数据结构与算法（10）：查找

一、基本概念

查找（Search）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）。在计算机应用中，查找是常用的基本运算，例如编译程序中符号表的查找。先说明几个概念：

查找表（Search Table）：由同一类型的数据元素（或记录）构成的集合
关键字（Key）：数据元素中某个数据项的值，又称为键值。
主键（Primary Key）：可唯一地标识某个数据元素或记录的关键字。

平均查找长度（Average Search Length，ASL）：需和指定key进行比较的关键字的个数的期望值，称为查找算法在查找成功时的平均查找长度。

对于含有n个数据元素的查找表，查找成功的平均查找长度为：$ASL = P_i * C_i$的和。

- P_i：查找表中第i个数据元素的概率。
- C_i：找到第i个数据元素时已经比较过的次数。

查找表按照操作方式可分为：

1. 静态查找表（Static Search Table）:只做查找操作的查找表。它的主要操作是：
2. 查询某个“特定的”数据元素是否在表中
3. 检索某个“特定的”数据元素和各种属性
4. 动态查找表（Dynamic Search Table）：在查找的同时进行插入或删除等操作：
5. 查找时插入数据
6. 查找时删除数据

按照查找表是否有序分为无序查找和有序查找：

- 无序查找：被查找数列有序无序均可；
- 有序查找：被查找数列必须为有序数列。

二、无序表查找

说明：顺序查找适合于存储结构为顺序存储或链接存储的线性表。

基本思想：顺序查找也称为线形查找，属于无序查找算法。从数据结构线形表的一端开始，顺序扫描，依次将扫描到的结点关键字与给定值k相比较，若相等则表示查找成功；若扫描结束仍没有找到关键字等于k的结点，表示查找失败。

算法分析：最好情况是在第一个位置就找到了，此为O(1)；最坏情况是在最后一个位置才找到，此为O(n)；所以平均查找次数为$(n+1)/2$，最终时间复杂度为$O(n)$

```
# 最基础的遍历无序列表的查找算法
# 时间复杂度O(n)

def sequential_search(lis, key):
    length = len(lis)
    for i in range(length):
        if lis[i] == key:
            return i
        else:
            return False
if __name__ == '__main__':
    LIST = [1, 5, 8, 123, 22, 54, 7, 99, 300, 222]
    result = sequential_search(LIST, 123)
    print(result)
```

三、有序表查找

查找表中的数据必须按照某个主键进行某种排序！

3.1 二分查找

说明：元素必须是有序的，如果是无序的则要先进行排序操作。
基本思想：也称为折半查找，属于有序查找算法。用给定值k先与中间结点的关键字比较，中间结点把线性表分成两个子表，若相等则查找成功；若不相等，再根据k与该中间节点关键字的比较结果确定下一步查找哪个字表，这样递归进行，知道查找到或查找结束发现表中没有这样的结点。

复杂度分析：最坏情况下,关键字比较次数为$log_2(n+1)$，且期望时间复杂度为$O(log_2 n)$

注意：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，一次排序后不再发生变化，折半查找能得到不错的效率。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。

```python
# 针对有序查找表的二分查找算法
# 时间复杂度O(log(n))

def binary_search(list, key):
    low = 0
    high = len(list) - 1
    time = 0
    while low <= high:
        time += 1
        mid = int((low + high) / 2)
        if key < list[mid]:
            high = mid - 1
        elif key > list[mid]:
            low = mid + 1
        else:
            # 打印折半的次数
            print("times: %s" % time)
            return mid
    print("times: %s" % time)
    return -1

if __name__ == '__main__':
    LIST = [1, 5, 7, 8, 22, 54, 99, 123, 200, 222, 444]
    result = binary_search(LIST, 99)
    print(result)
```

3.2 插值查找

在介绍插值查找之前，首先考虑一个新问题，为什么上述算法一定要是折半，而不是折四分之一或者折更多呢?

