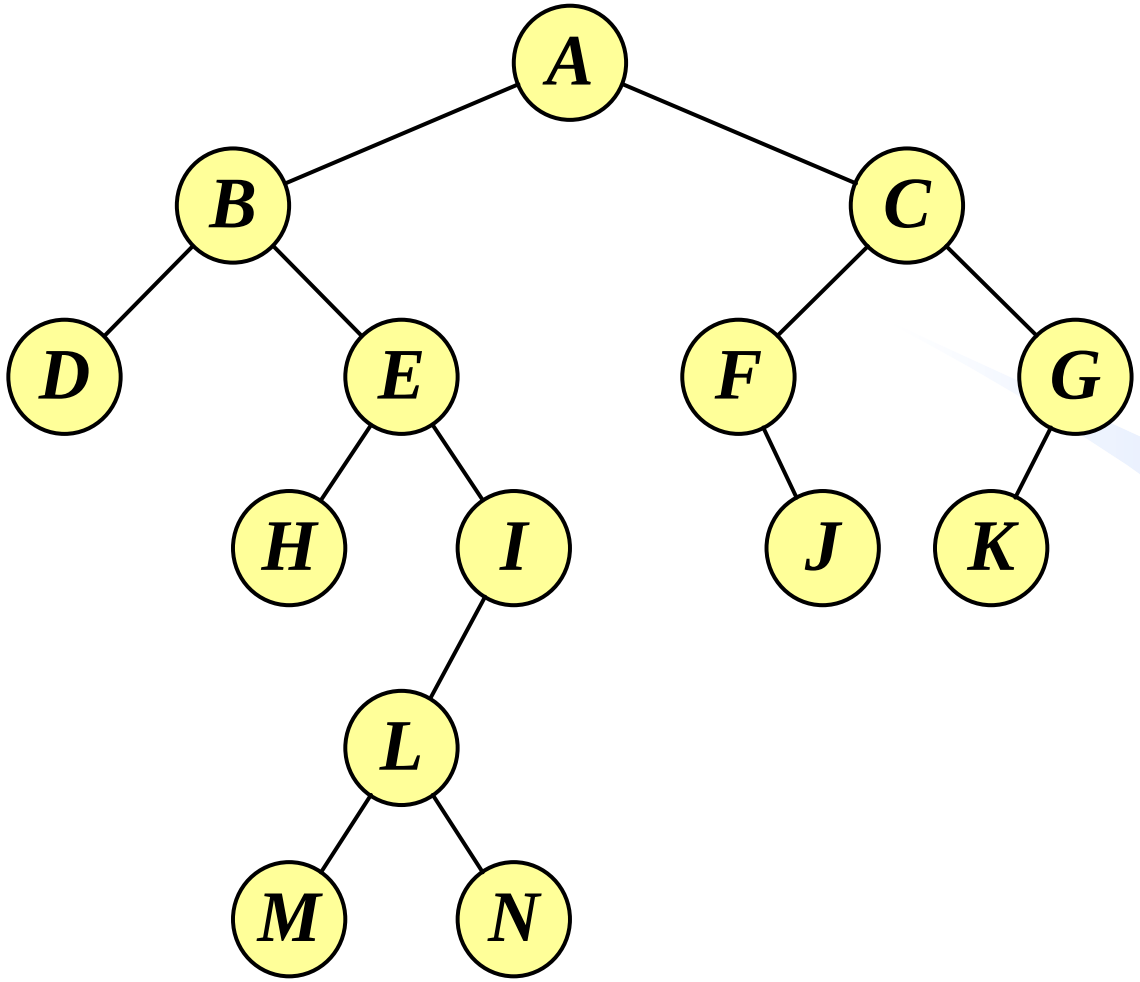

A
null


B
C
null


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

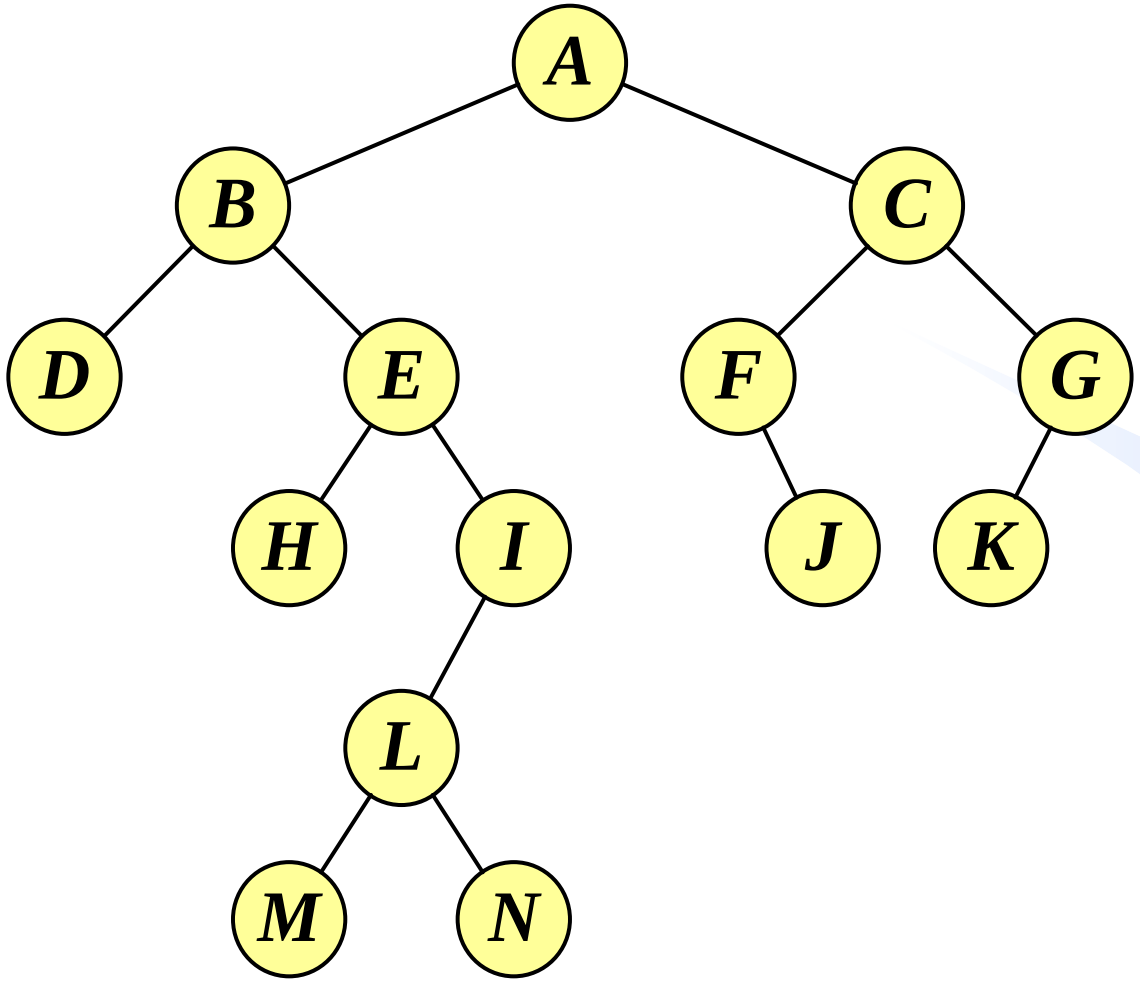

A
null
B


C
null
D
E


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


A
null
B
C


null
D
E
F
G



A null B C null

D E F G null

```
int LeafInEachLevel (TreeNode *root, int leafnum[]){
    if(root==NULL) return -1;
    Queue<TreeNode*> q ; int h=0;
    q.enQueue(root);
    q.enQueue(NULL); //NULL入队
    while (!q.empty()){
        TreeNode * p=q.deQueue(); //出队一个结点
        if(p==NULL) {                    //本层结束
            h++;
            if(!q.empty()) q.enQueue(NULL);
        }else{
            if(p->left==NULL && p->right==NULL) leafnum[h]++;
            if(p->left!=NULL) q.enQueue(p->left);
            if(p->right!=NULL) q.enQueue(p->right);
        }
    }
    return h-1; //返回二叉树的高度
};
```

