**江西师范大学计算机信息工程学院学生实验报告（6）**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **专业：** | **数据科学与大数据技术2班** | **姓名：** | **赖丽婷** | **学号：** |  | **日期：** | **2021.10.31** |

|  |  |  |  |
| --- | --- | --- | --- |
| 课程名称 | 数据结构 | 实验室名称 | 计算机综合实验室 |
| 实验名称 | 树 | | |
| 指导教师 |  | 成绩 |  |

1. **实验目的**（结出本次实验所涉及并要求掌握的知识点）

理解树的结构特征及各种存储方法

掌握前序序列递归建树算法

掌握树的前序，后序及层次遍历算法，设计并实现树的结构相关问题算法

如求高度，叶子节点树等

**2.实验内容**（结出实验内容具体描述）

2.1编写算法函数实现树的层次遍历

2.2编写一个非递归函数实现树的前序遍历算法

2.3编写一个非递归函数实现树的后序遍历算法

2.4编写一个函数，判断两颗给定的树是否等价

2.5根据输入树的括号表示字符串S,建立树的存储结构

**3.算法描述及实验步骤**（用适当的形式表达算法设计思想与算法实现步骤）

3.1首先我们知道，层次遍历是上下结构的遍历，也就是谁先进谁就先出，所以我们可以想到用队列，我们先把没一层的节点放进队列中，然后看队首节点是否有子树，如果有则再放入队尾，找完后，队首往后移一位，然后再循环查看队首的子节点，如果有放进队尾，直到最后遍历完整个子树，具体首先设置一个front跟一个top指针，front为队尾top为队尾，先输出队首的值，然后查看队首是否有子树，如果有则加入队尾后，此节点找完第一层子树后，队首后移一位，如此循环下去，就能完整遍历树的节点了。

3.2我们先看前序遍历过程中，打印树节点值时的顺序为根→左→右，我们需要知道当左边子树找完之后去访问哪棵子树，以及先访问哪颗子树，发现可以使用栈的先进后出的方法，把右边的树先放到栈中，先把左边的树出栈，注意我们先把右边的节点放进栈中，进栈过程中可以不需要把栈顶的最左边子树节点的存入栈中，也可以访问左边子树的时候就把树出栈，在出栈前，先打印栈顶的值，

3.3在后序遍历过程中，打印树节点值顺序为左→右→根，这里就不像上一题那么简单了，我们要找到每个节点的所有子树，然后根据节点的输出顺序进行输出，以为需要反方向打印存储的树节点这里我们还是可以利用栈的先进后出的方法，不过这里需要用到两个栈，一个栈用来访问pushstack，另一个栈用来保存输出顺序printstack，根节点先放入pushstack中，然后我们先访问根节点，把根节点出栈同时放入printstack中，把根节点的子树节点按照从左往右的方式进栈，然后访问栈顶，出栈顶同时放入printstack中，然后以同样的方法，直到把整棵树遍历完，然后遍历输出printstack中的值。

3.4判断两颗树是否相等可以想到树的递归前序，要注意子树两个为空，一个为空一个不为空，还有两个不为空时值是否相等的判断，还要看注意根节点是否相等

3.5根据树的括号表示法来创建树，我们需要建立一个根节点，然后因为遇到括号的时候代表前面那个子树节点有子树，遇到括号的闭合后，我们又要访问当前子树的父级，这样我们就可以想到栈的后进先出的特点，不过我们还需要一个数组来保存确定已存在有子树的树节点当前存有孩子的个数，注意在创建树的过程中我们需要先给他的全部孩子赋值为NULL，当遇到括号时，在根据字符串中的值来给子树赋值

1. **调试过程及运行结果**（详细记录在调试过程中出现的问题及解决方法。记录实验执行的结果）

**5. 总结**（对实验结果进行分析，问题回答，实验心得体会及改进意见）

利用好队列与栈的特点，实现一些非递归复杂程序设计，在特定情况下，我们可以稍微改变一下栈与队列的结构，书本上我们对树的前序遍历后序遍历都是用的递归程序设计，代码比较简洁，通过实验六的练习，对树的前序遍历后序遍历有了更深的理解，也让我们学会多种方式思考问题。

