实验 5 查找和排序

EXPERIMENT

5.1 基础性实验

5.1.1 查找操作

一、实验目的

1. 熟练掌握折半查找的查找方法；
2. 熟练掌握二叉排序树的构造方法和查找算法。

二、实验内容

1. 建立有序表，采用折半查找实现某一已知关键字的查找；
2. 随机产生一组关键字，利用二叉排序树的插入算法建立二叉排序树，然后查找某一指定关键字元素。

三、实验说明

1. 问题描述

静态查找表可以有不同的表示方法，在不同的表示方法中，实现查找操作的方法也不同。静态查找表如果是有序的顺序表，相应的查找方法为折半查找方法。动态查找表的特点是表结构本身在查找过程中动态生成，即对给定的关键字 key，若表中存在其关键字等于 key 的记录，则查找成功返回；否则插入关键字等于 key 的记录。完成二叉排序树建立、查找、插入程序。

2. 数据描述

类型定义如下：

```
typedef struct node
{  int key;
   struct node * lchild, * rchild;
} bstnode;
```

3. 算法描述

在该程序中编写构造二叉排序树的建立、插入函数、查找函数、中序遍历的结果放入数组中，并对数组进行折半查找。

1）二叉排序树的建立

从空的二叉树开始，每输入一个数据，就插入到当前已生成的二叉排序树中。

2）二叉排序的插入

插入结点的关键字 key，为将其插入，先要在二叉排序树中进行查找，若查找成功，待插入结点已存在，不用插入；若查找不成功，则插入。因此，新插入结点一定是作为叶子结点添加上去的。

3）二叉排序树查找

从其定义可见，二叉排序树的查找过程为：

(1) 若查找树为空，查找失败。

(2) 查找树非空，将给定值 kx 与查找树的根结点关键字比较。

(3) 若相等，查找成功，结束查找过程，否则，

① 当 kx 小于根结点关键字，查找将在以左子女为根的子树上继续进行，转步骤(1)；

② 当 kx 大于根结点关键字，查找将在以右子女为根的子树上继续进行，转步骤(1)。

4）折半查找

(1) low=1;high=length; //设置初始区间

(2) 当 low>high 时，返回查找失败信息 //表空，查找失败

(3) low≤high,mid=(low+high)/2; //取中点

① 若 kx<tbl. elem[mid]. key,high=mid−1;转② //查找在左半区进行

② 若 kx>tbl. elem[mid]. key,low=mid+1;转② //查找在右半区进行

③ 若 kx=tbl. elem[mid]. key,返回数据元素在表中位置 //查找成功

四、注意问题

1. 注意理解折半查找的适用条件(链表能否实现折半查找?)。

2. 注意建立二叉排序树相同元素的处理。

五、算法实现示例

```
#include<stdio.h>
#include<stdlib.h>
typedef struct node
{ int key;
struct node * lchild, * rchild;
} bstnode; /* 类型定义 */
/* 中序遍历 */
void inorder(bstnode * t,int a[],int * n)
{    if(t)
    {
```

```
        inorder(t->lchild,a,n);
        a[++(*n)]=t->key;
        inorder(t->rchild,a,n);
    }
}
/*将 s 结点插入到二叉排序树 t 中*/
bstnode *insertbst(bstnode  *t,int key)
{
/*在此嵌入代码*/
}
/*建立二叉排序树*/
bstnode *creatord()
{  bstnode *t;
    int key;
    t=0;
    scanf("%d",&key);
    while(key!=0)
    {
        t=insertbst(t, key);
        scanf("%d",&key);
    }
    return t;
}
/*二叉排序树 t 的查找*/
bstnode *searchbst(bstnode  *t,int key)
{
/*在此嵌入代码*/
}
/*折半查找*/
int bisearch(int a[],int n,int k)
{
/*在此嵌入代码*/
}
main()
{
    bstnode *root ;
    int k;                      //存放要查找的关键字
    int a[100];int n=0;         //存放数组的实际长度
    int i;
    printf("建立二叉排序树,请输入一组整数,以 0 结束\n");
    root=creatord();
    printf("请输入要查找的整数\n");
    scanf("%d",&k);
    if(searchbst(root,k)) printf("查找成功\n");else printf("查找失败\n");
```

```
    inorder(root,a,&n);
    printf("有序表为\n");
    for(i<=i=1; n;i++)
        printf("%d ",a[i]);
    printf("\n");
    printf("请输入要查找的整数,采用折半查找的方法\n");
    scanf("%d",&k);
    i=bisearch(a,n,k);
    if(i>0) printf("查找成功\n");else printf("查找失败\n");
}
```

六、测试用例

二叉排序树的结点数据采用测试用例：12,2,34,5,6,89,9,7,0,程序的执行过程如图 1-5-1 所示。

图 1-5-1　测试用例

七、思考与提高

1. 二叉排序树的插入函数是否可写为递归函数？
2. 折半查找的递归如何完成？

5.1.2　排序操作

一、实验目的

1. 熟练掌握各种排序的算法思想、方法及稳定性；
2. 掌握快速排序、堆排序、归并排序的方法实现；
3. 对已知一组数据，能写出其具体的排序过程、算法及完整程序，并上机调试。