***leafnum[h]*表示第*h*层的叶结点数。调用时*leafnum*数组各元素值应初始化为0。**

统计二叉树每层结点信息

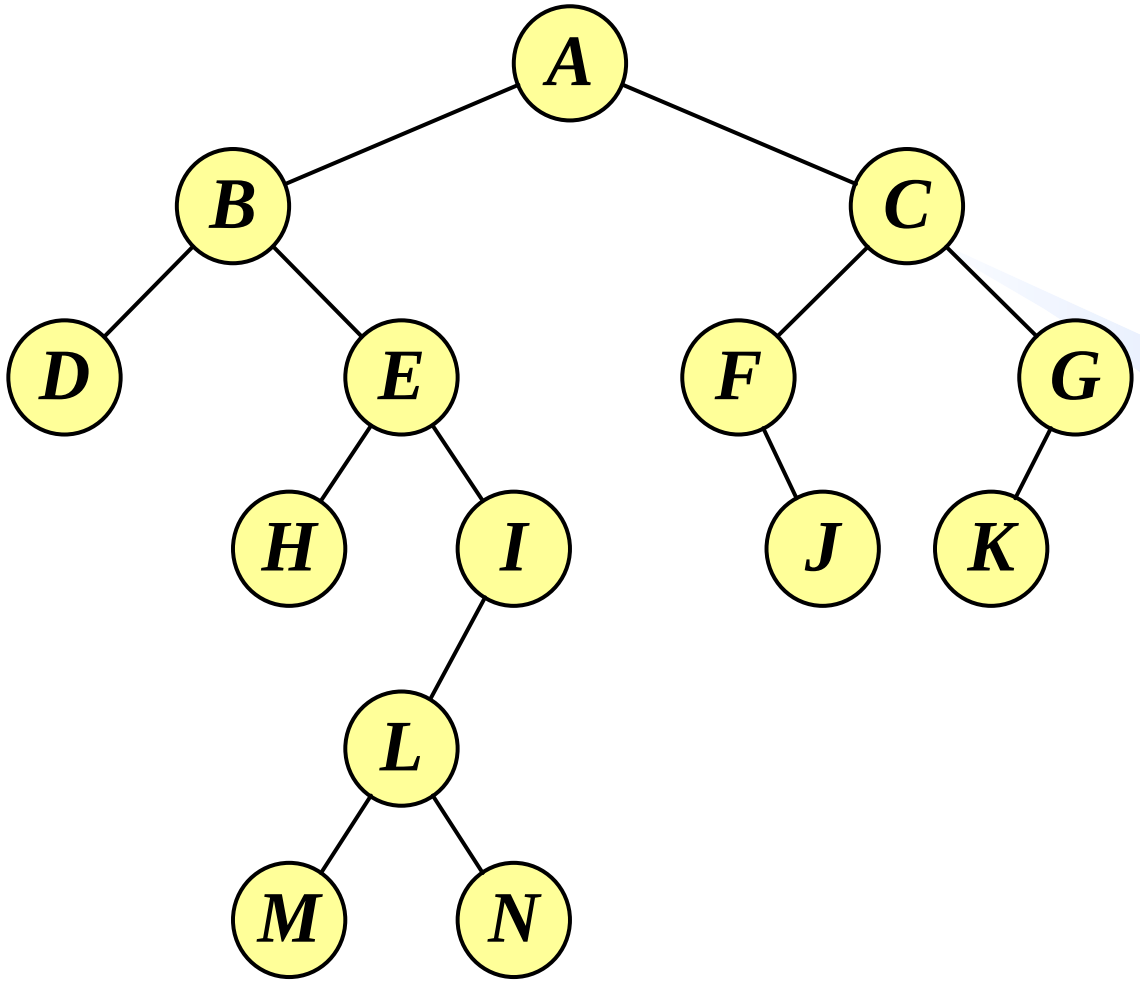
统计一棵二叉树中每层叶结点的数目【吉林大学上机考试题】。

➢需要识别每层的结束，每层结束后，统计该层信息。

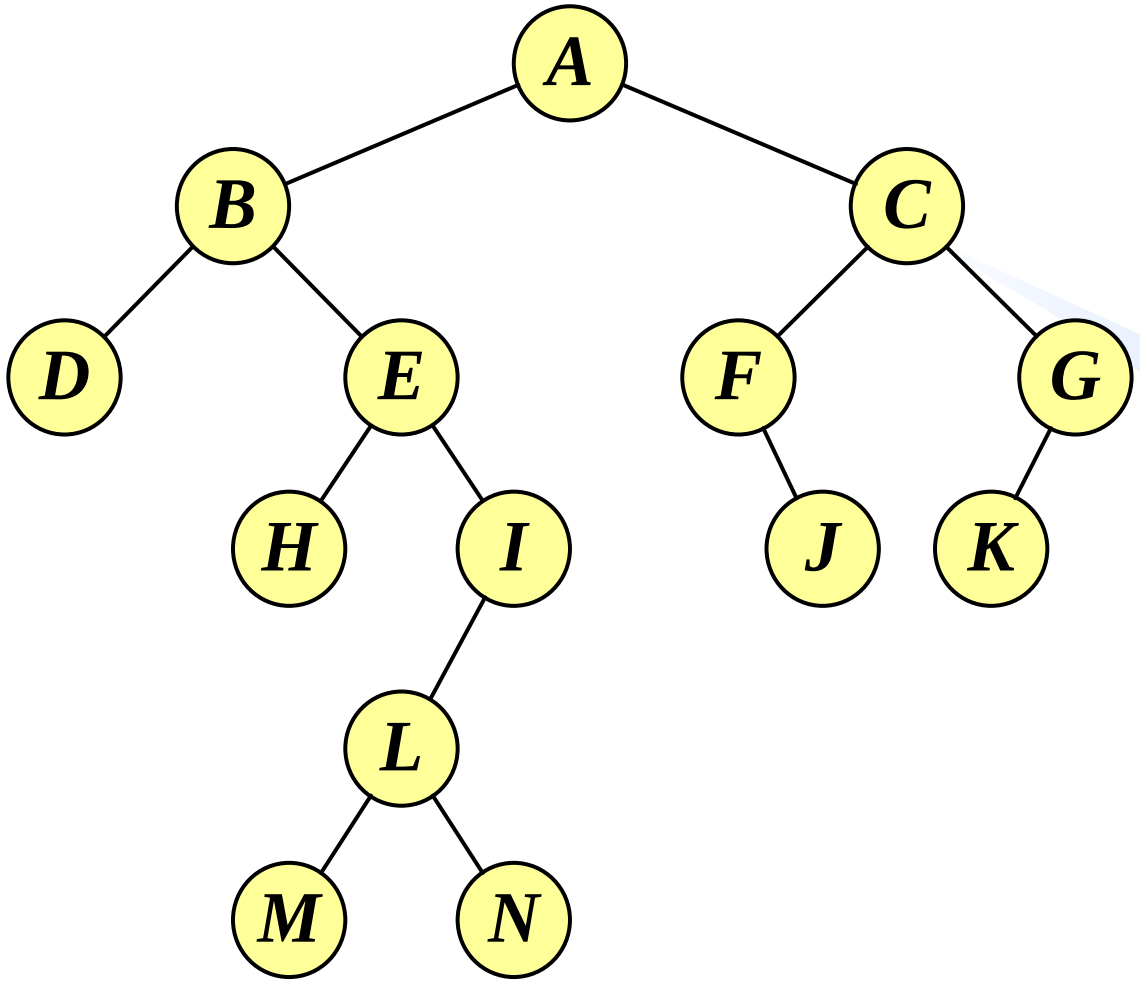
方案2:

