山东大学 计算机科学与技术 学院

数据结构与算法 课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：201705130113 | 姓名： 黄瑞哲 | | 班级：计科17.3 |
| 实验题目：二叉树操作 | | | |
| 实验学时：2 | | 实验日期：2018.11.29 | |
| 实验目的：   1. 掌握二叉树的基本概念，链表描述方法；二叉树操作的实现 | | | |
| 软件环境：  Visual Studio Community 2017 | | | |
| 1. 实验内容（题目内容，输入要求，输出要求） 2. 创建二叉树类。二叉树的储存结构使用链表。提供操作：前序遍历、中序遍历、后序遍历、层次遍历、计算二叉树节点操作、计算二叉树高度。 3. 对建立好的二叉树，执行上述各操作，输出各操作的结果。 4. 接受键盘输入的二叉树前序序列和中序序列（各元素不相同），输出该二叉树的后序序列。 5. 数据结构与算法描述 （整体思路描述，所需要的数据结构与算法） 6. 链表节点结构体有4个属性：element元素，leftChild左子树，rightChild右子树，father父亲节点 7. 计算节点的方法为左子树节点个数加右子树节点个数再加1，复杂度为O(n) 8. 计算树高的方法为左子树高度与右子树高度最大值加1，复杂度为O(n) 9. 根据前序序列和中序序列建立二叉树的方法为：当前根节点为\*preOrder，从inOrder中找到\*preOrder的位置i，i左边为左子树，i右边为右子树然后递归建立。 10. 测试结果（测试输入，测试输出，结果分析）   树：  输出     1. 分析与探讨（结果分析，若存在问题，探讨解决问题的途径）   从树的结构图中可以看出这棵树有11个节点，高度为5，前序遍历为ABDGHEFCKFIJ，中序遍历为GDHBEAKCIJF，后序遍历为GHDEBKJIFCA，层次遍历为ABCDEFGHIJ。结果正确。  如果只根据前序序列和后序序列是无法确定二叉树的，因为没有中序序列，左右子树的关系未知。   1. 附录：实现源代码（本实验的全部源程序代码，程序风格清晰易理解，有充分的注释）   /\*binaryTreeNode.h\*/  #ifndef binaryTreeNode\_  #define binaryTreeNode\_  template<typename T>  struct binaryTreeNode {  T element;  binaryTreeNode<T>\* leftChild;//左子树  binaryTreeNode<T>\* rightChild;//右子树  binaryTreeNode<T>\* father;//父亲  binaryTreeNode(const T& theElement) : element(theElement) {  leftChild = rightChild = father = nullptr;  }  };  #endif //binaryTreeNode\_  /\*binaryTree.h\*/  #ifndef binaryTree\_  #define binaryTree\_  template<typename T>  class binaryTree {  public:  virtual ~binaryTree() {};  virtual bool empty() const = 0;  virtual int size() const = 0;  virtual void preOrder(void(\*) (T\*)) = 0;  virtual void inOrder(void(\*) (T\*)) = 0;  virtual void postOrder(void(\*) (T\*)) = 0;  virtual void levelOrder(void(\*) (T\*)) = 0;  };  #endif //binaryTree\_  /\*linkedBinaryTree.h\*/  #ifndef linkedBinaryTree\_  #define linkedBinaryTree\_  #include "binaryTreeNode.h"  #include "binaryTree.h"  #include <queue/linkedQueue.h>  #include <iostream>  template<typename T>  class linkedBinaryTree : public binaryTree<binaryTreeNode<T> > {  public:  linkedBinaryTree() { root = nullptr; };  ~linkedBinaryTree() { erase(); };  bool empty() const { return size(root) == 0; };  int size() const { return size(root); };  //求树根  binaryTreeNode<T>\*& getRoot() {  return root;  }  //前序遍历  void preOrder(void(\*visit)(binaryTreeNode<T>\*) = linkedBinaryTree<T>::output) {  preOrder(root, visit);  };  //中序遍历  void inOrder(void(\*visit)(binaryTreeNode<T>\*) = linkedBinaryTree<T>::output) {  inOrder(root, visit);  };  //后序遍历  void postOrder(void(\*visit)(binaryTreeNode<T>\*) = linkedBinaryTree<T>::output) {  postOrder(root, visit);  };  //层次遍历  void levelOrder(void(\*visit)(binaryTreeNode<T>\*) = linkedBinaryTree<T>::output) {  levelOrder(root, visit);  };  //清空树  void erase() {  postOrder(dispose);  root = nullptr;  }  //建树  void makeTree(const T&, linkedBinaryTree<T>&, linkedBinaryTree&);  //求树高  int height() const { return height(root); };    protected:  binaryTreeNode<T>\* root;  static void preOrder(binaryTreeNode<T>\*, void(\*)(binaryTreeNode<T>\*));  static void inOrder(binaryTreeNode<T>\*, void(\*)(binaryTreeNode<T>\*));  static void postOrder(binaryTreeNode<T>\*, void(\*)(binaryTreeNode<T>\*));  static void