第六章作业答案

```
//6.47 层序遍历二叉树
//这一题还有一些同学用栈的，栈是后进先出，队列是先进先出
//另外Pop和Push操作是栈的，队列是EnQueue和DeQueue，有同学定义了队列，却使用pop和push操作
void LayerOrder(BiTree T){
    InitQueue(Q);
    EnQueue(Q, T);
    while (!QueueEmpty(Q)){
        DeQueue(Q, p);
        visit(p->data);
        if (p->lchild)
            EnQueue(Q, p->lchild);
        if (p->rchild)
            EnQueue(Q, p->rchild);
    }
}//LayerOrder

//6.48 寻找最近的共同祖先，一般题目中除非特别注明用三叉链表存储，否则都按二叉链表处理
//答案一：有些同学按照三叉链表的形式写的，但是也写不对，想当然的认为p和q在同一层了
BiTree FindAncestor(BiTree T, BiTree p, BiTree q) {
    /*
    典型错误：把p和q当成在同一层次上
    while(p != q){
        p = p->parent;
        q = q->parent;
    }
    */
    //下面是直接用循环写的，也可以用栈把p和q到根的路径存下来，然后出栈对比，找到第一个不相同的，其双亲就是答案
    BiTree tmp1, tmp2;
    tmp1 = p;
    while (tmp1 != tmp2) {
        tmp2 = q;
        while (tmp2 != T) {
            tmp2 = tmp2->parent;
            if (tmp1 == tmp2)
                break;
        }
        if (tmp1 != tmp2) {
            tmp1 = tmp1->parent;
        }
```

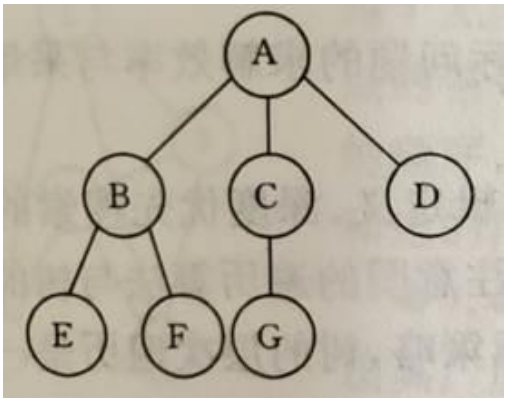
```
    }
    return tmp1;
}

//答案二，按照二叉链表来处理，可以先用递归找到从根到p和q的路径，然后对比路径得出最近的共同祖先
int found = FALSE;
BiTree Find_Near_Ancient(BiTree T, BiTree p, BiTree q) {//求二叉树T中结点p和q的最近共同祖先
    BiTree pathp[100], pathq[100] //设立两个辅助数组暂存从根到p,q的路径
    Findpath(T, p, pathp, 0);
    found = FALSE;
    Findpath(T, q, pathq, 0); //求从根到p,q的路径放在pathp和pathq中
    for (i = 0; pathp[i] == pathq[i] && pathp[i]; i++); //查找两条路径上最后一个相同结点
    return pathp[--i];
}//Find_Near_Ancient

void Findpath(BiTree T, BiTree p, BiTree path[], int i)//求从T到p路径的递归算法{
    if (T == p){
        found = TRUE;
        return; //找到
    }
    path[i] = T; //当前结点存入路径
    if (T->lchild)
        Findpath(T->lchild, p, path, i + 1); //在左子树中继续寻找
    if (T->rchild&& !found)
        Findpath(T->rchild, p, path, i + 1); //在右子树中继续寻找
    if (!found) path[i] = NULL; //回溯
}//Findpath

//6.54 根据顺序表构建二叉树，大家要注意题目给的是顺序表，不少同学把sa当成包含lchild和rchild域的结构体数组了
//也有一些同学直接把sa当成数组，sa是顺序表，sa.elem才是包含结点数据的数组
//顺序表中数组是按照层次遍历的顺序存储结点数据的，故而用队列来构建二叉树,也可直接根据双亲与孩子下标之间的关系来建立
void CreateBiTree(BiTree &T, SqList sa) {
    InitQueue(Q);
    T = (BiTree)malloc(sizeof(BiTNode));
    EnQueue(Q, T);
    i = 0;
    while (i < sa.last&& !QueueEmpty(Q)) {
        DeQueue(Q, p);
        p->data = sa.elem[i++];
```