✓每层结点全部出队后，队列中的元素即为下一层的全部结点

✓出队时，在while循环里面做个for循环，根据队列中元素个数，把同层结点连续出队。

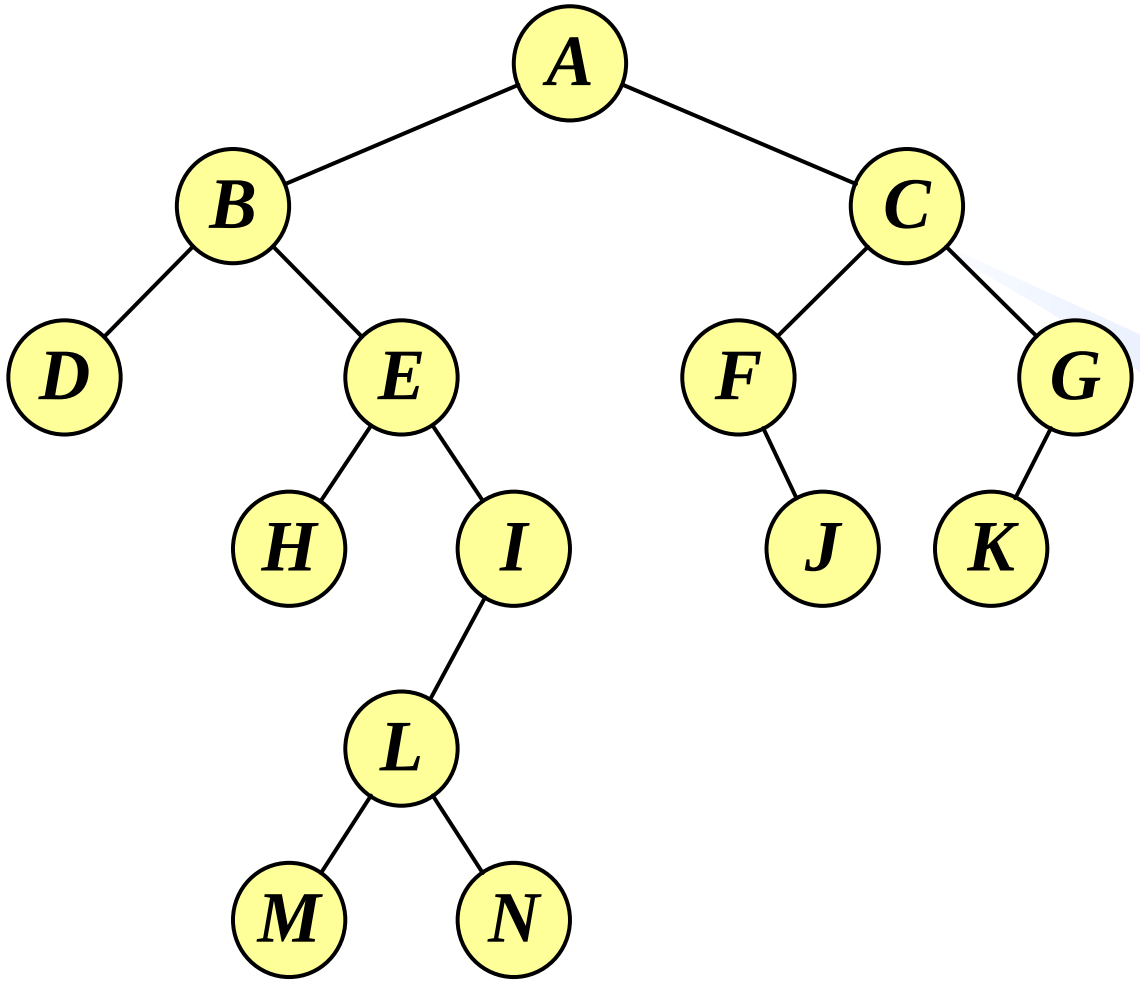


A



A

B C



A B C

D E F G

```
int LeafInEachLevel (TreeNode *root, int leafnum[]){
    if(root==NULL) return -1;
    Queue<TreeNode*> q ; int h=0;
    q.enQueue(root);
    while (!q.empty()){
        int size=q.size(); //size为本层结点数
        for(int i=0; i<size; i++){ //连续出队并访问本层结点
            TreeNode * p=q.deQueue();
            if(p->left==NULL && p->right==NULL)leafnum[h]++;
            if(p->left!=NULL) q.enQueue(p->left);
            if(p->right!=NULL) q.enQueue(p->right);
        }
        h++;     //本层结束，准备访问下一层
    }
    return h-1; //返回二叉树的高度
};
```

***leafnum[h]*表示第*h*层的叶结点数。调用时*leafnum*数组各元素值应初始化为0**

统计二叉树每层结点信息

统计一棵二叉树中每层叶结点的数目【吉林大学上机考试题】。

➢需要识别每层的结束，每层结束后，统计该层信息。

方案3：

✓采用先根遍历，用变量*k*标识递归深度，每进入一层递归（向下一层访问结点），*k*加1，递归深度*k*其实就是当前访问的结点所在的层数。

✓用数组*leafnum*[]记录各层叶结点数目，每访问到一个叶结点时，*leafnum*[*k*]++，表示第*k*层叶节点数目加1。

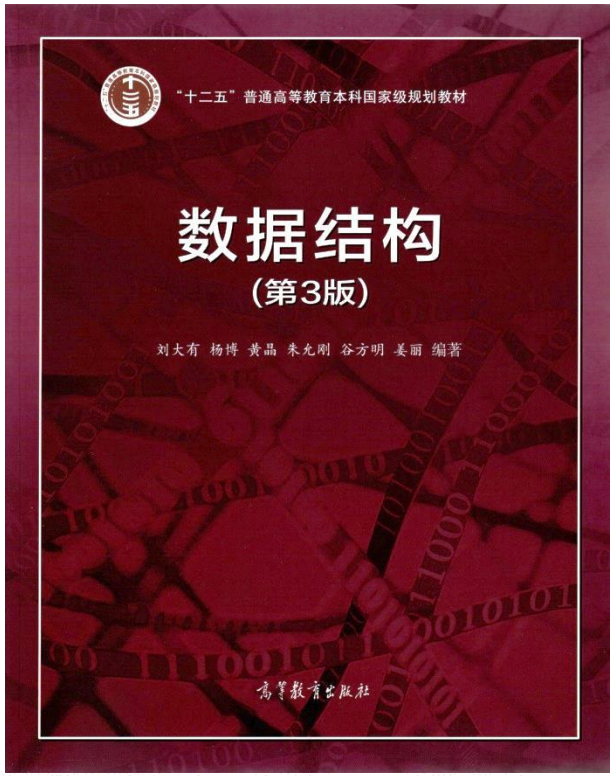
✓当遍历完成后，*leafnum*[*i*]数组就存储了第*i*层叶节点数目。

```
void LeafInEachLevel(TreeNode *t,int k,int leafnum[],int &h){
    //计算每层叶结点数存入leafnum[]数组,h为二叉树高度即最大层数
    //k为递归深度，每往深递归一层k加1，k亦为当前访问结点所在层数
    if(t==NULL) return; //空树
    if(k>h) h=k; //若当前层数超过最大树高，则更新树高
    if(t->left==NULL && t->right==NULL) leafnum[k]++;
    LeafInEachLevel(t->left, k+1, leafnum, h);
    LeafInEachLevel(t->right, k+1, leafnum, h);
}
```

初始调用

```
int h=-1;  //h为遍历过程中遇到的最大层数，初值≤0
LeafInEachLevel(root, 0, leafnum, h)
```

//leafnum[i]表示第i层的叶结点数，初始调用前数组各元素值初始化为0



二叉树的分层遍历和路径

- 二叉树的分层遍历
- 二叉树的路径

数据之法
结构之美
算法之道

zhuyungang@jlu.edu.cn

Last updated on 2024.11

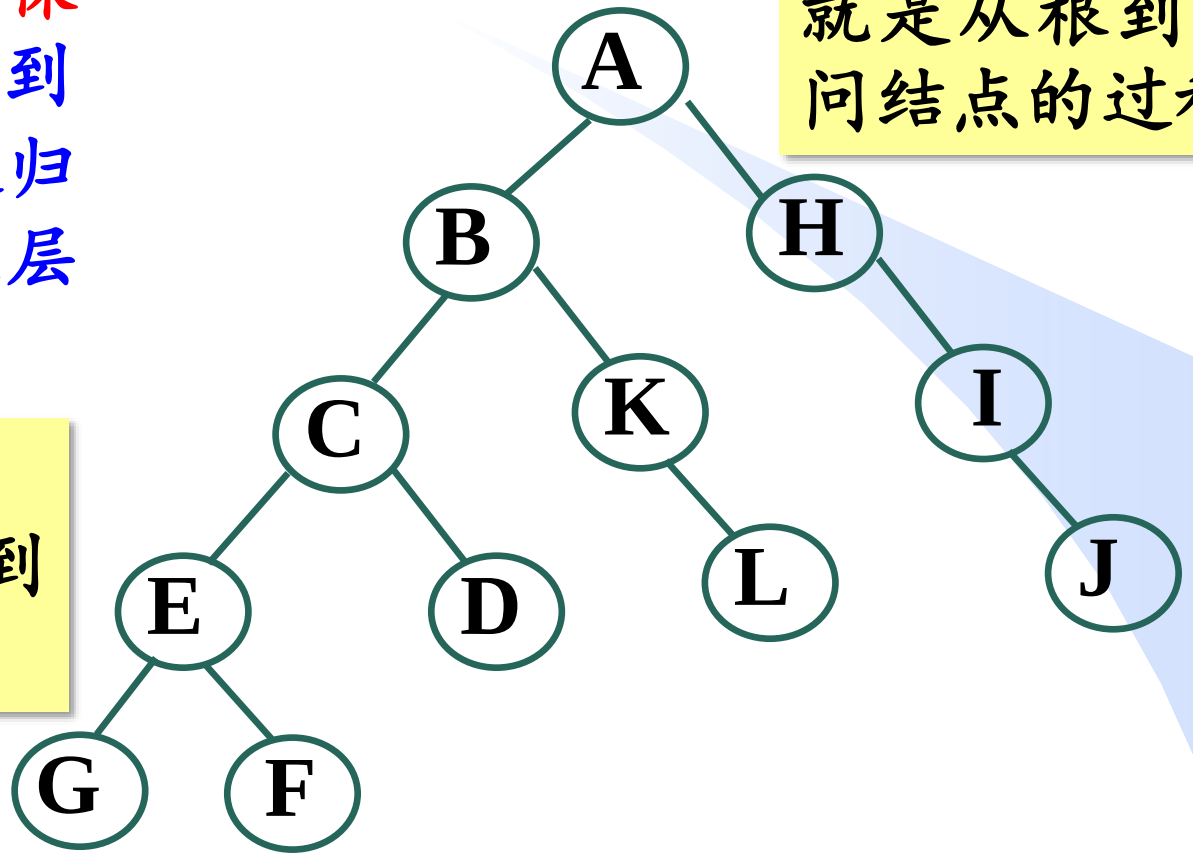
二叉树的路径

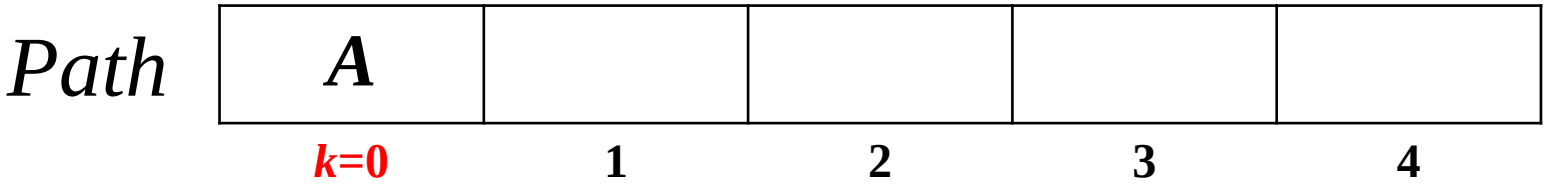
先根遍历，将沿途访问的结点保存在*Path*数组中，即得到从根到当前访问结点的一条路径。递归深度对应当前访问的结点所在层数，亦即当前路径的长度。