二、实验内容

1. 实现简单选择排序、直接插入排序和冒泡排序；
2. 实现快速排序算法；
3. 实现堆排序算法；

4. 采用几组不同数据测试各个排序算法的性能(比较次数和移动次数)。

三、实验说明

1. 问题描述

完成直接插入排序、直接选择排序、冒泡排序、快速排序和堆排序的排序过程。

2. 数据描述

```
typedef struct
{   int elem[100]          //线性表占用的数组空间
    int length;            //元素的实际长度
} SqList;
```

3. 算法描述

1) 直接插入排序

仅有一个记录的表总是有序的,因此,对 n 个记录的表,可从第二个记录开始直到第 n 个记录,逐个向有序表中进行插入操作,得到 n 个记录按关键字有序的表。

2) 冒泡排序

冒泡排序方法:对 n 个记录的表,第一趟冒泡得到一个关键字最大的记录 $r[n]$,第二趟冒泡对 $n-1$ 个记录的表,再得到一个关键字最大的记录 $r[n-1]$,如此重复,直到得到 n 个记录按关键字有序的表。

3) 简单选择排序

选择排序方法:第一趟,从 n 个记录中找出关键字最小的记录与第一个记录交换;第二趟,从第二个记录开始的 $n-1$ 个记录中再选出关键字最小的记录与第二个记录交换;如此,第 i 趟,则从第 i 个记录开始的 $n-i+1$ 个记录中选出关键字最小的记录与第 i 个记录交换,直到整个序列按关键字有序。

4) 快速排序

思想:通过一趟排序将待排序列分成两部分,使其中一部分记录的关键字均比另一部分小,再分别对这两部分排序,以达到整个序列有序。

(1) 一趟划分。

关键字通常取第一个记录的值为基准值。

做法:附设两个指针 low 和 high。

首先从 high 所指位置起向前搜索,找到第一个小于基准值的记录与基准记录交换。

然后从 low 所指位置起向后搜索,找到第一个大于基准值的记录与基准记录交换。

重复这两步直至 low=high 为止。

(2) 快速排序。

首先对无序的记录序列进行“一次划分”,之后分别对分割所得两个子序列“递归”进行快速排序。

5) 堆排序

首先将 n 个元素按关键字建成堆,将堆顶元素输出,得到 n 个元素中关键字最小(或最大)的元素。然后,再对剩下的 $n-1$ 个元素建成堆,输出堆顶元素,得到 n 个元素中关

键字次小(或次大)的元素。如此反复,便得到一个按关键字有序的序列。称这个过程为堆排序。

建堆方法:对初始序列建堆的过程,就是一个反复进行筛选的过程。对于 n 个结点的完全二叉树来说,最后一个结点是第$\lfloor \frac{n}{2} \rfloor$个结点的孩子。对以第$\lfloor \frac{n}{2} \rfloor$个结点为根的子树进行筛选,使该子树成为堆,之后向前依次对各结点为根的子树进行筛选,使之成为堆,直到根结点。

堆排序:对 n 个元素的序列进行堆排序,先将其建成堆,以根结点与第 n 个结点交换;调整前 $n-1$ 个结点成为堆,再以根结点与第 $n-1$ 个结点交换;重复上述操作,直到整个序列有序。

四、注意问题

注意理解各种算法的思想,了解算法的适用情况及时间复杂度,能够根据实际情况选择合适的排序方法。

五、算法实现示例

```
#include<stdio.h>
#include<stdlib.h>
typedef  struct {
    int   *elem;
    int   Length;
} SqList;
/*直接插入排序*/
void InsertSort(SqList  L)
{
/*在此嵌入代码*/
}
/*冒泡排序*/
void bubble_sort(SqList L)
{
/*在此嵌入代码*/
}
/*简单选择排序*/
void SelectSort(SqList L)
{
/*在此嵌入代码*/
}
/*快速排序*/
int Partition(SqList  L,int low,  int  high)
{
/*在此嵌入代码*/
}
```

```
void Qsort (SqList  L,int low,  int  high)
{  int  pivotloc;
    if  (low<high)
    {  pivotloc=Partition(L, low, high) ;
        Qsort (L, low, pivotloc-1) ;
        Qsort (L, pivotloc+1, high);
     }
}
/*堆排序*/
typedef SqList HeapType;          //堆采用顺序表存储
void HeapAdjust(HeapType  H, int s, int m)
{
/*在此嵌入代码*/
}
void HeapSort(HeapType  H)
{
/*在此嵌入代码*/
}
void main()
{   char ch;int i;
    int k;
    SqList r;
    printf("要进行排序操作吗? 如果进行则按 Y 或者 y,否则退出排序");
    scanf("%c",&ch);
    while(ch=='Y'||ch=='y')
    {system("cls");
        printf("请输入待排记录的个数");
        scanf("%d",&r.Length);
    r.elem=(int*)malloc(sizeof(int)*(r.Length+1));
    printf("请输入待排记录");
    for(i=1;i<=r.Length;i++)
        scanf("%d",&r.elem[i]);
    printf("请选择适用的排序方法:\n 使用直接插入排序请按 1\n 使用冒泡排序请按 2\n 使
用简单选择排序请按 3\n 使用快速排序请按 4\n 使用堆排序请按 5\n");
    scanf("%d",&k);
    switch(k)
    {
    case 1: InsertSort(r);break;
    case 2: bubble_sort(r);break;
    case 3:SelectSort(r);;break;
    case 4:Qsort(r,1,r.Length); break;
    case 5: HeapSort(r);break;
    }
    scanf("%c",&ch);
```

```
        printf("有序序列为: \n");
        for(i=1;i<=r.Length;i++)
            printf("%d ",r.elem[i]);
        printf("\n 要进行排序操作吗? 如果进行,则按 Y 或者 y,否则退出排序");
        scanf("%c",&ch);
        }
    }
```

