|  |  |
| --- | --- |
| 小组编号 | 6 |

|  |  |
| --- | --- |
| 成绩 |  |

华中师范大学计算机科学系

实 验 报 告 书

实验题目： 二叉树

课程名称： 数据结构实验

主讲教师： 刘巍

小组成员姓名：刘子鹏，罗楠，逯文叙，

周锦，Atayeva Gulshirin

实验时间： 2023/11/20

1. 实验目的：
2. 掌握二叉树的基本操作；
3. 掌握层次遍历和中序遍历；
4. 掌握形象化输出二叉树。
5. 实验内容：

这个算法通过构建一个二叉树的数据结构，然后对该二叉树进行不同方式的遍历（层次遍历和中序遍历），并输出二叉树的深度。

1. 定义了一个TreeNode类，表示二叉树的节点。每个节点包含一个值、左子节点和右子节点；
2. 创建十个节点，并按照完全二叉树的性质连接节点，返回根节点。
3. 实现二叉树的层次遍历。通过队列按层次遍历节点，将节点值存储在结果列表中。
4. 中序遍历：实现二叉树的中序遍历。通过递归方式按左、根、右的顺序遍历节点，将节点值存储在结果列表中。
5. 深度计算：计算二叉树的深度。通过递归方式返回左右子树深度的最大值加1。
6. 形象化输出二叉树：利用递归的方式，从根节点开始，按照二叉树的结构逐层打印节点，并通过合适的字符绘制树的形状，使得二叉树的结构清晰可见。
7. 实验环境：

Dev C++/Visual Studio

1. 问题描述

建立一个含有十个节点的二叉树，并以直观的形式输出该二叉树，同时要求对该树进行层次遍历和中序遍历，并输出该树的深度。

五、主程序流程和函数说明

#include "BiTreeFun.cpp"

int depth;

char ch;

vector<TElemType> s;

vector<TElemType> res;

int main(){//主函数

BiTree T;

InitBiTree(T);

CreateBiTree(T, s); //ABDH##I##EJ###CF##G##

//AB#DE###C##

Traverse(T);

printf("二叉树的深度：\n");

cout<<BiTreeDepth(T)<<endl;

printf("二叉树的中序遍历：\n");

preTraversal(T, res);

show(res);

printf("二叉树的层次遍历：\n");

res.clear();

levelTraversal(T, res);

show(res);

return 0;

}

//中序遍历函数接口

Status inTraversal(BiTree T){

if(T!=NULL){

inTraversal(T->lchild);

cout<<T->data<<" ";

inTraversal(T->rchild);

}

return 1;

}

cout<<res[i]<<" ";

cout<<endl;

}

//改变光标位置

void gotoxy(int x, int y)

{

// 更新光标位置

COORD pos;

HANDLE hOutput = GetStdHandle(STD\_OUTPUT\_HANDLE);

pos.X = x;

pos.Y = y;

SetConsoleCursorPosition(hOutput, pos);

}

//将树分为左子树和右子树

Status breakTree(BiTree& T, BiTNode\*& L, BiTNode\*& R){

if(T==NULL) return 0;

L=T->lchild;

R=T->rchild;

return 1;

}

BiTNode\* L,R;

/\*

\* 递归打印打印出树形

\* T 正在打印的树

\* depth 目前在打印树的第几层

\* right 该子树是否为右子树

\* tap 目前子树需要的相对偏移数量

\*/

Status Traverse\_R(BiTree T, int depth, int right, int tap) {

if (T == NULL) return 1;

// 获取一次树的初始高度，用于计算相对偏移数量

static int treeDepth = BiTreeDepth(T);

// 记录当前光标位置，用于在递归中记录当前递归时树的位置

int x, y;

// 拆分树，将树的左右子树拆分开来

BiTree L, R;

breakTree(T, L, R);

// 计算父树的偏移量

int tap1 = tap\*pow(2, treeDepth-depth);

