**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：

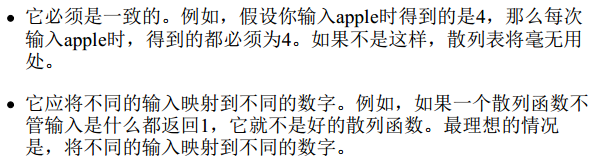
数组：支持随机访问（立即访问到一个随机位置的元素），但是需要申请一大段连续内存，插入元素可能会导致整个数组转移到其他地方。（可以预留，但是有两个问题：一是预留的可能用不上造成浪费，二是预留的位置可能还是会被用完从而整个数组要转移）

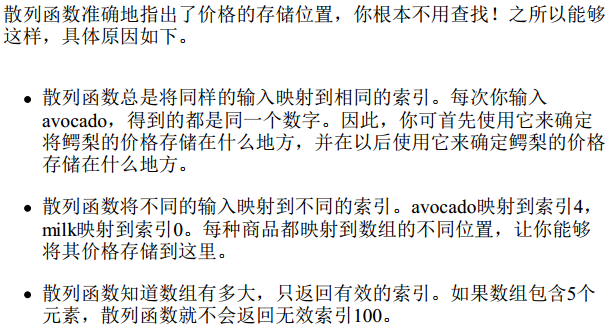
链表：只能顺序访问（逐个元素读取），但是插入和删除元素操作很快。

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：





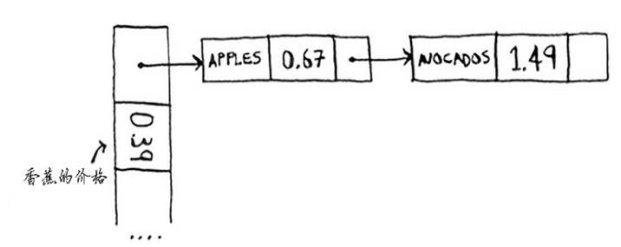
散列函数和数组结合起来创建了被称为散列表的数据结构（也称哈希表）。

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。



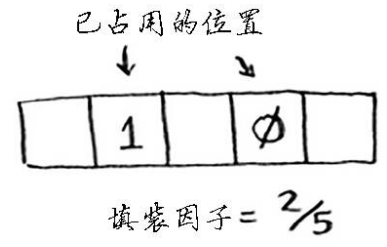
平均情况下散列表的访问速度跟数组一样快，插入和删除的速度和链表一样快，所以只要避免最糟糕的情况，它就可以兼具两者的优点。

