数据结构与算法

第三章 树

P121.12

设一棵二叉树T，按图所给例子形式放在内存中，尝试给出一个算法，用来求T的高度，并对T的每一个结点赋予一个层号。

Left	right	Level
2	6	
3	4	
0	0	
0	5	
0	0	
7	8	
0	0	
0	9	
0	0	

基本思路：

观察以上表格，我们会发现在这个二叉树中，存在许多叶结点。根据定义，树的高定义为根结点的高，即该棵树所有结点中最大的层号。而结点的层也就是从根到该结点的路长+1。

一个十分直观的想法是赋予层号，比较层号，取最大值为二叉树的高。

算法设计：

```
node[1].level = 1;
if(node->child != 0)
  node[node->child].level++;
//左右子结点同理。
height = max(node[i].level);
```

P121.13

试证明：任一棵高为h>1的二叉树，其内部结点(除根结点和叶结点之外的结点)的个数小于[2^(h-1)]-1,而叶结点的个数小于等于2^(h-1)

> 性质3.1 在二叉树中第i层的结点数最多为2^(i-1) (i>=1)

> 性质3.2 高度为k的二叉树其结点总数最多为2^k -1。

证明如下：

首先由性质3.1知，在二叉树第i层的结点数最多为2^(i-1) (i>=1)，显然，当且仅当二叉树为满二叉树(Full Binary Tree)时，其叶结点树达到最大值，为2^(i-1) (i>=1), 所以任一棵h>1的二叉树，其叶结点个数小于等于2^(h-1)。

同样，由性质3.1与性质3.2得知，

内部结点个数 < 2^h - 2^(h-1) -1 -1 = 2^(h-1) - 2 < 2^(h-1) - 1

P121.14

> *求二叉树中节点的最大距离... 如果我们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"为两节点之间边的个数。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。*

思路：用BFS(Breadth-First-Search)

1.从任意一个节点u开始做第一遍BFS，得到距离u最远的那个节点v

2. 从节点v开始做第二遍BFS，得到距离v最远的节点 e, 那 v 到 e 就是直径。

这里我们需要证明v必然在树的直径路径上

证明：

如果u在直径路径上

用反证法，假设v不在直径上，则根据直径定义，必然存在一点v2在直径上，使得 distance(u->v2) > distance(u->v)，这就与bfs算法v是从u出发到达的所有节点中离最远的相矛盾。

u ----- w ------- v

/

x ----------y ----------z

上图中，u-->v 是第一遍BFS算出来的路径，x-->z 是直径路径，反证法假设v 不在直径路径上，如图所示。根据树和联通图的定义，u->v中必然存在一点w, 和x->z中的某点y 相连通，或者说必然存在一个路径 w--y ,链接uv和xz。

根据直径的定义： Dist(xz) = Dist(xy) + Dist(yz) ， **Dist(yz) >= Dist(yw) + Dist(wv)** , 不然直径就变成x->y->w->v了。

根据BFS的定义： Dist(uv) = Dist(uw) + Dist(wv) , **Dist(wv) >= Dist(wy) + Dist(yz),** 不然BFS最短路径就变成 u->w->y->z了。

要让上面的不等式成立，唯一的可能是 Dist(yw)=0 并且 Dist(wv)=Dist(yz), **也就是说 uv 必须和xz相交，并且相交后的部重合，也就意味着v 一定在最终的直径路径上。**

所以对于本题，我们只需要实现

> 1.对每个一个节点u做一次BFS

即可。

```
图的广度优先遍历BFS算法是一个分层搜索的过程，和树的层序遍历算法类同，它也需要一个队列以保持遍历过的顶点顺序，以便按出队的顺序再去访问这些顶点的邻接顶点。
1.连通图的广度优先遍历算法思想。
（1）顶点v入队列。
（2）当队列非空时则继续执行，否则算法结束。
（3）出队列取得队头顶点v；访问顶点v并标记顶点v已被访问。
（4）查找顶点v的第一个邻接顶点col。
（5）若v的邻接顶点col未被访问过的，则col入队列。
（6）继续查找顶点v的另一个新的邻接顶点col，转到步骤（5）。直到顶点v的所有未被访问过的邻接点处理完。转到步骤（2）。
```

P123.34

试设计一个算法，将一棵二叉树按如下形式打印出来:

(key: LT,RT)

其中,key为根结点的关键字,LT是按同一形式表示的左子树，RT是右子树。此外，对空二叉树什么也不输出。由一个结点x组成的二叉树应打印(x),而不是(x;,)。

算法如下：

```
void PrintTree(BinTree T)

{

    //打印二叉树(以前序遍历实现)

        if(T)//非空树

{

    printf("%c",T->data);

    if(T->lchild||T->rchild)

        printf("(");
    PrintTree(T->lchild);
     if(T->lchild && T->rchild)
       printf(",");
    PrintTree(T->rchild);
         if(T->lchild||T->rchild)
           printf(")");
}
}
void PrintTree1(BinTree T)
{
  if(T){
             printf("%c",T->data);
             PrintTree1(T->lchild);
             if(T->lchild&&T->rchild)
                 printf(",");
             PrintTree1(T->rchild);
             printf(")");
  }
}
```

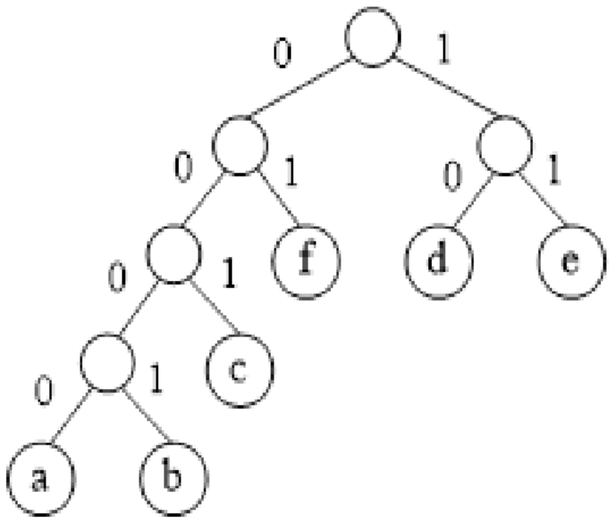
P124.36

假设字符A、B、C、D、E、F的使用频率分别是 0.07,0.09,0.12,0.22,0.23,0.27,

(1) 画出这棵哈夫曼树

(2)最优的Huffman编码，求编码平均长度

(1)



(2)

由此可得哈夫曼编码分别为：

a：0000 b：0001 c：001

d：01 e：10 f：11

$$ASL = 0.07*4+0.09*4+0.12*3+0.22*2+0.23*2+0.27*2=2.44$$