

# 数据结构课程设计报告

设计题目：线索二叉树

学生姓名：杨程锦

专 业：计算机科学与技术

班 级：21-1班

学 号：2021214710

指导教师：李培培

完成日期：2022/07/15

# （一）需求和规格说明

**问题描述：**设计线索二叉树的相关运算及其演示系统

**编程任务：**

1. 实现动态存储方式的线索二叉树类；
2. 要求能够从文件读取数据构建线索二叉树，进行遍历、节点的插入和删除，并保存到文件中；
3. 任意构建二叉树，实现3种线索化操作，及其前驱与后继的查找；
4. 实现线索二叉树的图形用户界面操作。

# （二）设计

## 1．设计思想

本题主要考察二叉树的线索化操作。我选取的数据存储方式是带左右孩子指针，左右标记，父节点指针的二叉树。一开始我认为进行三种线索化不需要带有父指针，但是直到自己进行操作时候发现没有记录父节点的指针对于后续线索化的进行相当困难，所以最后添加了一个指向父节点的指针。在进行创建二叉树时，我进行了先序创建二叉树，通过fstream类进行文件的读写，当节点为空时，文件读取内容为“#”，不为空时创建节点。后续，进行线索化时，根据其遍历顺序，对根节点、左右节点按照一定的顺序进行线索化，添加线索。遍历时，将根节点的地址传入，通过循环语句对其根节点、左右节点进行遍历输出。

在查找前驱时，我将每个节点入栈，进行栈顶元素的查找，直到找到这个节点，然后将这个节点出栈，下一个栈顶节点即为该节点的前驱。

在查找后继时，我将每个结点入队，进行队首元素的查找，直到找到这个节点，然后将这个节点出队，下一个队首节点即为该节点的后继。

## 2. 设计表示

1. 结点类：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **类名** | **成员类别** | **类型** | **成员名** | **描述** |
| bTHRNode | 数据 | char | data | 存储的数据 |
| bTHRNode\* | leftChild | 左孩子节点 |
| int | leftTag | 左标记 |
| bTHRNode\* | rightChild | 右孩子节点 |
| int | rightTag | 右标记 |
| bTHRNode\* | bTHRNode\* | 父节点 |

1. 线索二叉树类：

typedef bTHRNode\* Node2;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **类名** | **成员类别** | **类型** | **成员名** | **描述** |
| bTHRTree | 函数 | void | createTree(Node2 &T, Node2 parent) | 创建二叉树 |
| void | visit(Node2 T, int x) | 访问二叉树节点 |
| void | inThreading1(Node2 &T, Node2 &pre) | 中序线索化 |
| void | TinOrder(Node2 T, int x) | 中序遍历线索二叉树 |
| void | inThreading2(Node2 &T, Node2 &pre) | 先序线索化 |
| void | PreOrder(Node2 T, int x) | 先序遍历线索二叉树 |
| void | inThreading3(Node2 &T, Node2 &pre) | 后序线索化 |
| void | PostOrder(Node2 T, int x) | 后序遍历线索二叉树 |
| void | reverseThread(Node2 T) | 逆转线索化 |
| void | deleteNode(Node2 T, char x) | 删除节点 |
| void | insertNode(Node2 P, Node2 R) | 插入节点 |
| void | initStack(Node2 T, int type) | 节点入栈 |
| Node2 | searchPreNode(char x); | 查找前驱 |
| void | initQueueNode2 T, int type) | 节点入队 |
| Node2 | searchNextNode(char x) | 查找后继 |
| 数据 | Node2 | pre | 前驱节点 |
| Node2 | T | 头节点 |
| Node2 | parent | 父节点 |
| stack<Node2> | stack | 存放节点 |
| queue<Node2> | queue | 存放节点 |

