# 实验六 查找

【实验项目】：完成二叉树的基本运算，从而了解其基本特征，基本运算算法

## 一、实验目的

1. 掌握查找的含义。

2. 掌握基本查找操作的算法与实现。

3. 掌握二叉排序树查找的算法与实现。

4. 掌握Hash表查找的算法与实现。

## 二、实验要求（本次实验要求上交）

**【题目】--验证型实验**

**1. 建立一个线性表，对表中数据元素存放的先后次序没有任何要求。输入待查数据元**

**素的关键字进行查找。（为了简化算法，数据元素只含一个整型量关键字字段，数据元素的其余数据部分忽略不考虑。）**

**顺序查找的基本思想及程序实现（参考代码见文件夹6-1）**

对于给定的关键字k，从表的一端开始，逐个进行数据元素的关键字和给定值的比较，若当前扫描到的结点关键字与k相等则查找成功；若扫描结束后，仍未找到关键字等于k的节点，则查找失败。

建立一个顺序表，数据元素从下标为1的单元开始放入，下标为0的单元起监视哨作用，将待查的关键字存入下标为0的单元，顺序表从后向前查找，若直到下标为0时才找到关键字则说明查找失败；若不到下标为0时就找到关键字，则查找成功。

2. **查找表的存储结构为有序表，即表中记录按关键字大小排序存放。输入待查数据元**

**素的关键字进行查找。（为了简化算法，记录只含一个整型量关键字字段，记录的其余数据部分略不考虑。程序要求对整型量关键字数据的输入按从小到大排序输入。）**

**折半查找的基本思想及程序实现：（参考代码见文件夹6-1）**

设查找表中的元素存放在数组r中，数据元素的下标范围为[low,high]，要查找的关键字值为key，中间元素的下标为mid=(low+high)／2 (向下取整)，令key与r[mid]的关键字比较：

若key=r[mid].key，查找成功，下标为m的记录即为所求，返回mid。

若key<r[mid].key，所要找的记录只能在左半部分记录中，再对左半部分使用折半查找法继续进行查找，搜索区间缩小了一半。

若key>r[mid].key，所要找的记录只能在右半部分记录中，再对右半部分使用折半查找法继续进行查找，搜索区间缩小了一半。

重复上述过程，直到找到查找表中某一个数据元素的关键字的值等于给定的值key说明查找成功；或者出现low的值大于high的情况，说明查找不成功。

建立一个有序表，数据元素从下标为1的单元开始放入。实现查找算法时，首先将low赋值为l，high等于最后一个数据元素的下标，然后将给定的关键字的值与查找区间[low，high]中间的数据元素的关键字比较， 实现查找过程。

3. **编程实现二叉排序树的创建、查找、插入、输出等算法。**

**二叉排序树基本思想及程序实现（参考代码见文件夹6-1）**

二叉排序树就是指将原来已有的数据根据大小构成一棵二叉树，二叉树中的所有

结点数据满足一定的大小关系，所有的左子树中的结点均比根结点小，所有的右子树的结点均比根结点大。

二叉排序树查找是指按照二叉排序树中结点的关系进行查找，查找关键字首先同根结点进行比较，如果相等则查找成功；如果比根结点小，则在左子树中查找；如果比根结点大，则在右子树中进行查找。这种查找方法可以快速缩小查找范围，大大减少查找关键字的比较次数，从而提高查找效率。

算法的关键过程实际上就是二叉排序树的创建和查找两个步骤。

首先分析二叉排序树的创建运算，由于二叉排序树中的所有结点都有一个大小关系，因此，每个结点必须在二叉排序树中寻找其合适的位置。创建二叉排序树的第一步就是创建一个根结点，第二步就是将其他结点不断插入到二叉排序树中的合适位置。二叉排序树进行结点插入时，首先要为被插入结点寻找合适的插入位置。这个过程实际上就是一个关键值不断的比较的过程。

第一次比较是与二叉排序树的根结点比较，如果比根结点的关键值小，则插入到他的左子树中；如果比根结点的关键值大，则插入到它的右子树中。在子树中重复这样过程，直到找到合适的位置为止。

二叉排序树的查找运算实际上同二叉排序树的创建运算是一致的。区别在于，创建过程中要为二叉排序树中没有位置的关键字建立结点，而查找过程中，只是在二叉排序树中查找是否等于查找关键字值的结点存在，不管有还是没有，它都不会创建一个新的结点。

**【题目2】（注意：该程序将作为平时成绩的一部分计入期末总分，请认真完成。另外：请独立完成，在批改过程中若发现是抄袭的，相同版本第二份及其以上做不合格处理）**

编程实现一个开放式的高校本科招生最低录取分数线的查询系统，供师生和家长查询。高校自愿放入该校的信息，可能随时有高校加入。要求实现的查询功能有：①查询等于用户给定分数的高校；②查询大于(或小于)用户给定分数的高校；③查询最低录取分数线在用户给定的分数段中的高校。

该实验主要的功能是查找，查找表为高校最低录取分数信息的集合。根据题意可知，该查找表中的元素个数可能随时增减，所以它是一个动态查找表，可采用树状结构保存。为了提高查询速度，可建立二叉排序树并在二叉排序树中实现查找。

第一个要求可以直接利用一般的二叉排序树的查找算法实现。

第二个要求可以利用比较二叉排序树根结点的关键字值和给定的分数实现，若前者大于或等于后者，根结点及其右子树中的结点全部满足要求，再在根结点的左子树中查找；否则只在根结点的右子树中查找即可。