当前访问的结点存入*Path*[*k*]，则*Path*[0]…*Path*[*k*]即为从根到当前访问结点的路径。

*k*为先根遍历的递归深度

先根遍历的过程就是从根到叶访问结点的过程





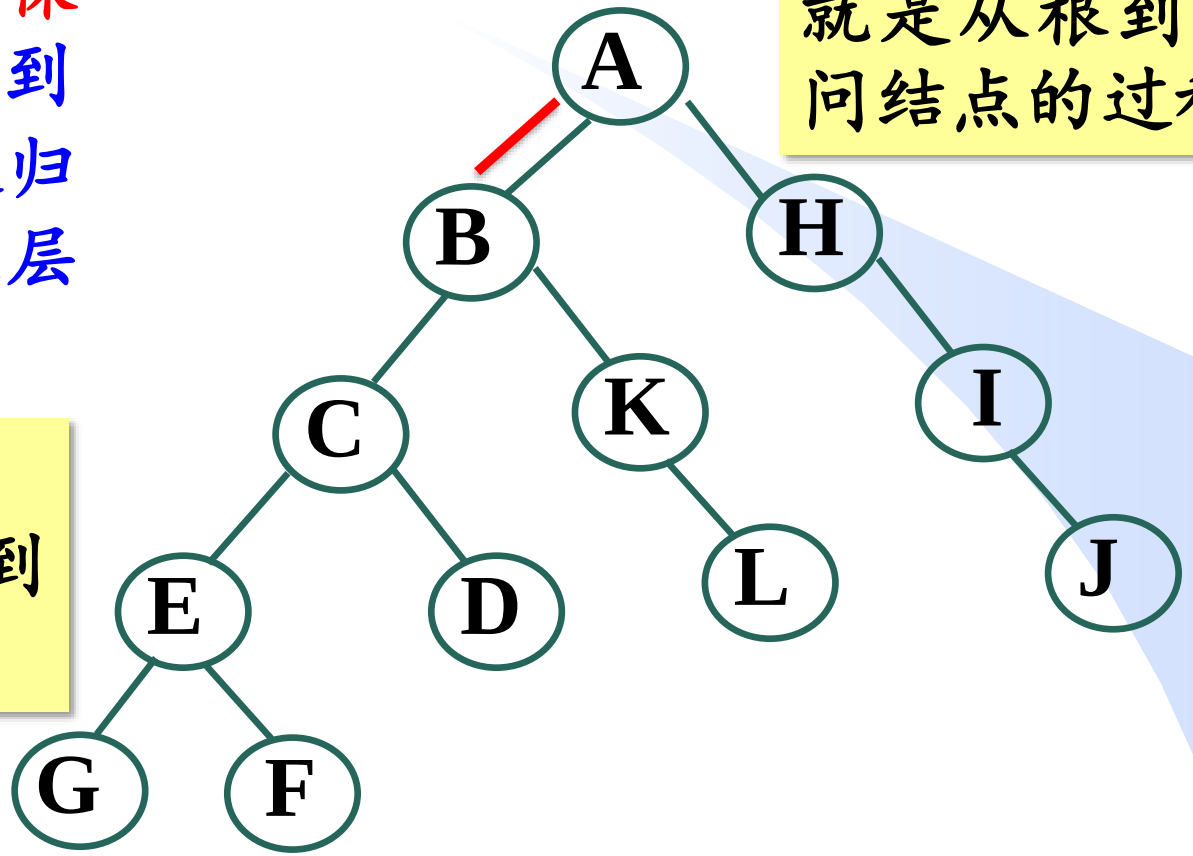
二叉树的路径

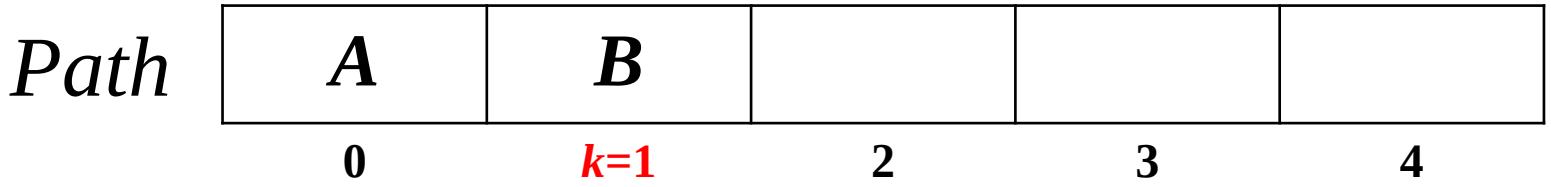
先根遍历，将沿途访问的结点保存在*Path*数组中，即得到从根到当前访问结点的一条路径。递归深度对应当前访问的结点所在层数，亦即当前路径的长度。

当前访问的结点存入*Path*[*k*]，则*Path*[0]…*Path*[*k*]即为从根到当前访问结点的路径。

*k*为先根遍历的递归深度

先根遍历的过程就是从根到叶访问结点的过程





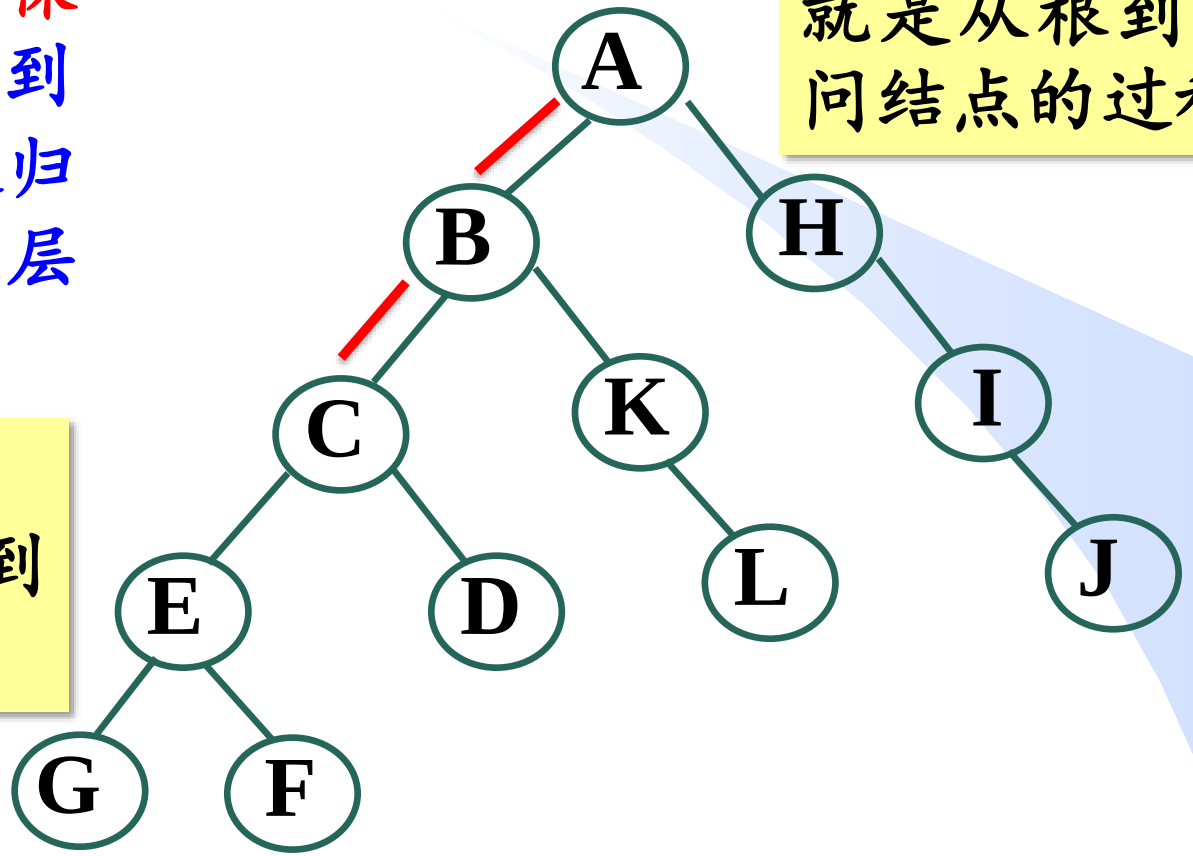
二叉树的路径

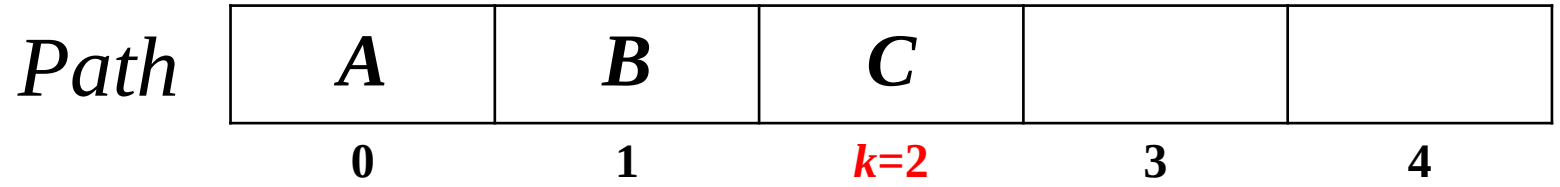
先根遍历，将沿途访问的结点保存在*Path*数组中，即得到从根到当前访问结点的一条路径。递归深度对应当前访问的结点所在层数，亦即当前路径的长度。

