# 查找

## 实验简介

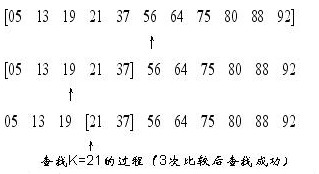
介绍二分查找和散列查找，二分查找是对于有序序列，每次都缩小一半查找范围的查找方法，而散列查找是关键字与在数据集中的位置一一对应，通过这种对应关系能快速地找到数据。

## 一、二分查找

前面几章我们已经讲了几种线性和非线性的数据结构，了解了数据的存储方式，这一章我们来讲下怎么查找这些数据。

顺序查找想必大家都知道，就是从头到尾比较数据集中的每一个数据，以此来获得想要的数据，但当数据集中拥有的数据较多时，这种方法的效率就会很低。

二分查找是比顺序查找效率高的一种查找算法，但它只适用于有序的数据集。二分查找也叫折半查找，它的查找步骤为：首先，假设表中元素是按升序排列，将表中间位置记录的关键字与查找关键字比较，如果两者相等，则查找成功；否则利用中间位置记录将表分成前、后两个子表，如果中间位置记录的关键字大于查找关键字，则进一步查找前一子表，否则进一步查找后一子表。重复以上过程，直到找到满足条件的记录，使查找成功，或直到子表不存在为止，此时查找不成功，如下图所示。



下面是二分查找的代码实现：

#include <stdio.h>#include <stdlib.h>

int BinarySearch(int \*array, int key, int low, int high){

int mid;

while (low <= high)

{

mid = (low + high) / 2;

if (key == array[mid])

{

return mid;

}

else if (key < array[mid])

{

high = mid - 1;

}

else

{

low = mid + 1;

}

}

return 0;

}

int main(){

int n, i, key, position;

int \*array;

printf("请输入有序数组的大小：");

scanf("%d", &n);

array = (int\*) malloc(sizeof(int) \* n);

printf("请按升序输入数据：\n");

for (i = 0; i < n; i++)

{

scanf("%d", &array[i]);

}

printf("请输入想要查找的数：");

scanf("%d", &key);

if (position = BinarySearch(array, key, 0, n - 1))

{

printf("%d的位置为：%d\n", key, position);

}

else

{

printf("%d不存在\n", key);

}

}

## 二、散列查找

### 1. 散列表

通常我们查找数据都是通过一个一个地比较来进行，那么有没有可能有这样的一种方法，要寻找的数据与其在数据集中的位置存在一种对应的关系，通过这种关系就能找到数据的位置。其实这种方法已经存在了，这个对应关系称为散列函数（哈希函数），由这个思想建立的表就称为散列表（哈希表）。

### 2. 散列函数的构造

要构造哈希表首先需要有散列函数，并且这个散列函数需要尽可能地减少冲突，通常有下面几种构造方法：

(1) 直接定址法

我们通过一个例子来讲解，如果我们现在要对0-20岁的进行人口统计，那么我们对年龄这个关键字就可以直接用年龄的数字作为地址。此时f(key) = key。



这个时候，我们可以得出这么个哈希函数：f(0) = 0，f(1) = 1，……，f(20) = 20。这个是根据我们自己设定的直接定址来的。人数我们可以不管，我们关心的是如何通过关键字找到地址。

如果我们现在要统计的是80后出生年份的人口数，那么我们对出生年份这个关键字可以用年份减去1980来作为地址。此时f (key) = key-1980。



假如今年是2000年，那么1980年出生的人就是20岁了，此时 f(2000) = 2000 - 1980，可以找得到地址20，地址20里保存了数据“人数500万”。

也就是说，我们可以取关键字的某个线性函数值为散列地址，即：

f(key) = a × key + b

这样的散列函数优点就是简单、均匀，也不会产生冲突，但问题是这需要事先知道关键字的分布情况，适合査找表较小且连续的情况。由于这样的限制，在现实应用中，直接定址法虽然简单，但却并不常用。

