**实验报告**

**学院（系）名称：**计算机与通信工程学院

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **姓名** | | 王帆 | | **学号** | | 20152180 | | | **专业** | | 计算机科学与技术 | |
| **班级** | | 15级1班 | | **实验项目** | | 实验五：查找算法应用 | | | | | | |
| **课程名称** | | | | 数据结构与算法 | | | | | **课程代码** | | 0668016 | |
| **实验时间** | | | | 2017年6月2日 第5-8节 | | | | | **实验地点** | |  | |
| 考核标准 | 实验过程  25分 | | 程序运行  20分 | | 回答问题  15分 | | 实验报告  30分 | 特色  功能  5分 | | 考勤违纪情况  5分 | **成绩** |  |
| 成绩栏 |  | |  | |  | |  |  | |  | 其它批改意见:  教师签字： | |
| 考核内容 | 评价在实验课堂中的表现，包括实验态度、编写程序过程等内容等。 | | □功能完善,  □功能不全  □有小错  □无法运行 | | ○正确  ○基本正确  ○有提示  ○无法回答 | | ○完整  ○较完整  ○一般  ○内容极少  ○无报告 | ○有  ○无 | | ○有  ○无 |
|  | | | | | | | | | | | | |
| **一、实验目的**  运用折半查找算法解决实际问题；理解二叉排序树查找、插入、删除、建立算法的思想及程序实现；掌握散列存储结构的思想，能选择合适散列函数，实现不同冲突处理方法的散列表的查找、建立。  **二、实验题目与要求**  每位同学从下面题目中至少选择2题实现：其中第1题必做，2-4题中至少选择1题。  1.折半查找算法  1）问题描述：从键盘读入一串整数和一个待查关键字，查找在该整数串中是否有这个待查关键字。如果有，输出它在整数串中的位置；如果没有，输出-1  2）实验要求：   * 利用折半查找算法查找 * 用递归和非递归两种方式实现折半查找算法   3) 实现提示：   * 递归实现参考书上折半查找算法的实现 * 非递归算法利用栈实现   2.构造二叉排序树，并进行中序遍历  1）问题描述：从键盘读入一串整数构造一棵二叉排序树，并对得到的二叉排序树进行中序遍历，得到有序序列。  2）实验要求：该二叉排序树以二叉链表存储  3）实现提示：二叉排序树的构成，可从空的二叉树开始，每输入一个结点数据，就建立一个新结点插入到当前已生成的二叉排序树中，所以它的主要操作是二叉排序树插入运算。在二叉排序树中插入新结点，只要保证插入后仍符合二叉排序树的定义即可。  3.哈希表查找  1）问题描述：针对某个集体的“人名”构造哈希表，解决按“人名”进行查找的索引结构。  2）实验要求：要求表的平均查找长度不超过R，完成相应的建表和查表程序。  3) 实现提示：假设人名为汉语拼音形式。代填入哈希表人名共30个，取平均查找长度的上限为2。哈希函数用除留余数法构造，用再散列法处理冲突。  4. 拼写检查  1）问题描述：现在有一些英语单词需要做拼写检查，你的工具是一本词典。需要检查的单词，有的是词典中的单词，有的与词典中的单词相似，你的任务是发现这两种情况。单词A与单词B相似的情况有三种：  ① 删除单词A的一个字母后得到单词B；  ② 用任意一个字母替换单词A的一个字母后得到单词B；  ③ 在单词A的任意位置增加一个字母后得到单词B。  2）实验要求：发现词典中与给定单词相同或相似的单词。   1. **实验过程与实验结果**   **实验5-01：折半查找算法**  **数据结构定义：**  typedef int KeyType;  typedef struct ElemType{  KeyType key;  }ElemType;  **算法设计思路简介：**  折半查找是一种优化的查找方法，其遵从二分的思想，但是需要将关键字有序排列（升序/降序）并采用顺序存储的方法存储。实现方案为先有序构造顺序表，后利用二分法实现。  **算法描述：**  流程图见下页   1. 初始化low/high，开始二分。 2. 比较low与high，如果不符合逻辑（low>high）则退出，返回-1；否则，求解mid=(low+high)/2 3. 判断mid对应关键字是否为待查关键字，如果是则返回mid，否则进一步判断mid对应关键字与目标关键字的大小关系，从而确定下次二分区间。 4. 重复2,3步骤，直到获取到最终二分区间   **D:\360极速浏览器下载\1.png**  **图1 折半查找算法流程图**  **算法的实现和测试结果（参考OJ）**    **实验5-02：二叉排序树的构造与（中序）遍历**  **数据结构定义：**  typedef int KeyType;  typedef struct BiTNode{  KeyType data;  BiTNode \*lchild, \*rchild;  }BiTNode;  typedef BiTNode \*BiTree;  **算法设计思路简介：**  二叉排序树是一种具有特定序列的二叉树，其定义是递归实现的，其左孩子关键字均小于根节点关键字，而右孩子关键字均大于根节点关键字。