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

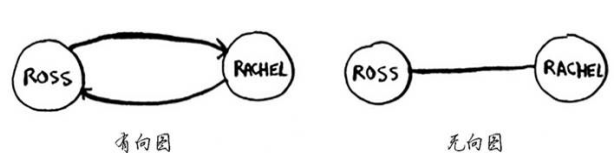


它度量的是散列表中有多少位置是空的。

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。

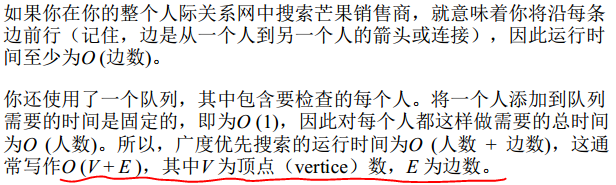
BFS算法可以用来计算最短路径问题。

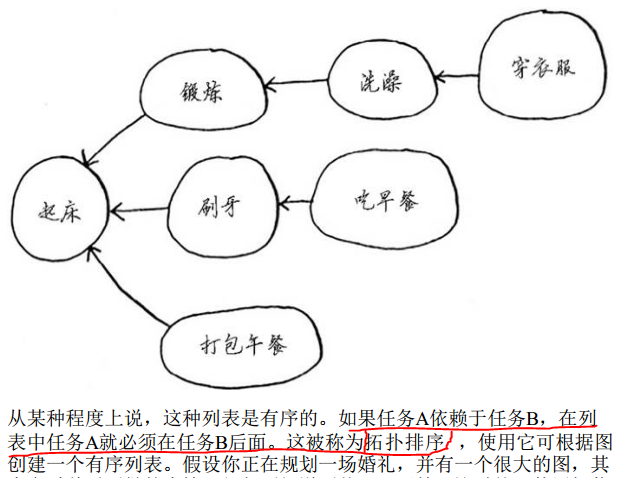


无向图的关系是双向的，上述两个图是等价的，而且他们其实都是环。

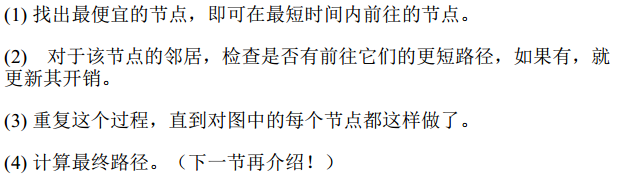
BFS的时候，你要把检查过的人标记，不再检查他，如果不这样做可能会导致无限循环。

BFS的时间复杂度：





迪杰斯特拉算法（最短路径）的步骤：

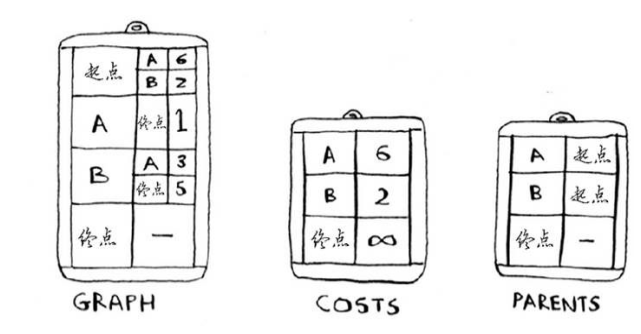


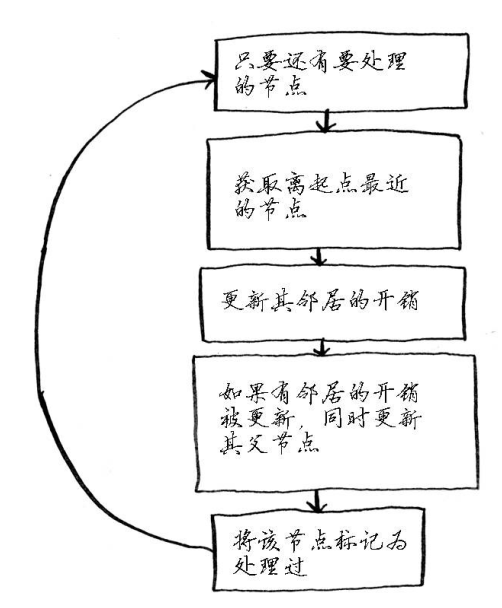
无向图中每条边都是一个环，迪杰斯特拉算法只适用于**有向无环图。**

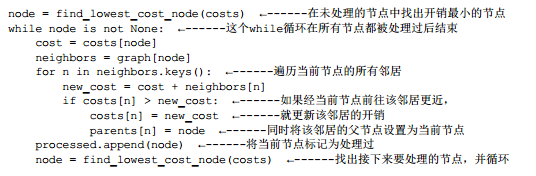
如果有负权边，迪杰斯特拉算法也会不管用。

原因是该算法这样假设：对于处理过的节点，没有前往该节点的更短的路径。这种假设只有在没有负权边的时候才成立。对于含有负权边的图，可以使用另一种算法来找出最短路径，**Bellman-Ford算法**。