打个比方，在英文字典里面查“apple”，你下意识翻开字典是翻前面的书页还是后面的书页呢?如果再让你查“zoo”，你又怎么查？很显然，这里绝对不会是从中农间开始查起，而是有一定目的的往前或往后翻。

同样的，比如要在取值范围1~10000之间100个元素从小到大均匀分布的数组中查找5，我们自然会从数组下标较小的开始查找。

经过上面的分析，折半查找这种查找方式，不是自适应的（也就是说是傻瓜式的）。二分查找中查找点计算如下:

$$mid = (low + high)/2$$

，即

$$mid = low + (high - low)/2$$

通过类比，我们可以将查找的点改进为如下：

$$mid = low + \frac{key - list[low]}{list[high] - list[low]} \times (high - low)$$

也就是将上述的比例参数1/2改进为自适应的，根据关键字在整个有序表中所处的位置，让mid值的变化更靠近关键字key，这样也就间接地减少了比较次数。

基本思想：基于二分查找算法，将查找点的选择改进为自适应选择，可以提高查找效率。当然，插值查找也属于有序查找。

注意：对于表长较大，而关键字分布又比较均匀的查找表来说，插值查找算法的平均性能比折半查找要好的多。反之，数组中如果分布非常不均匀，那么插值查找未必是很合适的选择。

复杂度分析：查找成功或者失败的时间复杂度均为$O(log2(log2n))$。

```
# 插值查找算法
# 时间复杂度O(log(n))

def binary_search(lis, key):
    low = 0
    high = len(lis) - 1
    time = 0
    while low <= high:
        time += 1
        # 计算mid值是插值算法的核心代码
        mid = low + int((high - low) * (key - lis[low])/(lis[high] - lis[low]))
        print("mid=%s, low=%s, high=%s" % (mid, low, high))
        if key < lis[mid]:
            high = mid - 1
        elif key > lis[mid]:
            low = mid + 1
        else:
            # 打印查找的次数
            print("times: %s" % time)
            return mid
    print("times: %s" % time)
    return -1

if __name__ == '__main__':
    LIST = [1, 5, 7, 8, 22, 54, 99, 123, 200, 222, 444]
    result = binary_search(LIST, 444)
    print(result)
```

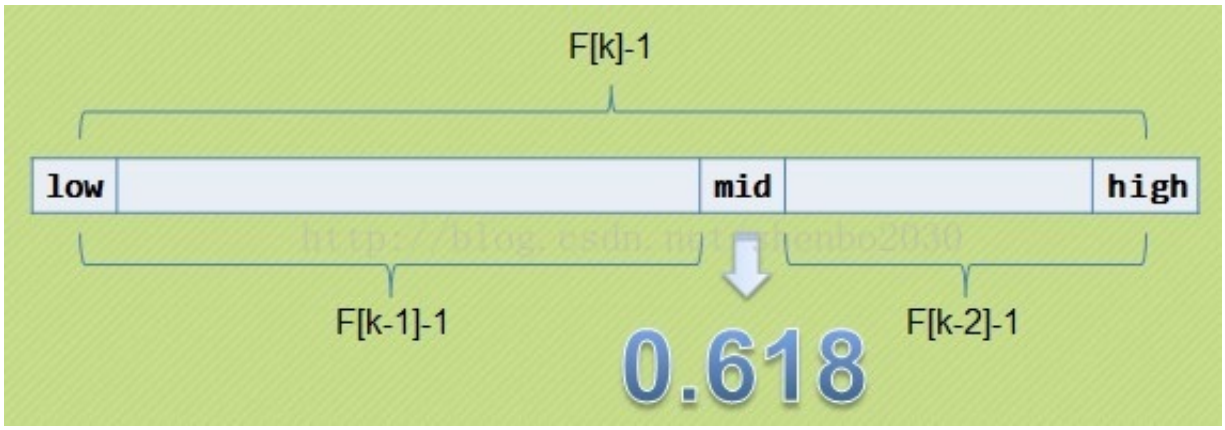
3.3 斐波那契查找

在介绍斐波那契查找算法之前，我们先介绍一下和它很紧密相连并且大家都熟知的一个概念——黄金分割。

黄金比例又称为黄金分割，是指事物各部分间一定的数学比例关系，即将整体一分为二，较大部分与较小部分之比等于整体与较大部分之比，其比值约为1：0.618。

0.618倍公认为是最具有审美意义的比例数字，这个数值的作用不仅仅体现在诸如绘画、雕塑、音乐、建筑等艺术领域，而且在管理、工程设计等方面有着不可忽视的作用。因此被称为黄金分割。