1. 函数：

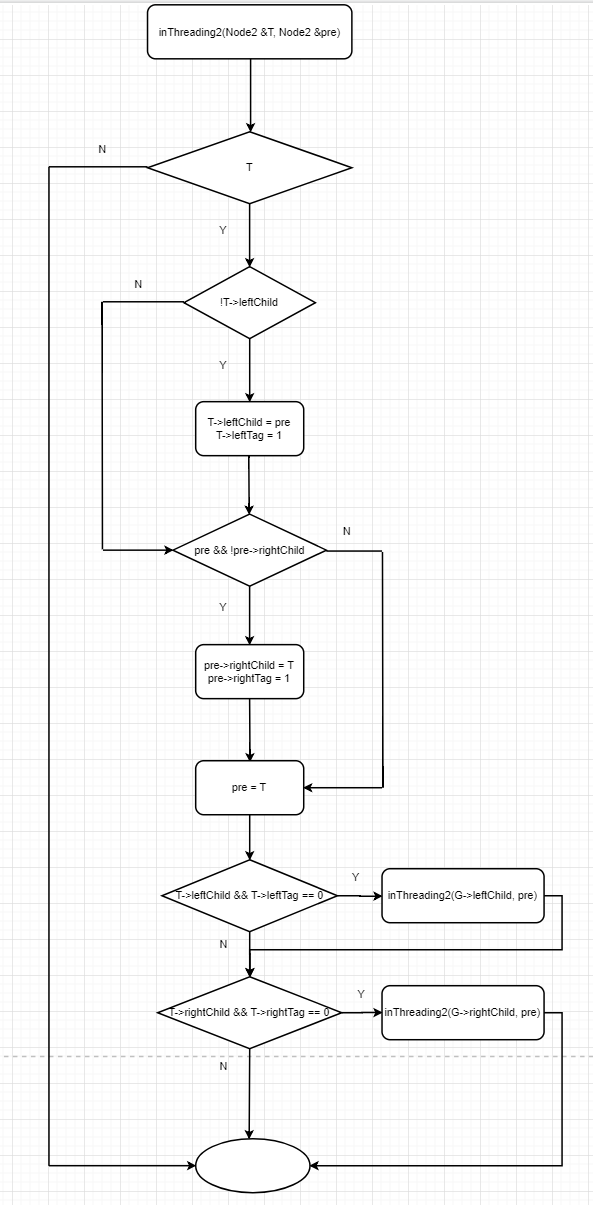
|  |  |  |
| --- | --- | --- |
| **数据类型** | **函数名称** | **描述** |
| void | menu() | 显示主菜单 |

## 3. 核心算法

该程序的核心算法是三个线索化操作，即inThreading1、inThreading2、inThreading3，三个函数。

开始程序第一步是根据文件进行先序建立二叉树，之后通过面板菜单进行选择操作，核心算法就是三个线索化算法。

先序线索化是先进行根节点的访问，然后依次进行左右节点的访问。对于左子树存在的节点，则不会有前驱；同样，右子树存在的节点，不会存在后继。只需要寻找节点的左右子树，并且判断是否为空，若为空则需要考虑前驱后继的问题。 若左子树为空，则需要把左孩子指向前驱这个节点，前驱节点是在每次该节点线索化结束之后用pre这个节点记录当前节点，即为前驱节点。把该节点线索化之后再对其左右节点分别线索化。中序、后序线索化步骤与先序基本一致，只是顺序不同。

图：线序二叉树线索化

# （三）用户手册

程序运行时，首先显示菜单栏，有七个选项，输入1-7七个数字，可以调用不同的功能。

在主函数开头，先创建一个bTHRTree类的实例，然后在switch语句中根据输入的值不同，调用不同的成员函数，不同的成员函数实现了不同的方法。选择1：进行二叉树的中序线索化，并且输出到文件中。

