山东大学 软件 学院

数据结构 课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号： | 姓名： | | 班级： |
| 实验题目：堆和搜索树 | | | |
| 实验学时：4小时 | | 实验日期：2018.11.20 | |
| 实验目的：  掌握堆和搜索树的基本概念，插入、删除方法。 | | | |
| 硬件环境： 笔记本电脑 | | | |
| 软件环境：Win10+Vistual Studio 2017 | | | |
| 实验步骤与内容：  实验内容：  1、输入一系列不为零的正整数（最多不超过20个），遇到0代表输入结束（不包含0）。  2、根据上面输入的数据序列，用初始化方法创建最大堆（不要用节点依次插入的办法创建最大堆），然后输出最大堆的层次序列。  3、输出用堆排序后的排序结果。  4、根据上面输入的数据，创建二叉搜索树，输出二叉搜索树的前序序列、中序序列（分行输出）。  实验步骤：  binaryTreeNode.h  #pragma once  #include <iostream>  struct binaryTreeNode  {  friend class bsTree;  public:  binaryTreeNode() { leftChild = rightChild = NULL; }  binaryTreeNode(const int& theElement)  {  element = theElement;  leftChild = rightChild = 0;  }  binaryTreeNode(const int& theElement, binaryTreeNode \*theleftChild, binaryTreeNode \*therightChild)  {  element = theElement;  leftChild = theleftChild;  rightChild = therightChild;  }  private:  int element;  binaryTreeNode \*leftChild; //左子树  binaryTreeNode \*rightChild; //右子树  };  bsTree.h  #pragma once  #include "binaryTreeNode.h"  class bsTree  {  public:  bsTree(int max) { root = NULL; }  int\* rootElement() const;  int size(); //返回树中的节点数  int getSize(binaryTreeNode \*p);  void insert(const int & x);  private:  int treeSize;  binaryTreeNode \*root;  };  maxHeap.h  #pragma once  class maxHeap  {  public:  maxHeap(int max);  int size() const { return heapSize; }  int max() { return heap[1]; }  maxHeap& insert(const int & x);  maxHeap& deleteMax(int & x);  void initialize(int \*theHeap, int theSize, int arraySize);  void levelOrder();  void heapSort();  private:  int heapSize, arrayLength;  int \*heap;  };  bsTree.cpp  #include <iostream>  #include "bsTree.h"  using namespace std;  int t = 0;  int\* bsTree::rootElement() const  {  //返回NULL，如果是空树。否则，返回根节点的element域  if (treeSize == 0)  return NULL; // no root  else  return &root->element;  }  int bsTree::size()  {  return getSize(root);  }  int bsTree::getSize(binaryTreeNode \*p)  {  if (p == NULL)  return 0;  else  return(getSize(p->leftChild) + 1 + getSize(p->rightChild));  }  void bsTree::insert(const int & thePair)  {  //插入thePair。如果存在与其关键字相同的数对，则覆盖  binaryTreeNode \*p = root;  binaryTreeNode \*pp = NULL; //pp为p的父节点  while (p != NULL)  {  //检查元素p->element  pp = p;  //p移动到它的一个孩子节点  if (thePair < p->element)  p = p->leftChild;  else  if (thePair > p->element)  p = p->rightChild;  else  {  //出现重复元素，覆盖旧值  p->element = thePair;  return;  }  }  //为thePair建立一个节点，然后与pp链接  binaryTreeNode \*newNode = new binaryTreeNode(thePair);  if (root != NULL)  {  //树不空  if (pp->element > thePair)  {  pp->leftChild = newNode;  }  else  {  pp->rightChild = newNode;  }  }  else  {  root = newNode;//插入空树  }  }  maxHeap.cpp  #include <iostream>  #include <queue>  #include "maxHeap.h"  using namespace std;  void maxHeap::heapSort()  {  int x;  int size = heapSize;  for (int i = heapSize; i > 1; i--)  {  deleteMax(x);  heap[i] = x;  }  for (int n = 1; n < size; n++)  {  cout << heap[n] << ",";  }  cout << heap[size] << endl;  }  maxHeap::maxHeap(int max)  {  arrayLength = max;  heap = new int[max + 1]; //这个数组用于存储最大堆，但要注意：该数组的0号位空置不用，即从1号位计起  heapSize = 0;  }  maxHeap& maxHeap::insert(const int &theElement)  {  if (heapSize == arrayLength - 1)  {  cout << "数组过小" << endl;  }  //为元素theElement寻找插入位置  //currentNode 从新叶向上移动  int currentNode = ++heapSize;  while (currentNode != 1 && theElement > heap[currentNode / 2])  {  //不能把元素theElement插入在heap[currentNode]  heap[currentNode] = heap[currentNode / 2];//把元素向下移动  currentNode /= 2; //currentNode移向双亲  }  heap[currentNode] = theElement;  return \*this;  }  maxHeap& maxHeap::deleteMax(int &x)  {  //取得当前堆中最后一个元素  x = heap[1];  //删除最后一个元素，然后重建堆  int lastElement = heap[heapSize--];  //从根开始，为最后一个元素寻找位置  int currentNode = 1;  int child = 2; //currentNode的孩子  while (child <= heapSize)  {  //heap[child]应该是currentNode的更大的孩子  if (child < heapSize && heap[child] < heap[child + 1])  {  child++;  }  //可以把lastElement放在heap[currentNode]吗？  