学号 WA2214014 专业 人工智能 姓名 杨跃浙
实验日期 6 月 10 号 教师签字 成绩

实验报告

【实验名称】 数和二叉树

【实验目的】

二叉树是一种最常用的树形结构，掌握二叉树的 5 个性质。

二叉树有两种常用的存储表示：顺序存储和链式存储，链式存储是二叉树常用的存储结构，也称二叉链表存储表示法。掌握二叉树的两种存储表示法，尤其是二叉链表存储表示法；掌握不同存储结构的特点和适用场合。

二叉树的遍历算法是其他运算的基础，掌握二叉树的先序，中序和后序遍历算法，能够灵活运用三种遍历算法实现二叉树的其他操作。例如，二叉树的创建算法，计算二叉树的高度，计算二叉树的结点总数，计算叶子结点总数等。

输入中序和后序的遍历顺序，输出先序遍历顺序。

【实验原理】

使用二叉链表表示如图所示的二叉树，递归输出先序、中序和后序遍历的结果

统计二叉树中叶子节点和所有节点的数量

删除该二叉树

利用一个队列，实现对一颗二叉树的层序遍历

输入中序和后序的遍历顺序，输出先序遍历顺序

```
constructBitree(char * pPost, char * pMid, int iLen)
{
    构造树节点 pNode
    查找*(pPost+iLen-1)在 pMid 中的位置 iPos
    pNode->lchild=constructBitree(pPost,pMid,iPos)
    pNode->rchild=constructBitree(pPost+iPos,pMid+iPos+1iLen-1-iPos)
    return pNode
}
```

【实验内容】

使用二叉链表表示如图所示的二叉树，递归输出先序、中序和后序遍历的结果

统计二叉树中叶子节点和所有节点的数量

删除该二叉树

利用一个队列，实现对一颗二叉树的层序遍历

输入中序和后序的遍历顺序，输出先序遍历顺序

```
#include <iostream>
#include <string>
using namespace std;
#define OK 1
#define ERROR 0
#define OVERFLOW -2
#define MAXLEN 255
```

```
typedef char TElemType;
typedef int Status;
typedef struct BiTNode {
	TElemType data;
	struct BiTNode* lchild, * rchild;
}BiTNode,*BiTree;
typedef BiTree QElemType;
typedef struct QNode
{
	QElemType data;
	struct QNode* next;
}QNode, * QueuePtr;
typedef struct
{
	QueuePtr front;
	QueuePtr rear;
}LinkQueue;
void CreatBiTree(BiTree& T)
{
	char ch;
	cin >> ch;
	if (ch == '#') T = NULL;
	else
	{
		T = new BiTNode;
		T->data = ch;
		CreatBiTree(T->lchild);
		CreatBiTree(T->rchild);
	}
}
void PreOrderTraverse(BiTree T)
{
	if (T)
	{
		cout << T->data;
		PreOrderTraverse(T->lchild);
		PreOrderTraverse(T->rchild);
	}
}
void InOrderTraverse(BiTree T)
{
	if (T)
	{
		InOrderTraverse(T->lchild);
		cout << T->data;
```

```
            InOrderTraverse(T->rchild);
        }
}
void PostOrderTraverse(BiTree T)
{
        if (T)
        {
            PostOrderTraverse(T->lchild);
            PostOrderTraverse(T->rchild);
            cout << T->data;
        }
}
void FreeBiTree(BiTree T)
{
        if (T)
        {
            FreeBiTree(T->lchild);
            FreeBiTree(T->rchild);
            delete(T);
        }
}
int NodeCount(BiTree T)
{
        if (!T) return 0;
        return NodeCount(T->rchild) + NodeCount(T->lchild) + 1;
}
int NodeCount_Leaf(BiTree T)
{
        if (!T) return 0;
        if ((!T->lchild) && (!T->rchild)) return NodeCount_Leaf(T->rchild) + NodeCount_Leaf(T->lchild) + 1;
        return NodeCount_Leaf(T->rchild) + NodeCount_Leaf(T->lchild);
}

Status Init_LinkQueue(LinkQueue& Q)
{
        Q.front = Q.rear = new QNode;
        Q.front->next = NULL;
        return OK;
}
Status En_LinkQueue(LinkQueue& Q, QElemType e)
{
        QNode* p = new QNode;
        p->data = e;
        p->next = NULL;
        Q.rear->next = p;
```

```
    Q.rear = p;
    return OK;
}
Status De_LinkQueue(LinkQueue& Q, QElemType& e)
{
    if (Q.front == Q.rear) return ERROR;
    QNode* p = Q.front->next;
    e = p->data;
    Q.front->next = p->next;
    if (Q.front->next == NULL) Q.rear = Q.front;
    delete p;
    return OK;
}
void LevelOrderTraverse(BiTree T)
{
    LinkQueue Tree;
    BiTree e;
    Init_LinkQueue(Tree);
    En_LinkQueue(Tree,T);
    while (De_LinkQueue(Tree, e))
    {
        cout << e->data;
        if (e->lchild) En_LinkQueue(Tree, e->lchild);
        if (e->rchild) En_LinkQueue(Tree, e->rchild);
    };
};
int find(char Node, char* Tree)
{
    for (int i=0;i<strlen(Tree);i++)
        if (Node == Tree[i]) return i;
}
BiTree constructBitree(char* pPost, char* pMid, int iLen)
{
    if (iLen == 0) return NULL;
    BiTree pNode = new BiTNode;
    pNode->data = *(pPost + iLen - 1);
    int iPos=find(*(pPost + iLen - 1), pMid);
    pNode->lchild = constructBitree(pPost, pMid, iPos);
    pNode->rchild = constructBitree(pPost + iPos, pMid + iPos + 1, iLen - 1 - iPos);
    return pNode;
}
void CreatBiTree_ByInandPost()
{
    char pMid[MAXLEN], pPost[MAXLEN] = {'\0'};
    int iLen;
```

```
	BiTree T;
	cin >> pMid;
	cin >> pPost;
	iLen = strlen(pPost);
	T = constructBitree(pPost, pMid, iLen);
	PreOrderTraverse(T);
	cout << endl;
}
int main()
{
	BiTree Tree;
	CreatBiTree(Tree);
	PreOrderTraverse(Tree);
	cout << endl;
	InOrderTraverse(Tree);
	cout << endl;
	PostOrderTraverse(Tree);
	cout << endl;
	cout << NodeCount(Tree)<<endl;
	cout << NodeCount_Leaf(Tree)<<endl;
	LevelOrderTraverse(Tree);
	cout << endl;
	FreeBiTree(Tree);
	_CrtDumpMemoryLeaks();
	CreatBiTree_ByInandPost();
	return 0;
}
//ABD###CEG###FH##I##
//ABDCEGFHI
//DBAGECHFI

//DBGEHIFCA
```

【小结或讨论】

通过该次实验我掌握了二叉树的创建与删除，二叉树的遍历算法包括先序遍历，中序遍历，后序遍历和层序遍历，并能够通过遍历算法实现二叉树的其他操作，如统计二叉树中叶子节点和所有节点的数量，通过输入中序和后序的遍历顺序，输出先序遍历顺序等。