选择2：进行二叉树的先序线索化，并且输出到文件中。

选择3：进行二叉树的后序线索化，并且输出到文件中。

选择4：插入节点，选择二叉树线索化种类，主函数中写入插入的位置，然后进行插入节点。

选择5：删除节点，选择二叉树线索化种类，主函数中写删除节点的位置，然后进行删除节点。

选择6：查找前驱，选择二叉树线索化种类，然后调用函数查找前驱函数，最后输出结果。

选择7：查找后继，选择二叉树线索化种类，然后调用函数查找后继函数，最后输出结果。

# （四）调试及测试

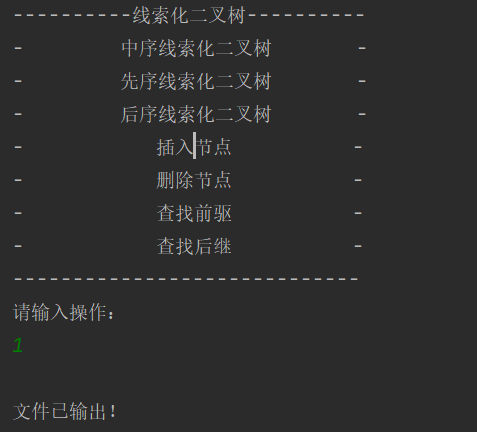
## 1. 测试数据：



**二叉树数据**

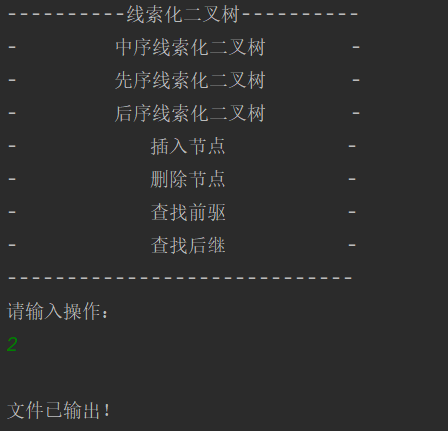
## 2. 测试结果：

（1）中序线索化二叉树



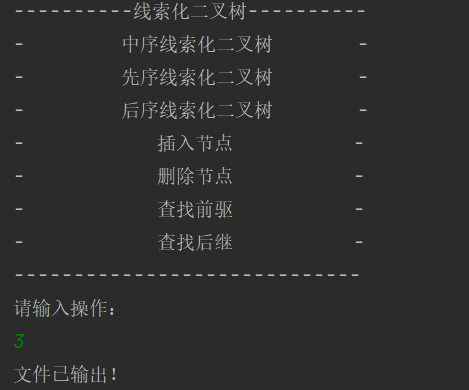


1. 先序线索化二叉树



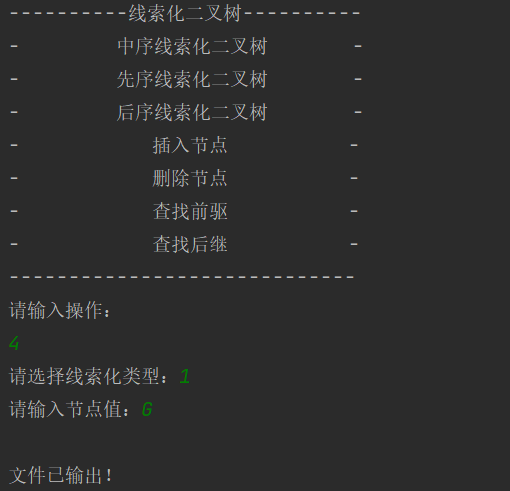


（3）后序线索化二叉树



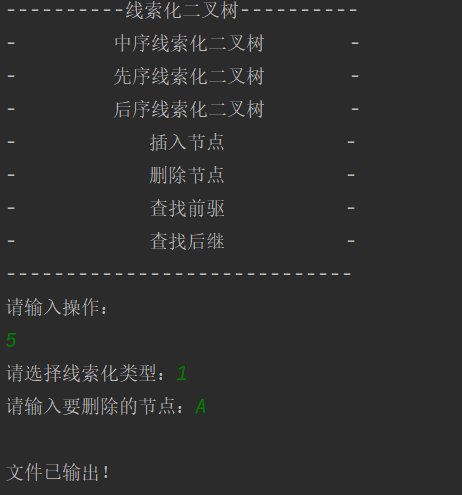


（4）插入节点



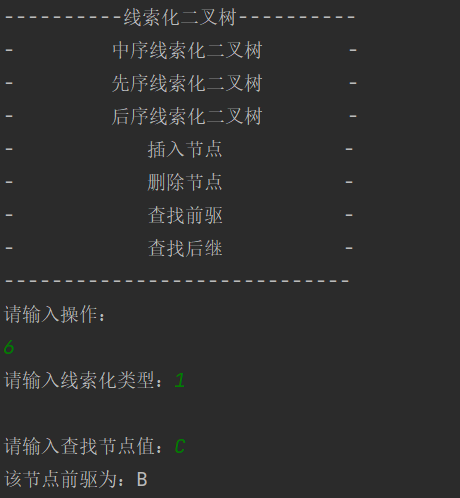


（5）删除节点



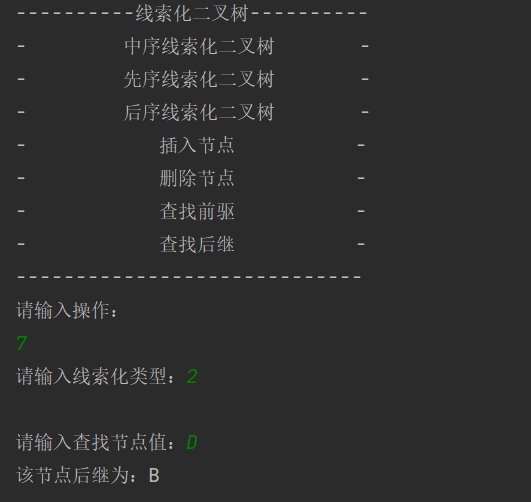


（6）查找前驱





（7）查找后继





## 3. 进一步改进：

（1）原本我没有写visit函数，在进行节点入栈、入队时候需要更改遍历时候的输出方法，例如把输出到文件改为入栈、入队。后面我进行visit函数的编写，在调用入栈、入队函数时候通过固定的visit函数的参数，调用不同的输出方法。使得程序相对更加高效。