大家记不记得斐波那契数列：1，1，2，3，5，8，13，21，34，55，89.....（从第三个数开始，后面每一个数都是前两个数的和）。然后我们会发现，随着斐波那契数列的递增，前后两个数的比值会越来越接近0.618，利用这个特性，我们就可以将黄金比例运用到查找技术中。



基本思想：也是二分查找的一种提升算法，通过运用黄金比例的概念在数列中选择查找点进行查找，提高查找效率。同样地，斐波那契查找也属于一种有序查找算法。

相对于折半查找，一般将待比较的key值与第mid=（low+high）/2位置的元素进行比较，比较结果分为三种情况：

1. 相等，mid位置的元素即为所求
2. >,low=mid+1
3. <,high=mid-1

斐波那契查找和折半查找很相似，它是根据斐波那契序列的特点对有序表进行分割的。他要求开始表中记录的个数为某个斐波那契小1，即$n = F(k) - 1$

开始将k值与第F(k-1)位置的记录进行比较（即mid=low+F(k-1)-1），比较结果也分为三种

1. 相等，mid位置的元素即为所求
2. >,low=mid+1,k-=2：说明，low=high+1说明待查找的元素在[mid+1,high]范围内，k-=2说明

范围[mid,high]内的元素个数为
$n-(F(k-1))=Fk-1-F(k-1)=Fk-F(k-1)-1=f(k-2)-1$个，所以可以递归地应用斐波那契查找。

3. <,high=mid-1,k-=1:说明，low=mid+1，说明待查找的元素在[mid+1,high]范围内，k-=1说明范围[low,mid-1]内的元素个数为F(k-1)-1个，所以可以递归的应用斐波那契查找

复杂度分析：最坏情况下，时间复杂度为$O(log_2n)$，且其期望复杂度也为$O(log_2n)$。就平均性能，要优于二分查找。但是在最坏情况下，比如这里如果key为1，则始终处于左侧半区查找，此时其效率要低于二分查找。

总结：二分查找的mid运算是加法与除法，插值查找则是复杂的四则运算，而斐波那契查找只是最简单的加减运算。在海量数据的查找中，这种细微的差别可能会影响最终的查找效率。因此，三种有序表的查找方法本质上是分割点的选择不同，各有优劣，应根据实际情况进行选择。

