

**教学上机实验报告**

**课程名称：**   **数据结构**

**任课教师姓名：**  **贾盼盼**

**学生学号：**  312105010207

**学生姓名：**  刘晨阳

**学生专业班级：** 计算2106

**2021 ～ 2022 学年 第一学期**

|  |
| --- |
| **河南理工大学**  **教学上机实验报告评价分值标准** |
| |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 序号 | 评价指标 | 分值 | 评价等级及参考分值 | | | | | 评价分 | | 优 | 良 | 中 | 合格 | 差 | | 1 | 实验报告内容完整充实 | 10 | 10 | 8 | 7 | 6 | 3 |  | | 2 | 实验内容书写规范、字迹工整认真 | 10 | 10 | 8 | 7 | 6 | 3 |  | | 3 | 实验过程叙述详细、概念正确，语言表达准确，结构严谨，调理清楚，逻辑性强，自己努力完成，没有抄袭。 | 30 | 30 | 26 | 23 | 20 | 10 |  | | 4 | 对实验过程中存在的问题分析详细透彻、深刻、全面、规范、，结合实验内容，有自己的个人见解和想法，并能结合该实验提出相关问题，给出解决方法。 | 30 | 30 | 26 | 23 | 20 | 10 |  | | 5 | 实验结果、分析和结论正确无误 | 20 | 20 | 17 | 15 | 13 | 6 |  | | 总得分 | | | | | | | |  |   签名（签章）：  日期： 年 月 日 |
|  |