（2）目前主函数不能在自己选择线索化之后再调用，功能不够随心所欲，无法在线索化之后再次走选择语句调用删除或者插入等功能，只能通过选择功能，在功能内部选择线索化种类。我认为可以增加一个goto语句，虽然不够完美，但是相对来说改进还是比较方便。

# (五) 感想

这道题并不算难，但是功能较多，内容较为复杂，运用了多种知识，例如栈、队列、二叉树。刚开始做这个题，我首先面临许多问题，我先在网上学习了三种线索化的基本知识，认识到了后序线索化的难度，网络资料较少，我面临了一些难题，再经过了思考之后，我发现必须添加一个指向父节点的指针才可以进行后序的线索化。我在建树时候，通过把该节点的父节点记录下来，然后进行先序建树，就得到了一棵节点类型为两个孩子节点、一个父节点、两个左右标记的二叉树。

在编写查找节点的前驱和后继时，我发现对于先序前驱和后序后继较难实现，我发现对于节点的遍历可以使用队列和栈进行操作。例如查找前驱节点，可以将所有的节点遍历之后挨个入栈，然后再进行数据的匹配，直到将栈顶元素的数据与查询数据相同时，结束出栈，然后再次出栈，剩余的栈顶元素则为该要求节点的前驱节点。后继节点同样是将节点入队，不断出队，直到与要求节点数据相同，然后再次出队，剩余的队首元素则为该节点的后继节点。

此次实验总体难度较小，但是对于某些操作比如查找前驱后继、后序线索化二叉树来说，还是比较难的。需要自己创新，找到一种合适的方法来进行操作。总体来说，通过此次课设我学到了很多内容。

# 附录：

## bTHRTree.h

#include <iostream>

#include <fstream>

#include <stack>

#include <queue>

using namespace std;

class bTHRNode{

public:

char data;

bTHRNode\* leftChild;

bTHRNode\* rightChild;

bTHRNode\* parent;

int leftTag;

int rightTag;

bTHRNode();