第三个要求可以利用比较二叉排序树根结点的关键字值和给定的分数段实现，若前者在给定的分数段中，根结点就满足要求，则在根结点的左、右子树中继续查找即可：若前者

给定的分数段的左侧，只在根结点的右子树中查找即可；若前者在给定的分数段的右侧，只根节点的左子树中查找即可。

|  |
| --- |
| 请将源程序附录于此： |
| #include <stdio.h>  #include <malloc.h>  #define MaxSize 100  typedef int KeyType; //定义关键字类型  typedef char InfoType;  typedef struct node //记录类型  {  KeyType key; //关键字项  InfoType data; //其他数据域  struct node \*lchild,\*rchild; //左右孩子指针  } BSTNode;  int path[MaxSize]; //全局变量,用于存放路径  void DispBST(BSTNode \*b); //函数说明  int InsertBST(BSTNode \*&p,KeyType k) //在以\*p为根节点的BST中插入一个关键字为k的节点  {  if (p==NULL) //原树为空, 新插入的记录为根节点  {  p=(BSTNode \*)malloc(sizeof(BSTNode));  p->key=k;p->lchild=p->rchild=NULL;  return 1;  }  else if (k==p->key)  return 0;  else if (k<p->key)  return InsertBST(p->lchild,k); //插入到\*p的左子树中  else  return InsertBST(p->rchild,k); //插入到\*p的右子树中  }  BSTNode \*CreatBST(KeyType A[],int n)  //由数组A中的关键字建立一棵二叉排序树  {  BSTNode \*bt=NULL; //初始时bt为空树  int i=0;  while (i<n)  if (InsertBST(bt,A[i])==1) //将A[i]插入二叉排序树T中  {  printf(" 第%d步,插入%d:",i+1,A[i]);  DispBST(bt);printf("\n");  i++;  }  return bt; //返回建立的二叉排序树的根指针  }  void SearchBST1(BSTNode \*bt,KeyType k,KeyType path[],int i)  //以非递归方式输出从根节点到查找到的节点的路径  {  int j;  if (bt==NULL){  printf("不存在该分数的学校\n");  return;  }  else if (k==bt->key) //找到了节点  {  path[i+1]=bt->key; //输出其路径  printf("你要查询的学校是:%d",path[i+1]);  printf("\n");  return;  }  else  {  path[i+1]=bt->key;  if (k<bt->key)  SearchBST1(bt->lchild,k,path,i+1); //在左子树中递归查找  else  SearchBST1(bt->rchild,k,path,i+1); //在右子树中递归查找  }  }  int SearchBST2(BSTNode \*bt,KeyType k,KeyType p)  //以递归方式输出从根节点到查找到的节点的路径  {  if (bt==NULL)  return 0;  else if (k<=bt->key && p>=bt->key)  {  printf("%d ",bt->key);  SearchBST2(bt->rchild,k,p);  SearchBST2(bt->lchild,k,p);  return 1;  }  else if (k>bt->key)  SearchBST2(bt->rchild,k,p); //在右子树中递归查找  else if (p<bt->key)  SearchBST2(bt->lchild,k,p); //在右子树中递归查找  }  void SearchBST3(BSTNode \*bt,KeyType k,KeyType path[],int i)  {  if (bt==NULL)  return;  else if (k<=bt->key)  {  if(k==bt->key){  path[i+1]=bt->key;  printf("%d ",path[i+1]);  SearchBST3(bt->rchild,k,path,i+1); //在右子树中递归查找  }else{  path[i+1]=bt->key;  printf("%d ",path[i+1]);  SearchBST3(bt->rchild,k,path,i+1); //在右子树中递归查找  SearchBST3(bt->lchild,k,path,i+1); //在左子树中递归查找  }  }  else  SearchBST3(bt->rchild,k,path,i+1); //在右子树中递归查找  }  void DispBST(BSTNode \*bt)  {  if (bt!=NULL)  {  printf("%d",bt->key);  if (bt->lchild!=NULL || bt->rchild!=NULL)  {  printf("(");  DispBST(bt->lchild);  if (bt->rchild!=NULL) printf(",");  DispBST(bt->rchild);  printf(")");  }  }  }  KeyType predt=-32767;  int JudgeBST(BSTNode \*bt) //判断bt是否为BST  {  int b1,b2;  if (bt==NULL)  return 1;  else  {  b1=JudgeBST(bt->lchild);  if (b1==0 || predt>=bt->key)  return 0;  predt=bt->key;  b2=JudgeBST(bt->rchild);  return b2;  }  }  void main()  {  BSTNode \*bt;  KeyType i,j,k,p;  int a[]={4,9,0,1,8},n=5;  printf("创建一棵BST树:");  printf("\n");  bt=CreatBST(a,n);  while(1){  printf("请输入你想查找的分数:");  scanf("%d",&k);  SearchBST1(bt,k,path,-1);  printf("大于等于你给的分数的学校是:");  SearchBST3(bt,k,path,-1);  printf("\n请输入你要查询的分数段:");  scanf("%d,%d",&k,&p);  printf("最低录取分数线在该分数段的学校有:");  SearchBST2(bt,k,p);  printf("\n你是否想要继续插入高校,是请输入1，否输入0:");  scanf("%d",&i);  if(i==1){  printf("输入你要插入高校的分数线:");  scanf("%d",&j);  InsertBST(bt,j);  }else  break;  }  printf("\n");  DispBST(bt);  printf("\n插入查询操作结束");  } |
| 请将运行结果图附录于此 |
| C:\Users\Administrator\AppData\Roaming\Tencent\Users\1061892125\QQ\WinTemp\RichOle\~_@0LW%ALM)5LPOID2%Y}1M.png |

## 实验上传方法：

将源程序文件和本word文档（添加了相应的源程序和截图）上传。多个文件请将其放入一个文件夹压缩后上传。请于指定时间前上交，过时不补！