《计算机实践》—数据结构
上机实验题和要求

南京航空航天大学
自动化学院
2022 年 10 月

实验 4. 查找和排序实验

实验目的：

掌握顺序查找和二分查找的操作实现；掌握插入排序、选择排序和冒泡排序的相关操作实现。

（以下实验 4.1、实验 4.2、实验 4.3 任选一节完成）

实验 4.1 查找和排序的基本实验

1. 实验内容

实现查找表的顺序查找和二分查找；实现插入排序、选择排序和冒泡排序。

2. 实验要求

顺序查找利用顺序表进行，二分查找利用顺序存储的有序表进行，设计算法并编程实现此过程。设计菜单，选择排序方法对查找表的数据进行排序。实现 3 种排序算法时，可以使用随机数，或者键盘输入数据。

3. 输入输出要求

输入数据：查找操作时，建立输入处理，输入查找表里的数据；待查找的关键字要求从键盘输入。排序操作时，可随机生成待排序列，或者键盘输入待排序列，数据元素 10 个以内。

输出形式：分别输出查找表元素以及查找结果（找到的话返回关键字的位置，没找到返回相应的查找失败信息）。对排序操作，输出原始数据序列和每一趟排序序列。

4. 自行设计合理的交互界面，设计合理的算法验证方法。

5. 可选内容：输出查找和排序算法中的相关原操作的次数，如元素比较次数和元素移动次数等。

实验 4.2 查找和排序的应用—寻找舞会中的“落单客人”

1. 问题描述

“单身贵族”是中文对于单身人士的一种爱称。本题要求从上万人的大型舞会中找出落单的客人，以便给予特殊的照顾。

2. 实验要求

（1）输入说明：输入的第一行给出正整数 N（≤50000），表示已知伴侣（或夫妻）的对数；随后 N 行，每行给出一对伴侣（每人使用一个 5 位数字表示的 ID 号，范围 00000~99999），ID 之间用空格隔开；随后给出一个正整数 M（≤10000），表示参加舞会的总人数；随后一行给出这 M 个客人的 ID，以空格分开。输入数据应确保每个人没有重婚（或者一个人有两个或两个以上伴侣）的情况。

（2）输出说明：第一行输出落单客人（既包括“单身贵族”，也包括有伴侣，但独自一人参加舞会的客人）的总人数；随后第二行按照 ID 递增的顺序列出这些落单的客人。每个 ID 之间用 1 个空格分隔。

3. 测试用例

序号	输入	输出	说明
1	3 11111 22222 33333 44444 55555 66666 7 55555 44444 10000 88888 22222 11111 23333	5 10000 23333 44444 55555 88888	一般情况，有单身的，也有独自参加舞会的
2	4 00000 99999 00001 99998 50002 99997 50000 49999 6 00000 50000 99997 49999 99999 50002	0	无落单客人
3	1 10001 10002 1 10003	1 10003	最小 N 和 M
4	略	略	随机产生

4. 提示

（1）本题分为两个任务：1）读入并存储伴侣关系；2）对每个参加舞会的客人，检查其有无伴侣，若无，则为单身；否则找到其伴侣，判断其伴侣是否也参加舞会，若未参加，说明是落单客人。

（2）输出要求有序。可以将参加派对的人先进行排序，然后逐个判断是否为落单客户。查找非单身客人的伴侣时，也可以使用二分查找方法。

（3）存储伴侣关系时，可以有两种方法：1）使用图来存储；2）使用一个整数数组 Couple[]

来存储两个人 P1 和 P2 的关系，若是伴侣，则 Couple[P1]=Couple[P2]，Couple[P2]=Couple[P1]。数组元素的初始化值为-1。