if (lastElement >= heap[child])  break; //可以  //不可以  heap[currentNode] = heap[child]; //把孩子child向上移动  currentNode = child; //向下移动一层寻找位置  child \*= 2;  }  heap[currentNode] = lastElement;  return \*this;  }  void maxHeap::initialize(int \*theHeap, int theSize, int arraySize)  {  //在数组heap[1:theSize]中建立大根堆  delete[] heap;  heap = theHeap;  heapSize = theSize;  arrayLength = arraySize;  //堆化  for (int root = heapSize / 2; root >= 1; root--)  {  int rootElement = heap[root];  //为元素rootElement寻找位置  int child = 2 \* root;//孩子child的双亲是元素rootElement的位置  while (child <= heapSize)  {  //heap[child]应该是兄弟中的较大者  if (child < heapSize && heap[child] < heap[child + 1])  {  child++;  }  //可以把元素rootElement放在heap[child / 2]吗？  if (rootElement > heap[child])  break;//可以  //不可以  heap[child / 2] = heap[child]; //把孩子向上移  child \*= 2; //移到下一层  }  heap[child / 2] = rootElement;  }  }  void maxHeap::levelOrder()  {  int count = 0;  queue<int> q;  int p = 1;  q.push(1);  int leftChild, rightChild;  while (!q.empty())  {  count++;  p = q.front();  q.pop();  cout << heap[p];  if (count < heapSize)  cout << ",";  leftChild = p \* 2;  if (leftChild <= heapSize)  q.push(leftChild);  rightChild = leftChild + 1;  if (rightChild <= heapSize)  q.push(rightChild);  }  cout << endl;  }  源.cpp  #include <iostream>  #include <queue>  #include <string>  #include "binaryTreeNode.h"  #include "maxHeap.h"  #include "bsTree.h"  using namespace std;  int z = 0;  struct binarySearchTreeNode  {  binarySearchTreeNode(int x) { element = x; leftChild = rightChild = 0; }  int element;  binarySearchTreeNode \*leftChild;  binarySearchTreeNode \*rightChild;  };  void insert(binarySearchTreeNode \*root, const int & thePair)  {  //插入thePair。如果存在与其关键字相同的数对，则覆盖  binarySearchTreeNode \*p = root;  binarySearchTreeNode \*pp = NULL; //pp为p的父节点  while (p != NULL)  {  //检查元素p->element  pp = p;  //p移动到它的一个孩子节点  if (thePair < p->element)  p = p->leftChild;  else  if (thePair > p->element)  p = p->rightChild;  else  {  //出现重复元素，覆盖旧值  p->element = thePair;  return;  }  }  //为thePair建立一个节点，然后与pp链接  binarySearchTreeNode \*newNode = new binarySearchTreeNode(thePair);  if (root != NULL)  {  //树不空  if (pp->element > thePair)  {  pp->leftChild = newNode;  }  else  {  pp->rightChild = newNode;  }  }  else  {  root= newNode;//插入空树  }  }  void preOrder(binarySearchTreeNode \*root, int num)  {  if (root)  {  z++;  cout << root->element;  if (z < num)  cout << ",";  preOrder(root->leftChild, num);  preOrder(root->rightChild, num);  }  }  void inOrder(binarySearchTreeNode \*root, int num)  {  if (root)  {  inOrder(root->leftChild, num);  z++;  cout << root->element;  if (z < num)  cout << ",";  inOrder(root->rightChild, num);  }  }  int main()  {  cout << "Input" << endl;  maxHeap \*mh = new maxHeap(20);  bsTree \*bst = new bsTree(20);  int a[20];  int b[20];  int i;  cin >> i;  int count = 0;  for (; i != 0; cin >> i)  {  count++;  a[count] = i;  b[count] = i;  bst->insert(b[count]);  }    cout << "Output" << endl;  mh->initialize(a, count, 20);  mh->levelOrder();  mh->heapSort();  binarySearchTreeNode \*root = new binarySearchTreeNode(b[1]);  for (int n = 2; n <= count; n++)  {  insert(root, b[n]);  }  int y = bst->size();  preOrder(root,y);  cout << endl;  z = 0;  inOrder(root,y);  cout << endl;  z = 0;  cout << "End" << endl;  system("pause");  return 0;  } | | | |
| 结论分析与体会：  运行结果：    体会： | | | |