2 - 2 数组、链表、跳表的基本实现和特性

Array 数组

- 语言格式：

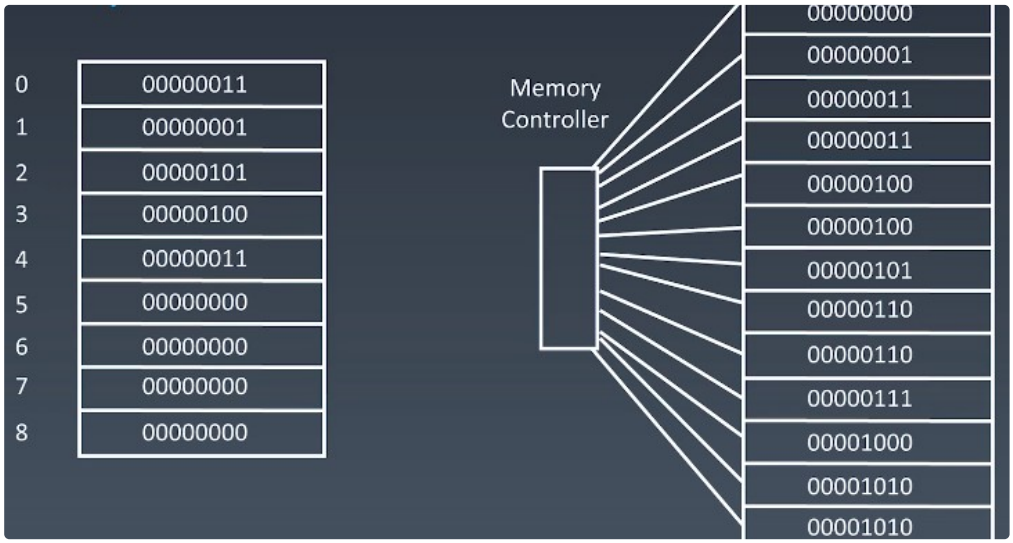
Java, C++： int a[100];

Python : list = []

JavaScript : let x = [1,2,3]

