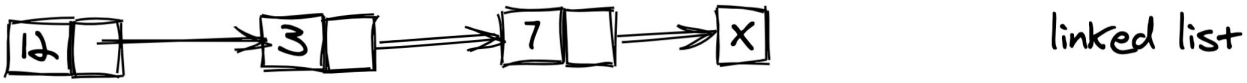
跳表

虽然在面试中出现的频率不大，但是在工业中，跳表会经常被用到。力扣中关于跳表的题目只有一个。但是跳表的设计思路值得我们去学习和思考。 其中有很多算法和数据结构技巧值得我们学习。比如空间换时间的思想，比如效率的取舍问题等。

解决的问题

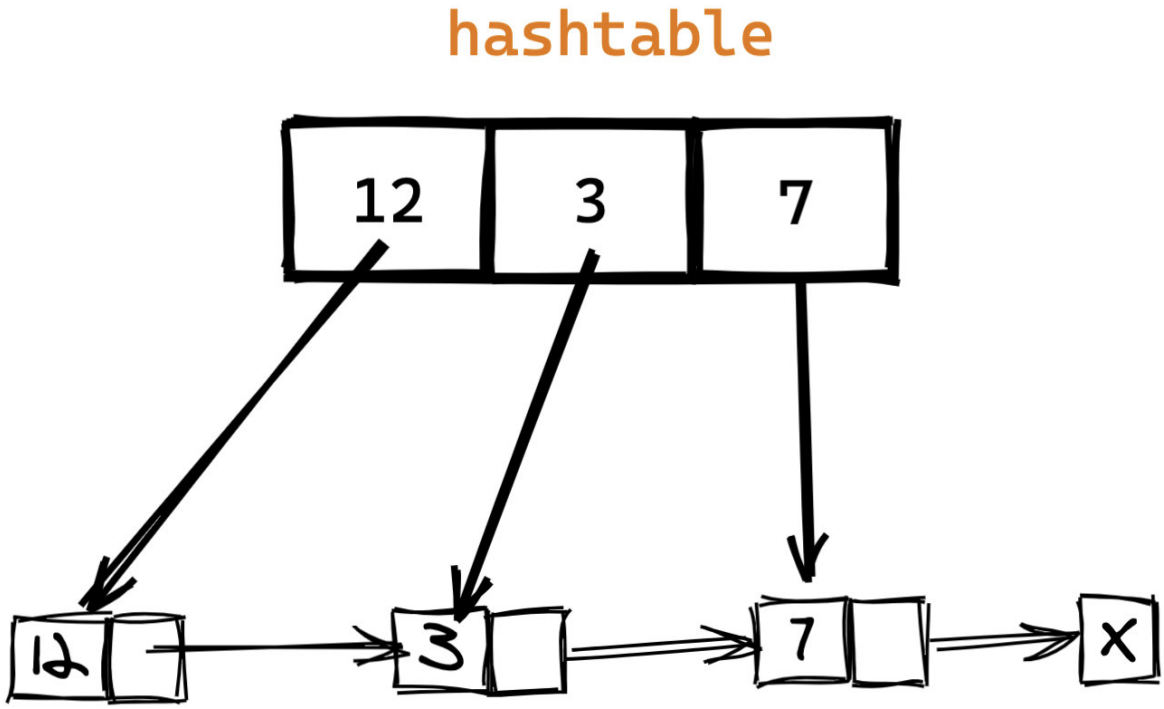
只有知道跳表试图解决的问题， 后面学习才会有针对性。实际上，跳表解决的问题非常简单，一句话就可以说清楚，那就是**为了减少链表长度增加，查找链表节点时带来的额外比较次数**。

不借助额外空间的情况下，在链表中查找一个值，需要按照顺序一个个查找，时间复杂度为 $O(N)$，其中 N 为链表长度。



（单链表）

当链表长度很大的时候， 这种时间是很难接受的。 一种常见的的优化方式是**建立哈希表，将所有节点都放到哈希表中，以空间换时间的方式减少时间复杂度**，这种做法时间复杂度为 $O(1)$，但是空间复杂度为 $O(N)$。



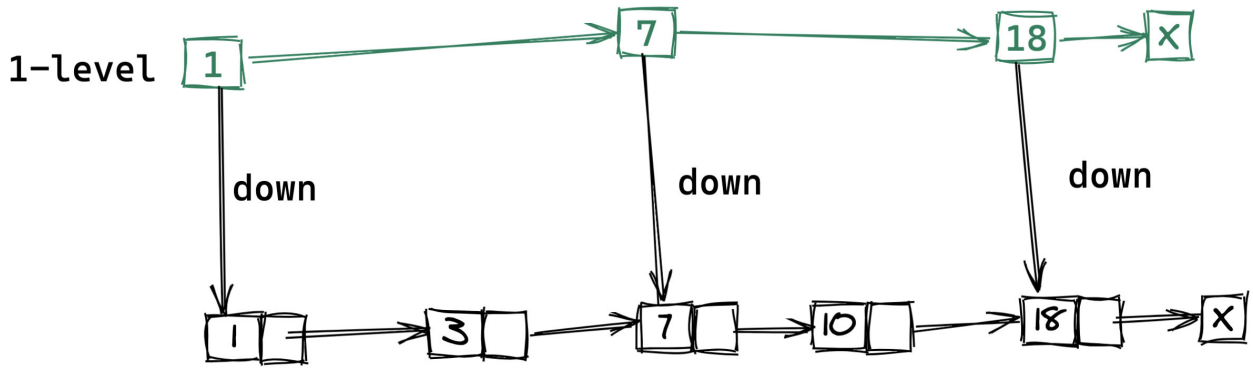
（单链表 + 哈希表）

为了防止链表中出现重复节点带来的问题，我们需要序列化节点，再建立哈希表，这种空间占用会更高，虽然只是系数级别的增加，但是这种开销也是不小的 。

为了解决上面的问题，跳表应运而生。

如下图所示，我们从链表中每两个元素抽出来，加一级索引，一级索引指向了原始链表，即：通过一级索引 7 的 down 指针可以找到原始链表的 7 。那怎么查找 10 呢?

