|  |  |  |  |
| --- | --- | --- | --- |
| **《数据结构与算法》实验报告** | | | |
| **学生姓名** | 郭茁宁 | **院（系）** | 计算机科学与技术 |
| **学 号** | 1183710109 | **专 业** | 软件工程 |
| **实验时间** | 2019年11月22日（周五） | **实验地点** | 格物213室 |
| **实验项目** | **实验2/5：树型结构的建立与遍历（3学时）** | | |
| **实验目的：**将课程的基本原理、技术和方法与实际应用相结合，训练和提高学生组织、存储和处理信息的能力，以及复杂问题的数据结构设计能力和程序设计能力，培养软件设计与开发所需要的实践能力。  **实验要求：**灵活运用基本的数据结构和算法知识，对实际问题进行分析和抽象；结合程序设计的一般过程和方法为实际问题设计数据结构和有效算法；用高级语言对数据结构和算法进行编程实现、调试，测试其正确性和有效性。 | | | |
| **实验内容：**  **树型结构的遍历是树型结构算法的基础，本实验要求编写程序演示二叉树的存储结构的建立方法和遍历过程。**   1. **编写建立二叉树的二叉链表存储结构（左右链表示）的程序，并以适当的形式显示和保存二叉树；** 2. **采用二叉树的二叉链表存储结构，编写程序实现二叉树的先序、中序和后序遍历的递归和非递归算法以及层序遍历算法，并以适当的形式显示和保存二叉树及其相应的遍历序列；** 3. **给定一个二叉树，** 编写算法完成下列应用:（二选一）    1. 判断其是否为完全二叉树；    2. 求二叉树中任意两个结点的公共祖先。 | | | |
| 数据结构定义：  Class Tree{init, add}; 树的节点  Class node{}; 链表节点（继承Tree）  Class Queue{empty, push, pop, front, back}; 队列（继承node）  Class Stack{push, pop, get\_top}; 栈（继承node） | | | |
| 算法设计与分析（要求画出核心内容的程序流程图）：  首先用class封装数据结构（面向对象）：   1. 树的节点Tree：节点值val，儿子节点指针son[0], son[1], 父亲节点指针fa；初始化指针指向空，建立节点后用成员函数init赋值val，用成员函数add连接父子节点，leaf判断是否是叶子节点。 2. 链表节点node：继承Tree树的节点，有Tree \*data和node \*next。 3. 队列Queue：继承node类，支持调用Tree成员。头指针head，尾指针tail，成员函数empty判断队列是否为空，push在队头插入节点，pop弹出队尾节点，front查询队头节点，back查询队尾节点。总体与链表维护相似。 4. 栈Stack：继承node类，支持调用Tree成员。栈顶指针top，push在栈顶插入节点，pop弹出栈顶节点，get\_top查询栈顶节点。   在依次实现遍历功能：   1. 层序遍历：通过队列这一数据结构，每次拓展访问节点的子节点；因为同一层情况下，左子树的节点被访问必定先于右子树，不同层的情况下深度更深的节点一定晚于浅的被访问，所以依据队列“先进先出，后进后出”的特点可以达到层序遍历的的效果。下图是遍历顺序和队列情况：      1. 递归先序、中序、后序遍历：这三种遍历要求在任意一棵子树上都要按照遍历顺序进行遍历，在每一个节点的操作都是一样的，即按照顺序深搜和输出，可以直观想到用递归的方法。以先序遍历为例：      1. 非递归先序、中序遍历：不断搜索左子树整棵子树，直到搜索完后，退回某一节点，搜索右儿子，将右儿子当作新的一棵子树的根节点，依照前面的方法继续搜索。   void 非递归前序/中序遍历(Tree \*头节点)  {      声明栈 class Stack      默认为头节点      while (栈不空)      {          if (x不为空)          {  （先序输出）              x压栈;              搜索左儿子;          }          Else//子树为空          {  （中序输出）              搜索栈顶（前一个节点）右子树；          }      }  }   1. 非递归后序遍历：先搜索左子树，再搜索右子树，最后输出局部根节点。由于前面的方法，在访问右子树之前能访问根节点，而访问右子树之后不能返回访问根节点，所以不适用与后序遍历。后序遍历的输出顺序是“左🡪右🡪根”，可以逆向压栈，在弹栈时输出。   void 非递归后序遍历(Tree \*头节点)  {      头节点压栈;      while (栈不空)      {           if ((栈顶节点左右儿子均为空)  ||(上一个访问的节点是栈顶节点的父亲)))          {              弹栈输出;              保存此时节点;          }          else          {              右儿子进栈（弹栈输出，故逆向）;              左儿子进栈;          }      }  }   1. 判断是否是完全二叉树：完全二叉树的的特点是，每个结点只有两种情况：叶子节点或有两个儿子节点。而且在层序遍历中，叶子节点总是连续的，依据这个性质，可以通过“若层序遍历上一个节点为叶子节点，则下一个也应该是叶子节点”来判断是否是完全二叉树。      1. 寻找两点的最近公共祖先（LCA）：在加入边时，设置一个指向父亲的指针，两个点分别有两条通向根节点的路径，这个问题就变成“以两个点为起点的链表，找交叉点”。   char LCA(Tree \*head,Tree \*x,Tree \*y)  {      Tree \*a=x,\*b=y;      while (a!=b)      {          a=(a==head?y:a->fa);          b=(b==head?x:b->fa);      }      return a->val;  } | | | |
| 实验测试结果及结果分析：  样例：    结果：    level\_trave: A B C D E H G F N M  pre\_order\_recursion: A B C E D H F N G M  in\_order\_recursion: A E C B F H N D G M  post\_order\_recursion: E C F N H M G D B A  pre\_order\_non\_recur: A B C E D H F N G M  in\_order\_non\_recur: A E C B F H N D G M  post\_order\_non\_recur: E C F N H M G D B A  complete\_binary\_tree: false  lowest\_common\_ancestor: A | | | |
| 问题及解决方法：   1. 无上限节省空间！？不用数组！树、队列、栈全部使用链表！！！无一例外！！！ 2. 调用地址很麻烦？怎么办呢？面向对象！！！功能全部封装为成员函数！！！ 3. 构建类的栈和队列？继承再继承！！！node继承Tree，Queue和Stack继承node。 4. 派生类沿用成员？可以的！设为public，类有多态性，多重继承连起来写！！ 5. 封装类！！！！🡪🡪🡪🡪提高程序质量！核心代码可读性！大工程浓缩！面向对象省去指针！继承避免冗余！工业级看起来很高级！！！！！！！ | | | |
| 源程序名称：lab2.cpp | | | |

注意：正文文字为宋体小4号，图中文字为宋体5号。行距为多倍行距1.25。

源程序与此报告打包提交，压缩包采用学号命名。