（2）思考题

1.分析：

如果已经理解了第 6 题此题就容易解决，此题同样可以用重要思想提示中讲到的思想去解决。这里我们用一个栈来保存路径上的结点，当 p 自上至下走的时候将所经过的结点依次入栈，当 p 自下至上走的时候，将 p 所经过的结点依次出栈，当 p 来到叶子结点的时候，自底至顶输出栈中元素就是根到叶子的路径。因此我们可以写出以下代码：

```
int t ;
int top=-1;
char pathstack[MAX];              //此处两句定义了存储路径用的栈。
void allPath(BTNode *p)
{
    if(p!=NULL)
    {
        pathstack[top]=p->data; //此处即为图 4.7 中程序模板的（1）处，在此处点
        top++;                  //让所访问结点入栈，即 p 自上至下走的时候结入栈。
        if(p->lchild==NULL&&p->rchild==NULL) //如果当前结点是叶子结点则
        {                                    //打印路径。
            for(i=0;i<top;i++)
                cout<<pathstack[i];
        }
        allPath(p->lchild);
        allPath(p->rchild);
        top--;                  //此处即为图 4.7 中程序模板的（3）处，在此处让
                                //所访问结点出栈，即 p 自下至上走的时候结点出栈。
    }
}
```

2.解：

（1）满 K 叉树个层的结点数构成一个首相为 1，公比为 K 的等比数列，则各层结点数位：K^{h-1}，其中 h 为层号。

（2）因为该树每层上均有 K^{h-1} 个结点，从根开始编号为 1，则结点 i 的从右向左数第 2 个孩子的结点编号为 K×i。设 n 为结点 i 的子女，则关系式 $(i-1)\times K+2\leqslant n\leqslant i\times K+1$ 成立。因 i 是整数，故结点 n 的双亲 i 的编号为：$\left\lfloor\frac{n-2}{K}\right\rfloor+1$。

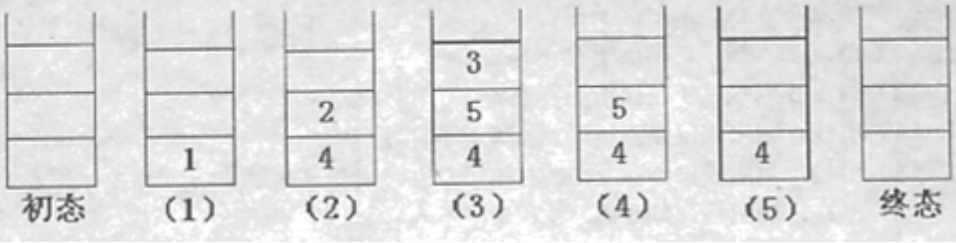
（3）结点 n 的前一结点编号为 n-1，其最右边子女编号是 $(n-1)\times K+1$，故结点 n 的第 i 个孩子的编号是 $(n-1)\times K+1+i$。

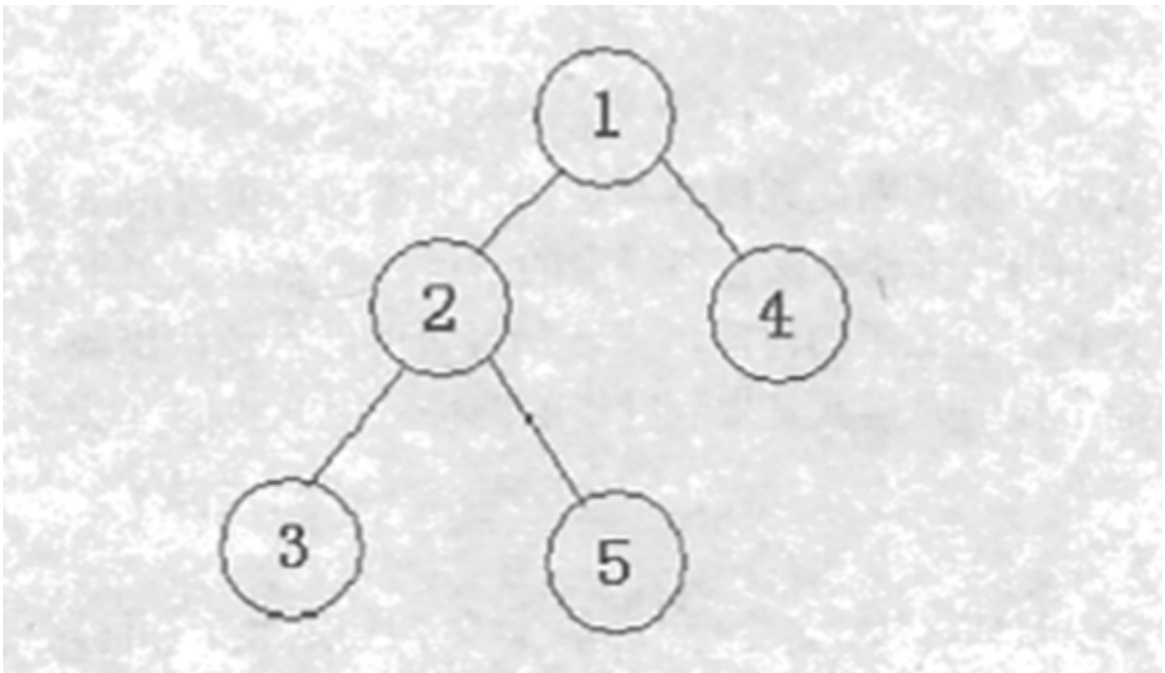
（4）结点 n 有右兄弟的条件是，它不是双亲的从右数的第一子女，即 $(n-1)\text{Mod } K\neq 0$，其右兄弟编号是 n+1。