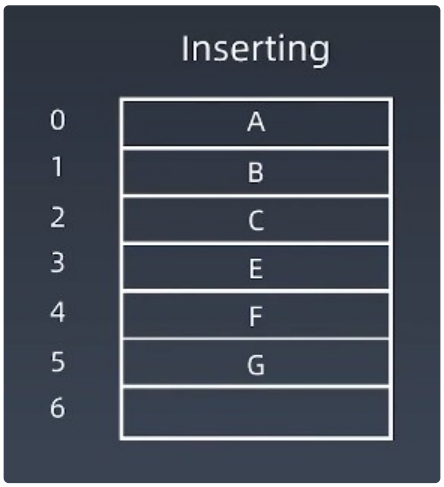
- 内存管理器：

每当申请数组时，计算机实际上在内存中开辟了一段连续的地址，每一个地址就可以通过内存管理器进行访问。



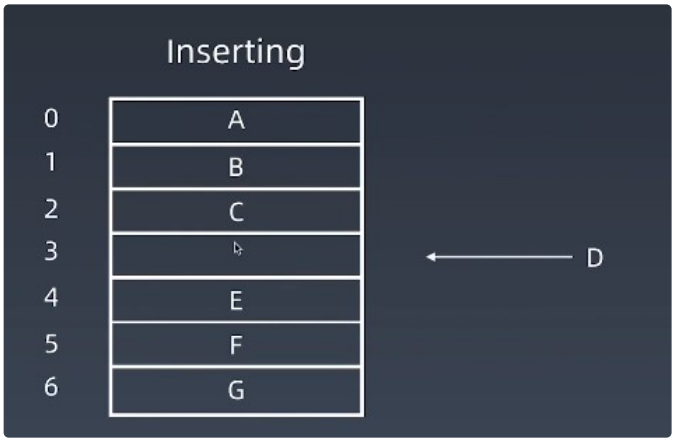
- 插入操作（Inserting）：

初识状态：



将D插入 index = 3 的位置 ：

首先，将 index = 3 后面的元素向下移动，然后将 D 插入到腾空的 3 号位置

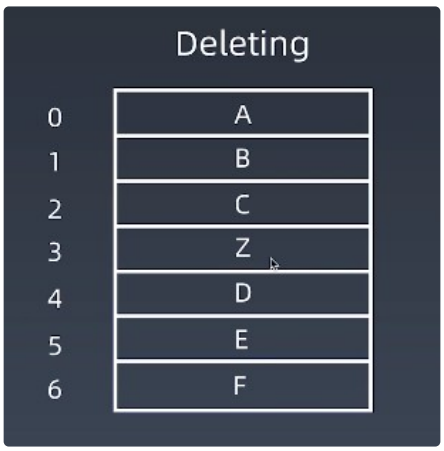


时间复杂度：

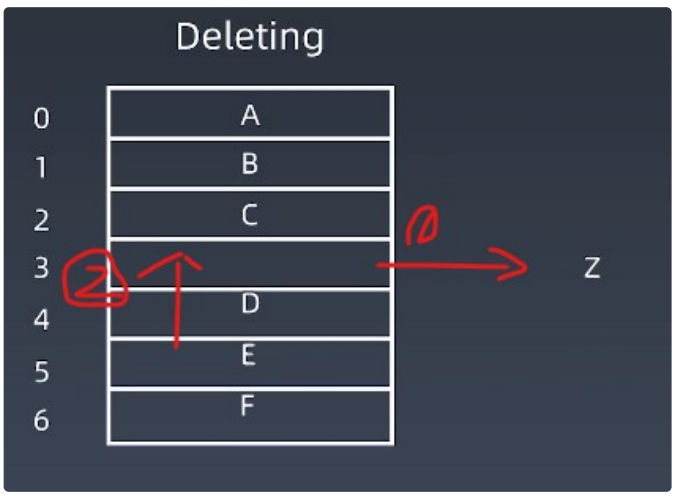
- 最好情况：O(1)
- 最坏情况：O(n)
- 平均情况：O(n)

- 删除操作（Deleting）：

初始数组，删除 元素 Z



首先，先把 Z 拿出去，然后将 Z 后的元素提前



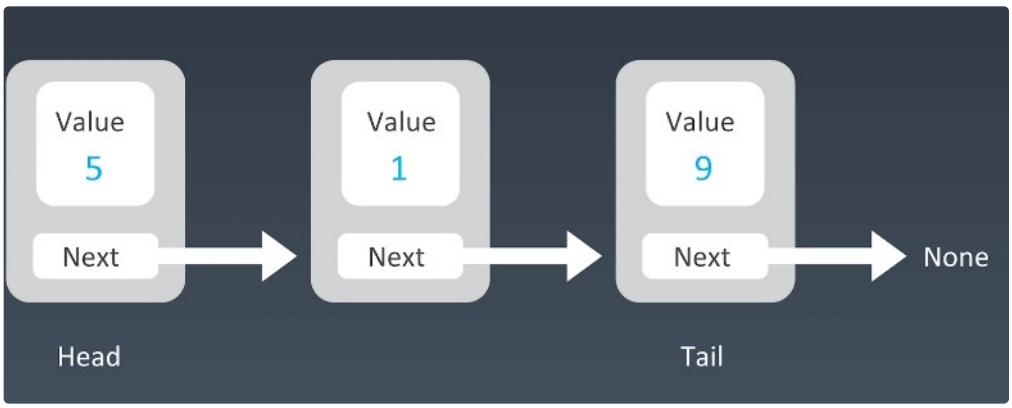
- 时间复杂度：
  - prepend ： O(1)

    注：正常情况下数组的 prepend 的操作时间时间复杂度是 O(n)，但是可以进行特殊优化到 O(1)。采用的方式是申请稍大一些的内存空间，然后在数组最开始预留一部分空间，然后 prepend 的操作则是把头下标前移一个为止即可。