(2) 数字分析法

数字分析法是在知道关键字的情况下，取关键字的尽量不重复的几位值组成散列地址。

(3) 平方取中法

平方取中法就是取关键字平方后的中间几位为散列地址。

(4) 折叠法

折叠法是将关键字分为位数相等的几部分，最后一部分的位数可以不等，然后把这几部分的值（舍去进位）相加作为散列地址。

(5) 除留余数法

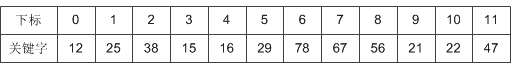
除留余数法此方法为最常用的构造散列函数方法。对于散列表长为m的散列函数公式为：

f( key ) = key mod p ( p ≤ m )

mod是取模（求余数）的意思。事实上，这方法不仅可以对关键字直接取模，也可在折叠、平方取中后再取模。

很显然，本方法的关键就在于选择合适的p，p如果选得不好，就可能会容易产生同义词。下面我们来举个例子看看：

有一个关键字，它有12个记录，现在我们要针对它设计一个散列表。如果采用除留余数法，那么可以先尝试将散列函数设计为f(key) = key mod 12的方法。比如29 mod 12 = 5，所以它存储在下标为5的位置。



不过这也是存在冲突的可能的，因为12=2×6=3×4。如果关键字中有像18(3×6)、30(5×6)、42(7×6)等数字，它们的余数都为6，这就和78所对应的下标位置冲突了。

使用除留余数法的一个经验是，若散列表表长为m，通常p为小于或等于表长（最好接近m)的最小质数或不包含小于20质因子的合数。实践证明，当P取小于散列表长的最大质数时，产生的散列函数较好。

(6) 随机数法

随机数法是选择一个随机函数，取关键字的随机函数值作为散列地址。

### 3. 处理冲突

前面在散列函数的构造中我们发现散列地址可能会产生冲突，所以处理冲突也是构造散列表中重要的一部分，通常处理冲突的方法有下面几种：

(1) 开发定址法

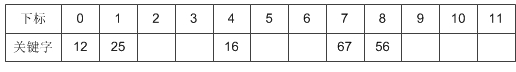
所谓的开放定址法就是一旦发生了冲突，就去寻找下一个空的散列地址，只要散列表足够大，空的散列地址总能找到，并将记录存入，公式为：

fi(key) = (f(key)+di) MOD m (di=1,2,3,......,m-1)

用开放定址法解决冲突的做法是：当冲突发生时，使用某种探测技术在散列表中形成一个探测序列。沿此序列逐个单元地查找，直到找到给定的关键字，或者碰到一个开放的地址（即该地址单元为空）为止（若要插入，在探查到开放的地址，则可将待插入的新结点存入该地址单元）。查找时探测到开放的地址则表明表中无待查的关键字，即查找失败。

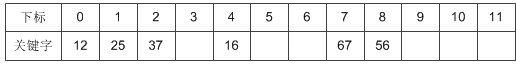
比如说，我们的关键字集合为{12,67,56,16,25,37,22,29,15,47,48,34}，表长为12。 我们用散列函数f(key) = key mod l2。

当计算前S个数{12,67,56,16,25}时，都是没有冲突的散列地址，直接存入：

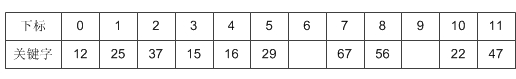


计算key = 37时，发现f(37) = 1，此时就与25所在的位置冲突。

于是我们应用上面的公式f(37) = (f(37)+1) mod 12 = 2。于是将37存入下标为2的位置。这其实就是房子被人买了于是买下一间的作法：。



接下来22,29,15,47都没有冲突，正常的存入：



到了 key=48，我们计算得到f(48) = 0，与12所在的0位置冲突了，不要紧，我们f(48) = (f(48)+1) mod 12 = 1，此时又与25所在的位置冲突。于是f(48) = (f(48)+2) mod 12=2，还是冲突……一直到 f(48) = (f(48)+6) mod 12 = 6时，才有空位，机不可失，赶快存入：

IMG_263

我们把这种解决冲突的开放定址法称为线性探测法。

**二次探测法**

考虑深一步，如果发生这样的情况，当最后一个key=34，f(key)=10，与22所在的位置冲突，可是22后面没有空位置了，反而它的前面有一个空位置，尽管可以不断地求余数后得到结果，但效率很差。

