课程知识回顾与整理

查找

目录/CONTENTS

1 查找概论

2 顺序表查找

3 有序表查找

4 总结

01

查找概论

只要打开电脑，就会涉及到查找技术。如炒股软件中查股票信息、硬盘文件中找照片、在光盘中搜DVD，甚至玩游戏时在内存中查找攻击力、魅力值等数据修改用来作弊等，都要涉及到查找。当然，在互联网上查找信息就更加是家常便饭。所有这些需要被查的数据所在的集合，我们给它一个统称叫查找表。

查找表（Search Table）是由同一类型的数据元素（或记录）构成的集合。例如下图就是一个查找表。

关键字（Key）是数据元素中某个数据项的值，又称为键值，用它可以标识一个数据元素。也可以标识一个记录的某个数据项（字段），我们称为关键码，如下图中①和②所示。

若此关键字可以唯一地标识一个记录，则称此关键字为主关键字（Primary Key）。注意这也就意味着，对不同的记录，其主关键字均不相同。主关键字所在的数据项称为主关键码，如下图中③和④所示。

那么对于那些可以识别多个数据元素（或记录）的关键字，我们称为次关键字（Secondary Key）。次关键字也可以理解为是不以唯标识一个数据元素（或记录）的关键字，它对应的数据项就是次关键码。

查找（Searching）就是根据给定的某个值，在查找表中确定一个其关键字等于给定值的数据元素（或记录）。

02

顺序表查找

顺序查找（Sequential Search）又叫线性查找，是最基本的查找技术，它的查找过程是：从表中第一个（或最后一个）记录开始，逐个进行记录的关键字和给定值比较，若某个记录的关键字和给定值相等，则查找成功，找到所查的记录;如果直到最后一个（或第一个）记录，其关键字和给定值比较都不等时，则表中没有所查的记录，查找不成功。

顺序查找的算法实现如下：

```c
/* 顺序查找，a为数组，n为要查找的数组个数，key 为要查找的关键字 */
int Sequential_Search(int *a,int n,int key)
{
    int i;
    for (i=1;i<-n;i++)
    {
        if (a[i]==key)
            return i;
    }
    return 0;
}
```

这段代码非常简单，就是在数组a（注意元素值从下标 1 开始）中查看有没有关键字（key），当你需要查找复杂表结构的记录时，只需要把数组a与关键字key定义成你需要的表结构和数据类型即可。

到这里并非足够完美，因为每次循环时都需要对i是否越界，即是否小于等于n作判断。事实上，还可以有更好一点的办法，设置一个哨兵，可以解决不需要每次让i与n作比较。看下面的改进后的顺序查找算法代码。

```
1  /* 有哨兵顺序查找 */
2  int Sequential_Search2(int *a,int n,int key)
3  {
4      int i;
5      a[0]=key;    /* 设置a[0]为关键字值，我们称之为"哨兵" */
6      i=n;         /* 循环从数组尾部开始 */
7      while(a[i]!=key)
8      {
9          i--;
10     }
11     return i;    /* 返回0则说明查找失败 */
12 }
```

此时代码是从尾部开始查找，由于a[0]=key，也就是说，如果在a[i]中有key则返回i值，查找成功。否则一定在最终的a[0]处等于key，此时返回的是 0，即说明a[1]~a[n]中没有关键字key，查找失败。

这种在查找方向的尽头放置 “哨兵” 免去了在查找过程中每一次比较后都要判断查找位置是否越界的小技巧，看似与原先差别不大，但在总数据较多时，效率提高很大，是非常好的编码技巧。当然，“哨兵” 也不一定就一定要在数组开始， 也可以在末端。

03

有序表查找

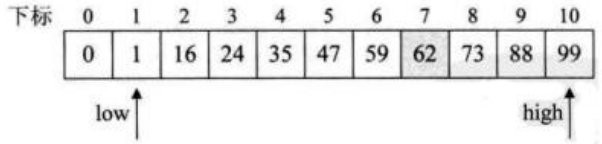
折半查找

折半查找（Binary Search）技术，又称为二分查找。它的前提是线性表中的记录必须是关键码有序（通常从小到大有序），线性表必须采用顺序存储。折半查找的基本思想是：在有序表中，取中间记录作为比较对象，若给定值与中间记录的关键字相等，则查找成功；若给定值小于中间记录的关键字，则在中间记录的左半区继续查找；若给定值大于中间记录的关键字，则在中间记录的右半区继续查找。不断重复上述过程，直到查找成功，或所有查找区域无记录，查找失败为止。

假设我们现在有这样一个有序表数组 {0,1,16,24,35,47,59,62,73,88,99}， 除 0 下标外共 10 个数字。对它进行查找是否存在 62 这个数。我们来看折半查找的算法是如何工作的。

