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：201805130155 | 姓名： 赵雨晗 | | 班级： 18计科3班 |
| 实验题目：二叉树操作 | | | |
| 实验学时：2 | | 实验日期： 2019.11.13 | |
| 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 | | | |
| 软件开发工具：  MAC OS: Vs Code + MingW | | | |
| 1. 实验内容（题目内容，输入要求，输出要求）   （1） 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。  （2） 对建立好的二叉树，执行上述各操作，输出各操作的结果。  （3） 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。   1. 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）   （1） 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  （2） 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)  （3） 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)  （4） 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。   1. 测试结果（测试输入，测试输出，结果分析）   输入  5  1 2 3 4 5  4 2 5 1 3  输出  4 5 2 3 1    结果：与手动模拟结果一致，符合题目要求。   1. 分析与探讨（结果分析，若存在问题，探讨解决问题的途径）   从树的结构可以看出这个树有五个结点，前序遍历为1 2 4 5 3，中序遍历为4 2 5 1 3，后序遍历为4 5 2 3 1，答案输出正确   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）   #include<cstdio>  #include<iostream>  #include<cstdlib>  #include<stdexcept>  #include<queue>  #include<functional>  using namespace std;  template<typename T>  struct binaryTreeNode{  T element;  int height, size;  binaryTreeNode<T>\* leftChild;  binaryTreeNode<T>\* rightChild;  binaryTreeNode<T>\* father;  binaryTreeNode() {height = 0, size = 0;}  binaryTreeNode(const T& theElement) : element(theElement) {  leftChild = rightChild = father = nullptr;  }  bool operator<(const binaryTreeNode<T>& x) const {  return element < x.element;  }  };  template<typename T>  class linkedBinaryTree{  public:  linkedBinaryTree(binaryTreeNode<T>\* node = nullptr) {root = node;}  virtual ~linkedBinaryTree() {}  bool operator <(const linkedBinaryTree<T>& x) const{  return root ->element < x.root->element;  }    void size() const {  visit = linkedBinaryTree<T>::size;  postOrder(root);  }  //求树根  binaryTreeNode<T>\*& getRoot(){  return root;  }  //先序遍历  void preOrder(void(\*\_visit)(binaryTreeNode<T>\*) = linkedBinaryTree<T>::output) {  visit = \_visit;  preOrder(root);  }  //中序遍历  void inOrder(void(\*\_visit)(binaryTreeNode<T>\*) = linkedBinaryTree<T>::output) {  visit = \_visit;  inOrder(root);  }  //后序遍历  void postOrder(void(\*\_visit)(binaryTreeNode<T>\*) = linkedBinaryTree<T>::output) {  visit = \_visit;  postOrder(root);  }  //层次遍历  void levelOrder(void(\*\_visit)(binaryTreeNode<T>\*) = linkedBinaryTree<T>::output) {  visit = \_visit;  levelOrder(root);  }  //清空树  void erase(){  postOrder(linkedBinaryTree<T>::dispose);  root = nullptr;  }  //求树高  void height() const {  visit = linkedBinaryTree<T>::height;  postOrder(root);  }    protected:  int tmp;  int tmpArray[100010];  T \*tmp1;  binaryTreeNode<T> \*root;  static void (\*visit)(binaryTreeNode<T>\*);  static void preOrder(binaryTreeNode<T>\*);  static void inOrder(binaryTreeNode<T>\*);  static void postOrder(binaryTreeNode<T>\*);  static void levelOrder(binaryTreeNode<T>\*);  static void dispose(binaryTreeNode<T>\* t) {delete t;}  static void size(binaryTreeNode<T>\*);  static void height(binaryTreeNode<T>\*);  static void output(binaryTreeNode<T>\* t) {cout << t -> element << ' ';}  };  template<typename T>  void (\*linkedBinaryTree<T>::visit)(binaryTreeNode<T>\*) = linkedBinaryTree<T>::output;  template<typename T>  void linkedBinaryTree<T>::preOrder(binaryTreeNode<T>\* t){  if (t == nullptr) return;  linkedBinaryTree<T>::visit(t);  preOrder(t ->leftChild);  preOrder(t ->rightChild);  }  template<typename T>  void linkedBinaryTree<T>::inOrder(binaryTreeNode<T>\* t){  if (t == nullptr) return;  inOrder(t ->leftChild);  linkedBinaryTree<T>::visit(t);  inOrder(t ->rightChild);  }  template<typename T>  void linkedBinaryTree<T>::postOrder(binaryTreeNode<T>\* t) {  if (t == nullptr) return;  postOrder(t ->leftChild);  postOrder(t ->rightChild);  linkedBinaryTree<T>::visit(t);  }  template<typename T>  void linkedBinaryTree<T>::levelOrder(binaryTreeNode<T>\* t) {  queue<binaryTreeNode<T>\* >q;  q.push(t);  while(!q.empty()){  binaryTreeNode<T>\* p = q.front(); q.pop();  if (p == nullptr) continue;  linkedBinaryTree<T>::visit(p);  q.push(p ->leftChild);  q.push(p ->rightChild);  }  }  template<typename T>  void linkedBinaryTree<T>::height(binaryTreeNode<T>\* t){  t ->height = 1;  if (t->leftChild) t->height = max(t->height, t->leftChild->height + 1);  if (t->rightChild) t->height = max(t->height, t->rightChild->height + 1);  }  template<typename T>  void linkedBinaryTree<T>::size(binaryTreeNode<T>\* t){  t ->size = 1;  if (t ->leftChild) t ->size += t ->leftChild ->size;  if (t ->rightChild) t ->size += t ->rightChild ->size;  }  //P281.33 时间复杂度为O(N)  binaryTreeNode<int> \*tmp;  binaryTreeNode<int>\* makeTree(int \*preOrderArray, int \*inOrderArray, int sz){  if (sz == 0) return nullptr;  int pos = 0;  tmp = new binaryTreeNode<int>(preOrderArray[0]);  binaryTreeNode<int> \*node = tmp;  if (sz == 1) {return node;}  while(pos < sz && preOrderArray[0] != inOrderArray[pos]) pos++;  node ->leftChild = makeTree(preOrderArray + 1, inOrderArray, pos);  node ->rightChild = makeTree(preOrderArray + pos + 1, inOrderArray + pos + 1, sz - pos - 1);  return node;  }  binaryTreeNode<int>\* tr;  inline void solve(){  int n;  scanf("%d", &n);  int \*preO = new int[n + 2];  int \*inO = new int[n + 2];  for (int i = 1; i <= n; ++i) scanf("%d", &preO[i]);  for (int i = 1; i <= n; ++i) scanf("%d", &inO[i]);  linkedBinaryTree<int> Tree(makeTree(preO + 1, inO + 1, n));  Tree.postOrder();  }  int main(){  solve();  system("pause");  } | | | |