levelOrder(binaryTreeNode<T>\*, void(\*)(binaryTreeNode<T>\*));  static void dispose(binaryTreeNode<T>\* t) { delete t; }  static int size(binaryTreeNode<T>\*);  static int height(binaryTreeNode<T>\*);  static void output(binaryTreeNode<T>\* t) { std::cout << t->element << ' '; }  };  template<typename T>  void linkedBinaryTree<T>::preOrder(binaryTreeNode<T>\* t, void(\*visit)(binaryTreeNode<T>\*)) {  if (t == nullptr) return;  visit(t);  preOrder(t->leftChild, visit);  preOrder(t->rightChild, visit);  }  template<typename T>  void linkedBinaryTree<T>::inOrder(binaryTreeNode<T>\* t, void(\*visit)(binaryTreeNode<T>\*)) {  if (t == nullptr) return;  inOrder(t->leftChild, visit);  visit(t);  inOrder(t->rightChild, visit);  }  template<typename T>  void linkedBinaryTree<T>::postOrder(binaryTreeNode<T>\* t, void(\*visit)(binaryTreeNode<T>\*)) {  if (t == nullptr) return;  postOrder(t->leftChild, visit);  postOrder(t->rightChild, visit);  visit(t);  }  template<typename T>  void linkedBinaryTree<T>::levelOrder(binaryTreeNode<T>\* t, void(\*visit)(binaryTreeNode<T>\*)) {  linkedQueue<binaryTreeNode<T>\* >q;  q.push(t);  while (!q.empty()) {  binaryTreeNode<T>\* p = q.front(); q.pop();  if (p == nullptr) continue;  visit(p);  q.push(p->leftChild);  q.push(p->rightChild);  }  }  template<typename T>  void linkedBinaryTree<T>::makeTree(const T& theElement, linkedBinaryTree<T>& left, linkedBinaryTree<T>& right) {  root = new binaryTreeNode<T>(theElement);  root->leftChild = left.root;  root->rightChild = right.root;  left.root->father = right.root->father = root;  left.root = right.root = nullptr;  }  template<typename T>  int linkedBinaryTree<T>::height(binaryTreeNode<T>\* t) {  if (t == nullptr) return 0;  int hl = height(t->leftChild);  int hr = height(t->rightChild);  //树高等于左右子树的最大高度加一  return (hl > hr ? hl : hr) + 1;  }  template<typename T>  int linkedBinaryTree<T>::size(binaryTreeNode<T>\* t) {  if (t == nullptr) return 0;  //节点个数等于左右子树节点个数之和再加一  return size(t->leftChild) + size(t->rightChild) + 1;  }  #endif linkedBinaryTree\_  /\*exe9.cpp\*/  // exe9.cpp : 此文件包含 "main" 函数。程序执行将在此处开始并结束。  //  #include "pch.h"  #include <iostream>  template<typename T>  void build(const T preOrder[], const T inOrder[], int n, binaryTreeNode<T>\*& t) {  if (n < 1)  {  t = nullptr;  return;  }  int i;  for (i = 0; i < n&&inOrder[i] != \*preOrder; ++i);//当前的根节点是\*preOrder,在中序遍历中找到\*preOrder的位置i  delete t;  t = new binaryTreeNode<T>(\*preOrder);//建立根节点  //inOrder[i]的左边是\*preOrder的左子树，递归建立左子树  build(preOrder + 1, inOrder, i, t->leftChild);  //inOrder[i]的右边是\*preOrder的右子树，递归建立右子树  build(preOrder + i + 1, inOrder + i + 1, n - i - 1, t->rightChild);  if (t->leftChild != nullptr) t->leftChild->father = t;  if (t->rightChild != nullptr) t->rightChild->father = t;  //这里可以输出后序遍历  //cout << \*preOrder << ' ';  }  int main()  {  char pre[] = "abdgheckfij";  char in[] = "gdhbeakcijf";  linkedBinaryTree<char> t;  build(pre, in, 5, t.getRoot());  cout << "树的结点个数: " << t.size() << endl;  cout << "树的高度: " << t.height() << endl;  cout << "前序遍历: ";  t.preOrder();  cout << endl;  cout << "中序遍历: ";  t.inOrder();  cout << endl;  cout << "后序遍历: ";  t.postOrder();  cout << endl;  cout << "层次遍历: ";  t.levelOrder();  cout << endl;  return 0;  } | | | |