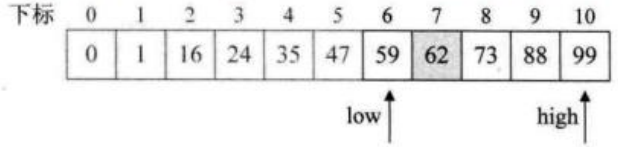
```c
/* 折半查找 */
int Binary_Search(int *a,int n,int key)
{
    int low, high, mid;
    low=1;      /* 定义最低下标为记录首位 */
    high=n;     /* 定义最高下标为记录末位 */
    while(low<=high)
    {
        mid=(low+high)/2;   /* 折半 */
        if (key<a[mid])     /* 若查找值比中值小 */
            high=mid-1;     /* 最高下标调整到中位下标小一位 */
        else if (key>a[mid])    /* 若查找值比中值大 */
            low=mid+1;      /* 最低下标调整到中位下标大一位 */
        else
            return mid;     /* 若相等则说明mid即为查找到的位置 */
    }
    return 0;
}
```

1. 程序开始运行，参数 `a={0,1,16,24,35,47,59,62,73,8899}`，`n=10`，`key=62`，第 3~5 行，此时 `low=1`，`high=10`，如下图所示。

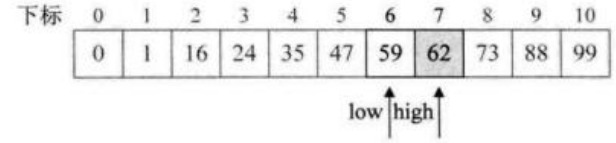


2. 第 6~15 行循环，进行查找。

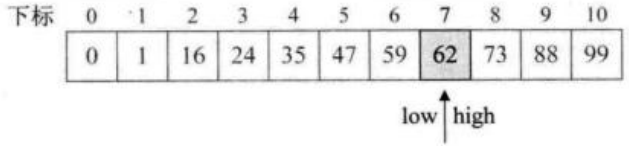
3. 第 8 行，`mid` 计算得 5，由于 `a[5]=47<key`，所以执行了第 12 行，`low=5+1=6`，如下图所示。



4. 再次循环，`mid=[6+10]/2=8`，此时 `a[8]=73>key`，所以执行第 10 行，`high=8-1=7`，如下图所示。



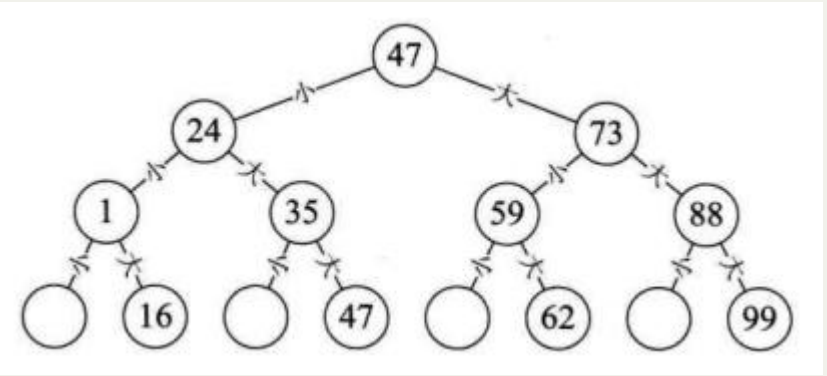
5. 再次循环，`mid=(6+7)/2=6`，此时 `a[6]=59<key`，所以执行 12 行，`low=6+1=7`，如下图所示。



6. 再次循环，`mid=(7+7)/2=7`，此时 `a[7]=62=key`，查找成功，返回 7。

该算法还是比较容易理解的，同时我们也能感觉到它的效率非常高。但到底高多少？关键在于此算法的时间复杂度分析。

首先，我们将这个数组的查找过程绘制成一棵二叉树，如下图所示，从图上就可以理解，如果查找的关键字不是中间记录 47 的话，折半查找等于是把静态有序查找表分成了两棵子树，即查找结果只需要找其中的一半数据记录即可，等于工作量少了一半，然后继续折半查找，效率当然是非常高了。



我们之前讲的二叉树的性质 4，有过对“具有 n 个结点的完全二叉树的深度为 $[\log_2 n]+1$”性质的推导过程。在这里尽管折半查找判定二叉树并不是完全二叉树，但同样相同的推导可以得出，最坏情况是查找到关键字或查找失败的次数为 $[\log_2 n]+1$。

有人还在问最好的情况？那还用说吗，当然是 1 次了。

因此最终我们折半算法的时间复杂度为 $O(logn)$，它显然远远好于顺序查找的 $O(n)$ 时间复杂度了。

不过由于折半查找的前提条件是需要有序表顺序存储，对于静态查找表，一次排序后不再变化，这样的算法已经比较好了。但对于需要频繁执行插入或删除操作的数据集来说，维护有序的排序会带来不小的工作量，那就不建议使用。

总结

ADD YOUR TITLE HERE

首先我们要弄清楚查找表、记录、关键字、主关键字、静态查找表、动态查找表等这些概念。

然后，对于顺序表查找来说，尽管很土（简单），但它却是后面很多查找的基础，注意设置“哨兵”的技巧，可以使得本已经很难提升的简单算法里还是提高了性能。

有序查找，着重讲了折半查找的思想，它在性能上比原来的顺序查找有了质的飞跃，由 $O(n)$ 变成了 $O(logn)$。之后我们又讲解了另外两种优秀的有序查找：插值查找和斐波那契查找，三者各有优缺点。

线性索引查找，讲了稠密索引、分块索引和倒排索引。索引技术被广泛的用于文件检索、数据库和搜索引擎等技术领域，是进一步学习这些技术的基础。

感 谢 聆 听