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：202000130138 | 姓名：宋璎航 | | 班级： 20.3 |
| 实验题目：二叉树操作 | | | |
| 实验学时：2 | | 实验日期： 2021/11/25 | |
| 实验目的：  掌握二叉树的基本概念，链表描述方法；二叉树操作的实现。 | | | |
| 软件开发环境：  Vsc | | | |
| 1. 实验内容   **1、题目描述**：  创建二叉树类。二叉树的存储结构使用链表。提供操作:前序遍历、中序遍历、后序遍历、层次遍历、计算二叉树结点数目、计算二叉树高度。  **输入输出格式**：  **输入**：  第一行为一个数字n (10<=n<=100000)，表示有这棵树有n个节点，编号为1~n。之后n行每行两个数字，第 i 行的两个数字a、b表示编号为 i 的节点的左孩子节点为 a，右孩子节点为 b，-1表示该位置没有节点。保证数据有效，根节点为1。  **输出**：  第一行，n个数字，表示该树的层次遍历。  第二行，n个数字，第i个数字表示以 i 节点为根的子树的节点数目。  第三行，n个数字，第i个数字表示以 i 节点为根的子树的高度。  **2、题目描述**：  接收二叉树前序序列和中序序列(各元素各不相同)，输出该二叉树的后序序列。  **输入输出格式**：  **输入**：  第一行为数字n；  第二行有n个数字，表示二叉树的前序遍历；  第三行有n个数字，表示二叉树的中序遍历。  **输出**：  输出一行，表示该二叉树的后序遍历序列。   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）   建立二叉树类，建立指向每个节点的指针的数组，在初始化操作时直接对此数组进行初始化即可。  前序与中序序列转后序即利用递归，每次将中序序列切割成左右子树分别建树即可。   1. 测试结果（测试输入，测试输出）  输入 5  2 3  4 5  -1 -1  -1 -1  -1 -1  **输出**  1 2 3 4 5  5 3 1 1 1  3 2 1 1 1 输入 5  1 2 4 5 3  4 2 5 1 3 输出 4 5 2 3 1   1. 分析与探讨（结果分析，若存在问题，探讨解决问题的途径）   均正确   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 "bits/stdc++.h"  template <class T>  struct binaryTreeNode  {  T element;  binaryTreeNode<T> \*leftChild, \*rightChild;  int No;  binaryTreeNode() { leftChild = rightChild = NULL; }  binaryTreeNode(const T &theElement)  {  element = theElement;  leftChild = rightChild = NULL;  }  binaryTreeNode(const T &theElement, const binaryTreeNode \*theleftChild, const binaryTreeNode \*therightChild)  {  element = theElement;  leftChild = theleftChild;  rightChild = therightChild;  }  };  template <class T>  class linkedBinaryTree  {  binaryTreeNode<T> \*root;  binaryTreeNode<T> \*\*pp; //指向第i个结点的指针  int tot;  int \*treeSize;  int \*treeheight;  public:  linkedBinaryTree(int n)  {  pp = new binaryTreeNode<T> \*[n + 1];  treeSize = new int[n + 1];  treeheight = new int[n + 1];  tot = n;  for (int i = 1; i <= n; ++i)  {  pp[i] = new binaryTreeNode<T>(i);  treeSize[i] = -1;  treeheight[i] = -1;  }  root = pp[1];  }  void input()  {  for (int i = 1; i <= tot; ++i)  {  int a, b;  std::cin >> a >> b;  if (a == -1)  pp[i]->leftChild = NULL;  else  pp[i]->leftChild = pp[a];  if (b == -1)  pp[i]->rightChild = NULL;  else  pp[i]->rightChild = pp[b];  }  }  void levelOrder()  {  std::queue<binaryTreeNode<T> \*> q;  binaryTreeNode<T> \*t = root;  while (t != NULL)  {  std::cout << t->element << ' ';  if (t->leftChild != NULL)  q.push(t->leftChild);  if (t->rightChild != NULL)  q.push(t->rightChild);  if (q.empty())  {  std::cout << std::endl;  return;  }  t = q.front();  q.pop();  }  std::cout << std::endl;  }  int heightcalcu(int i)  {  int lh = 0, rh = 0;  if (pp[i]->leftChild != NULL)  lh = heightcalcu(pp[i]->leftChild->element);  if (pp[i]->rightChild != NULL)  rh = heightcalcu(pp[i]->rightChild->element);  treeheight[i] = std::max(lh, rh) + 1;  return treeheight[i];  }  int numcalcu(int i)  {  int lh = 0, rh = 0;  if (pp[i]->leftChild != NULL)  lh = numcalcu(pp[i]->leftChild->element);  if (pp[i]->rightChild != NULL)  rh = numcalcu(pp[i]->rightChild->element);  treeSize[i] = lh + rh + 1;  return treeSize[i];  }  void output()  {  for (int i = 1; i <= tot; ++i)  std::cout << treeSize[i] << ' ';  std::cout << '\n';  for (int i = 1; i <= tot; ++i)  std::cout << treeheight[i] << ' ';  std::cout << '\n';  }  };  int main()  {  int n;  std::cin >> n;  linkedBinaryTree<int> B(n);  B.input();  B.levelOrder();  B.heightcalcu(1);  B.numcalcu(1);  B.output();  }  #include "bits/stdc++.h"  template <class T>  struct binaryTreeNode  {  T element;  binaryTreeNode<T> \*leftChild, \*rightChild;  int No;  binaryTreeNode() { leftChild = rightChild = NULL; }  binaryTreeNode(const T &theElement)  {  element = theElement;  leftChild = rightChild = NULL;  }  binaryTreeNode(const T &theElement, const binaryTreeNode \*theleftChild, const binaryTreeNode \*therightChild)  {  element = theElement;  leftChild = theleftChild;  rightChild = therightChild;  }  };  binaryTreeNode<int> \*f(int \*pre, int \*in, int length)  {  if (length == 0)  return NULL;  binaryTreeNode<int> \*node = new binaryTreeNode<int>(\*pre);  int i;  for (i = 0; i < length; ++i)  {  if (in[i] == \*pre)  break;  }  node->leftChild = f(pre + 1, in, i);  node->rightChild = f(pre + i + 1, in + i + 1, length - i - 1);  std::cout << node->element << ' ';  return node;  }  int main()  {  int n;  int \*pre, \*in;  std::cin >> n;  pre = new int[n];  in = new int[n];  for (int i = 0; i < n; ++i)  std::cin >> pre[i];  for (int i = 0; i < n; ++i)  std::cin >> in[i];  f(pre, in, n);  } | | | |