

漫画：什么是跳跃表？

2017-07-04 算法爱好者

(点击上方公众号，可快速关注)

来源：伯乐专栏作者/玻璃猫，微信公众号 - 梦见 (dreamsee321)

如有好文章投稿，请点击 → [这里了解详情](#)

这是发生在很多年以前的故事.....

小灰，听说你们最近在开发一款网络游戏？



是啊，现在正在为性能优化的事情发愁呢。



是什么地方性能不好，有什么方案，说来听听？



哎，这件事说来话长……



几天以前……

游戏策划：咱们需要做一个拍卖行系统，用来查阅和出售游戏中的道具。玩过《魔兽世界》吗？就是要做成那样子的拍卖行。



浏览拍卖

名字	等级范围	稀有程度	可用物品	搜索	
全部	Lv	剩余时间	出售者	上一页 下一页	
武器	图样：不褪之力头盔	525	非常长	扎克斯的 剑 一...	40,000 99 99 50,000 99 99
护甲	图样：刃影护腿	525	非常长	扎克斯的 剑 一...	40,000 99 99 50,000 99 99
容器	图样：流静头盔	525	非常长	扎克斯的 剑 一...	40,000 99 99 50,000 99 99
消耗品	图样：自然勇士护腿	525	非常长	扎克斯的 剑 一...	40,000 99 99 50,000 99 99
雕文	夜盲腰带	85	非常长	扎克斯的 剑 一...	50,000 99 99 60,000 99 99
商品	强效牛角符文	85	长	纯种黑牛 一...	40,000 12 50 41,000 0 0
配方	强效牛角符文	85	非常长	Vurtnefury 一...	42,001 20 0 46,668 0 0
珠宝	强效牛角符文	85	非常长	道道 一...	46,666 12 50 46,669 0 0
其它					
任务					
Pet					

5.000 0 0 0 竞标 一口价 关闭 浏览 竞标 拍卖

咱们的拍卖行商品需要支持四种排序方式：按价格、按等级、按剩余时间、按出售者 ID 排序。而且要尽可能快地展现给玩家。



基于品类的查询先不要，但我们要支持输入道具名称的精确查询和不输入名称的全量查询。小灰，实现这个系统有问题没？



放心吧，交给我没问题的！



几天之后……

小小的拍卖行系统还真不好做，
尤其是性能的保障……



拍卖行的商品总数量有几十万件，对应数据库商品表的几十万条记录。

如果是按照商品名称精确查询还好办，可以直接从数据库查出来，最多也就上百条记录。

如果没有商品名称的全量查询怎么办？总不可能把数据库里的所有记录全查出来吧，而且还要支持不同字段的排序。

所以，只能提前在内存中存储有序的全量商品集合，每一种排序方式都保存成独立的集合，每次请求的时候按照请求的排序种类，返回对应的集合。

比如按价格字段排序的集合：

名称	价格	等级
青铜剑	30	5
草药	50	1
铁盾牌	100	10
红宝石	300	2

比如按等级字段排序的集合：

名称	价格	等级
草药	50	1
红宝石	300	2
青铜剑	30	5
铁盾牌	100	10

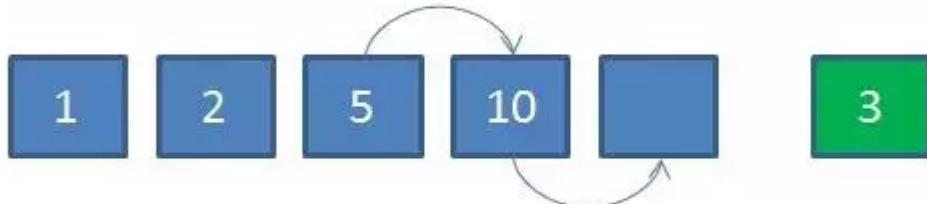
需要注意的是，当时还没有Redis这样的内存数据库，所以小灰只能自己实现一套合适的数据结构来存储。