因此我们可以改进di = 12, -12, 22, -22,……, q2, -q2 (q <= m/2)，这样就等于是可以双向寻找到可能的空位置。

对于34来说，我们取di即可找到空位置了。另外增加平方运算的目的是为了不让关键字都聚集在某一块区域。我们称这种方法为二次探测法。

fi(key) = (f(key)+di) MOD m (di = 12, -12, 22, -22,……, q2, -q2, q <= m/2)

**随机探测法**

还有一种方法，是在冲突时，对于位移量di采用随机函数计算得到，我们称之为随机探测法。

此时一定会有人问，既然是随机，那么查找的时候不也随机生成吗？如何可以获得相同的地址呢？这是个问题。这里的随机其实是伪随机数。

伪随机数是说，如果我们设置随机种子相同，则不断调用随机函数可以生成不会重复的数列，我们在査找时，用同样的随机种子，它每次得到的数列是相同的，相同的di当然可以得到相同的散列地址。

fi(key) = (f(key)+di) MOD m (di是一个随机数列)

总之，开放定址法只要在散列表未填满时，总是能找到不发生冲突的地址，是我们常用的解决冲突的办法。

(2) 再哈希法

再哈希法是当散列地址冲突时，用另外一个散列函数再计算一次，这种方法减少了冲突，但增加了计算的时间。

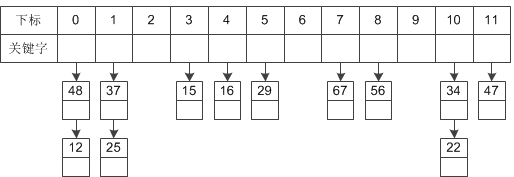
(3) 链地址法

前面我们谈到了散列冲突处理的开放定址法，它的思路就是一旦发生了冲突，就去寻找下一个空的散列地址。那么，有冲突就非要换地方吗？我们直接就在原地处理行不行呢？

可以的，于是我们就有了链地址法。

将所有关键字散列地址相同的记录存储在一个单链表中，我们称这种表为同义词子表，在散列表中只存储所有同义词子表的头指针。

对于关键字集合{12,67,56,16,25,37, 22,29,15,47,48,34}，我们用12为除数，进行除留余数法：



此时，已经不存在什么冲突换址的问题，无论有多少个冲突，都只是在当前位置给单链表增加结点的问题。链地址法解决冲突的做法是：将所有关键字散列地址相同的结点链接在同一个单链表中。若选定的散列表长度为m，则可将散列表定义为一个由m个头指针组成的指针数组T[0..m-1]。凡是散列地址为i的结点，均插入到以T[i]为头指针的单链表中。T中各分量的初值均应为空指针。在拉链法中，装填因子α可以大于1，但一般均取α≤1。

链地址法的优势是对于可能会造成很多冲突的散列函数来说，提供了绝不会出现找不到地址的保障。当然，这也就带来了査找时需要遍历单链表的性能损耗，不过性能损耗在很多场合下也不是什么大问题。

(4) 建立公共溢出区

这种方法的基本思想是：将散列表分为基本表和溢出表两部分，凡是和基本表发生冲突的元素，一律填入溢出表。

## 三、小结

这一章我们讲了二分查找和散列查找，二分查找是对于有序序列，每次都缩小一半查找范围的查找方法，而散列查找是关键字与在数据集中的位置一一对应，通过这种对应关系能快速地找到数据，散列查找中散列函数的构造和处理冲突的方法尤为重要，常见散列函数的构造方法有直接定址法、数字分析法、平方取中法、折叠法、除留余数法和随机数法，处理冲突的方法有开放定址法、再哈希法、链地址法和建立公共溢出区。

## 作业

1、代码实现散列查找，散列函数使用除留余数法，冲突处理方法使用开放地址法中的二次探测再散列。

2、在一个二维数组中，每一行都按照从左到右递增的顺序排序，每一列都按照从上到下递增的顺序排序。请完成一个函数，输入这样的一个二维数组和一个整数，判断数组中是否含有该整数。

3、思考：有 1000 瓶药水，其中 1 瓶有剧毒，一滴毙命，但 24 小时才生效。提供数只小狗做实验，在24小时内找出有毒的药水。至少需要多少只小狗？