要实现迪杰斯特拉算法需要创立三个散列表：







贪婪算法：

每一步都采取最优的做法（局部最优解），最终得到全局最优解。该算法容易理解和实现，但是并不是任何情况下都行之有效。

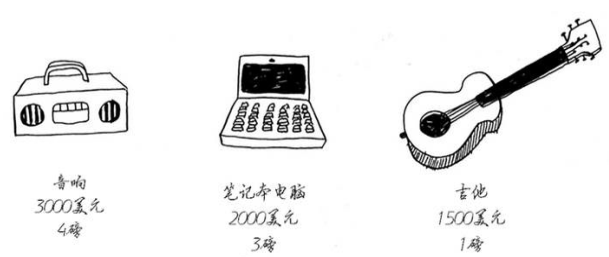
尽管有时候贪婪算法得不到最优解，但也能得到与之相当接近的解。

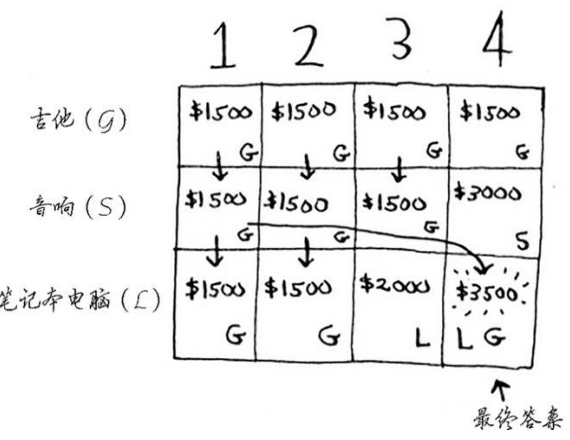
判断近似算法优劣的标准有两个：1.它的速度有多快 2.得到的近似解跟最优解有多接近