究竟用什么样的数据结构好呢？
数组？链表？



拍卖行商品列表是线性的，最容易表达线性结构的自然是数组和链表。可是，无论是数组还是链表，在插入新商品的时候，都会存在性能问题。

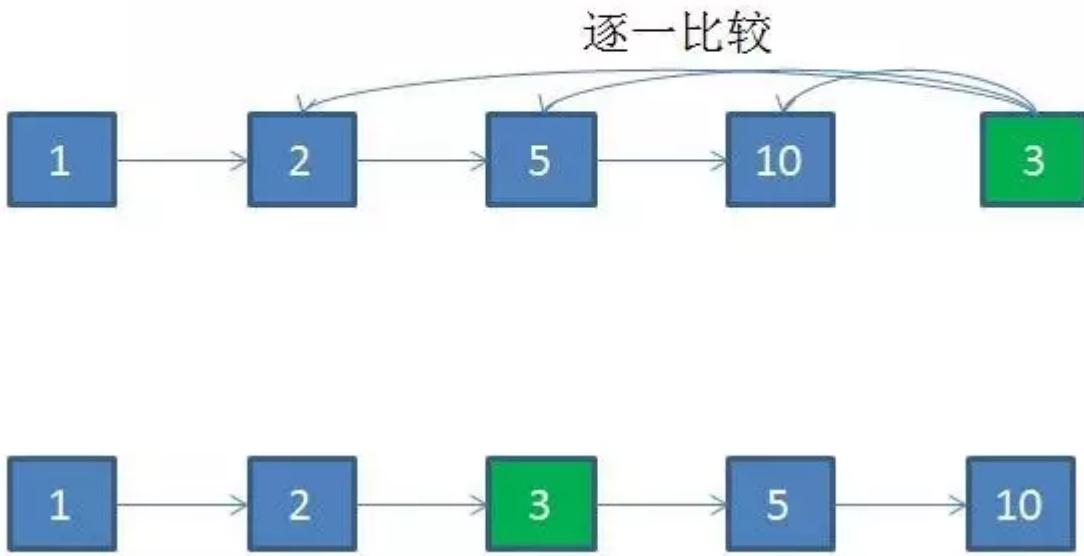
按照商品等级排序的集合为例，如果使用数组，插入新商品的方式如下：



如果要插入一个等级是3的商品，首先要知道这个商品应该插入的位置。使用二分查找可以最快定位，这一步时间复杂度是 $O(\log N)$ 。

插入过程中，原数组中所有大于3的商品都要右移，这一步时间复杂度是 $O(N)$ 。所以总体时间复杂度是 $O(N)$ 。

如果使用链表，插入新商品的方式如下：



如果要插入一个等级是3的商品，首先要知道这个商品应该插入的位置。链表无法使用二分查找，只能和原链表中的节点逐一比较大小来确定位置。这一步的时间复杂度是 $O(N)$ 。

插入的过程倒是很简单，直接改变节点指针的目标，时间复杂度 $O(1)$ 。因此总体的时间复杂度也是 $O(N)$ 。

这对于拥有几十万商品的集合来说，这两种方法显然都太慢了。

数组也不行，链表也不行，到底该怎么办呢？



事情就是这样，愁人啊.....



哈哈，小灰，你听说过【跳跃表】这种数据结构吗？



跳跃表？那是什么鬼？



跳跃表 (skiplist) 是一种基于有序链表的扩展，简称跳表。



在我详细介绍之前请你来思考一件事，怎样能更快查找到一个有序链表的某一节点呢？



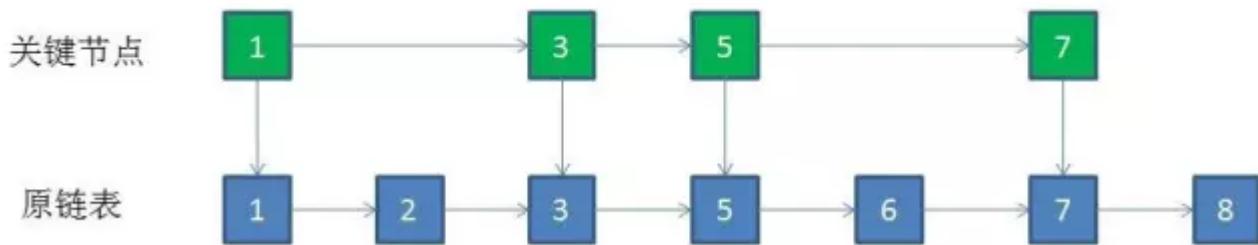
既然是链表，肯定不能用二分查找了... 让我想想啊.....