};

typedef bTHRNode\* Node2;

class bTHRTree{

public:

void createTree(Node2 &T, Node2 parent);

void visit(Node2 T, int x);

//中序线索化

void inThreading1(Node2 &T, Node2 &pre);

void TinOrder(Node2 T, int x); //线索化过程

//先序线索化

void inThreading2(Node2 &T, Node2 &pre);

void PreOrder(Node2 T, int x);

//后序线索化

void inThreading3(Node2 &T, Node2 &pre);

void PostOrder(Node2 T, int x);

//逆转线索化

void reverseThread(Node2 T);

//删除节点

void deleteNode(Node2 T, char x);

//插入节点

void insertNode(Node2 P, Node2 R);

//节点入栈

void initStack(Node2 T, int type);

//查找前驱

Node2 searchPreNode(char x);

//节点入队

void initQueue(Node2 T, int type);

//查找后继

Node2 searchNextNode(char x);

Node2 pre = NULL;

Node2 T = new bTHRNode();

Node2 parent;

stack<Node2> stack;

queue<Node2> queue;

};

## bTHRTree.cpp

#include "bTHRTree.h"

ifstream ifs("D:\\TREE.txt");

ofstream ofs("D:\\storeOrderTREE.txt");

bTHRNode::bTHRNode() {

this->rightChild = NULL;

this->leftChild = NULL;

this->parent = NULL;

this->leftTag = 0;

this->rightTag = 0;

}

void bTHRTree::visit(Node2 T, int x) {

switch (x) {

case 1:

ofs << T->data << " ";

break;

case 2:

stack.push(T);

break;

case 3:

queue.push(T);

break;

}

}

//有孩子为0 没孩子为1

void bTHRTree::createTree(Node2 &T, Node2 parent) {

char ch;

ifs >> ch;

if (ch == '#') {

T = NULL;

} else {

T = new bTHRNode();

T->data = ch;

T->parent = parent;

createTree(T->leftChild, T);

createTree(T->rightChild, T);

}

}

//中序 左根右

void bTHRTree::inThreading1(Node2 &T, Node2 &pre){

Node2 G = T;

if(G != NULL){

inThreading1(G->leftChild, pre);

if(G->leftChild == NULL){

G->leftTag = 1;

G->leftChild = pre;

} else {

G->leftTag = 0;

}

if(pre != NULL && pre->rightChild == NULL){

pre->rightTag = 1;

pre->rightChild = G;

}

pre = G;

inThreading1(G->rightChild, pre);

}

}

void bTHRTree::TinOrder(Node2 T, int x) {

Node2 G = T;

while (G){

while (G->leftTag == 0){

G = G->leftChild;

}

visit(G, x);

while (G->rightTag == 1){

G = G->rightChild;

visit(G, x);

}

G = G->rightChild;

}

cout << endl;

// cout << "文件已输出！";

}

//先序 根左右

void bTHRTree::inThreading2(Node2 &T, Node2 &pre) {

Node2 G = T;

if (G != NULL){

if (G->leftChild == NULL){

G->leftChild = pre;

G->leftTag = 1;

}

if (pre != NULL && pre->rightChild == NULL){

pre->rightChild = G;

pre->rightTag = 1;

}

pre = G;

if (G->leftTag == 0 && G->leftChild != NULL){

inThreading2(G->leftChild, pre);

}

if (G->rightTag == 0 && G->rightChild != NULL){

inThreading2(G->rightChild, pre);

}

}

}

void bTHRTree::PreOrder(Node2 T, int x) {

Node2 G = T;

while (G){

while (G->leftTag == 0){

visit(G, x);

G = G->leftChild;

}

visit(G, x);

G = G->rightChild;

}

cout << endl;

// cout << "文件已输出！";

}

//后序 左右根

void bTHRTree::inThreading3(Node2 &T, Node2 &pre) {

Node2 G = T;

if (G != NULL){

if (G->leftTag == 0 && G->leftChild != NULL){

inThreading3(G->leftChild, pre);