四、线性索引查找

对于海量的无序数据，为了提高查找速度，一般会为其构造索引表。索引就是把一个关键字与他相对应的记录进行关联的过程。

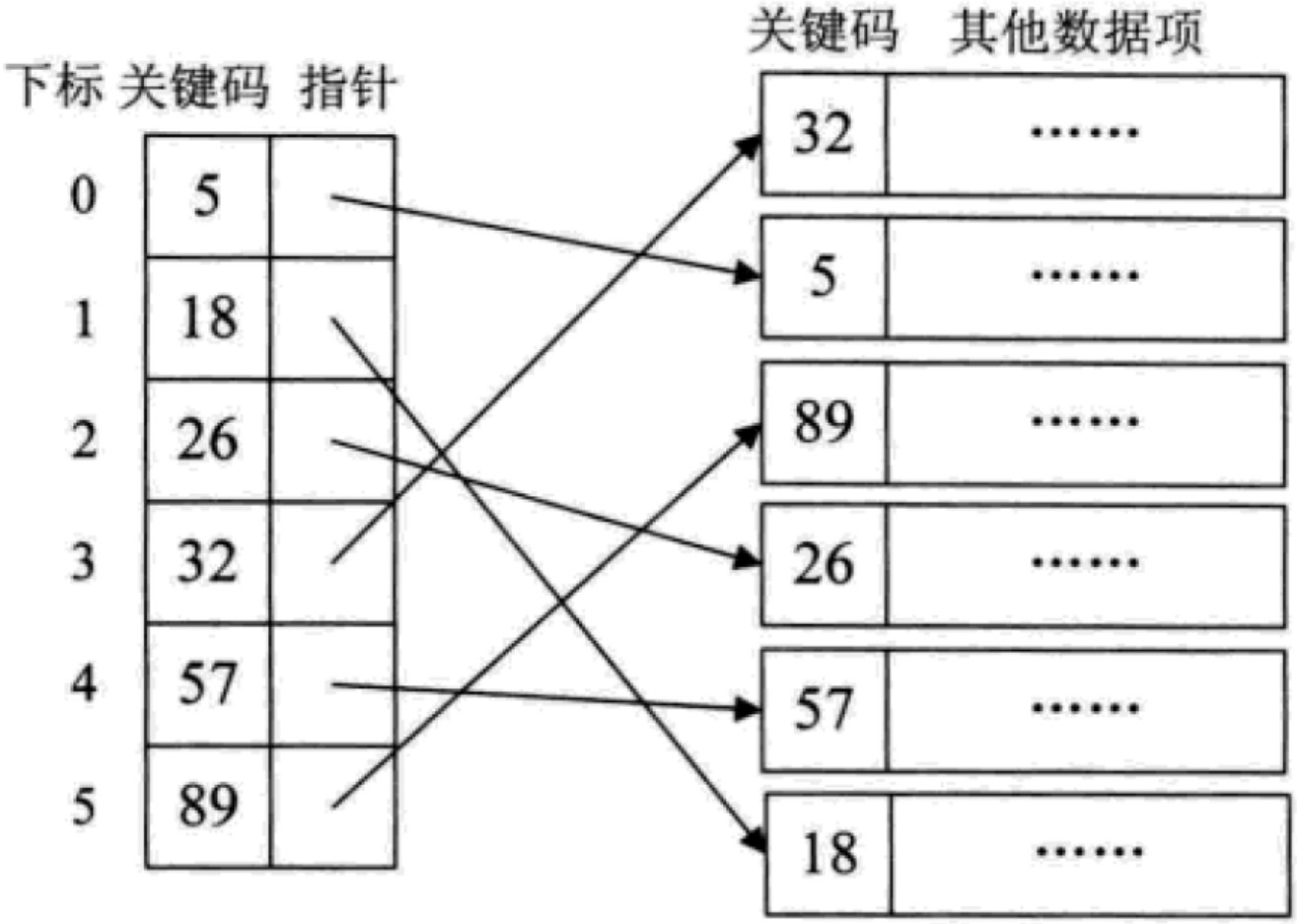
一个索引由若干个索引项构成，每个索引项至少包含关键字和其对应的记录在存储器中的位置等信息。