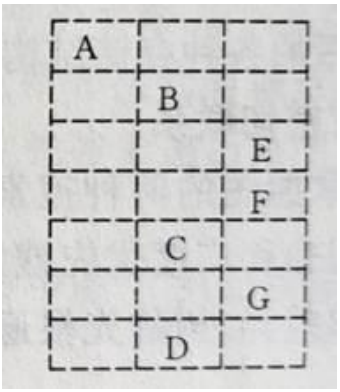
```
            if (2 * i - 1< sa.last) {
                p->lchild = (BiTree)malloc(sizeof(BiTNode));
                EnQueue(Q, p->lchild);
            }
            if (2 * i < sa.last) {
                p->rchild = (BiTree)malloc(sizeof(BiTNode));
                EnQueue(Q, p->rchild);
            }
        }
}

//6.68 创建孩子兄弟链表，书上有该结构体的定义，如果同学们要用和书上不一样的结构体，需给出定义

//先写一个创建结点的函数
CSNode* CreateCSNode(ElemType data) {
    CSNode* p;
    p = (CSNode*)malloc(sizeof(CSNode));
    if (p == NULL)
        return NULL;
    p->data = data;
    p->firstchild = NULL;
    p->nextsibling = NULL;
    return p;
}

Status CreateCSTreeByDegree(CSTree& T, ElemType node[], int degree[], int n){
    //node包含结点数据，degree包含每个结点的度，n为结点数目，这些都在接收结点信息输入的时候可以获得
    CSTree *t;
    t = (CSTree*)malloc(n * sizeof(CSTree));
    t[0] = CreateCSNode(node[0]);
    T = t[0];
    if(!T)
        return ERROR;
    int i = 0;//结点序号
    int j = 1;//孩子序号
    for(i = 0; i < n; i++){
        int d = degree[i];
        if(d == 0)
            continue;//叶子结点
        t[j] = CreateCSNode(node[j]);
        t[i]->firstchild = t[j];
        j++;
```

```
            //接下来添加兄弟nextsibling
            for(int k = 2; k <= d; k++){
                t[j] = CreateCSNode(node[j]);
                t[j-1]->nextsibling = t[j];
                j++;
            }
        }
        return OK;
}//CreateCSTreeByDegree

//6.69 根据示例，可知字母前面的空格与其所在层次相关，且打印顺序为RDL，用递归可实现
void PrintBiTree(BiTree T, int i) {//i表示结点所在层次,初次调用时i=0
    if (T->rchild)
        PrintBiTree(T->rchild, i + 1);
    for (j = 1; j <= i; j++)
        printf(" "); //打印i个空格以表示出层次
    printf("%c\n", T->data);
    if (T->lchild)
        PrintBiTree(T->lchild, i + 1);
}//PrintBiTree

//这道题也可以用栈，但有些同学写的时候，没有把握好层次信息的表示，以致错乱
void PrintBiTree(BiTree T) {
    InitStack(S);
    Push(S, T);
    i = 0;
    while (!StackEmpty(S)) {
        while (GetTop(S, p)&& p) {
            Push(S, p->rchild);//向右走到尽头
            i++;
        }
        Pop(S, p);//空指针出栈
        i--;
        if (!StackEmpty(S)) {
            Pop(S, p);
            for (j = 0; j < i; j++)
                printf(" ");
            printf("%c\n", p->data);
            Push(S, p->lchild);//向左走一步
        }
    }
}
```

6.71

假设树上每个结点所含的数据元素为一个字母，并且以**孩子-兄弟链表**为树的存储结构，试写一个按**凹入表方式打印**一棵树的算法。例如：左下所示树印为右下形状。





以下两种算法是大家写的最多的两种算法，这两种算法都可以，代码如下：

算法一：

```
void problem_6_71(CSTree T, int level)
{
      int i = 0;
      if (!T)              //如果 T 为空的话，应该返回
            return;
      for (i = 0; i < level; i++)
            printf("_"); //打印 level 个空格
      printf("%c\n", T->data);
//注意打印的是字符，并且打印字符后要换行
    //遍历孩子结点，打印的空格数应加 1
      problem_6_71(T->firstChild, level + 1);
    //递归遍历兄弟节点，兄弟节点的空格数和当前空格数相同
      problem_6_71(T->nextSibling, level);
}
```

算法二：

```
void problem_6_71(CSTree T, int level)
{
      CSTree p;
      for (i = 0; i < level; i++)
            printf("_"); //打印 level 个空格
      printf("%c\n", T->data);
//注意打印的是字符，并且打印字符后要换行
      for (p = T->firstChild; p; p = p->nextSibling)
            problem_6_71(p, level + 1);
}
```