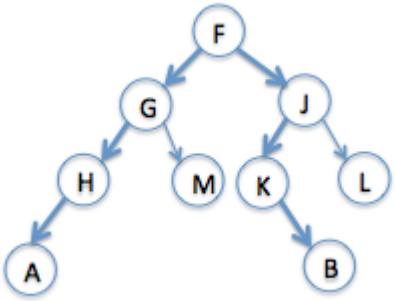
问题来源：《编程之美》**3.8** 求二叉树节点的最大距离

如果把二叉树看成一个图，父子节点之间的连线看成是双向的，我们姑且定义"距离"为两个节点之间的个数。

写一个程序求一棵二叉树中相距最远的两个节点之间的距离。

如下图所示，粗箭头的边表示最长距离：



树中相距最远的两个节点是 **A，B**

分析可知：对于二叉树，若要两个节点 **U**，**V** 相距最远，有两种情况：

1，从 **U** 节点到 **V** 节点之间的路径经过根节点

2，从 **U** 节点到 **V** 节点之间的路径不经过根节点，这种情况下，**U**，**V** 节点必定在根节点的左子树或者右子树上，这样就转化为求以根节点的孩子节点为根节点的二叉树中最远的两个节点间的距离

上面所说的经过根节点，是指路径中包含根节点，例如：加入上图中只有左子树 **FGHA**，那么最长距离的两个节点是 **F，A**，该路径中包含根节点 **F**，也称为经过根节点。

于是我们可以递归求解，按照二叉树的中序遍历方式遍历二叉树，在遍历的过程中寻找相距最远的两个节点。

程序描述如下：

[cpp] view plaincopyprint?

```cpp
typedef struct Node {
    struct Node *pleft;     //左孩子
    struct Node *pright;    //右孩子
    char chValue;           //该节点的值

    int leftMaxValue;       //左子树最长距离
    int rightMaxValue;      //右子树最长距离
}LNode, *BinTree;

void findMaxLen(BinTree root, int *maxLen) {
    //遍历到叶子结点，返回
    if(root == NULL)
        return;

```

```
    //如果左子树为空，那么该节点左边最长距离为0
    if(root->pleft == NULL)
        root->leftMaxValue = 0;

    //如果右子树为空，那么该节点右边最长距离为0
    if(root->pright == NULL)
        root->rightMaxValue = 0;

    //如果左子树不为空，递归寻找左子树最长距离
    if(root->pleft != NULL)
        findMaxLen(root->pleft, maxLen);

    //如果右子树不为空，递归寻找右子树最长距离
    if(root->pright != NULL)
        findMaxLen(root->pright, maxLen);

    //计算左子树中距离根节点的最长距离
    if(root->pleft != NULL) {
        if(root->pleft->leftMaxValue > root->pleft->rightMaxValue)
            root->leftMaxValue = root->pleft->leftMaxValue + 1;
        else
            root->leftMaxValue = root->pleft->rightMaxValue + 1;
    }

    //计算右子树中距离根节点的最长距离
    if(root->pright != NULL) {
        if(root->pright->leftMaxValue > root->pright->rightMaxValue)
            root->rightMaxValue = root->pright->leftMaxValue + 1;
        else
            root->rightMaxValue = root->pright->rightMaxValue + 1;
    }

    //更新最长距离
    if(root->leftMaxValue + root->rightMaxValue > *maxLen)
        *maxLen = root->leftMaxValue + root->rightMaxValue;
}
```

int *maxLen 中始终存储的是当前两个节点间的最远距离，在遍历的过程中更新。

下面的程序描述是为了测试上面的代码是否正确，包括建立二叉树，销毁二叉树，打印二叉树：

[cpp] view plaincopyprint?

```
#include <stdlib.h>
```

```
#include <stdio.h>

typedef struct Node {
    struct Node *pleft;     //左孩子
    struct Node *pright;    //右孩子
    char chValue;           //该节点的值

    int leftMaxValue;       //左子树最长距离
    int rightMaxValue;      //右子树最长距离
}LNode, *BinTree;

void findMaxLen(BinTree root, int *maxLen) {
    //遍历到叶子结点，返回
    if(root == NULL)
        return;

    //如果左子树为空，那么该节点左边最长距离为0
    if(root->pleft == NULL)
        root->leftMaxValue = 0;

    //如果右子树为空，那么该节点右边最长距离为0
    if(root->pright == NULL)
        root->rightMaxValue = 0;

    //如果左子树不为空，递归寻找左子树最长距离
    if(root->pleft != NULL)
        findMaxLen(root->pleft, maxLen);

    //如果右子树不为空，递归寻找右子树最长距离
    if(root->pright != NULL)
        findMaxLen(root->pright, maxLen);

    //计算左子树中距离根节点的最长距离
    if(root->pleft != NULL) {
        if(root->pleft->leftMaxValue > root->pleft->rightMaxValue)
            root->leftMaxValue = root->pleft->leftMaxValue + 1;
        else
            root->leftMaxValue = root->pleft->rightMaxValue + 1;
    }

    //计算右子树中距离根节点的最长距离
    if(root->pright != NULL) {
        if(root->pright->leftMaxValue > root->pright->rightMaxValue)
            root->rightMaxValue = root->pright->leftMaxValue + 1;
```

```
46.         else
47.             root->rightMaxValue = root->pright->rightMaxValue + 1;
48.     }
49.
50.     //更新最长距离
51.     if(root->leftMaxValue + root->rightMaxValue > *maxLen)
52.         *maxLen = root->leftMaxValue + root->rightMaxValue;
53. }
54.
55. //创建二叉树
56. void buildBinTree(BinTree *root)
57. {
58.     char ch;
59.     scanf("%c", &ch);     //输入一个元素
60.     fpurge(stdin);
61.     if(ch == ' ')         //若输入的是空格符，表明二叉树为空，置*root 为 NULL
62.         *root = NULL;
63.     else {                //若输入的不是空格符，则将该值赋值给根节点的 chValue，递
    归建立左子树和右子树
64.         *root = (BinTree)malloc(sizeof(LNode));
65.         (*root)->chValue = ch;
66.         (*root)->leftMaxValue = 0;
67.         (*root)->rightMaxValue = 0;
68.
69.         buildBinTree(&(*root)->pleft);
70.         buildBinTree(&(*root)->pright);
71.     }
72. }
73.
74. //销毁二叉树，释放内存
75. void destroyBinTree(BinTree *root)
76. {
77.     if(*root != NULL) {
78.         destroyBinTree(&(*root)->pleft);
79.         destroyBinTree(&(*root)->pright);
80.
81.         free(*root);
82.         *root = NULL;
83.     }
84. }
85.
86. //前序遍历二叉树
87. void preOrderTraverse(BinTree root)
88. {
```

```
89.     if(root != NULL) {
90.         preOrderTraverse(root->pleft);
91.         printf("%c", root->chValue);
92.         preOrderTraverse(root->pright);
93.     }
94. }
95. 
96. int main() {
97.     BinTree root;
98.     buildBinTree(&root);
99.     preOrderTraverse(root);
100.     printf("\n");
101.     int maxLen = 0;
102.     findMaxLen(root, &maxLen);
103.     printf("maxLen = %d\n", maxLen);
104.     destroyBinTree(&root);
105. }
```