第二章·算法入门

第一节·简单查找算法

1. ACM/ICPC 算法预览。

在学习基础的算法之前，先让我们粗略的了解一下 ACM/ ICPC 之中常用的算法。见附录 A。

简单排序算法，是 ACM/ ICPC 常用算法中的基础中的基础。下面，我们开始学习简单排序算法。这可能是你在 ACM 备赛中学习到的第一个算法系列。

2. 简单的查找算法简介

2.1 什么是查找？

定义：

在一些（有序的/无序的）数据元素中，通过一定的方法找出与给定关键字相同的数据元素的过程叫做查找。

这个定义，有两个中心点：**数据元素**，**关键字**。也就是说，一个查找操作，肯定含有关键字，和把关键字包含在其中的数据元素。找准这两个中心点，对我们进行查找非常重要。

2.2 为什么查找？

为了获得目标元素，并对目标元素进行操作（增删等），从而达到一定目的的目的。

2.3 简单的查找算法简介

2.3.1 顺序查找(Sequence Search)

基本思想：在数据集合里，一个一个的遍历所有的数据，并按照数据的关键字比较，直到找到与目标关键字相同的数据。

2.3.2 折半查找(Binary Search)

基本思想：减少查找序列的长度，分而治之的进行关键紫的查找。先确定将被查找的数据元素集合所在的范围，然后逐渐缩小范围直至查找成功或失败。设查找数据的范围下限为 l=1，上限为 h=5，求中点 m=（l+h）/2，用 X 与中点元素 a[m]比较，若 X 等于 a[m]，即找到，停止查找；否则，若 X 大于 a[m]，替换下限 l=m+1，到下半段继续查找；若 X 小于 a[m]，换上限 h=m-1，到上半段继续查找；如此重复前面的过程直到找到或者 l>h 为止。如果 l>h，说明没有此数，打印找不到信息，程序结束。

3. 简单的查找算法的理解与用法详解

3.1 顺序查找(Sequence Search)

在 2.3.1 中已经对顺序查找有了了解。就是：从第一个元素开始，遍历所有的元素，找到匹配的元素后退出。它的一般实现如下列代码：

例一：

```
int sq_serch(keytype key[],int n,keytype
key)
{
    for(int i=0; i<n; i++)
    {
        if (key[i] == key)
        {
            return i;
        }
    }
    return -1;
}
```

这其中，n 表示待查找的数据元素的个数。Key 为要查找的元素的关键字。Key[]表示待查找元素的数据元素的关键字列表，如第 0 个元素的关键字是 key[0]。

