**数据结构与算法笔记**

讨论一个算法的快慢时，我们考虑的是随着输入规模的增加，其操作数（或运行时间）会以什么样的速度增长。

数组和链表的优缺点：

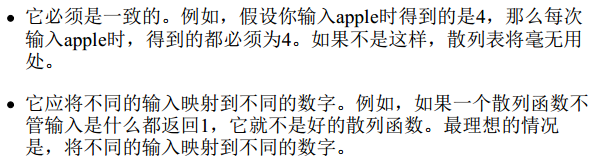
数组：支持随机访问（立即访问到一个随机位置的元素），但是需要申请一大段连续内存，插入元素可能会导致整个数组转移到其他地方。（可以预留，但是有两个问题：一是预留的可能用不上造成浪费，二是预留的位置可能还是会被用完从而整个数组要转移）

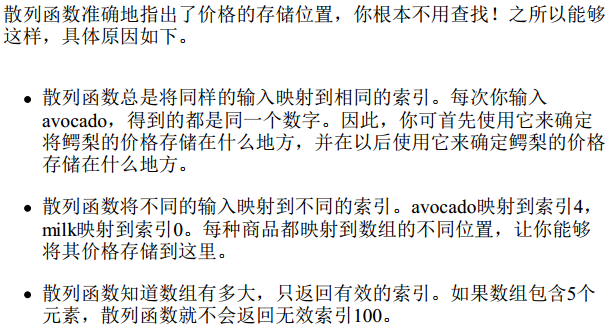
链表：只能顺序访问（逐个元素读取），但是插入和删除元素操作很快。

每个递归都需要有两个部分：停止条件（或称基线条件）和递归条件。基线条件避免递归形成无限循环。

散列函数：无论你给它什么数据，它都还你一个数字。

散列函数必须满足一些要求：





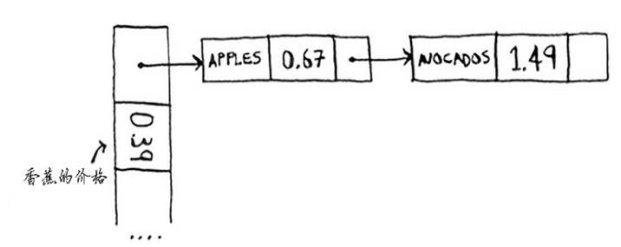
散列函数和数组结合起来创建了被称为散列表的数据结构（也称哈希表）。

在需要创建映射或者需要查找时，散列表时很不错的选择。

散列表也被用于网站缓存数据。

冲突：散列表给两个键分配了相同的内存地址。

处理冲突：如果两个键映射到同一位置，就在这个位置存储一个链表。



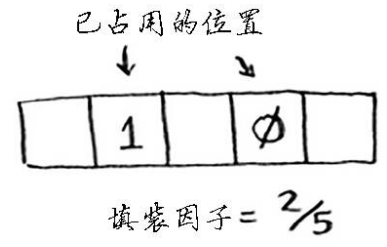
平均情况下散列表的访问速度跟数组一样快，插入和删除的速度和链表一样快，所以只要避免最糟糕的情况，它就可以兼具两者的优点。

避免冲突的方法：

1.需要较低的填装因子

2.需要好的散列函数（散列函数很重要。最理想的情况是散列函数把键均匀地映射到散列表地不同位置）

填装因子：散列表包含的元素数 / 位置数

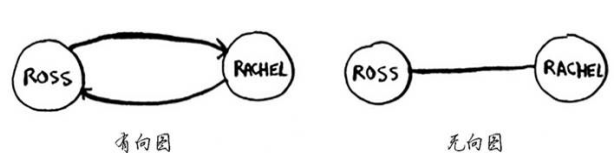


它度量的是散列表中有多少位置是空的。

填装因子大于1说明位置是不够的，这时候需要调整散列表的长度（通常是把数组增长一倍）

填装因子越低代表发生冲突的可能性越小，散列表的性能越高。一个不错的经验规则是：填装因子大于0.7就增大散列表的长度。

BFS算法可以用来计算最短路径问题。



无向图的关系是双向的。

BFS的时候，你要把检查过的人标记，不再检查他，如果不这样做可能会导致无限循环。

BFS的时间复杂度：

