**《数据结构》实验报告**

**姓名： 杨孟衡 学号： 8002118240**

**班级： 软件工程1809班 专业： 软件工程**

**报告日期： 2019 年 11 月 20 日**

**实验六 树**

**一、问题描述与分析**

1、理解树的结构特征及各种存储方法；

2、掌握前序序列递归建树算法；

3、掌握树的前序、后序及层次遍历算法，设计并实现树结构相关问题算法，如求高度、叶子结点数等。

1. 编写算法void levelorder(tree t)实现树的层次遍历。(lab6\_01.c)
2. 假设树采用指针方式的孩子表示法存储，试编写一个非递归函数void preorder(tree root), 实现树的前序遍历。 (lab6\_02.c)
3. 假设树采用指针方式的孩子表示法存储，试编写一个非递归函数void Postorder(tree root), 实现树的后序遍历。 (lab6\_03.c)
4. 假设树采用指针方式的孩子表示法存储，试编写一个函数int isequal(tree t1, tree t2), 判断两棵树是否等价。 (lab6\_04.c)
5. 假设树采用指针方式的孩子表示法存储，试编写一个函数tree Ct(char s[]), 根据输入的树的括号表示字符串s，建立树的存储结构。例如，若要建立教材图6.4所示的树，应输入A(b(e,f),c,d(g(I,j,k),h)) (lab6\_05.c)

2.分析

(1) 根据层次遍历的定义，首先访问第一层的根结点，其次按自左向右的顺序访问第二层中的所有结点，而第二层的所有结点恰巧为第一层根结点的子女，接下来访问第三层中的所有结点，就循环直到所有结点都被访问了；

(2) 由于非递归实现前序遍历，需要用一个栈保存未访问的根结点，所以先定义一个空栈，然后对每一个结点遍历，按照先根结点，再按照自左向右依次按前序遍历的方式遍历所有结点，若中途遇到部分子树还未访问就先压入栈中，待之后的循环遍历；

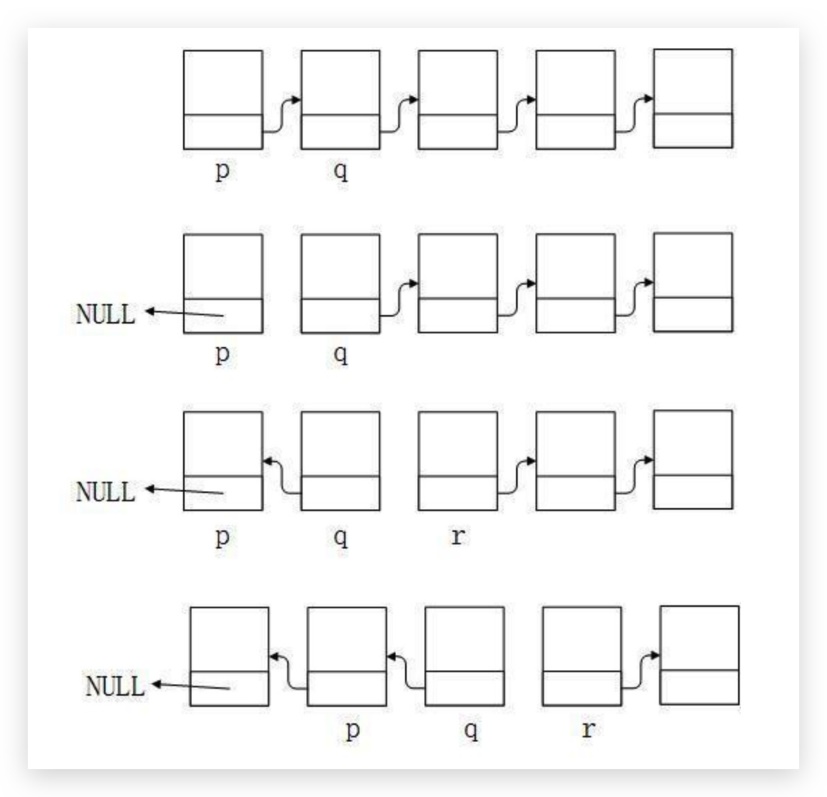
(3) 同前序遍历的基本思想，只不过顺序相反；

(4)两棵树相等的充要条件是所有结点位置的结点值相等，依次遍历两棵树对应然后对对应的结点值比较；

（5）从左到右扫描树的括号表示，每当遇到左括号时，其前一个结点进栈，并读下一个符号，每当遇到右括号时，栈顶元素出栈，说明以栈顶元素为根的数（子树）构造完毕，此时若栈为空，算法结束，否则度下一个符号；每当遇到结点，则它一定为栈顶元素的子女，将其挂到栈顶元素的某子女位置上，并读下一个符号，每当遇到“，”，则略过该符号，并读下一个符号；