}

if (G->rightTag == 0 && G->rightChild != NULL){

inThreading3(G->rightChild, pre);

}

if (G->leftChild == NULL){

G->leftChild = pre;

G->leftTag = 1;

}

if (pre != NULL && pre->rightChild == NULL){

pre->rightChild = G;

pre->rightTag = 1;

}

pre = G;

}

}

void bTHRTree::PostOrder(Node2 T, int x) {

if (T != NULL){

Node2 G = T;

pre = NULL;

while (pre != T){

while (G->leftChild != NULL && G->leftTag == 0) {

G = G->leftChild;

}

while (G != NULL && G->rightTag == 1){

visit(G, x);

pre = G;

G = G->rightChild;

}

while (pre != T && G->rightChild == pre){

visit(G, x);

pre = G;

if (pre != T){

G = G->parent;

}

}

if (G->rightTag == 0){

G = G->rightChild;

}

}

}

// cout << "文件已输出！";

}

//逆转线索化

void bTHRTree::reverseThread(Node2 T) {

if (T != NULL){

if (T->leftTag == 1 && T->rightTag == 1){

T->leftChild = NULL;

T->rightChild = NULL;

T->leftTag = 0;

T->rightTag = 0;

}

if (T->leftTag == 0 && T->rightTag == 1){

T->rightChild = NULL;

T->rightTag = 0;

reverseThread(T->leftChild);

}

if (T->leftTag == 1 && T->rightTag == 0){

T->leftChild = NULL;

T->leftTag = 0;

reverseThread(T->rightChild);

}

if (T->leftTag == 0 && T->rightTag == 0){

reverseThread(T->leftChild);

reverseThread(T->rightChild);

}

}

}

//删除节点

void bTHRTree::deleteNode(Node2 T, char x) {

Node2 G = T;

if (G != NULL){

if (G->data > x){

pre = G;

deleteNode(G->leftChild,x);

} else if (G->data < x){

pre = G;

deleteNode(G->rightChild, x);

} else if (G->data == x){

if (pre->rightChild == G){

pre->rightChild = NULL;

pre->rightTag = 1;

} else if (pre->leftChild == G){

pre->leftChild = NULL;

pre->leftTag = 1;

}

free(G);

}

}

}

//插入节点

void bTHRTree::insertNode(Node2 P, Node2 R) {

if (P->leftChild != NULL && P->rightChild != NULL){

cout << "无法插入节点" << endl;

}

if (P->leftChild != NULL && P->rightChild == NULL){

P->rightChild = R;

R->parent = P;

}

if (P->leftChild == NULL && P->rightChild != NULL){

P->leftChild = R;

R->parent = P;

}

if (P->leftChild == NULL && P->rightChild == NULL){

P->leftChild = R;

R->parent = P;

}

}

//节点入栈

void bTHRTree::initStack(Node2 T, int type) {

switch (type) {

case 1:{

//中序

inThreading1(T, pre);

TinOrder(T, 2);

break;

}

case 2:{

//先序

inThreading2(T, pre);

PreOrder(T, 2);

break;

}

case 3:{

//后序

inThreading3(T, pre);

PostOrder(T, 2);

break;

}

}

}

//查找前驱

Node2 bTHRTree::searchPreNode(char x) {

while (stack.top()->data != x){

stack.pop();

}

stack.pop();

if (!stack.empty()){

return stack.top();

} else {

return NULL;

}

}

//节点入队

void bTHRTree::initQueue(Node2 T, int type) {

switch (type) {

case 1:{

//中序

inThreading1(T, pre);

TinOrder(T, 3);

break;

}

case 2:{

//先序

inThreading2(T, pre);

PreOrder(T, 3);

break;

}

case 3:{

//后序

inThreading3(T, pre);

PostOrder(T, 3);

break;

}

}

}

//查找后继

Node2 bTHRTree::searchNextNode(char x) {

while (queue.front()->data != x){

queue.pop();

}

queue.pop();

if (!queue.empty()){

return queue.front();