我知道了，我们可以利用类似索引的思想，提取出链表中的部分关键节点。

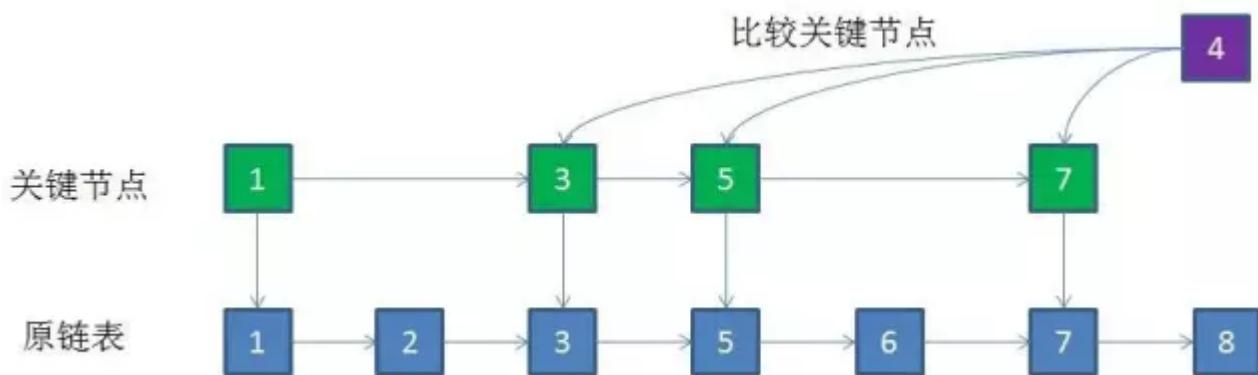
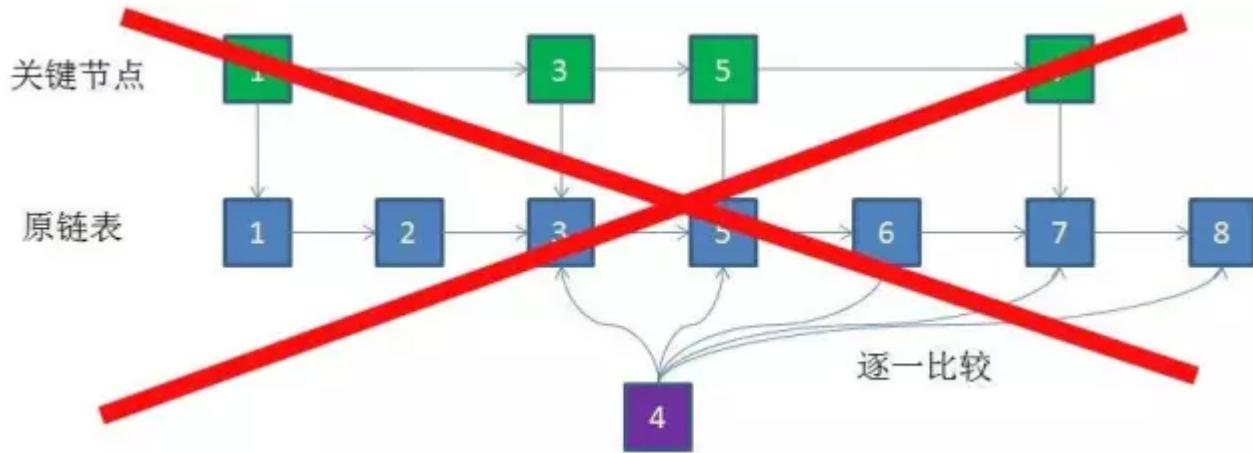