旅行商问题的复杂度是n!，属于NP完全问题，但是可以用贪婪算法得到近似的解。

背包问题可以使用动态规划来解决，先解决子问题(小背包最多能装多少价值)，然后逐步解决原来的问题。

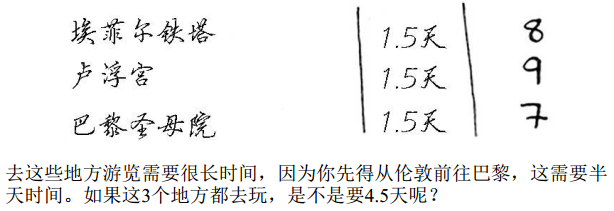
例：列是背包空间，行是物品。

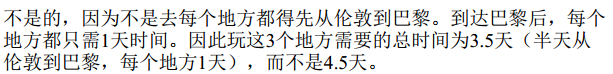




最终答案不会随着行的排列顺序变化而产生变化。

如果可以偷商品的一部分，那么可以使用贪婪算法。

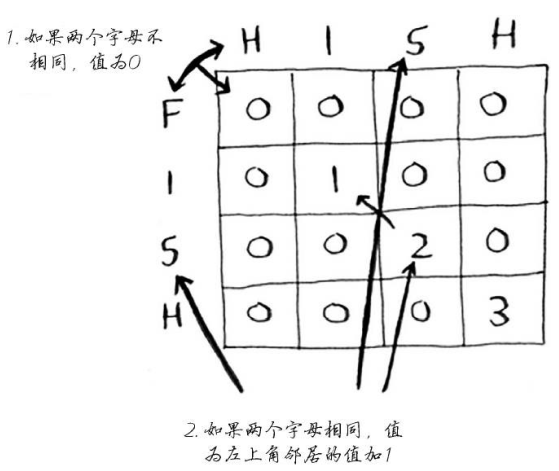


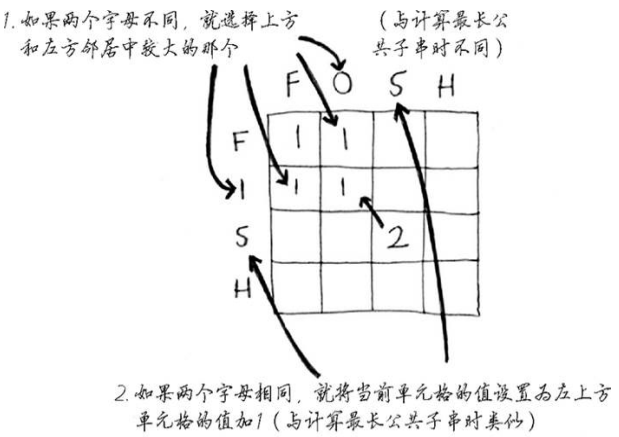


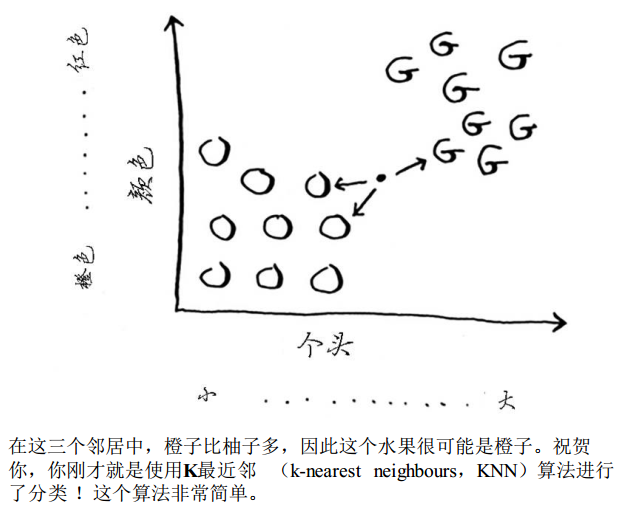
此时子问题之间有依赖。

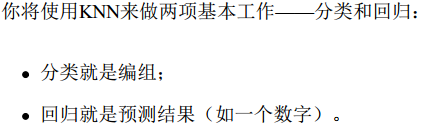
仅当每个子问题都是离散的，即子问题和子问题之间没有任何依赖，才可以使用动态规划。