} else {

return NULL;

}

}

## main.cpp

#include "bTHRTree.h"

void menu(){

cout << "----------线索化二叉树----------" << endl;

cout << "- 中序线索化二叉树 -" << endl;

cout << "- 先序线索化二叉树 -" << endl;

cout << "- 后序线索化二叉树 -" << endl;

cout << "- 插入节点 -" << endl;

cout << "- 删除节点 -" << endl;

cout << "- 查找前驱 -" << endl;

cout << "- 查找后继 -" << endl;

cout << "-----------------------------" << endl;

cout << "请输入操作：" << endl;

}

int main(){

bTHRTree b1;

int i;

b1.createTree(b1.T, b1.parent);

menu();

cin >> i;

switch (i) {

case 1:{

//中序线索二叉树

b1.inThreading1(b1.T, b1.pre);

b1.TinOrder(b1.T, 1);

break;

}

case 2:{

//先序线索二叉树

b1.inThreading2(b1.T, b1.pre);

b1.PreOrder(b1.T, 1);

break;

}

case 3:{

//后序线索二叉树

b1.inThreading3(b1.T, b1.pre);

b1.PostOrder(b1.T, 1);

break;

}

case 4:{

//插入节点

cout << "请选择线索化类型：";

int temp;

cin >> temp;

switch (temp) {

case 1:

//中序

b1.inThreading1(b1.T, b1.pre);

break;

case 2:

//先序

b1.inThreading2(b1.T, b1.pre);

break;

case 3:

//后序

b1.inThreading3(b1.T, b1.pre);

break;

}

Node2 R = new bTHRNode();

cout << "请输入节点值：";

cin >> R->data;

b1.reverseThread(b1.T);

b1.insertNode(b1.T->leftChild->leftChild, R);

b1.pre = NULL;

switch (temp) {

case 1:

//中序

b1.inThreading1(b1.T, b1.pre);

b1.TinOrder(b1.T, 1);

break;

case 2:

//先序

b1.inThreading2(b1.T, b1.pre);

b1.PreOrder(b1.T, 1);

break;

case 3:

//后序

b1.inThreading3(b1.T, b1.pre);

b1.PostOrder(b1.T, 1);

break;

}

break;

}

case 5:{

//删除节点

cout << "请选择线索化类型：";

int temp;

cin >> temp;

switch (temp) {

case 1:

//中序

b1.inThreading1(b1.T, b1.pre);

break;

case 2:

//先序

b1.inThreading2(b1.T, b1.pre);

break;

case 3:

//后序

b1.inThreading3(b1.T, b1.pre);

break;

}

b1.reverseThread(b1.T);

cout << "请输入要删除的节点：";

char s;

cin >> s;

b1.deleteNode(b1.T, s);

b1.pre = NULL;

switch (temp) {

case 1:

//中序

b1.inThreading1(b1.T, b1.pre);

b1.TinOrder(b1.T, 1);

break;

case 2:

//先序

b1.inThreading2(b1.T, b1.pre);

b1.PreOrder(b1.T, 1);

break;

case 3:

//后序

b1.inThreading3(b1.T, b1.pre);

b1.PostOrder(b1.T, 1);

break;

}

break;

}

case 6:{

//查找前驱

int s;

cout << "请输入线索化类型：";

cin >> s;

b1.initStack(b1.T, s);

cout << "请输入查找节点值：";

char f;

cin >> f;

Node2 x = b1.searchPreNode(f);

if (x){

cout << "该节点前驱为：" << x->data << endl;

} else {

cout << "该节点无前驱！";

}

break;

}

case 7:{

//查找后继

int s;

cout << "请输入线索化类型：";

cin >> s;

b1.initQueue(b1.T, s);

cout << "请输入查找节点值：";

char f;

cin >> f;

Node2 x = b1.searchNextNode(f);

if (x){

cout << "该节点后继为：" << x->data << endl;

} else {

cout << "该节点无前驱！";

}

break;

}

default:

break;

}

}