比如给定一个长度是 7 的有序链表，节点值依次是 1->2->3->5->6->7->8。那么我们可以取出所有值为奇数的节点作为关键节点。

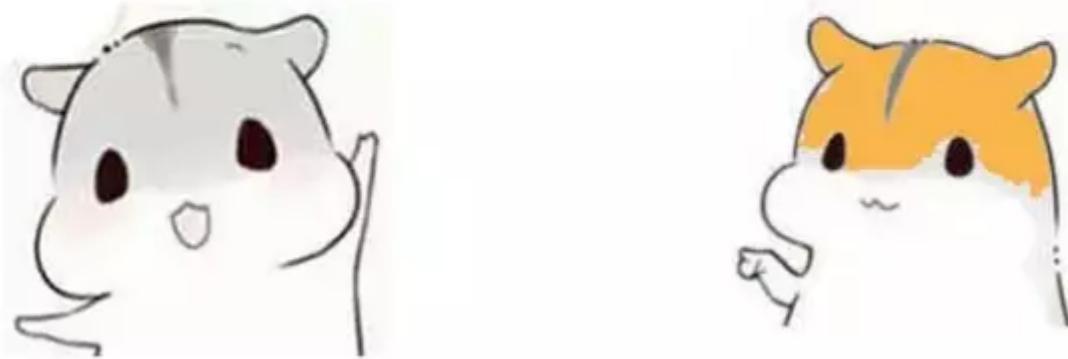


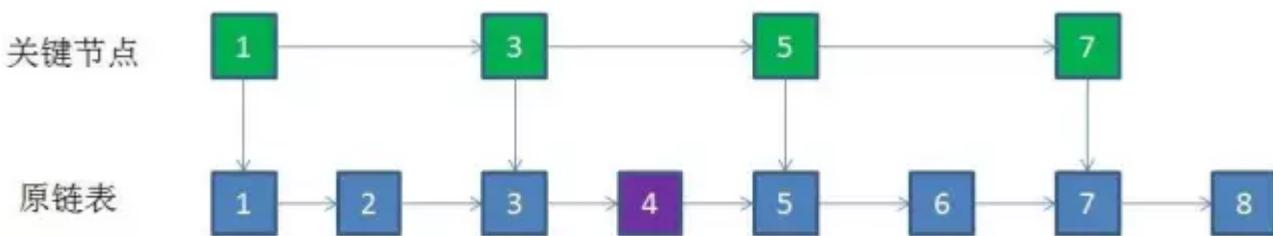
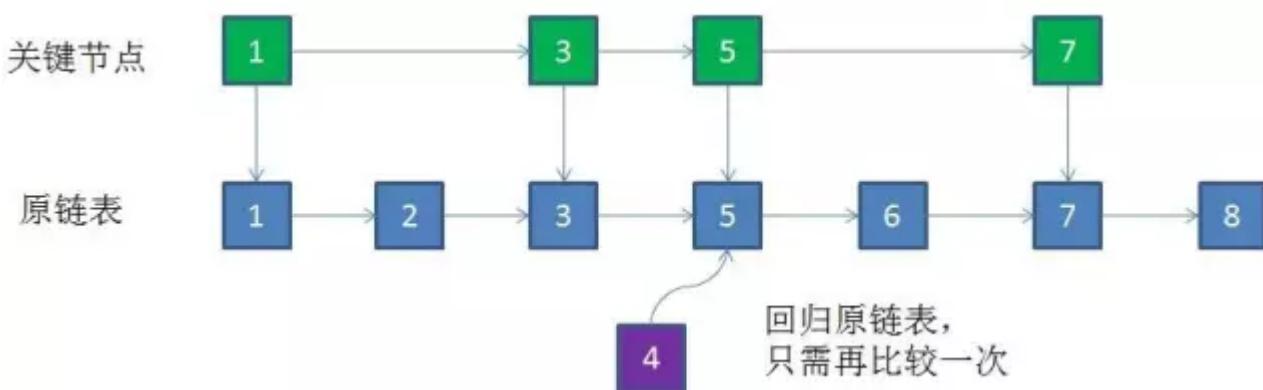
此时如果要插入一个值是 4 的新节点，不再需要和原节点 8, 7, 6, 5, 3 逐一比较，只需要比较关键节点 7, 5, 3。





确定了新节点在关键节点中的位置（3 和 5 之间），就可以回到原链表，迅速定位到对应的位置插入（同样是 3 和 5 之间）。





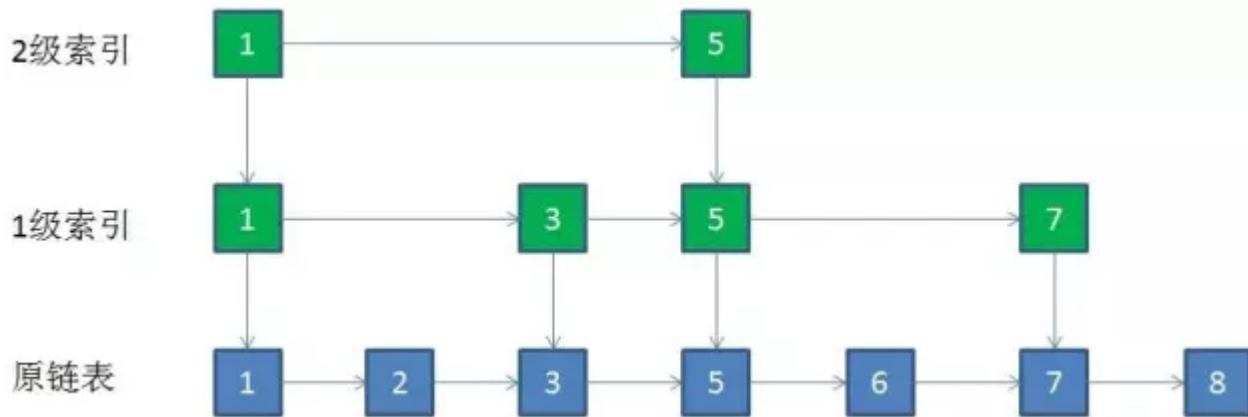
现在节点数目少，优化效果还不明显，如果
链表中有 1W 甚至 10W 个节点，比较次数就
整整减少了一半！



