山东大学 软件学院 学院

软件测试 课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：201800301202 | 姓名：李成 | | 班级： 2018级软件4班 |
| 实验题目：堆和搜索树 | | | |
| 实验学时：4 | | 实验日期2019-12-01 | |
| 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：  1、笔记本PC | | | |
| 软件环境：   1. Windows10 2. Eclipse 3. Java | | | |
| 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（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）  实验步骤：   1. 对二叉树的节点进行定义。   class treeNode{    public:    int value=0;    treeNode \*leftNode=NULL,\*rightNode=NULL;      treeNode(){  }    treeNode(int value){    this->value = value;    }    treeNode(int value , treeNode \*leftNode,treeNode \*rightNode){    this->value = value;  this->leftNode = leftNode;  this->rightNode = rightNode;    }    };   1. 由于需要用到队列创建一个队列类。   class queue{    private:    treeNode \*\*q;    int capacity;    int begin;    int end;    public :    queue(int n){    q = new treeNode\* [n];    for(int i = 0 ; i < n ; i++){    q[i] = NULL;    }  capacity = n;  begin = 0 ;  end = 0 ;    }    void push(treeNode \*T){    end = (end+1)%capacity;    q[end] = T;    }    treeNode\* read(){    return q[(begin+1)%capacity];    }    void pop(){    begin = (begin+1)%capacity;    delete q[begin];  }    };   1. 创建一个最大堆的类，并且写出堆排序的方法。   class bigTree{    private:    treeNode \*root = NULL;  treeNode \*p;  int count=0;  int number;  int \*heap;    public:    void bT(int a[], int n , int num , treeNode \*p2){    number = n;    if(2\*num+1<n){    p2->leftNode = new treeNode(a[2\*num+1]);    bT(a,n,2\*num+1,p2->leftNode);    }    if(2\*num+2<n){    p2->rightNode = new treeNode(a[2\*num+2]);    bT(a,n,2\*num+2,p2->rightNode);    }  }    treeNode\* search(int a){    queue q2(number);    int count2=0;    q2.push(root);    treeNode\* p3;    p3 = q2.read();    if(p3->value==a){  return p3;  }    count2++;    while(count2<number){    if(p3->leftNode!=NULL){    q2.push(p3->leftNode);    count2++;  }    if(p3->rightNode!=NULL){    q2.push(p3->rightNode);    count2++;    }    q2.pop();  if(q2.read()!=NULL)  p3 = q2.read();    if(p3->value==a){  return p3;  }  }  }    void order(int a[] , int n){    for(int root = n/2-1 ; root>=0 ; root--){    int rootElement = a[root];    int child = 2\*root+1;    while(child<=n-1){    if(child<n-1&&a[child]<a[child+1])    child++;    if(rootElement>=a[child])  break;    a[(child-1)/2] = a[child];    child = child\*2+1;    }    a[(child-1)/2] = rootElement;    }    }  bigTree(int a[],int n){    order(a,n);    heap = new int[n];    for(int i = 0 ; i < n ; i++){    heap[i] = a[i];    }    if(root == NULL){    root = new treeNode(a[count]);    p = root;    }    bT(a,n,count,root);    }    void Heap(){    int sort[number];  int number2 = number;    for(int i = 0 ; i < number ; i++){    int x = 0 ;  int child=1;  sort [i] = heap[x];    int lastElement = heap[--number2];    while(child<=number2-1){    if(child<number2-1&&heap[child]<heap[child+1]){    child++;  }    if(lastElement>=heap[child])  break;    heap[x] = heap[child];    x = child;  child = 2\*child+1;  }  heap[x] = lastElement;    }  cout<<endl;  cout<<sort[number-1];  if(number>=2)  for(int i = number-2 ;i >= 0; i--)  cout<<","<<sort[i];    }  void levelRead(){    queue q1(number);    int ccount=0;  q1.push(root);    treeNode\* p2;    p2 = q1.read();    cout<<p2->value;    ccount++;    while(ccount<number){    if(p2->leftNode!=NULL){    q1.push(p2->leftNode);    cout<<","<<p2->leftNode->value;    ccount++;  }    if(p2->rightNode!=NULL){    q1.push(p2->rightNode);    cout<<","<<p2->rightNode->value;    ccount++;    }    q1.pop();  if(q1.read()!=NULL)  p2 = q1.read();    }    }    };   1. 需要用到二叉树的方法，二叉树的类也必不可少。   class binaryTree{    private:    treeNode \*root;  treeNode \*p;  int Middlei = 0;      public:    void judge(int a,treeNode \*w){    if(a < w->value){    if(w->leftNode!=NULL){    w = w->leftNode;    judge(a,w);  }else{    w->leftNode = new treeNode(a);    }    }else if(a>w->value){    if(w->rightNode!=NULL){    w = w->rightNode;    judge(a,w);    }else{    w->rightNode = new treeNode(a);    }  }else{  }  }  binaryTree(int a[] , int n){    for(int i = 0 ; i < n ; i++ ){    if(i==0){    root = new treeNode(a[i]);  p = root;  }  else{    judge(a[i],root);    }  }  }  void PreRead(treeNode \*a){    if(a->leftNode!=NULL){    cout<<","<<a->leftNode->value;    PreRead(a->leftNode);    }  if(a->rightNode!=NULL){    cout<<","<<a->rightNode->value;    PreRead(a->rightNode);  }  }    void preRead(){    cout<<root->value;    PreRead(root);    }  void MiddleRead(treeNode \*a){    if(a->leftNode!=NULL){    MiddleRead(a->leftNode);    cout<<","<<a->value;    }    else{    if(Middlei == 0){    cout<<a->value;    Middlei++;    }    else    cout<<","<<a->value;    }    if(a->rightNode!=NULL){    MiddleRead(a->rightNode);  }  }    void middleRead(){  cout<<endl;  MiddleRead(root);    }  };  5，main()方法  int main(){    int array[20];  int array2[20];  int num = 0;    for(int i = 0 ; i < 20 ; i++){    array[i] = 0;  array2[i] = 0;  }    int a = 0 ;  cout<<"Input"<<endl;  cin>>a;    while(a!=0){    array[num] = a;  array2[num] = a;    num++;    cin>>a;  }    bigTree big(array,num);    cout<<"Output"<<endl;    big.levelRead();    big.Heap();    binaryTree bin(array2,num);  cout<<endl;  bin.preRead();  bin.middleRead();  cout<<endl;  cout<<"End"<<endl;    return 0;  } | | | |
| 结论分析与体会：  通过这次实验，进一步深度体会到了二叉树的各种操作，以及它的三种遍历方法，与上次不一样的是，堆排序的应用，堆排序是一种时间复杂度比较好的排序。操作起来稍微有点复杂。 | | | |