|  |
| --- |
| **河南理工大学教学上机实验报告** |
| 上机时间 2022 年 11月29 日 |
| **实验题目：**  1.编写程序实现有序表二分查找的递归算法；  2.用非递归的方法实现二叉排序树上的查找（并求出查找成功时关键字所在的层次）。 |
| **实验目的和要求：**  **实验目的：**1.掌握顺序表的查找方法，尤其是二分查找方法。 　　 2.掌握二叉排序树的建立及查找过程，理解二叉排序树查找过程及插入和删除算法。  **实验要求：**1.编写程序实现有序表二分查找的递归算法； 2.用非递归的方法实现二叉排序树上的查找（并求出查找成功时关键字所在的层次）。 |
| **实验过程：**  **程序主要功能源代码：**  **实验一：**实现有序表二分查找的递归算法  核心源代码：    **实验二：**  非递归的方法实现二叉排序树上的查找  核心源代码：  //算法7.4　二叉排序树的递归查找  BSTree SearchBST(BSTree T,int key) {  //在根指针T所指二叉排序树中递归地查找某关键字等于key的数据元素  //若查找成功，则返回指向该数据元素结点的指针，否则返回空指针  if((!T)|| key==T->data.key) return T; //查找结束  else if (key<T->data.key) return SearchBST(T->lchild,key); //在左子树中继续查找  else return SearchBST(T->rchild,key); //在右子树中继续查找  } // SearchBST  //非递归实现查找成功时返回指向该数据元素结点的指针和所在的层次  BSTree Searchlev(BSTree T,char key,int &lev) {    //统计查找次数    BSTree p = T; //p为二叉树中工作指针  lev++;  while(T)  {  // cout<<lev;  if(key==T->data.key) return T;  if(key<T->data.key){  T=T->lchild; //在左子树查找  lev++;  } else{  T=T->rchild; //在右子树查找  lev++;  }  // cout<<lev;  }  // cout<<"end";  // return T;  return NULL;      }// Searchlev  //算法7.5　二叉排序树的插入  void InsertBST(BSTree &T,ElemType e ) {  //当二叉排序树T中不存在关键字等于e.key的数据元素时，则插入该元素  if(!T) { //找到插入位置，递归结束  BSTree S = new BSTNode; //生成新结点\*S  S->data = e; //新结点\*S的数据域置为e  S->lchild = S->rchild = NULL; //新结点\*S作为叶子结点  T =S; //把新结点\*S链接到已找到的插入位置  }  else if (e.key< T->data.key)  InsertBST(T->lchild, e ); //将\*S插入左子树  else if (e.key> T->data.key)  InsertBST(T->rchild, e); //将\*S插入右子树  }// InsertBST  //算法7.6　二叉排序树的创建  void CreateBST(BSTree &T ) {  //依次读入一个关键字为key的结点，将此结点插入二叉排序树T中  T=NULL;  ElemType e;  cin>>e.key; //???  while(e.key!=ENDFLAG){ //ENDFLAG为自定义常量，作为输入结束标志  InsertBST(T, e); //将此结点插入二叉排序树T中  cin>>e.key; //???  }//while  }//CreatBST  void DeleteBST(BSTree &T,int key) {  //从二叉排序树T中删除关键字等于key的结点  BSTree p=T;BSTree f=NULL; //初始化  BSTree q;  BSTree s;  /\*------------下面的while循环从根开始查找关键字等于key的结点\*p-------------\*/  while(p){  if (p->data.key == key) break; //找到关键字等于key的结点\*p，结束循环  f=p; //\*f为\*p的双亲结点  if (p->data.key> key) p=p->lchild; //在\*p的左子树中继续查找  else p=p->rchild; //在\*p的右子树中继续查找  }//while  if(!p) return; //找不到被删结点则返回  /\*―考虑三种情况实现p所指子树内部的处理：\*p左右子树均不空、无右子树、无左子树―\*/  if ((p->lchild)&& (p->rchild)) { //被删结点\*p左右子树均不空  q = p;  s = p->lchild;  while (s->rchild) //在\*p的左子树中继续查找其前驱结点，即最右下结点  {q = s; s = s->rchild;} //向右到尽头  p->data = s->data; //s指向被删结点的“前驱”  if(q!=p){  q->rchild = s->lchild; //重接\*q的右子树  }  else q->lchild = s->lchild; //重接\*q的左子树  delete s;  }//if  else{  if(!p->rchild) { //被删结点\*p无右子树，只需重接其左子树  q = p; p = p->lchild;  }//else if  else if(!p->lchild) { //被删结点\*p无左子树，只需重接其右子树  q = p; p = p->rchild;  }//else if  /\*――――――――――将p所指的子树挂接到其双亲结点\*f相应的位置――――――――\*/  if(!f) T=p; //被删结点为根结点  else if (q==f->lchild) f->lchild = p; //挂接到\*f的左子树位置  else f->rchild = p; //挂接到\*f的右子树位置  delete q;  }  }//DeleteBST  //算法 7.7　二叉排序树的删除  //中序遍历  void InOrderTraverse(BSTree &T)  {  if(T)  {  InOrderTraverse(T->lchild);  cout<<T->data.key<<" " ;  InOrderTraverse(T->rchild);  }  }  char predt=0;  int judgeBST(BSTree T)  {  int b1, b2;  if(T == NULL)  {  return 1;  }  else  {  b1 = judgeBST(T->lchild);  if(b1 == 0 || predt > T->data.key) return 0;  predt = T->data.key;  b2 = judgeBST(T->rchild);  return b2;  }  } |
| **实验结果：**  **实验一结果：**    **实验二结果：** |
| **实验分析：**  **二分查找：**  **主要使用循环或递归在每次比较后将查找空间划分为两半，依次进行查找。**  **7.4-7.7：**   1. **从表的一端开始逐个将记录的关键字和给定 K 值进行比较，若某个记录的关键字和给定 K 值相等，查找成功；否则，若扫描完整个表，仍然没有找到相应的记录，则查找失败** 2. **根结点就是第一次进行比较的中间位置的记录** 3. **排在中间位置前面的作为左子树的结点，排在中间位置后面的作为右子树的结点** 4. **在 BST 树中插入一个新结点 x 时，若 BST 树为空，则新结点 x 为插入后 B ST 树的根结点；否则，将结点 x 的关键字与根结点 T的关键字进行比较，若相等，不需要插入，若 x . keykey :结点 x 插入到 T 的左子树中，若 x . key > T -> key :结点 x 插入到 T 的右子树中** |
| **实验成绩：**  日期： 年 月 日 |