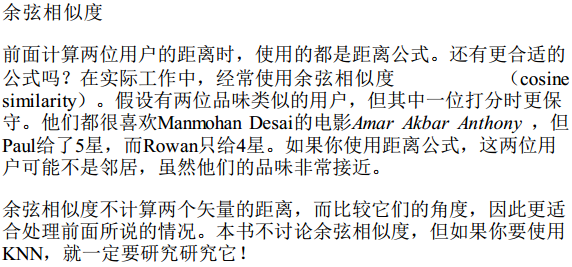
注意区分最长公共子串与最长公共子序列的区别，他们都可以使用动态规划来解决。



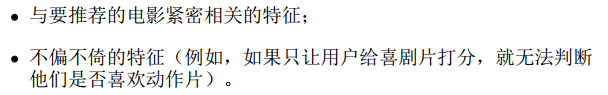




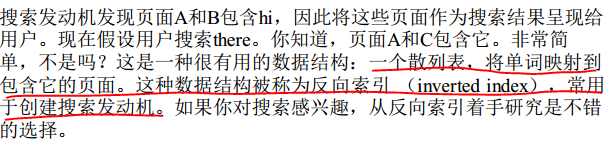




使用KNN算法的时候要选择合适的特征，这很重要。

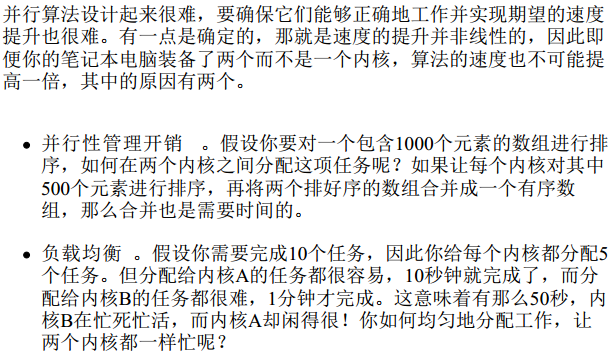




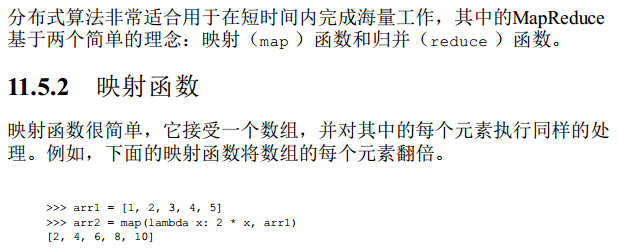


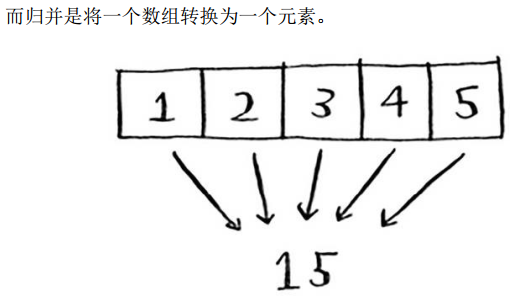


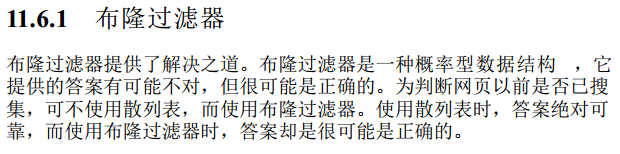
11.4 并行算法

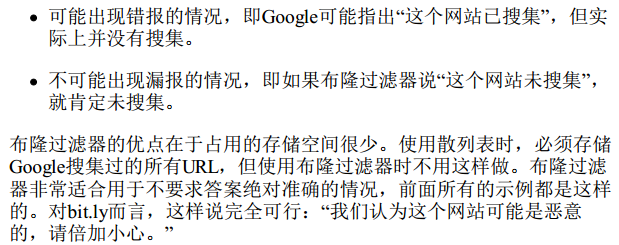


11.5 MapReduce算法









SHA算法（SECURE HASH ALGORITHM）安全散列算法

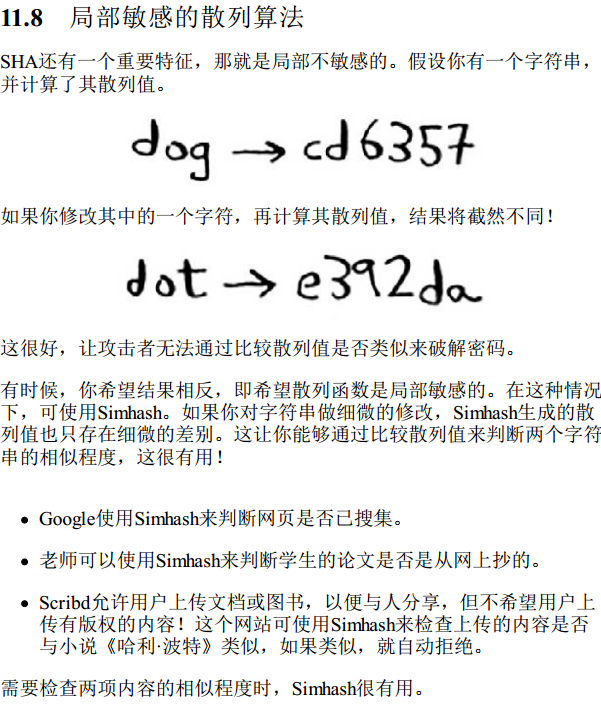
根据一个字符串生成另一个字符串，不同的字符串生成的散列值一般不同，同一字符串生成的散列值一定相同

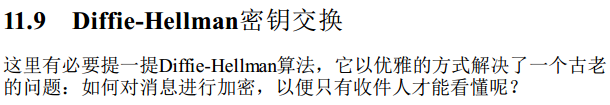
输入密码时，计算散列值，然后与数据库中的散列值进行比较，

这种算法是单向的，你可以根据字符串计算散列值，但不能根据散列值推断出原始的字符串

sha实际上是一系列的算法：sha-0，sha-1，sha-2，sha-3等

bcrypt是比较安全的密码散列函数







可以这样理解（公钥就是给所有人发一个带锁的箱子，这些锁只有我的私钥可以打开。如果有人要给我发消息但又不想被别人看到，就可以把消息装在箱子里锁上给我，这样只有我才可以看得到里面的内容）

RSA已经替代了Diffie-Hellman算法