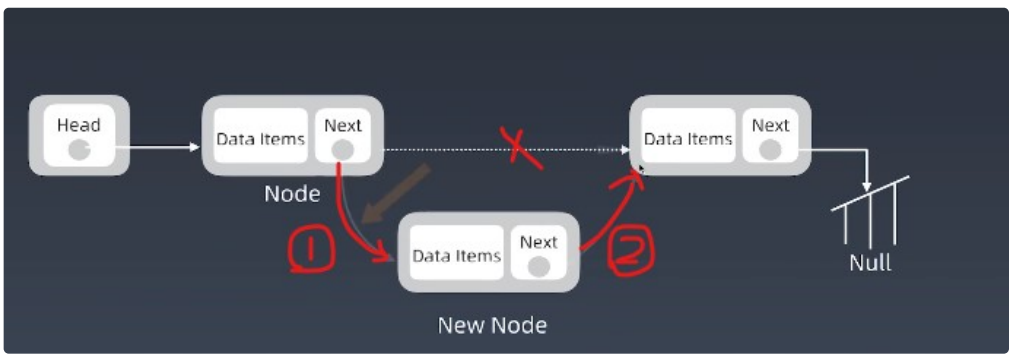
  - append : O(1)
  - lookup : O(1)
  - insert : O(n)
  - delete : O(n)
- 优缺点：
  - 优点：下标查询非常迅速 O(1);
  - 缺点：插入、删除缓慢 O(n).
- 实现代码：

🔗 **ArrayList Java 源码分析**

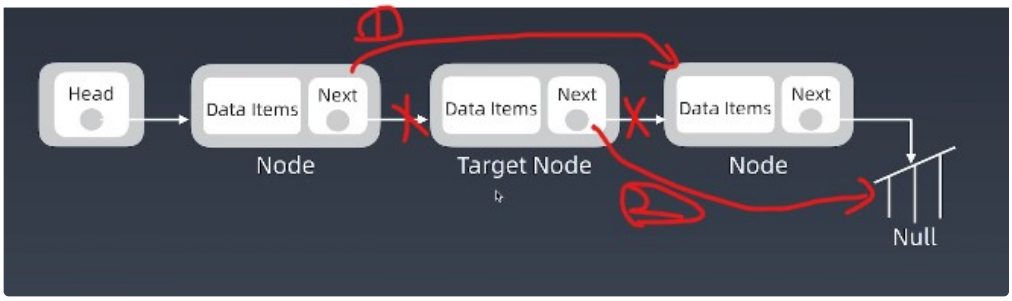
Linked List 链表

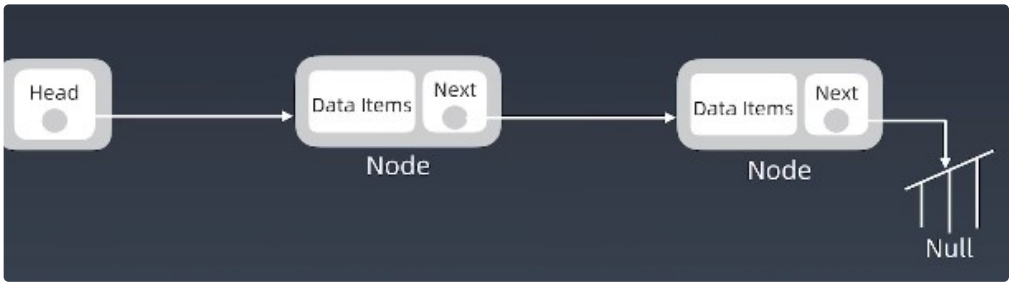


- 插入操作: O(1)



- 删除操作: O(1)





- 时间复杂度：
  - prepend ： O(1)
  - append ： O(1)
  - lookup ： O(n)
  - insert ： O(1)
  - delete ： O(1)
- LinkedList 优缺点：
  - 优点：插入迅速，O(1);
  - 缺点：查找缓慢，O(n).
- 实现代码：