（4）如果扫描到一个人，其伴侣也参加了舞会，可以将其伴侣的 Couple 值设置为特殊的标记（如-2），避免对于其伴侣进行重复的检索。

（5）为了便于调试，可以使用读文件的方式输入测试数据，参考网址（https://cplusplus.com/reference/cstdio/fscanf/）。

5. 自行设计合理的交互界面

6. 选作内容

分析算法的时间和空间复杂度。

实验 4.3 查找与排序算法的比较

1. 问题描述

在教材中，查找算法和排序算法的时间复杂度分析结果只给出了算法执行时间的阶，或大概执行时间。试通过随机数据比较各算法的时间复杂度（关键字比较次数、关键字移动次数），以取得直观感受。

2. 基本要求

（1）对常用的内部排序算法进行比较：直接插入排序、简单选择排序、冒泡排序和快速排序。

（2）利用随机函数产生 N（N=30000）个随机整数序列，作为输入数据作比较；比较的指标为关键字参加的比较次数和关键字的移动次数（关键字交换计为 3 次移动）。

（3）根据各排序方法的时间复杂度的理论值，对本实验得到的结果进行简要分析讨论。

（4）随机产生一个整数，分别实现顺序查找（在排序前的随机整数序列）和二分查找（排序后的随机整数序列），对比它们的查找时间是否符合理论值。

3. 测试数据

随机函数产生，可以参考（https://cplusplus.com/reference/cstdlib/rand/）。

4. 输入输出:

输入数据：参加排序的整数个数 N（N=30000，注：不得减少 N）；

输出数据：各种排序方法的关键字比较次数和移动次数（从小到大排列）。

5. 提示

（1）主要工作是设法在已知算法中适当位置插入对关键字的比较次数和移动次数的计数操作。注意采用分模块调试的方法。

（2）测试程序时，可以先设置 N 为一个较小值，程序调试正确后再将 N 设置为 30000。产生的随机整数序列可以用一个数组存放；测试各个排序方法时，使用另一个辅助数组复制原始随机整数序列，实现：1）所有排序方法使用同一组测试序列，2）避免对已经排序过的数据进行测试。

（3）快速排序算法

快速排序（Quick Sort）又称分区交换排序，它是对冒泡排序的一种改进，是目前已知的排序速度最快的排序方法之一。

快速排序方法的基本思想是任取待排记录序列中的某个记录（通常取第一个记录）作为**基准**，按照该记录的关键字大小，将整个记录序列划分为左右两个子序列：左侧子序列中所有记录的关键字都小于或等于基准记录的关键字；右侧子序列中所有记录的关键字都大于基准记录的关键字。基准记录则排在这两个子序列中间（这也是该记录最终应放的位置）。然后分别对这两个子序列重复实行上述过程，直到子序列长度不大于 1 为止。

在快速排序过程中，关键是如何选取记录的关键字 K_i 为基准，将一组记录划分为两个部分，即完成一趟快速排序（又称为一次划分）。

一次划分的具体做法是：从序列的两端开始交替扫描各个记录，即首先从序列末端开始向前搜索，并将关键字小于 K_i 的记录依次安置到序列的前边，然后从序列前段开始向后搜索，而将关键字大于 K_i 的记录依次安置到序列的后边。重复这两步，直到扫描完所有的记录。

假设使用数组 R 存放待排序记录。当前待排序记录的序列中的第一个记录的位置序号为 Low，最后一个记录的位置序号为 High。设两个变量 i 和 j，它们的初值分别是：i=Low，j=High。我们不妨设以第一个记录 R[Low]的关键字作为划分基准，则一趟快速排序的具体过程为：

（1）将记录 R[i]放在一个临时变量 x 中，腾空 R[i]所占的位置；

（2）从序列的 j 处开始，将 x.key 与 R[j].key 进行比较，若 x.key≤R[j].key，满足大小要求，无需移动记录，则令 j=j−1，然后继续进行比较下一个记录，直到 i=j 或者 x.key>R[j].key；若 x.key>R[j].key，则将 R[j]放到 R[i]处，腾空 R[j]所占的位置，并令 i=i+1，转向步骤（3）；

