

**数据结构课程设计**

**题目：** 求二叉树节点上的路径

**专业班级：** 计算机科学与技术 6班

**学 号：** 20204331

**姓 名：** 王博文

**指导教师：** 隋毅

**报告日期：** 2022年3月27日

计算机科学技术学院

**2022年3月27日**

一、课程设计内容

**设计要求：**在采用链式存储结构存储的二叉树上，以bt指向根结点，p指向任一给定的结点，编程实现求出从根结点到给定结点之间的路径。

菜单内容：

1. **建立二叉树存储结构**
2. **求二叉树的前序遍历**
3. **求二叉树的中序遍历**
4. **求二叉树的后续遍历**
5. **求指定结点的路径**
6. **退出系统**

**请选择：1 – 6：**

**提示：**

【采用非递归遍历的方法】

1. 二叉树的建立
2. 求指定结点的路径
3. 二叉树的前、中、后序非递归遍历算法
4. 查找函数

二、算法设计

**首先运用递归进行二叉树的创立，我的先序中序后序遍历都是采用非递归进行求解，每个函数内部都采用了栈进行操作，在第五题的思路上，我并没有运用递归方法进行求解，而是运用非递归的思路，也是用栈进行，首先先将所有左子树进栈，然后进栈的节点的falg全为1，如果这些满足不了，那么就出栈，然后看现在的栈顶有没有右子树，然后依次类推即可得出答案。**

1. 核心代码实现

#include<iostream>

#include<cstdio>

#include<cstdlib>

#include<cstring>

#include<stack>

#include<queue>

using namespace std;

struct BiTreeNode {

char data;

BiTreeNode\* left;

BiTreeNode\* right;

int flag;

};

BiTreeNode\* CreatBiTree(char\* s, int &i, int len) { //递归建立二叉树

BiTreeNode\* root = (BiTreeNode\*)malloc(sizeof(BiTreeNode));

if(s[i]=='#'||i==len) {

i++;

return root=NULL;

} else {

root->data=s[i++];

root->left = CreatBiTree(s, i, len);

root->right = CreatBiTree(s, i, len);

return root;

}

}

void PreOrder\_(BiTreeNode\* root) { //先序非递归

stack <BiTreeNode\*> q;

BiTreeNode \*head=root;

q.push(head);

printf("%c",head->data);

while(!q.empty()) {

while(head->left!=NULL) {

head=head->left;

q.push(head);

printf("%c",head->data);

}

BiTreeNode\* p;

p=q.top();

q.pop();

while(p->right==NULL&&!q.empty()) {

p=q.top();

q.pop();

}

if(p->right!=NULL) {

head=p->right;

q.push(p->right);

printf("%c",p->right->data);

}

}

}

void InOrder\_(BiTreeNode\* root) { //中序非递归

stack <BiTreeNode\*> q;

BiTreeNode \*head=root;

q.push(head);

while(!q.empty()) {

while(head->left!=NULL) {

head=head->left;

q.push(head);

}

BiTreeNode\* p;

p=q.top();

q.pop();

printf("%c",p->data);

while(p->right==NULL&&!q.empty()) {

p=q.top();

q.pop();

printf("%c",p->data);

}

if(p->right!=NULL) {

head=p->right;

q.push(p->right);

}

}

}

void PostOrder\_(BiTreeNode\* root) {

stack <BiTreeNode\*> q;

BiTreeNode\* p=root;

BiTreeNode\* r=NULL;

while(p||!q.empty()) {

if (p) {

q.push(p);

p=p->left;

} else {

p=q.top();

if (p->right&&p->right!=r) {

p=p->right;

q.push(p);

p=p->left;

} else {

q.pop();

printf("%c",p->data);

r=p;

p=NULL;

}

}

}

}

void Print\_(stack <BiTreeNode\*> q){

stack <BiTreeNode\*> p;

while(!q.empty()){

p.push(q.top());

q.pop();

}

cout<<"此二叉树存在此节点"<<endl;

while(!p.empty()){

BiTreeNode \*a;

a=p.top();

p.pop();

if(!p.empty()){

cout<<a->data<<"->";

}

else

cout<<a->data;

}

cout<<endl;

}

void Findtrace(BiTreeNode\* root){

cout<<"请输入想要查找的节点 "<<endl;

char s;

cin>>s;

stack <BiTreeNode\*> q;