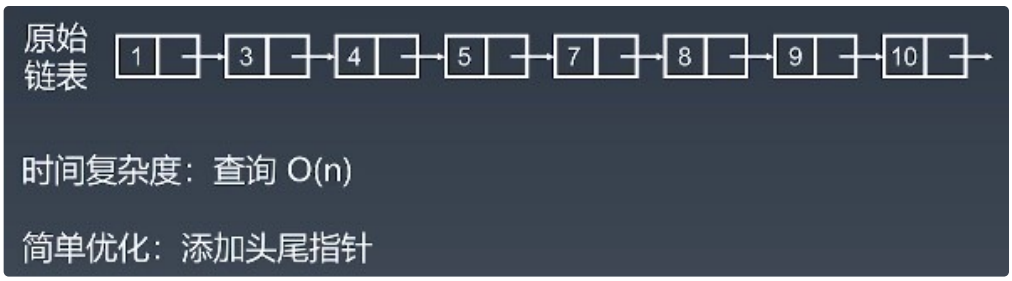
  Linked List 标准实现代码

  Linked List Java 源码分析

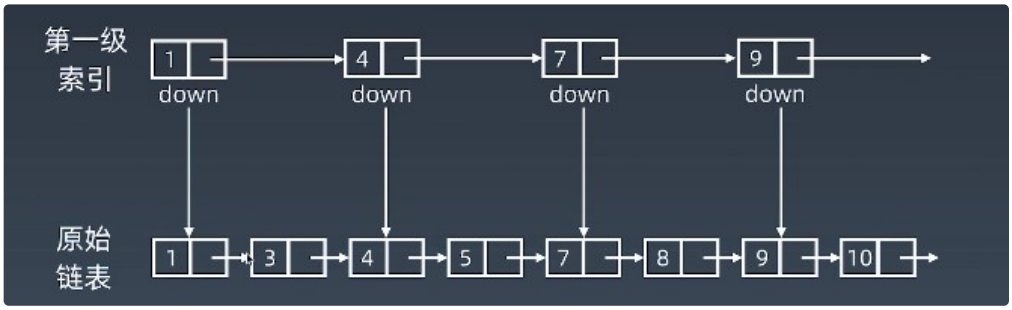
Skip List 调表（暂时只做了解，后期仔细讲解）

- 特点：只能用于元素有序的情况；
- 调表 (skip list) 对标的是平衡树 (AVL Tree) 和 二分查找，是一种和 插入、删除、搜索都是 O(log n) 的数据结构；（1989 年出现）
- 优点：原理简单、容易实现、方便扩展、效率更高。因此，在一些热门的项目离用来替代平衡树，如 Redis、LevelDB等。