3.分析：

二叉树的先序和中序遍历的递归算法中都用到了系统栈，我们要写出其遍历的非递归算法，主要任务就是用自己申请的栈来代替系统占的功能。系统栈在遍历过程中起什么作用呢？下图是一棵二叉树，比如进行先序遍历，来到结点 2 的时候下一个结点将要遍历 3，但是 2 还有一个分支将结点 5 没有遍历，因此我们要保存下 2 的位置，待 3 遍历完之后再返回 2 继续遍历其另一个分支结点 5，这就是系统栈在遍历时起到的作用，这同时也体现了之前讲过的栈的记忆特性。

1.对上图进行先序遍历，各个结点进展出栈情况如下：





初态栈空。

（1）结点 1 入栈。

（2）出栈，输出栈顶结点 1，并将 1 的左、右孩子结点（4 和 2）入栈；右孩子先入栈，左孩子后入栈，因为对左孩子的访问要先于右孩子，后入栈的会先出栈访问。

（3）出栈，输出栈顶结点 2，并将 2 的左、右孩子结点（5 和 3）入栈。

（4）出栈，输出栈顶结点 3，3 为叶子结点，无孩子，本步无结点入栈。

（5）出栈，输出栈顶结点 5。

出栈，输出栈顶结点 4，此时栈空，进入终态。

遍历序列为 1,2,3,5,4

由此可以写出以下代码：

```
void preorder(BTNode *bt)
{
    BTNode *Stack[MAX];//定义一个栈。
    int top=-1;          //初始化栈。
    BTNode *p;
    if(bt!=NULL)
    {
        Stack[++top]=bt;//根结点入栈。
        while(top!=-1)  //栈空循环退出，遍历结束。
```

```
            p=Stack[top--];//出栈并输出栈顶结点。
            cout<<p->data<<" ";
            if(p->rchild!=NULL)//栈顶结点右孩子存在，则右孩子入栈。
                Stack[++top]=p->rchild;
            if(p->lchild!=NULL)//栈顶结点左孩子存在，则左孩子入栈。
                Stack[++top]=p->lchild;
        }
    }
}
```

2．对上图进行中序遍历，各个结点进展出栈情况如下：

初太栈空。

（1）结点 1 入栈，1 左孩子存在。

（2）结点 2 入栈，2 左孩子存在。

（3）结点 3 入栈，3 左孩子不存在。

（4）出栈，输出栈顶结点 3，3 右孩子不存在。

（5）出栈，输出栈顶结点 2，2 右孩子存在，右孩子 5 入栈，5 左孩子不存在。

（6）出栈，输出栈顶结点 5，5 右孩子不存在。

（7）出栈，输出栈顶结点 1，1 右孩子存在，右孩子 4 入栈，4 左孩子不存在。

出栈，输出栈顶结点 4，此时栈空，进入终态。

遍历序列为 3,2,5,1,4

由上步骤可以看出，中序非递归遍历过程为：

（1）开始根结点入栈。

（2）循环执行如下操作：如果栈顶结点左孩子存在，则左孩子入栈；如果栈顶结点左孩子不存在，则出栈并输出栈顶结点，然后检查其右孩子是否存在，如果存在则右孩子入栈。

（3）当栈空时算法结束。

由此可以写出以下代码：

```
void inorder(BTNode *bt)
{
    BTNode Stack[MAX];
    int top=-1;
    BTNode *p;
    if(bt!=NULL)
        p=bt;
    /*下边这个循环完成中序遍历。注意：图（7）中进栈出栈过程中会出现栈空状态，但
    这时遍历还没有结束，因根结点的右子树还没有遍历，此时 p 非空，根据这一点来维持循
    环的进行。*/
        while(top!=-1||p!=NULL)
        {
            while(p!=NULL)//左孩子存在则左孩子入栈。
            {
                Stack[++top]=p;
                p=p->lchild;
            }
        }
        if(top!=-1)//在栈不空的情况下出栈并输出出栈结点。
        {
            p=Stack[top--];
            cout<<p->data<<" ";
            p=p->rchild;
        }
    }
```