> 注意这个算法要求链表是有序的。

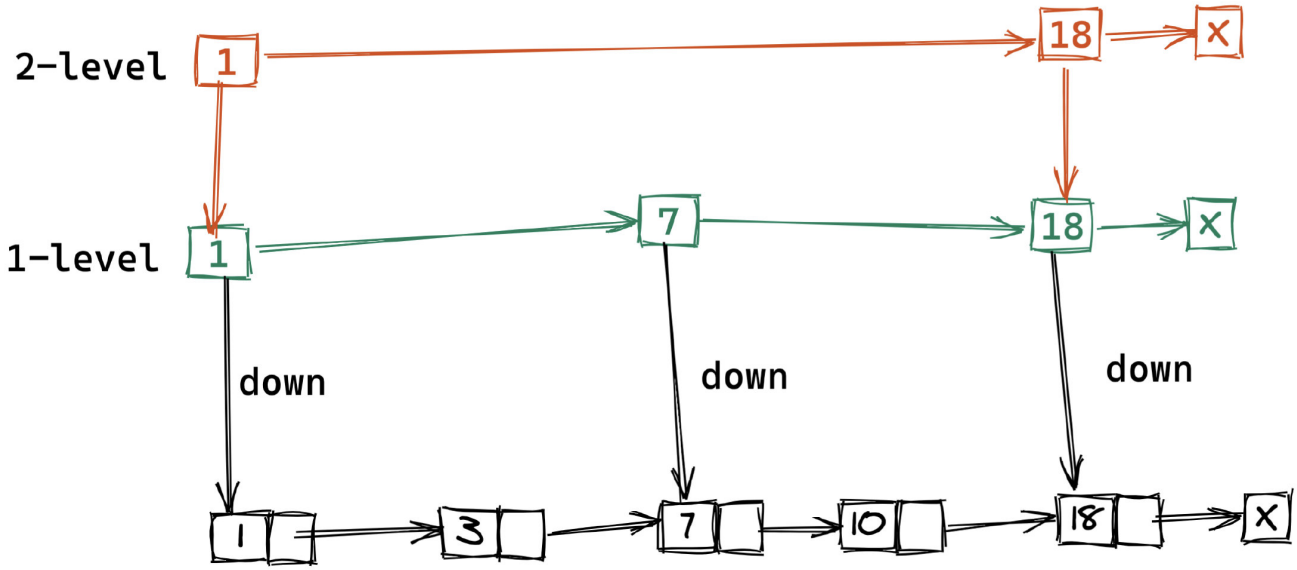


（建立一级索引）

我们可以：

- 通过现在一级跳表中搜索到 7，发现下一个 18 大于 10 ，也就是说我们要找的 10 在这两者之间。
- 通过 down 指针回到原始链表，通过原始链表的 next 指针我们找到了 10。

这个例子看不出性能提升。但是如果元素继续增大， 继续增加索引的层数，建立二级，三级。。。索引，使得链表能够实现二分查找，从而获得更好的效率。但是相应地，我们需要付出额外空间的代价。



（增加索引层数）

理解了上面的点，你可以形象地将跳表想象为玩游戏的**存档**。

一个游戏有 10 关。如果我想要玩第 5 关的某一个地方，那么我可以直接从第五关开始，这样要比从第一关开始快。我们甚至可以在每一关同时设置很多的存档。这样我如果想玩第 5 关的某一个地方，也可以不用从第 5 关的开头开始，而是直接选择**离你想玩的地方更近的存档**，这就相当于跳表的二级索引。

跳表的时间复杂度和空间复杂度不是很好分析。由于时间复杂度 = 索引的高度 * 平均每层索引遍历元素的个数，而高度大概为 $\log n$，并且每层遍历的元素是常数，因此时间复杂度为 $\log n$，和二分查找的空间复杂度是一样的。

空间复杂度就等同于索引节点的个数，以每两个节点建立一个索引为例，大概是 n/2 + n/4 + n/8 + … + 8 + 4 + 2 ，因此空间复杂度是 $O(n)$。当然你如果每三个建立一个索引节点的话，空间会更省，但是复杂度不变。

代码实现

关于代码，我先卖一个关子。这是因为跳表我们只准备了一道题，那就是**实现跳表**。因此我打算在每日一题的时候再给大家分析以及具体的代码。大家现在可以自己想想如何实现，然后对照官方题解看一下自己的实现和官方题解有什么不同，谁更好。

总结

- 跳表是可以实现二分查找的有序链表；
- 跳表由多层构成，最底层是包含所有的元素原始链表，往上是索引链表；
- 实际的设计中，需要做好取舍，设定合理数量的索引。
- 跳表查询、插入、删除的时间复杂度为 $O(\log N)$，空间复杂度为 $O(N)$；

上一页 下一页