（3）从序列的 i 处开始，将 x.key 与 R[i].key 进行比较，若 x.key>R[i].key，满足大小

要求，无需移动记录，则令 i=i+1，然后继续进行比较下一个记录，直到 i=j 或者 x.key<R[i].key。若 x.key<R[i].key，则将 R[i]放到 R[j]处，腾空 R[i]所占的位置，并令 j=j−1，转向步骤（2）；

（4）重复步骤（2）和（3），直到 i=j，此时 i 就是记录 x 在序列中应放置的位置。

一趟快速排序的算法描述如下：

```
/* R 为存放待排序记录的数组，low 和 high 分别为待排序记录序列的起始和终止位置，
使用 R[0]作为暂存记录的临时变量*/
void  QuickOnePass(RecType R[], int low, int high)
{   int i,j;
    i=low;j=high;    //初始化 i 和 j
    R[0]=R[i];       //对应上述步骤（1），使用 R[0]暂存基准记录信息
    do{
        //对应上述步骤（2）
        while((R[j].key>=R[0].key)&&(j>i))
        {
            j--;
        }
        if(j>i)
        {
            R[i]=R[j];
            i++;
        }

        //对应上述步骤（3）
        while((R[i].key<=R[0].key)&&(j>i))
        {
            i++;
        }
        if(j>i)
        {
            R[j]=R[i];
            j--;
        }
    }while(i!=j);

    //对应上述步骤（4）
    R[i]=R[0];
}
```


按照这个过程，对待排序的记录的关键字序列：37，28，56，80，60，14，25，50，进行一趟快速排序的过程如图 1.4.1 所示（使用 37 作为基准，不包含数据暂存的过程）。

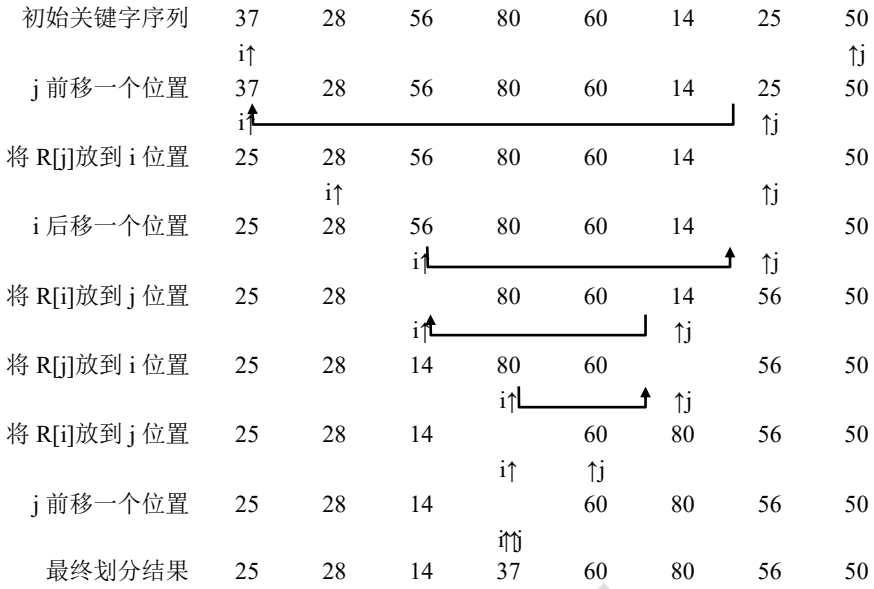

图 1.4.1　一趟快速排序过程的示例

当对待排序序列进行一趟快速排序之后，可采用同样的方法分别对基准记录两侧的两个子序列进行快速排序，直到各个子序列的记录个数都为 1 为止。这就完成了整个待排序序列的快速排序过程。对于上述例子，它整个的快速排序过程如图 1.4.2 所示。

初始记录的关键字序列	37	28	56	80	60	18	25	50
一趟快速排序后的序列	[25	28	14]	37	[60	80	56	50]
二趟快速排序后的序列	[14]	25	[28]	37	[50	56]	60	[80]
三趟快速排序后的序列	[14]	25	[28]	37	50	[56]	60	[80]
最后序列	14	25	[28]	37	50	[56]	60	[80]