// 计算子树的偏移量

int tap2 = right\*(pow(2, treeDepth- depth));

// 计算半偏移量

int tap3 = pow(2, treeDepth-depth-1);

// 获取根的坐标

x = tap1+tap2+tap3;

y = depth\*2+1;

// 打印根的位置

gotoxy(x\*2, y);

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

// 在打印子树时，当前层数+1

depth++;

// 计算子树的父偏移量

tap=tap\*2+(right==1?2:0);

if (L==NULL&&R==NULL) return 1;

else if (R==NULL) {

// 打印左子树的位置

gotoxy(x\*2-tap3,y+1);

printf("┏");

for (int i=0;i<tap3-1; i++) printf("━");

printf("┛");

Traverse\_R(L,depth,0,tap);

} else if (L==NULL) {

// 打印右子树的位置

gotoxy(x\*2, y+1);

printf("┗");

for (int i = 0;i<tap3-1; i++) printf("━");

printf("┓");

Traverse\_R(R,depth,1,tap);

} else {

// 打印左右子树的位置

gotoxy(x\*2-tap3, y+1);

printf("┏");

for (int i = 0; i<tap3-1; i++) printf("━");

printf("┻");

for (int i = 0; i<tap3-1; i++) printf("━");

printf("┓");

Traverse\_R(L, depth, 0, tap);

Traverse\_R(R, depth, 1, tap);

}

}

// 打印树形接口

Status Traverse(BiTree T) {

Traverse\_R(T, 0, 0, 0);

int x=0;int y=BiTreeDepth(T)\*2+1;

gotoxy(x,y);

return 1;

}

//层次遍历函数接口

Status levelTraversal(BiTree T, vector<TElemType>& res){

queue<BiTNode\*> q;

if(T==NULL) return 0;

q.push(T);

while(!q.empty()){

BiTNode\* p=q.front();q.pop();

res.push\_back(p->data);

if(p->lchild!=NULL) q.push(p->lchild);

if(p->rchild!=NULL) q.push(p->rchild);

}

return 1;

}

//遍历树元素接口

void show(vector<TElemType> res)

//改变光标位置接口

void gotoxy(int x, int y)

//将树分为左子树和右子树接口

Status breakTree(BiTree& T, BiTNode\*& L, BiTNode\*& R)

/\*

\* 递归打印打印出树形接口

\* T 正在打印的树

\* depth 目前在打印树的第几层

\* right 该子树是否为右子树

\* tap 目前子树需要的相对偏移数量

\*/

Status Traverse\_R(BiTree T, int depth, int right, int tap) {

if (T == NULL) return 1;

// 获取一次树的初始高度，用于计算相对偏移数量

static int treeDepth = BiTreeDepth(T);

// 记录当前光标位置，用于在递归中记录当前递归时树的位置

int x, y;

// 拆分树，将树的左右子树拆分开来

BiTree L, R;

breakTree(T, L, R);

// 计算父树的偏移量

int tap1 = tap\*pow(2, treeDepth-depth);

// 计算子树的偏移量

int tap2 = right\*(pow(2, treeDepth- depth));

// 计算半偏移量

int tap3 = pow(2, treeDepth-depth-1);

// 获取根的坐标

x = tap1+tap2+tap3;

y = depth\*2+1;

// 打印根的位置

gotoxy(x\*2, y);

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

// 在打印子树时，当前层数+1

depth++;

// 计算子树的父偏移量

tap=tap\*2+(right==1?2:0);

if (L==NULL&&R==NULL) return 1;

else if (R==NULL) {

// 打印左子树的位置

gotoxy(x\*2-tap3,y+1);

printf("┏");

for (int i=0;i<tap3-1; i++) printf("━");

printf("┛");

Traverse\_R(L,depth,0,tap);

} else if (L==NULL) {

// 打印右子树的位置

gotoxy(x\*2, y+1);

printf("┗");

for (int i = 0;i<tap3-1; i++) printf("━");

printf("┓");