索引按照结构可以分为：线性索引、树形索引和多级索引。
线性索引：将索引项的集合通过线性结构来组织，也叫索引表

线性索引可分为：稠密索引、分块索引和倒排索引。

- 稠密索引：

指的是在线性索引中，为数据集合中的每个记录都建立一个索引项。



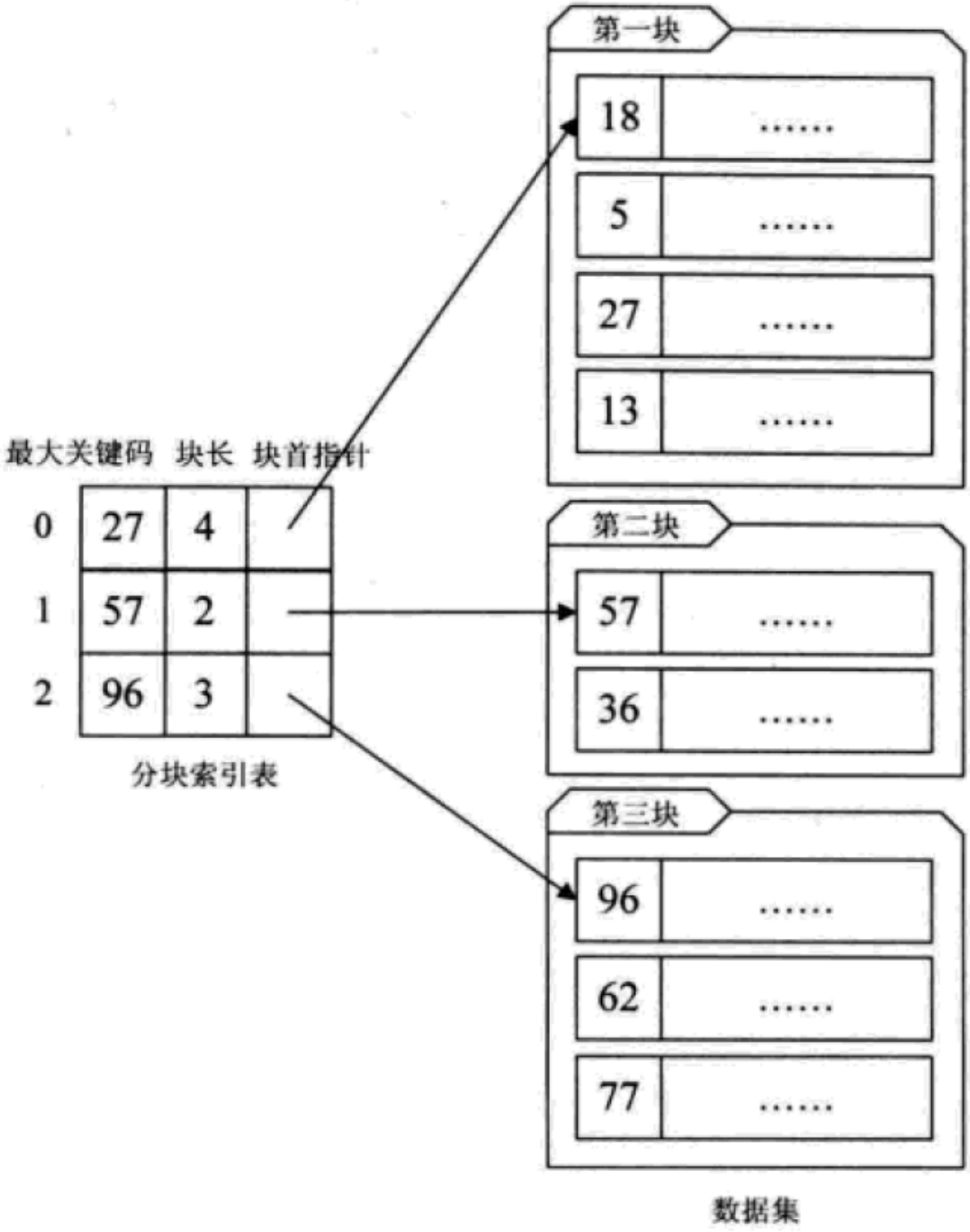
这其实就相当于给无序的集合，建立了一张有序的线性表，其索引项一定是按照关键码进行有序的排列。这也相当于把查找过程中需要的排序工作给提前做了。

- 分块索引：

分块查找又称索引顺序查找，是顺序查找的一种改进方法。

算法思想：将n个数据元素“按块有序”划分为m块（m<n）。每一块中的结点不必有序，但块与块之间必须“按块有序”；即第一块中人艺元素的关键字都必须小于第2块中任一元素的关键字；而第二块中任一元素又都必须小于第三块中的任一元素，.....

算法流程：
首先选取各块中的最大关键字构成一个索引表；查找分为两个部分：先对索引表进行二分查找或顺序查找，以确定待查记录在哪一块中；然后，在已确定的块中用顺序法进行查找。



这其实是有序查找和无序查找的一种中间状态或者说妥协状态。因为数据量过大，建立完整的稠密索引耗时耗力，占用资源过多；但如果不做任何排序或索引，那么遍历的查找也无法接受，只能这种，做一定程度的排序或索引。分块索引的效率比遍历查找的$O(n)$要高一些，但与二分查找的$O(logn)$还是要差不少。

- 倒排索引： 不是由记录来确定属性值，而是由属性值来确定记录的位置，这种被称为倒排索引。其中记录号表存储具有相同关键字的所有记录的地址或引用（可以是指向记录的指针或该记录的主关键字）。倒排索引是最基础的搜索引擎技术。