优点：

顺序查找的优点在于简单直观，对于被查找的数据元素在整个数据元素集合中的位置没有要求。在数据无序且查找数据次数较低时，使用。

缺点：

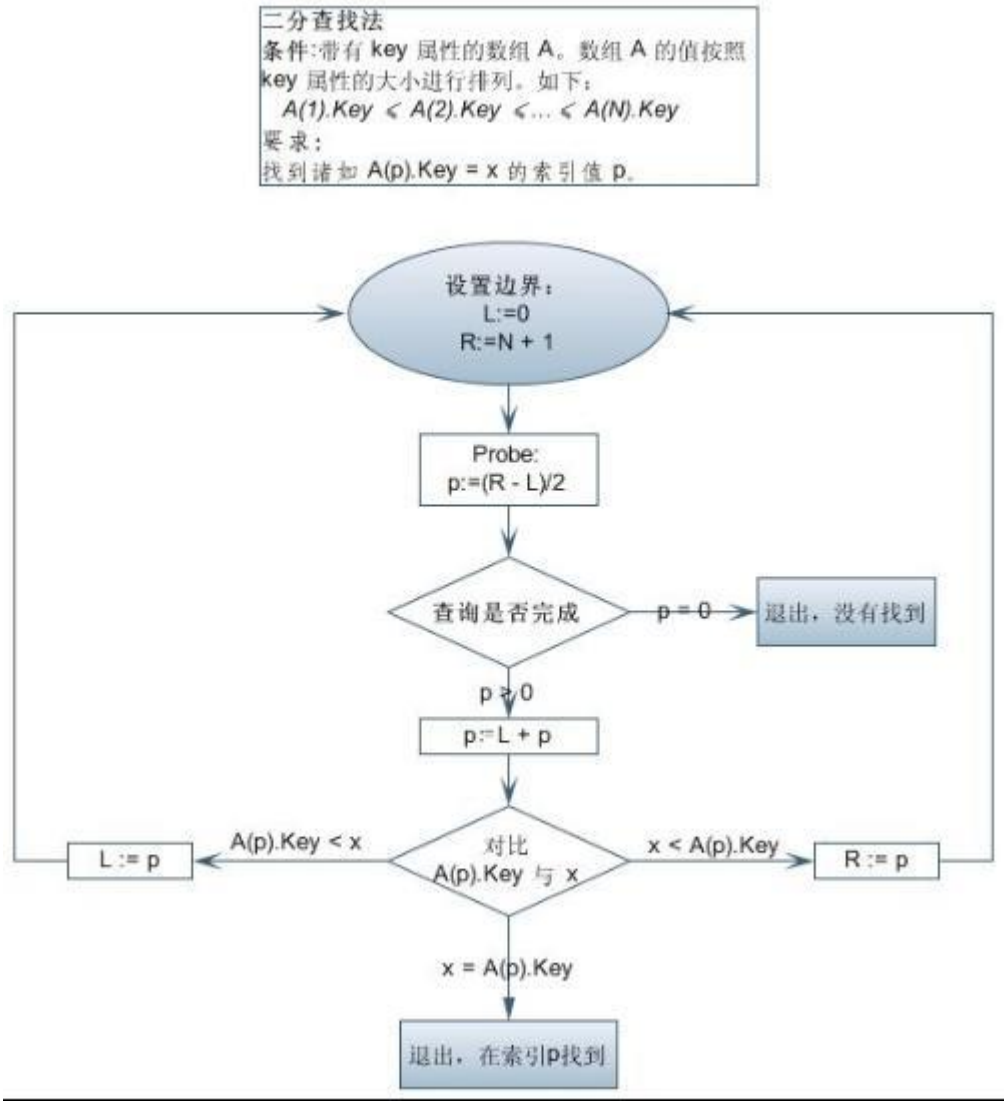
在查找范围较大时，效率极低。

评价：

除非没有别的办法，不然最好不要用。

3.2 折半查找(Binary Search)

折半查找，又称二分搜索。一般用在查找有序的数据元素集合。



以下，用一个例子详细理解折半查找。

例二：

现有一个序列：

K[11]={2 ,5 ,6 ,7 ,10 ,13 ,14 ,18 ,22 ,29 ,31 }

该序列包含 11 个元素，而且关键字单调递增。如果我们想查找关键字为 29 的元素。如果用顺序查找，需要从 2，开始一一遍历，共需比较 10 次。如果用折半查找那么我们可以这样做：

首先，设两个变量来“指向“这个序列的上下限。比如：

```
int head = k[0];

int tail = k[11];
```

然后，我们在设：

```
int mid = k[10/2];
```

假设要找的元素为 N=29：

1. 将 N 与 mid 作比较，由于 29>k[5]=13，所以，剩下的操作就只在 k[6]到 k[11]之间进行。将 head 调整到 k[6]的位置，将 mid 调整到 k[8]的位置 mid = k[(10+6)/2]。
2. 将 N 与 mid 作比较，由于 29>k[8]=22，所以，剩下的操作就只在 k[9]到 k[10]之间进行。将 head 调整到 k[9]的位置，将 mid 调整到 k[9]的位置 mid = k[(10+9)/2]。
3. 将 N 与 mid 作比较，由于 29 = k[10]=29，所以查找成功。

由此可以看到，使用折半查找，只需 3 次就可以找到目标元素。

折半查找的算法如下：

```
bin_search(int A[],int n,int key){
    int low,high,mid;
    low = 0;
    high = n-1;
    while(low<=high)
    {
        mid = (low + high)/2;
        if(A[mid]==key)return mid;              /*查找成功，返回mid*/
        if(A[mid]<key){
            low = mid + 1;                      /*在后半序列中查找*/
        }
        if(A[mid]>key){
            high = mid - 1;                     /*在前半序列中查找*/
        }
    }
    return -1;                                  /*查找失败，返回-1*/
}
```

优点：

比较次数少，查找速度快，平均性能好。

缺点：

要求待查找的数据为有序数据，且插入删除困难。

评价：

简单的查找算法里，效率比较可观。

4. 简单的查找算法的时间复杂度。

4.1 顺序查找

顺序查找的时间复杂度，在最好的情况下是 o(1),最坏是 o(n)。

4.2 折半查找

折半查找的时间复杂度，在最好的情况下是 o(1)，最坏是$\log_2(n)$

5. 简单的查找算法的空间复杂度。

5.1 顺序查找

顺序查找的空间复杂度是 o(n)。

5.2 折半查找

折半查找的空间复杂度是 o(n)。