Traverse\_R(R,depth,1,tap);

} else {

// 打印左右子树的位置

gotoxy(x\*2-tap3, y+1);

printf("┏");

for (int i = 0; i<tap3-1; i++) printf("━");

printf("┻");

for (int i = 0; i<tap3-1; i++) printf("━");

printf("┓");

Traverse\_R(L, depth, 0, tap);

Traverse\_R(R, depth, 1, tap);

}

}

// 打印树形接口

Status Traverse(BiTree T) {

Traverse\_R(T, 0, 0, 0);

int x=0;int y=BiTreeDepth(T)\*2+1;

gotoxy(x,y);

return 1;

}

六、实验调试、测试样例与结果分析（问题的发现、分析、解决方案与创新）

测试：

测试目的：形象化输出二叉树

测试过程：ABDH##I##EJ###CF##G##

A

┏━━━━━━━┻━━━━━━━┓

B C

┏━━━┻━━━┓ ┏━━━┻━━━┓

D E F G

┏━┻━┓ ┏━┛

H I J

测试目的：求二叉树的深度；

二叉树的深度：4

测试结论：二叉树的深度正常实现。

测试目的：求二叉树的中序遍历；

二叉树的中序遍历：

A B D H I E J C F G

测试目的：求二叉树的层次遍历；

测试结论：二叉树的中序遍历正常实现。

测试目的：二叉树的层次遍历：

二叉树的层次遍历：

A B C D E F G H I J

测试目的：二叉树的层次遍历。

测试结论：二叉树的层次遍历正常实现。

七、小组成员任务分配

刘子鹏：负责建立含有十个节点的二叉树的数据结构，确保二叉树的结构正确。

罗楠：实现二叉树的层次遍历算法。编写代码，确保能够按照层次遍历的顺序输出二叉树的节点。

逯文叙：实现二叉树的中序遍历算法。编写代码，确保能够按照中序遍历的顺序输出二叉树的节点。

周锦：在控制台或图形界面中以直观的形式输出该二叉树的树形结构。确保树的形状清晰可见。

Atayeva Gulshirin：计算二叉树的深度。编写代码，确保能够输出该二叉树的深度信息。

在任务进行的过程中，我们将保持密切的团队协作和沟通。每位成员应定期分享进展和遇到的问题，以便及时解决。合理分配任务的同时，鼓励成员们进行交流，共同解决可能出现的挑战。

通过分工合作，我们将能够构建、遍历和可视化一个完整的二叉树。每个人的贡献都是不可或缺的，而最终的成果将是我们团队协作的结晶。这个项目将为我们提供丰富的经

验，加深对二叉树数据结构和算法的理解，并提高团队合作的能力。

八、实验改进意见与建议

九、附录与说明

七、小组成员任务分配

刘子鹏：负责建立含有十个节点的二叉树的数据结构，确保二叉树的结构正确。

罗楠：实现二叉树的层次遍历算法。编写代码，确保能够按照层次遍历的顺序输出二叉树的节点。

逯文叙：实现二叉树的中序遍历算法。编写代码，确保能够按照中序遍历的顺序输出二叉树的节点。

周锦：在控制台或图形界面中以直观的形式输出该二叉树的树形结构。确保树的形状清晰可见。

Atayeva Gulshirin：计算二叉树的深度。编写代码，确保能够输出该二叉树的深度信息。

在任务进行的过程中，我们将保持密切的团队协作和沟通。每位成员应定期分享进展和遇到的问题，以便及时解决。合理分配任务的同时，鼓励成员们进行交流，共同解决可能出现的挑战。

通过分工合作，我们将能够构建、遍历和可视化一个完整的二叉树。每个人的贡献都是不可或缺的，而最终的成果将是我们团队协作的结晶。这个项目将为我们提供丰富的经

验，加深对二叉树数据结构和算法的理解，并提高团队合作的能力。

八、实验改进意见与建议

无

九、附录与说明

无