山东大学 软件 学院

数据结构 课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：202000300125 | 姓名：贾星宇 | | 班级：2020级5班 |
| 实验题目：堆和搜索树 | | | |
| 实验学时：4 | | 实验日期： 2021年11月23日星期二 | |
| 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：  lntel(R) Core(TM) i5-10210U CPU @ 1.60G Hz 2 .1 1 G Hz | | | |
| 软件环境：  CLion 2020.3.3 x64 | | | |
| 实验步骤与内容：  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（关键字不允许重复，如遇重复，则不重复插入该关键字），输出二叉搜索树的前序序列、中序序列（分行输出）。 | | | |
| 结论分析与体会：  1.初始化一个大根堆：  原来是一个数组形式的完全二叉树，长度为n，从第一个有叶节点的节点，也就是编号为n/2的结点开始，判断以此节点为根节点的子树是不是大根堆，如果不是，把它变成大根堆，方法： 当孩子索引值小于元素个数时，找到两个孩子的最大个，比较它与根节点的大小；如果大，则将孩子设置为根节点；同时去搜索孩子的孩子（孩子\*2），如果孩子的孩子大于孩子，则继续交换……如果遇到了小于根节点的元素，则break，同时把根节点放到小的元素的父节点上（此时父节点上的数已经上移）  2.弹出大根堆的最大元素，并且将其从大根堆删除，大根堆保持原有属性：  首先保存住大根堆的顶端元素，并将其删除；从大根堆最低端去一个元素并将大根堆的size--，将底端的元素插入大根堆内  插入方法：  类似于初始化中的一个循环，从顶端开始，如果最低端元素大于顶端两个子元素，则将该元素作为根节点；如果不是，则找到两个子元素的最大值作为根节点，再将空出的那个下面的两个子元素跟低端元素相比较，直到比较到地段元素变大或者到了最后一个；  如何找到下一个元素：大的元素索引值\*2； | | | |
|  | | | |

**本次实验主要代码：**

#include <iostream>

#include <queue>

using namespace std;

class maxHeap{

public:

int \* heap;

int arrayLength;

int heapSize;

void initialize(int \*theHeap, int theSize){

heap = theHeap;

heapSize = theSize;

for (int root = heapSize/2; root >= 1; --root) {

int theEle = heap[root];

int child = 2\*root;

while (child<=heapSize){

if (child<heapSize&&heap[child]<heap[child+1])child++;

if (theEle>=heap[child])break;

heap[child/2] = heap[child];

child\*=2;

}

heap[child/2] = theEle;

}

}

int pop(){

int result = heap[1];

int lastEle = heap[heapSize];

heapSize--;

int currentNode = 1,child = 2;

while(child<=heapSize){

if (child<heapSize&&heap[child]<heap[child+1])child++;

if (lastEle>=heap[child])break;

heap[currentNode]=heap[child];

currentNode=child;

child\*=2;

}

heap[currentNode] = lastEle;

return result;

}

};

void levelView(maxHeap mheap){

for (int i = 1; i <= mheap.heapSize; ++i) {

if (i==mheap.heapSize)cout << mheap.heap[i]<<endl;

else cout << mheap.heap[i]<<",";

}

}

void heapSort(maxHeap mheap){

int arr[mheap.heapSize];int len = mheap.heapSize;

for (int i = 0; mheap.heapSize>=1; ++i) {

arr[i] = mheap.pop();

}

for (int i = len-1; i >=0; --i) {

if (i==0)cout << arr[i]<<endl;

else cout << arr[i]<<",";

}

}

class tree{

public:

int ele;

tree \* left;

tree \* right;

tree(int tEle,tree\* tLeft,tree\* tRight){

ele = tEle;

left = tLeft;

right = tRight;

}

};

static int flag=0;

void insertEle(tree \*t\_tree,int ele){

if (t\_tree->ele>ele){

if (t\_tree->left==NULL)t\_tree->left=new tree(ele,NULL,NULL);

else insertEle(t\_tree->left,ele);

}

else if (t\_tree->ele<ele){

if (t\_tree->right==NULL)t\_tree->right=new tree(ele,NULL,NULL);

else insertEle(t\_tree->right,ele);

}

else {

flag++;

return;

}

}

tree\* createFindTree(int \*arr,int len){

tree \*result = new tree(arr[1],NULL,NULL);

for (int i = 2; i <= len; ++i) {

insertEle(result, arr[i]);

}

return result;

}

static int count=0;

void frontView(tree \* tTree,int len){

if (tTree==NULL)return;

else{

if (count==len-1) cout << tTree->ele<<endl;

else{

cout << tTree->ele<<",";

count++;

}

frontView(tTree->left,len);

frontView(tTree->right,len);

}

}

void inView(tree \* tTree,int len){

if (tTree==NULL)return;

else{

inView(tTree->left,len);

if (count==len-1) cout << tTree->ele<<endl;

else{

cout << tTree->ele<<",";

count++;

}

inView(tTree->right,len);

}

}

int main() {

cout <<"Input"<<endl;

int enters[21];int tempEnters[21];

int enter;int length = 0;

cin >> enter;

for (int i = 1; enter != 0; ++i) {

enters[i] = enter;

tempEnters[i] = enter;

length++;

cin>>enter;

}

cout <<"Output"<<endl;

maxHeap heap;

heap.initialize(enters,length);

levelView(heap);

heapSort(heap);

tree \*theTree = createFindTree(tempEnters,length);

frontView(theTree,length-flag);

count=0;

inView(theTree,length-flag);

cout <<"End0"<<endl;

return 0;

}