非常好！这样做虽然增加了 50% 的额外的空间，但是性能提高了一倍。



不过我们可以进一步思考。既然已经提取出了一层关键节点作为索引，那我们为何不能从索引中进一步提取，提出一层索引的索引？



我明白了。有了 2 级索引之后，新的节点可以先和 2 级索引比较，确定大体范围；然后再和 1 级索引比较；最后再回到原链表，找到并插入对应位置。



当节点很多的时候，比较次数会减少到原来的四分之一，厉害了！



没错。当节点足够多的时候，我们不止能提出两层索引，还可以向更高层次提取，保证每一层是上一层节点数的一半。



至于提取的极限，则是同一层只有两个节点的时候，因为一个节点没有比较的意义。这样的多层链表结构，就是所谓的【跳跃表】。



原来如此，这就是跳跃表呀。



有一个问题需要注意：当大量的新节点通过逐层比较，最终插入到原链表之后，上层的索引节点会渐渐变得不够用。



这时候需要从新节点当中选取一部分
提到上一层。可是究竟应该提拔谁、
忽略谁呢？



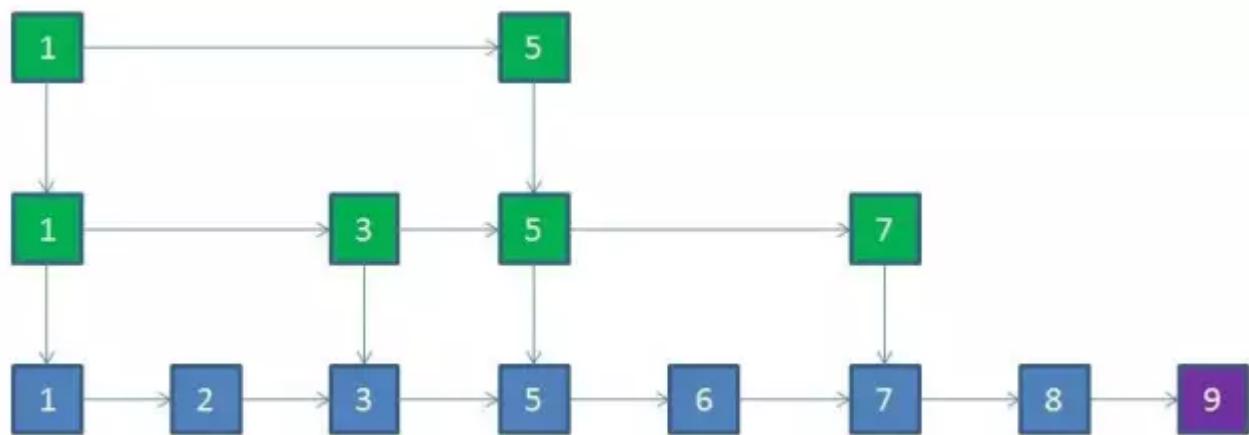