当前访问的结点存入*Path*[*k*]，则*Path*[0]…*Path*[*k*]即为从根到当前访问结点的路径。

*k*为先根遍历的递归深度

先根遍历的过程就是从根到叶访问结点的过程





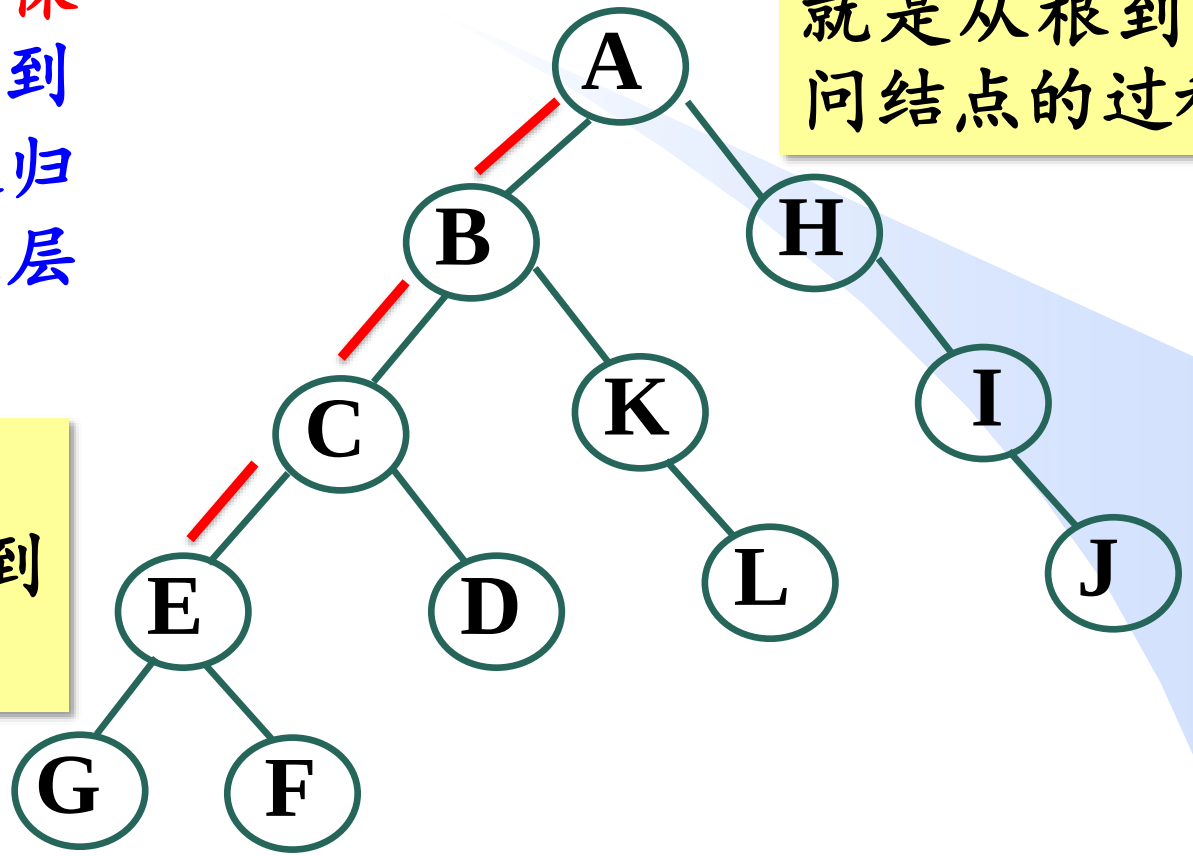
二叉树的路径

先根遍历，将沿途访问的结点保存在*Path*数组中，即得到从根到当前访问结点的一条路径。递归深度对应当前访问的结点所在层数，亦即当前路径的长度。

当前访问的结点存入*Path*[*k*]，则*Path*[0]…*Path*[*k*]即为从根到当前访问结点的路径。

*k*为先根遍历的递归深度

先根遍历的过程就是从根到叶访问结点的过程



Path	*A*	*B*	*C*	*E*	
	0	1	2	*k*=3	4

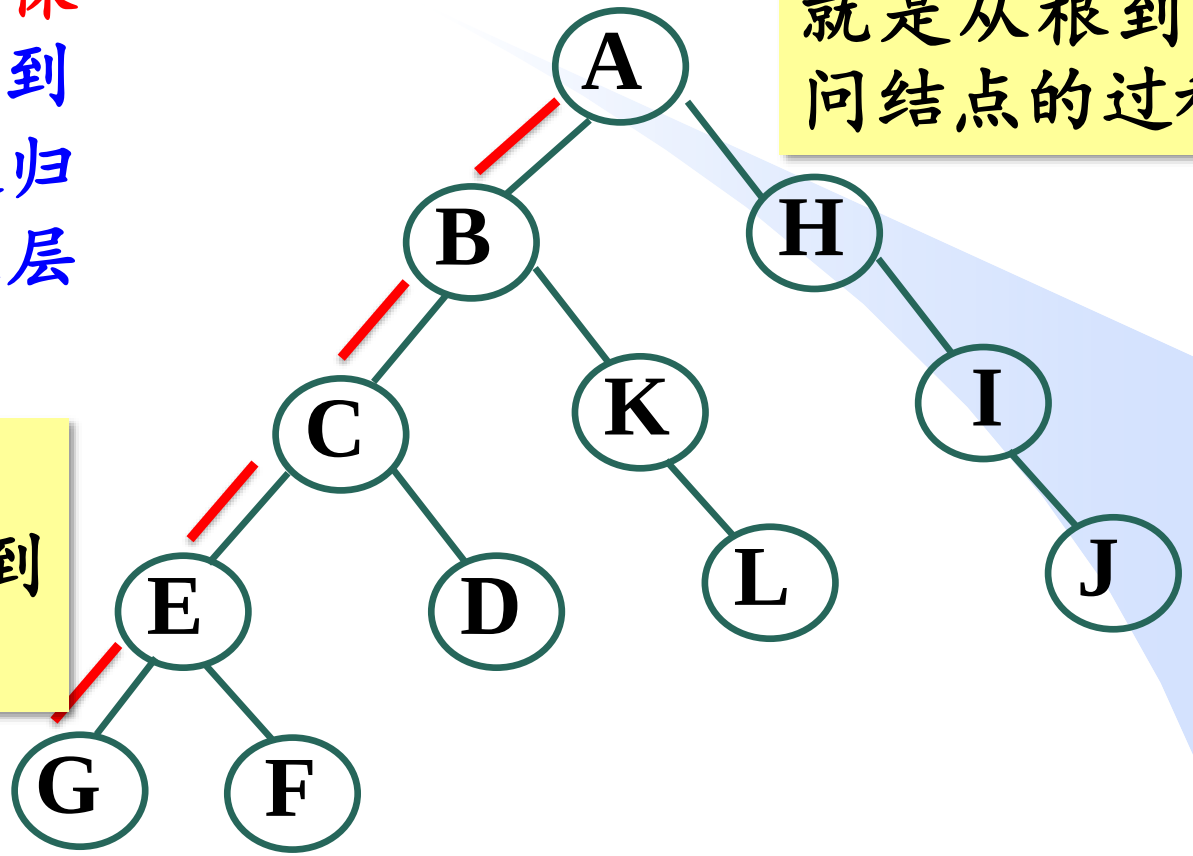
二叉树的路径

先根遍历，将沿途访问的结点保存在*Path*数组中，即得到从根到当前访问结点的一条路径。递归深度对应当前访问的结点所在层数，亦即当前路径的长度。

当前访问的结点存入*Path*[*k*]，则*Path*[0]…*Path*[*k*]即为从根到当前访问结点的路径。

*k*为先根遍历的递归深度

先根遍历的过程就是从根到叶访问结点的过程



Path	*A*	*B*	*C*	*E*	*G*
	0	1	2	3	*k*=4

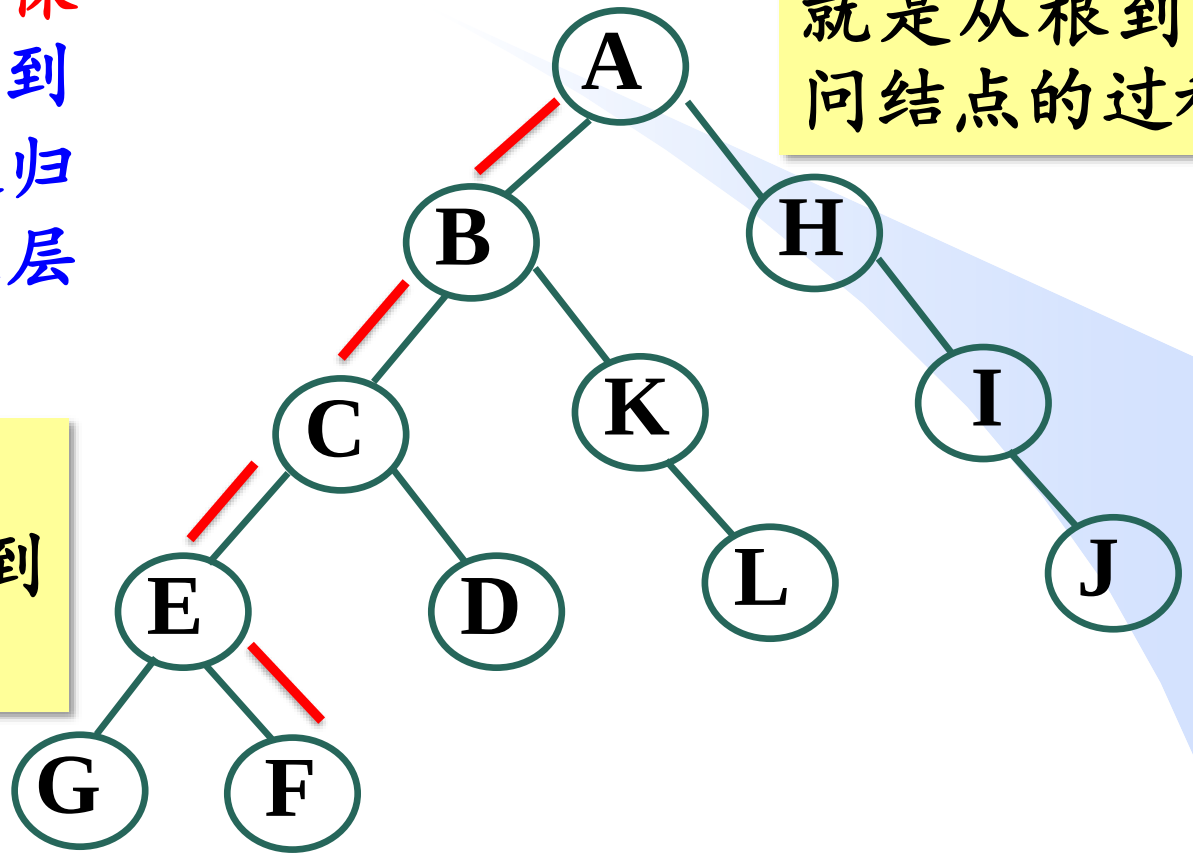
二叉树的路径

先根遍历，将沿途访问的结点保存在*Path*数组中，即得到从根到当前访问结点的一条路径。递归深度对应当前访问的结点所在层数，亦即当前路径的长度。

当前访问的结点存入*Path*[*k*]，则*Path*[0]…*Path*[*k*]即为从根到当前访问结点的路径。

*k*为先根遍历的递归深度