六、测试用例

待排记录的测试用例为 3,5,12,89,7,6,2,67,90,1 共 10 个整数时,程序的执行过程如图 1-5-2 所示。

图 1-5-2 测试用例

七、思考与提高

1. 如何在程序中加入性能测试?
2. 试着在该程序中加入归并排序和希尔排序的排序过程。

5.2 拓展性实验

5.2.1 个人通讯录

一、实验目的

1. 进一步理解和掌握课堂上所学各种基本抽象数据类型的逻辑结构、存储结构和操作实现算法;

2. 将理论知识和实际结合起来,锻炼分析解决实际问题的能力。

二、实验内容

【问题描述】

建立一个简易通讯录信息,包括姓名、工作单位、电话号码和 E-mail 地址等信息。能够对记录中的姓名和电话号码进行修改;可增加或删除记录;可显示所有保存的记录;可

按人名或电话号码进行查询。

【设计思路】

1. 首先对于通讯录中的各成员包含的信息：姓名、电话、工作单位等定义一个结构体；

2. 定义一个链表来储存成员，进行成员的插入以及信息修改、删除操作；

3. 对于通讯录重要实现的各种操作可以设置一个菜单清单。

【基本要求】

输入内容：

1. 建立通讯录；

2. 增加的记录信息；

3. 删除的信息；

4. 查找的信息。

输出内容：

1. 显示所有记录的内容；

2. 查找记录的内容。

【数据结构】

采用链表作为存储结构：

```
typedef struct {                    //通讯录结点类型
    char name[9];                   //姓名
    char phone[13];                 //电话
    char workunit[31];              //工作单位
    char email[3];                  //E-mail 地址
} DataType;
typedef struct node {               //结点类型定义
    DataType data;                  //结点数据域
    struct node * next;             //结点指针域
} ListNode, * LinkList;
```

【算法描述】

1. 建立通讯录

建立一个按照姓名排序的链表：

(1) 首先生成结点；

(2) 置结束标志为 0(假)。

```
while(结束标志不为真)
{
P 指向新生成的结点;
    读入一个通讯者数据至新结点的数据域;
    将新结点插入到链表适当位置;
        提示是否继续建表,读入一个结束的标志;
}
```

具体算法实现如下：

```
/*******建立带头结点的通讯录链表算法*******/
LinkList CreateList()
{
    LinkList head=(ListNode *)malloc(sizeof(ListNode));  //申请头结点
    ListNode *p;
    char ch;
    int flag=1;          //结束时为 0
    head->next=NULL;
    while(flag)
    {   p=(ListNode *)malloc(sizeof(ListNode));      //申请新结点
        printf("姓名(8) 电话(11)  工作单位(30)  email(20)\n");
        printf("-----------------------------------------\n");
        printf("\n 添加的姓名:\n");
        scanf("%s",p->data.name);
        printf("\n 电话:\n");
        scanf("%s",p->data.phone);
        printf("\n 工作单位:\n");
        scanf("%s",p->data.workunit);
        printf("\nE-mail:\n");
        scanf("%s",p->data.email);
        Insert(head,p);
        printf("继续建表? (1/0):");
        scanf("%d",&flag);
        scanf("%c",&ch);
    }
    return head;
}
```

2. 通讯者插入

链表结点的插入，要求将一个通讯者记录的数据结点按其姓名插入到有序通讯表的相应位置，以保持通讯录的有序性。

插入的基本思想是：使用指针变量依次访问链表中的结点，直到找到插入位置。具体实现算法如下：

```
/*********在通讯录链表 head 中插入结点**********/
void Insert(LinkList head,ListNode *p)
{
    ListNode *s;
    s=head;
    while(s->next&&strcmp(s->next->data.name,p->data.name)<0)
        s=s->next;
    p->next=s->next;
    s->next=p;        //新结点插入到 s 结点之后
```