哎呀，这倒是个问题，该怎么
选择索引呢？

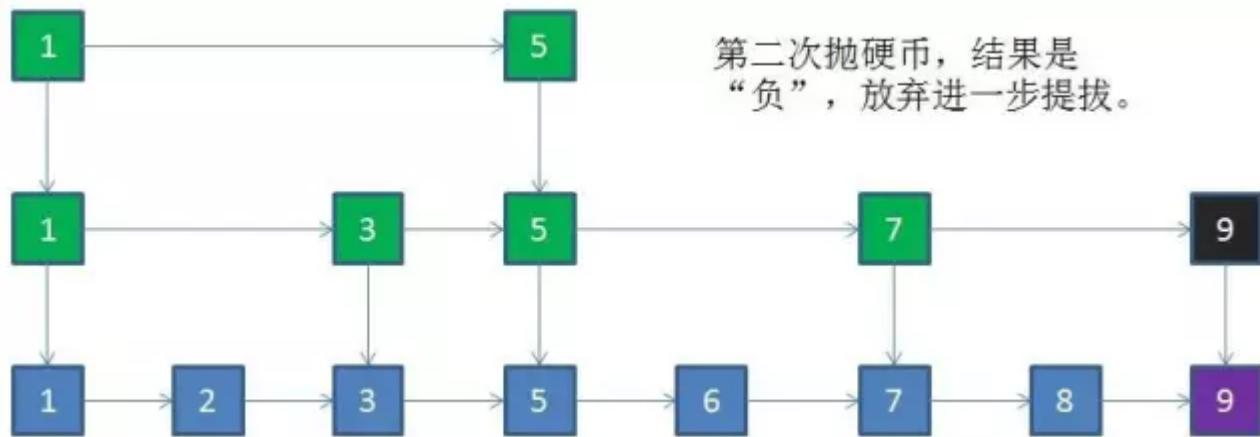
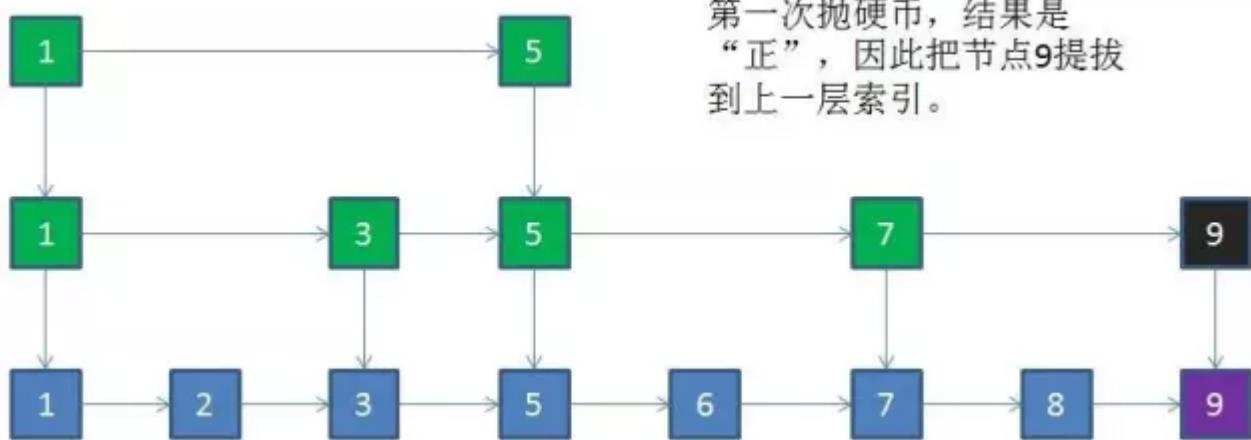


关于这一点，跳跃表的设计者采用了一种有趣的方法：【抛硬币】。也就是随机决定新节点是否提拔，每向上提拔一层的几率是 50%。



我们仍然借用刚才的例子，假如值为 9
新节点插入原链表





抛硬币？为什么要用这么奇怪的方法？



哈哈，因为跳跃表删除和添加的节点是不可预测的，很难用一种有效的算法来保证跳表的索引分部始终均匀。



随机抛硬币的方法虽然不能保证索引绝对均匀分布，却可以让大体趋于均匀。



总结一下，跳跃表插入节点的流程有以下几步：



1. 新节点和各层索引节点逐一比较，确定原链表的插入位置。O ($\log N$)
2. 把索引插入到原链表。O (1)
3. 利用抛硬币的随机方式，决定新节点是否提升为上一级索引。结果为“正”则提升并继续抛硬币，结果为“负”则停止。O ($\log N$)