图 1.4.3　快速排序示例

根据以上快速排序过程讨论，假设待排序记录放在 R[low]，R[low+1]，…，R[high]中，则快速排序的递归算法算法描述如下（由前述一次划分的算法扩充实现）：

```
/*快速排序的递归实现算法*/
   void  QuickSort(RecType R[],int low,int high)
   {  int i,j;
      i=low;j=high;

      R[0]=R[i];   //步骤（1）

      while(i<j)
      {
         //步骤（2）
         while((R[j].key>=R[0].key)&&(j>i))
         {
            j--;
         }
         if(j>i)
         {
            R[i]=R[j];
```

```
            i++;
        }
        //步骤（3）
        while((R[i].key<=R[0].key)&&(j>i))
        {
            i++;
        }
        if(j>i)
        {
            R[j]=R[i];
            j--;
        }
    }

    R[i]=R[0];    //步骤（4）。当前一次划分结束。

//对左侧子序列进行划分
if(low<i)
    QuickSort(R[],low,i-1);

//对右侧子序列进行划分
if(i<high)
    QuickSort(R[],j+1,high);
}
```


快速排序的时间主要耗费在划分操作上，对长度为 n 的序列进行快速排序所需要的比较次数 C(n)等于对长度为 n 的序列进行划分的比较次数 n−1 加上对两个子序列进行快速排序所需的比较次数。设划分后的一个子序列的长度为 k，则另一个子序列的长度为 n−1−k，即对 n 个记录进行快速排序所需要的比较次数可表示为：

$$C(n)=n-1+C(k)+C(n-1-k)$$

快速排序的最好情况是每次划分所得到的两个子序列长度大致相等（即两个子序列长度差不超过 1）。在这种情况下，快速排序过程中的比较次数 C(n)为

$$C(n) \leqslant n+2*C(n/2)$$
$$\leqslant n+2*[n/2+2*C(n/2/2)]$$
$$\leqslant 2n+4C(n/4)$$
$$\leqslant \cdots\cdots$$
$$\leqslant hn+2^h*(n/2^h)$$

当 $n/2^h=1$ 时，即当前待排序子序列的长度为 1 时，快速排序过程结束。因此，

$$C(n) \leqslant n\log_2 n+n*C(1)$$

其中，C(1)表示对长度为 1 的子序列进行快速排序所需要的比较次数，可以将它看作一个常数。故在最好的情况下，整个快速排序过程所需的比较次数 $C(n)=O(n\log_2 n)$。

快速排序的最坏情况是，每次划分所得到的子序列中有一个子序列为空，而另一个子序列的长度为 n−1。也就是说每次划分所选择的基准是当前待排序序列中最小（或最大）的记录关键字。这时需要进行 n−1 趟快速排序，整个排序过程中的比较次数为：

$$\text{比较次数}=\sum_{i=1}^{n-1}(n-i)=\frac{n(n-1)}{2}$$

当初始记录的关键字序列按照递增（或递减）序排列情况是快速排序的一种最坏情况。

由于快速排序过程中记录的移动次数不会超过关键字的比较次数，因此，快速排序的最好时间复杂度为 $O(n\log_2 n)$，最坏时间复杂度为 $O(n^2)$。

从所需要的附加空间来看，快速排序算法需要递归调用，系统内部需要一个栈来保存递归参量。当每次划分较为均匀时，栈的最大深度为$[\log_2 n]+1$(包括最外层参量进栈)；但若每次划分后极不平衡时，栈的最大深度为 n。所以最好情况下的空间复杂度为 $O(\log_2 n)$，最坏情况下的空间复杂度为 O(n)。就平均而言，空间复杂度仍为 $O(\log_2 n)$。

从排序的稳定性来看，快速排序是不稳定的。

6. 可选内容：

补充 1 种（或多种）其它更高效的进行排序方法，进行时间复杂度的分析，如希尔排序和归并排序等。