BiTreeNode \*head=root;

q.push(head);

while(!q.empty()){

BiTreeNode \*p;

p=q.top();

if(p->left!=NULL&&p->left->flag!=1){

q.push(p->left);

p->left->flag=1;

if(p->left->data==s){

Print\_(q);

return ;

}

}

else {

p=q.top();

if(p->right!=NULL&&p->right->flag!=1){

q.push(p->right);

p->right->flag=1;

if(p->right->data==s){

Print\_(q);

return ;

}

}

else

q.pop();

}

}

cout<<"查无此节点"<<endl;

}

int main() {

cout<<"1.建立二叉树存储结构"<<endl<<"2.求二叉树的前序遍历"<<endl<<"3.求二叉树的中序遍历"<<endl;

cout<<"4.求二叉树的后续遍历"<<endl<<"5.求指定结点的路径"<<endl<<"6.退出系统"<<endl;

char str[200];

int i = 0;

int len = int(strlen(str));

BiTreeNode\* root;

while(1) {

int z;

cin>>z;

switch(z) {

case 1:

scanf("%s", str);

root = CreatBiTree(str, i, len);

break;

case 2:

printf("先序非递归遍历：");

PreOrder\_(root);

printf("\n");

break;

case 3:

printf("中序非递归遍历：");

InOrder\_(root);

printf("\n");

break;

case 4:

printf("后序非递归遍历：");

PostOrder\_(root);

printf("\n");

break;

case 5:

Findtrace(root);

break;

case 6:

exit(0);

}

}

return 0;

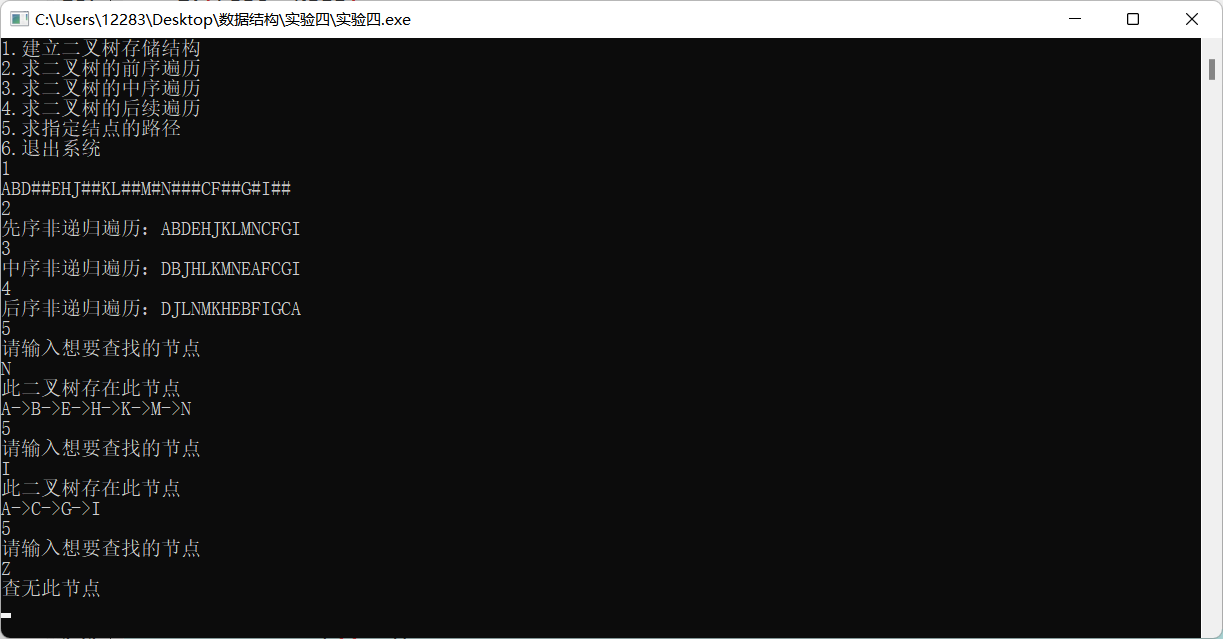
}

/\*

ABD##EHJ##KL##M#N###CF##G#I##

\*/

四、测试与分析



五、总结及体会

在这次的实验中，我几乎全部运用的是非递归的解法，这让我对栈的运用又更加的了解了不少，在这期间，我对于什么时候出栈的理解还是不够透彻，在那个点卡了不少时间，这方面还是要勤加练习，

1. 参考书目和网络资源

无