**6.附录**（程序源代码等）

6.1

#include "tree.h"

void levelorder(tree t) {

tree quene[100];

tree root;

int front, top, i;

front = 0;

top = 1;

quene[front] = t;

while (front < top) {

printf("%5c", quene[front]->data);

for (i = 0; i < m; i++) {

if (quene[front]->child[i]) {

quene[top++] = quene[front]->child[i];

}

}

front++;

}

}

int main() {

setbuf(stdout,NULL);

tree t;

printf("please input the preorder sequence of the tree:\n");

t = createtree();

printf("\nthe levelorder is:");

levelorder(t);

return 0;

}

6.2

#include "tree.h"

//第一种写法

void PreOrder1(tree root) {

tree stack[100];

int top = 0, i;

stack[top++] = root;

while (top != 0) {

if (root) {

printf("%5c", root->data);

for (i = m - 1; i > 0; i--) {

if (root->child[i]) {

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

}

}

root = root->child[0];

}else{

root = stack[--top];

}

}

}

//第二种写法

void PreOrder2(tree root) {

tree stack[100];

int top = 0, i;

stack[top++] = root;

while (top != 0) {

if (root) {

printf("%5c", root->data);

for (i = m - 1; i >= 0; i--) {

if (root->child[i]) {

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

}

}

}

root = stack[--top];

}

}

int main() {

tree root;

printf("please input the preorder sequence of the tree:\n");

root = createtree();

printf("is：\n");

PreOrder1(root);

return 0;

}

6.3

#include "tree.h"

void PostOrder1(tree root)

{

tree pushstack[100];

tree printstack[100];

int top = 0, i;

int ptop = 0;

pushstack[top++] = root;

while(top != 0)

{

root = pushstack[--top];

for(i = 0; i < m; i++)

{

if(root->child[i])

{

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

}

}

printstack[ptop++] = root;

}

while(ptop != 0)

{

printf("%5c", printstack[--ptop]->data);

}

}

int main ()

{

tree root;

printf("please input the preorder sequence of the tree:\n");

root =createtree();

printf("后序序列是：\n");

PostOrder1(root);

return 0;

}

6.4

#include "tree.h"

#define TRUE 1

#define FALSE 0

int equal(tree t1,tree t2)

{

int flag = TRUE,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)

{

for(i = 0; i < m; i++)

{

flag = flag && equal(t1->child[i], t2->child[i]);

}

}else{

return FALSE;

}

}

return flag;

}

int main ()

{

tree t1,t2;

printf("please input the preorder sequence of the tree:\n");

t1=createtree();

getchar();

printf("please input the preorder sequence of the tree:\n");

t2=createtree();

if ( equal(t1,t2) == TRUE)

{

printf ("两树相等\n");

}

else

{

printf ("两树不相等\n");

}

return 0;

}

6.5

#include "tree.h"

/\*请将本函数补充完整，并进行测试\*/

tree Ct(char s[MAXLEN])

{

tree stack[100], root = NULL, temp = NULL;

int children[m];

int top, length, i, j;

top = -1;

length = strlen(s);

for(i = 0; i < length; i++)

{

if(s[i] == ',')

{

continue;

}

else if(s[i] == '(')

{

stack[++top] = temp;

children[top] = 0;

}

else if(s[i] == ')')

{

top--;

}

else if(top != -1)

{

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

temp->data = s[i];

for(j = 0; j < m; j++)

{

temp->child[j] =NULL;

}

stack[top]->child[children[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;

}

int main ()

{

char s[MAXLEN];

tree root = NULL;

printf ("请用树的括号表示法输入一棵树:\n");

scanf ("%s",s);

root = Ct(s);

preorder(root); /\*前序遍历树\*/

return 0;

}