先根遍历的过程就是从根到叶访问结点的过程



Path	*A*	*B*	*C*	*E*	*F*
	0	1	2	3	*k*=4

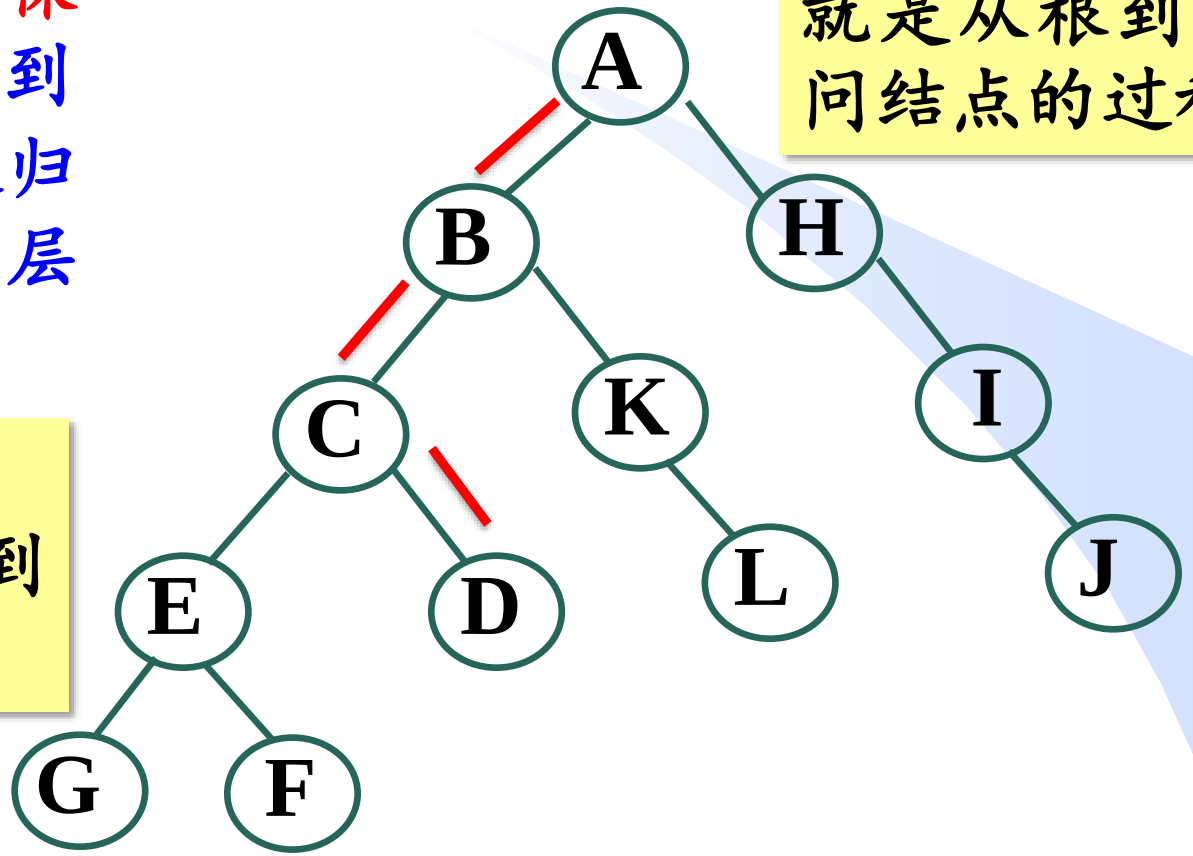
二叉树的路径

先根遍历，将沿途访问的结点保存在*Path*数组中，即得到从根到当前访问结点的一条路径。递归深度对应当前访问的结点所在层数，亦即当前路径的长度。

当前访问的结点存入*Path*[*k*]，则*Path*[0]…*Path*[*k*]即为从根到当前访问结点的路径。

*k*为先根遍历的递归深度

先根遍历的过程就是从根到叶访问结点的过程



Path	*A*	*B*	*C*	*D*	*F*
	0	1	2	*k*=3	4

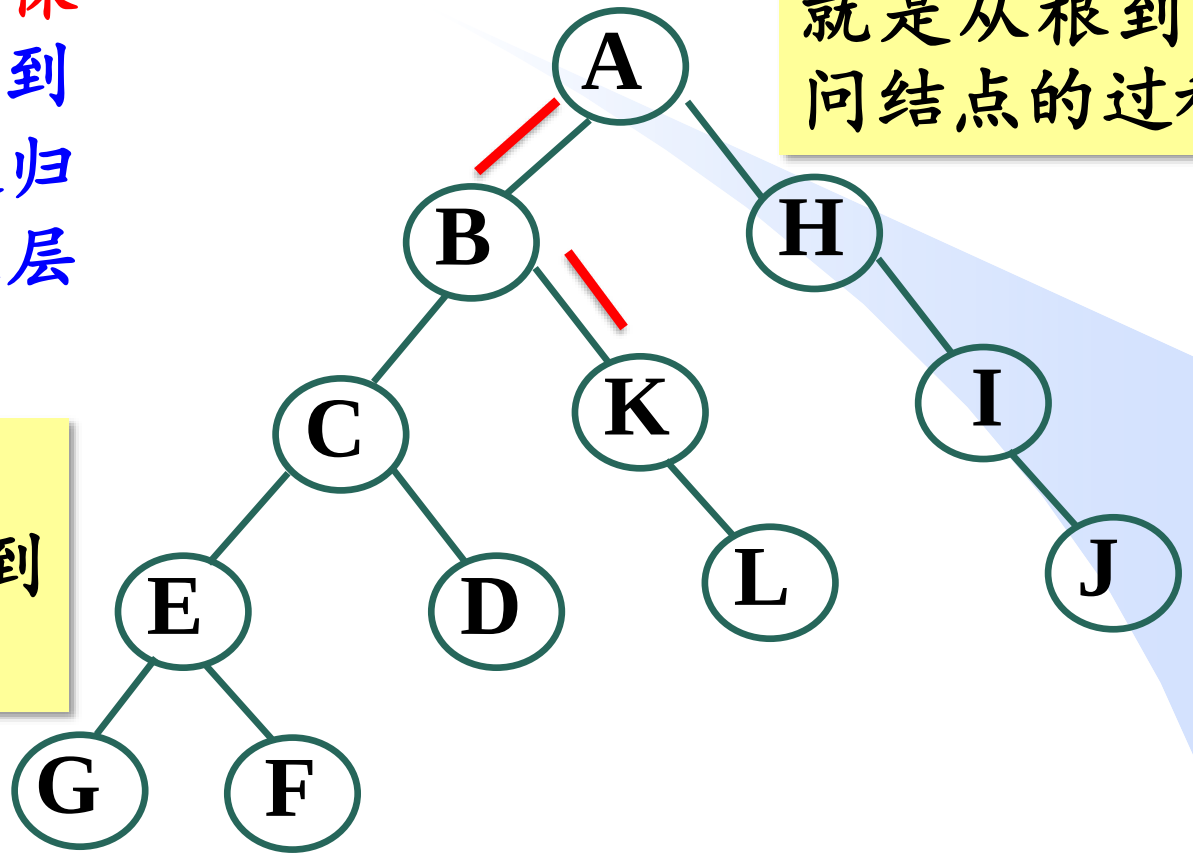
二叉树的路径

先根遍历，将沿途访问的结点保存在*Path*数组中，即得到从根到当前访问结点的一条路径。递归深度对应当前访问的结点所在层数，亦即当前路径的长度。

当前访问的结点存入*Path*[*k*]，则*Path*[0]…*Path*[*k*]即为从根到当前访问结点的路径。

*k*为先根遍历的递归深度

先根遍历的过程就是从根到叶访问结点的过程



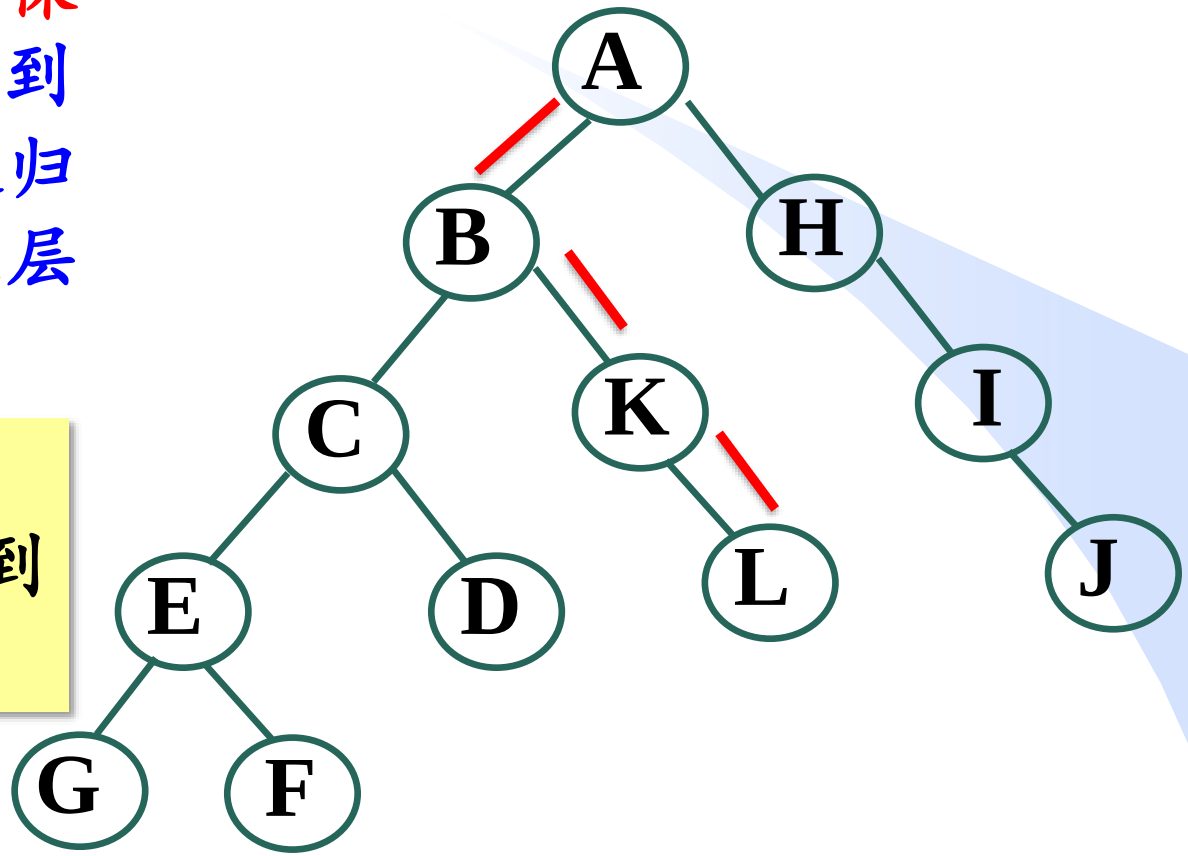
Path	*A*	*B*	*K*	*D*	*F*
	0	1	*k*=2	3	4

二叉树的路径

先根遍历，将沿途访问的结点保存在*Path*数组中，即得到从根到当前访问结点的一条路径。递归深度对应当前访问的结点所在层数，亦即当前路径的长度。

当前访问的结点存入*Path*[*k*]，则*Path*[0]…*Path*[*k*]即为从根到当前访问结点的路径。

*k*为先根遍历的递归深度



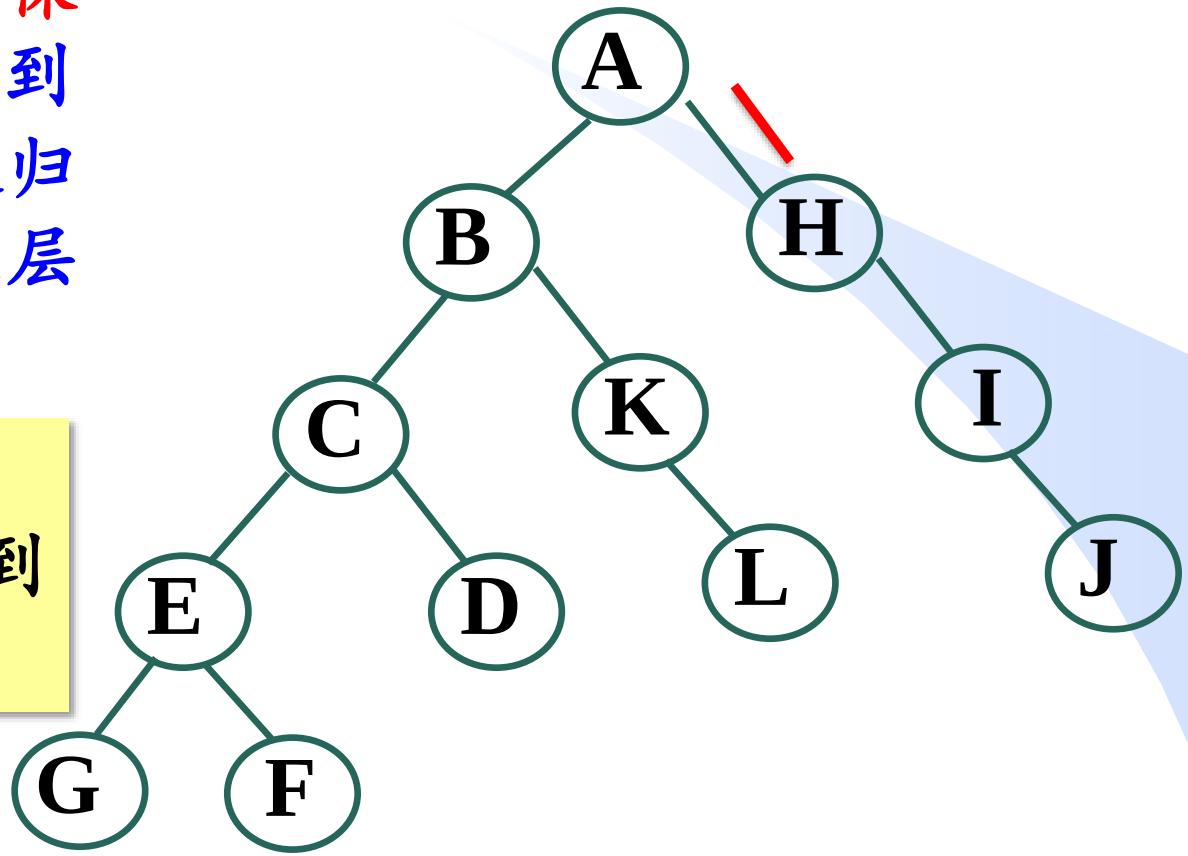