总体上，跳跃表插入操作的时间复杂度是O ($\log N$)，而这种数据结构所占空间是 $2N$ ，既空间复杂度是O (N)。

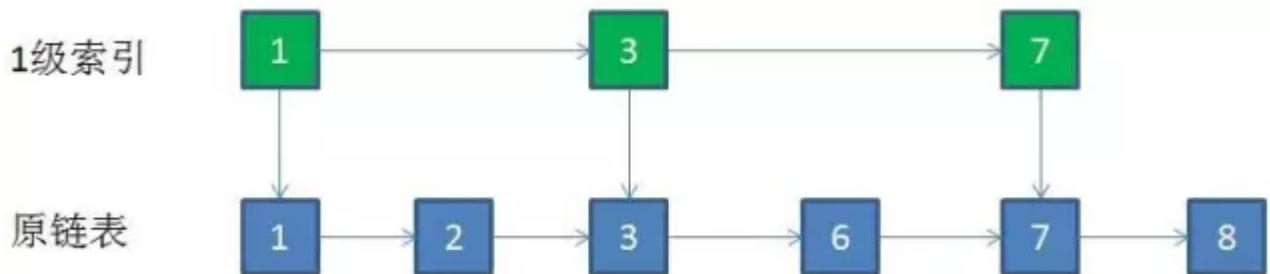
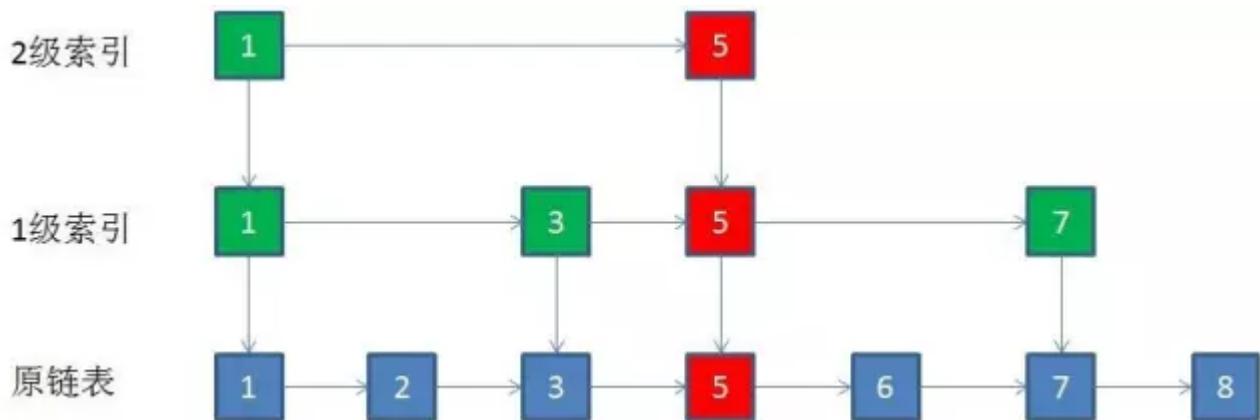
明白了，那么跳跃表的删除操作
怎么进行呢？



删除操作比较简单，只要在索引层
找到要删除的节点，然后顺藤摸瓜，
删除每一层的相同节点即可。



如果某一层索引在删除后只剩下一个节点，那么整个一层就可以干掉了。还用原来的例子，如果要删除的节点值是 5：



我们来总结一下跳跃表删除节点的步骤吧：



1. 自上而下，查找第一次出现节点的索引，并逐层找到每一层对应的节点。 $O(\log N)$
2. 删除每一层查找到的节点，如果该层只剩下1个节点，删除整个一层（原链表除外）。 $O(\log N)$

总体上，跳跃表删除操作的时间复杂度是 $O(\log N)$ 。

天了噜，跳跃表果然很强大。

我还有一个问题，跳跃表和二叉查找树的区别是什么呢？



跳跃表的优点是维持结构平衡的成本比较低，完全依靠随机。而二叉查找树在多次插入删除后，需要 Rebalance 来重新调整结构平衡。



所以说没有绝对优劣的数据结构，关键还要看应用场景。



小灰和大黄并不知道，他们的这一解决方案和若干年后Redis当中的**Sorted-set**不谋而合。而Sorted-set这种有序集合，正是对于跳跃表的改进和应用。

对于关系型数据库如何维护有序的记录集合呢？使用的是**B+树**。有关B+树的知识，将在以后的漫画中详细介绍。

小伙伴们，感谢支持！

觉得本文有帮助？请分享给更多人

关注「算法爱好者」，修炼编程内功

算法爱好者

专注算法相关内容



微信号：AlgorithmFans



长按识别二维码关注

伯乐在线旗下微信公众号

商务合作QQ：2302462408