构造时，根据起始元素顺序依次进行比较与插入的操作，从而最终形成二叉排序树。中序遍历实现方法同二叉树的中序遍历。  **算法描述：**  **算法的实现和测试结果（参考OJ）**    **实验5-03：哈希表查找**  **数据结构定义：**  **算法设计思路简介：**  **算法描述：**  **算法的实现和测试结果（参考OJ）**  **实验5-04：拼写检查**  **数据结构定义：**  **算法设计思路简介：**  **算法描述：**  **算法的实现和测试结果（参考OJ）**  **四、收获与体会**  **五、源代码清单**  **/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*///实验5-01：折半查找**  **/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**#include <iostream>  using namespace std;  typedef int KeyType;  typedef struct ElemType{  KeyType key;  }ElemType;  int n;  KeyType key;  int Binary\_Search(ElemType record[],KeyType k){  int low=1;  int high=n;  int mid;  while(low<=high){  mid=(low+high)/2;  if(record[mid].key==k) return mid;  else if(k<record[mid].key) high=mid-1;  else low=mid+1;  }  return -1;  }  ElemType Construction(ElemType str[]){  for(int i=1;i<n+1;i++){  cin>>str[i].key;  }  }    int main(){  cin>>n>>key;  ElemType str[n+1];  Construction(str);  cout<<Binary\_Search(str,key);  return 0;  }  **/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*///实验5-02：二叉排序树的构造与（中序）遍历**  **/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**  #include <iostream>  #include <queue>  #include <stack>  #define MAX\_NUM 100  using namespace std;  typedef int KeyType;  typedef struct BiTNode{  KeyType data;  BiTNode \*lchild, \*rchild;  }BiTNode;  typedef BiTNode \*BiTree;  int arr[MAX\_NUM];  int index=0;  void Insert(BiTree &T, KeyType k){  if(T==NULL){  T=(BiTree)malloc(sizeof(BiTNode));  T->data=k;  T->lchild=T->rchild=NULL;  }  else if(k<=T->data) Insert(T->lchild,k);  else if(k>T->data) Insert(T->rchild,k);  }  void Create\_BST(BiTree &T){  T=NULL;  KeyType k;  while(1){  cin>>k;  if(k==-1) break;  else Insert(T,k);  }  }  void Init\_SEQ(KeyType data){  arr[index]=data;  index++;  }  void InOrder(BiTree T){  if(T==NULL) return;  InOrder(T->lchild);  Init\_SEQ(T->data);  InOrder(T->rchild);  }  void show(){  int i;  for(i=0;i<index-1;i++){  cout<<arr[i]<<" ";  }  cout<<arr[index-1];  }  int main(){  BiTree T;  Create\_BST(T);  InOrder(T);  show();  return 0;  } | | | | | | | | | | | | |