Path	*A*	*B*	*K*	*L*	*F*
	0	1	2	*k*=3	4

二叉树的路径

先根遍历，将沿途访问的结点保存在*Path*数组中，即得到从根到当前访问结点的一条路径。递归深度对应当前访问的结点所在层数，亦即当前路径的长度。

当前访问的结点存入*Path*[*k*]，则*Path*[0]…*Path*[*k*]即为从根到当前访问结点的路径。

*k*为先根遍历的递归深度



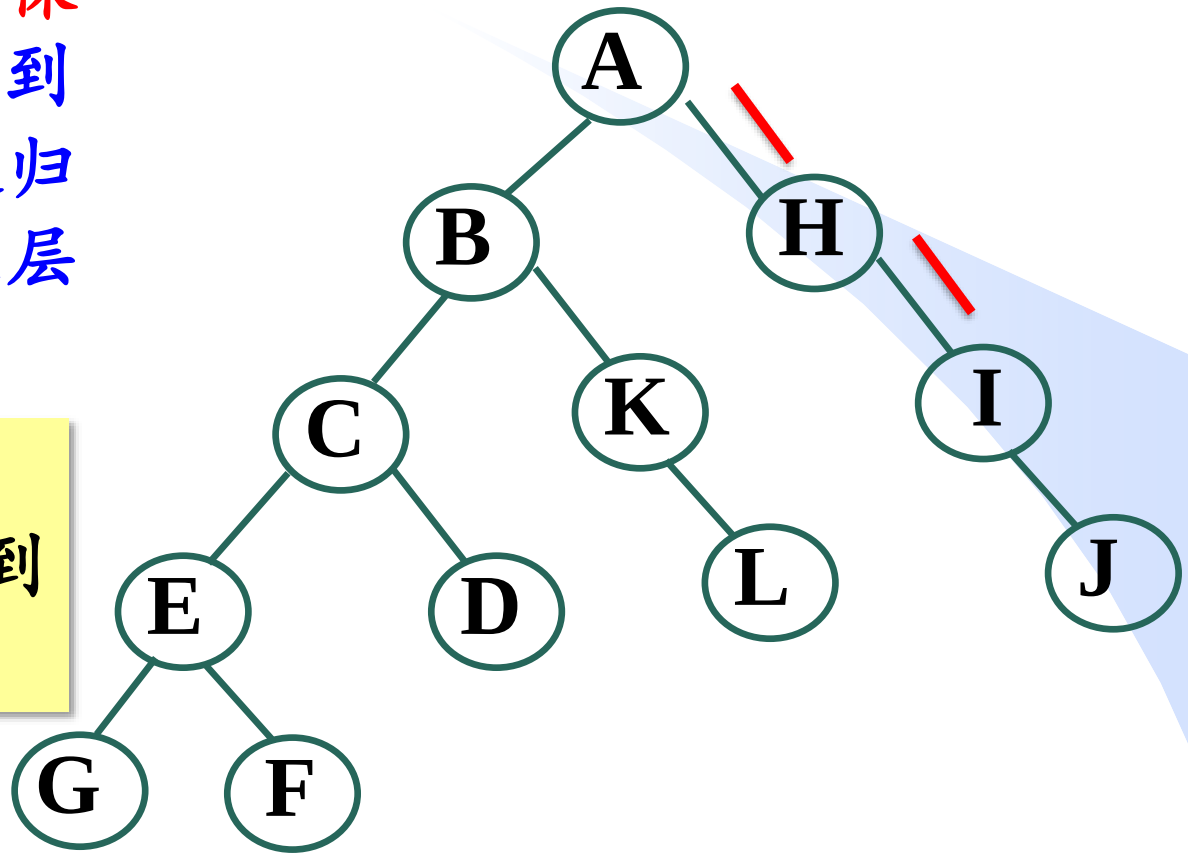
Path	*A*	*H*	*K*	*L*	*F*
	0	*k*=1	2	3	4

二叉树的路径

先根遍历，将沿途访问的结点保存在*Path*数组中，即得到从根到当前访问结点的一条路径。递归深度对应当前访问的结点所在层数，亦即当前路径的长度。

当前访问的结点存入*Path*[*k*]，则*Path*[0]…*Path*[*k*]即为从根到当前访问结点的路径。

*k*为先根遍历的递归深度



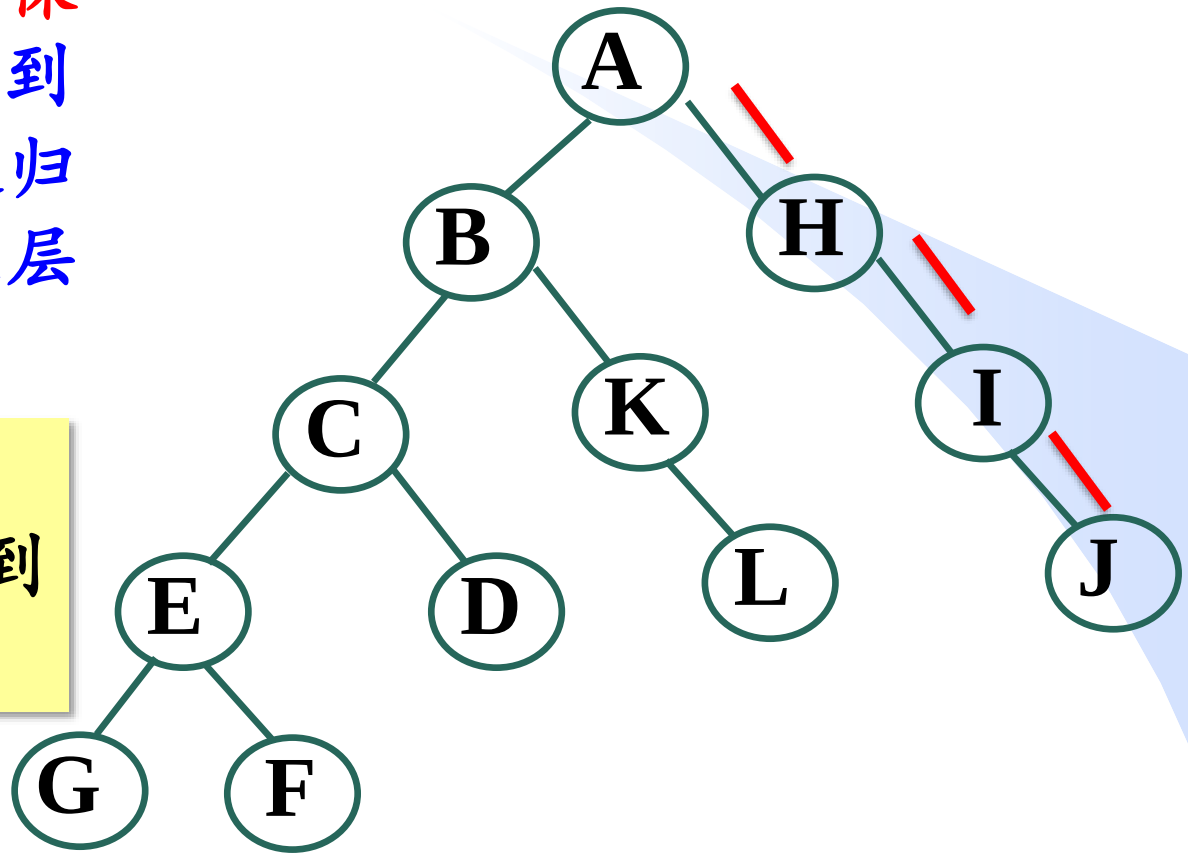
Path	*A*	*H*	*I*	*L*	*F*
	0	1	*k*=2	3	4

二叉树的路径

先根遍历，将沿途访问的结点保存在*Path*数组中，即得到从根到当前访问结点的一条路径。递归深度对应当前访问的结点所在层数，亦即当前路径的长度。

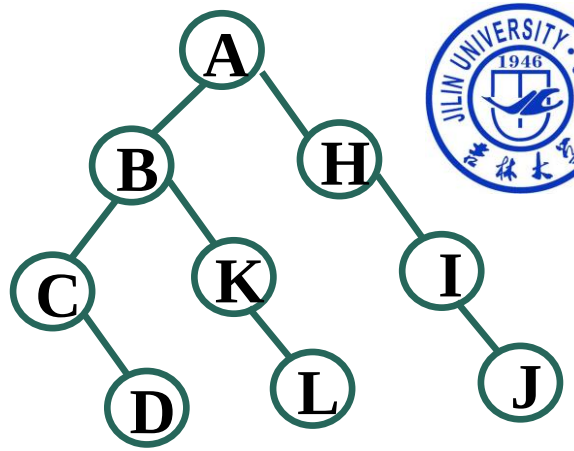
当前访问的结点存入*Path*[*k*]，则*Path*[0]…*Path*[*k*]即为从根到当前访问结点的路径。

*k*为先根遍历的递归深度



Path	*A*	*H*	*I*	*J*	*F*
	0	1	2	*k*=3	4

二叉树的路径




```
void findPath(TreeNode *t, int path[], int k){
    //输出从根到叶的所有路径，k为递归深度
    if(t==NULL) return;
    path[k] = t->data;
    if(t->left==NULL && t->right==NULL) { //找到一条路径
        for(int i=0; i<=k; i++)    //输出找到的路径
            printf("%d ", path[i]);
        printf("\n");
        return;
    }
    findPath(t->left, path, k+1);
    findPath(t->right, path, k+1);
}
```

先根遍历
回溯

初始调用
findPath(root,path,0)

二叉树的路径

```
bool findPath(TreeNode *t, int path[], int x, int k){
    //输出从根到数据值等于x的一条路径，k为递归深度
    if(t==NULL) return false;
    path[k] = t->data;
    if(t->data == x) { //找到一条路径
        for(int i=0; i<=k; i++)    //输出找到的路径
            printf("%d ", path[i]);
        printf("\n");
        return true;
    }
    if(findPath(t->left, path, x, k+1)) return true;
    if(findPath(t->right, path, x, k+1)) return true;
    return false;
}
```

初始调用

findPath(root,path,x,0)