**二、数据结构与算法设计**

链表图示如下：



**三、算法复杂度分析**

（1）嵌套循环遍历，算法复杂度为O（n^2）；

（2）单重循环，算法复杂度为O（n）；

（3）嵌套循环，算法复杂度为O（n^2）；

（4）单重循环，算法复杂度为O（n）；

（5）单重循环，算法复杂度为O（n）；

**四、测试计划**

1.编写目的

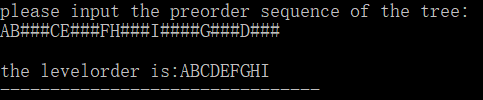
加深对数据结构链表的存储结构，递归程序的理解。

2.开发及运行环境Dev c++

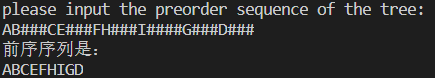
Visual Stdio Code 1.40

3.小项测试截图

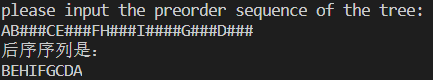
（1）层次遍历



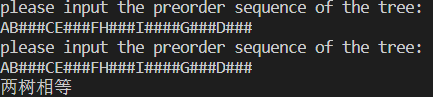
（2）前序遍历



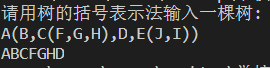
（3）后序遍历



（4）判断等价



（5）树的括号表示法



**五、源程序(仅贴算法函数部分)**

**<lab6\_01>:**

 void levelorder(tree t)    /\* t为指向树根结点的指针\*/

{

      tree queue[100];

      int f,r,i;

      tree p;

      f=0;

      r=1;

      queue[0]=t;

      while (f<r)

      {

        p=queue[f++];

        printf("%c",p->data);

        for (i=0;i<m;++i)

        {

          if (p->child[i])

          {

              queue[r++]=p->child[i];

          }

        }

      }

}

**<lab6\_02>:**

void  PreOrder1(tree root)

{

    tree stack[100];

    int top=-1;

    int i;

    while (root || top!=-1)

    {

        if (root)

        {   printf("%c",root->data);

            for (i=m-1;i>0;i--)

                if (root->child[i])

                 stack[++top]=root->child[i];

            root=root->child[0];

        }

        else

        if (top>-1)

        {

            root=stack[top--];

        }

   }

}

**<lab6\_03>:**

int PostOrder1(tree root)

{

    tree treeStack[MAXLEN];

    int top = -1;

    tree printStack[MAXLEN];

    int topp = -1;

    int i;

    if( root ) treeStack[++top] = root;

    while( top != -1 )

    {

        root = treeStack[top--];

        for(i=0;i<m;i++)

        {

            if( root->child[i] ) treeStack[++top] = root->child[i];

        }

        printStack[++topp] = root;

    }

    while( topp != -1 ) printf("%c",printStack[topp--]->data);

}

**<lab6\_04>:**

int equal(tree t1,tree t2)

{   int i;

    if (t1 == NULL && t2 == NULL)

    {

        return TRUE;

    }

    else if (t1 == NULL && t2 != NULL  ||   t1 != NULL && t2 == NULL)

    {

        return FALSE;

    }

    else

    {

        if (t1->data != t2->data)

        {

            return FALSE;

        }

        for (i = 0;i < m;i++)

        {

            if ( equal(t1->child[i],t2->child[i]) == FALSE)

            {

                return FALSE;

            }

        }

        return TRUE;

    }

}

**<lab6\_05>:**

tree Ct(char s[MAXLEN])

{

    int length;

    tree root;

    int i,j,top;

    tree stack[MAXLEN],temp = NULL,p;

    int childSeq[MAXLEN];

    top = -1;

    length = strlen (s);

    for (i = 0;i < length;i++)

    {

        if (s[i] == ',')

        {

            continue;

        }

        else if (s[i] == '(')

        {

            stack[++top] = temp;

            childSeq[top] = 0;

        }

        else if (s[i] == ')')

        {

            top--;

        }

        else if (top != -1)

        {

            p = (tree)malloc (sizeof (node));

            p->data= s[i];

            for (j = 0;j < m;j++)

            {

                p->child[j] = NULL;

            }

             temp = p;

             stack[top]->child[childSeq[top]++] = temp;

        }

        else

        {

            root = (tree)malloc (sizeof (node));

            root->data= s[i];

            for (j = 0;j < m;j++)

            {

                root->child[j] = NULL;

            }

            temp = root;

        }

    }

    return root;

}