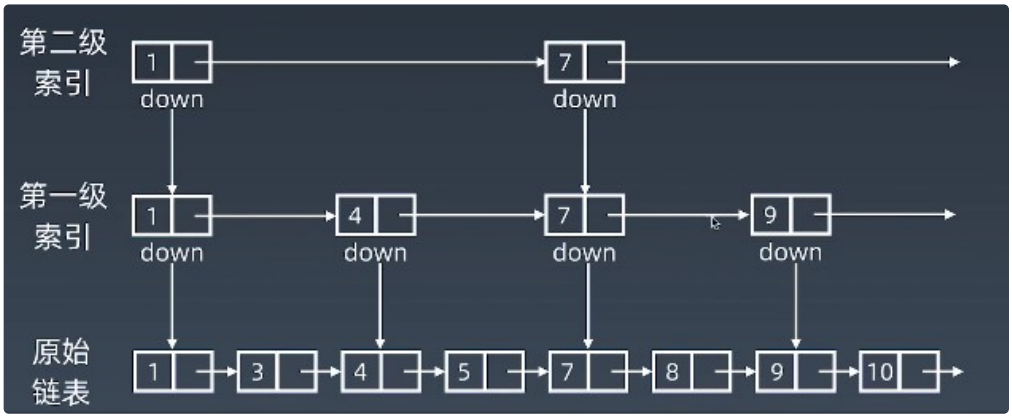
- Question：如何给有序的链表加速？or 如何提高链表线性查找的效率？



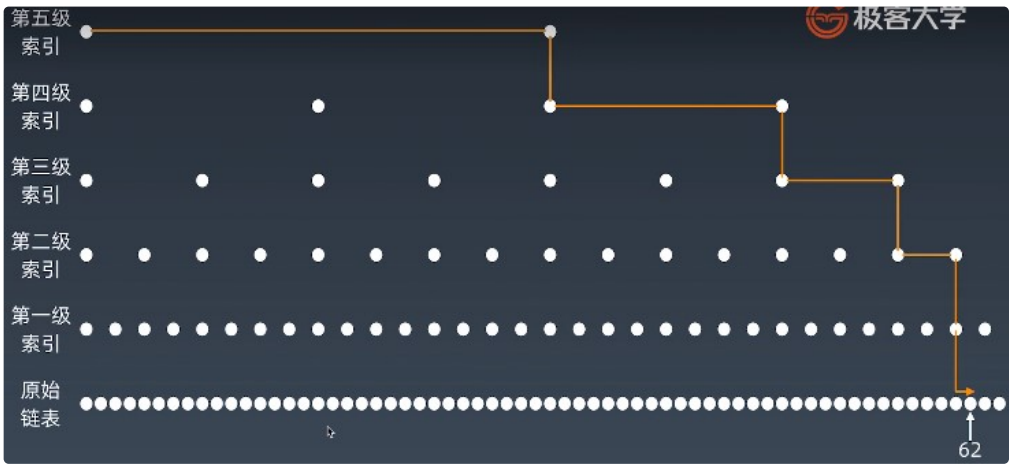
添加第一级索引：



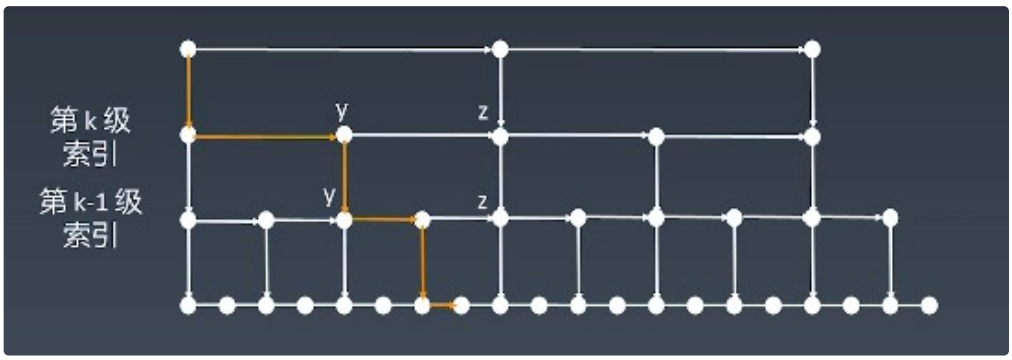
添加第二级索引：



由此类推，可以增加多级索引：

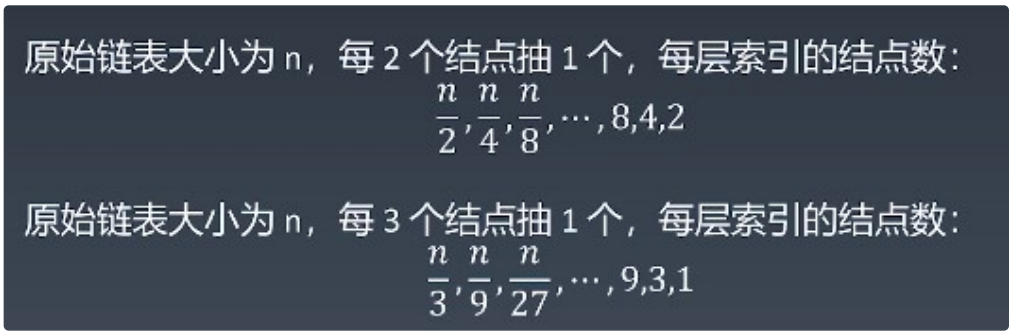


- 时间复杂度：
  - n/2、n/4、n/8、第k级索引节点的个数就是 $n/2^k$ ；
  - 假设索引有 h 级，最高级的索引有 2 个节点，$n/2^h = 2$,从而求得 $h = log2(n) - 1$ ；

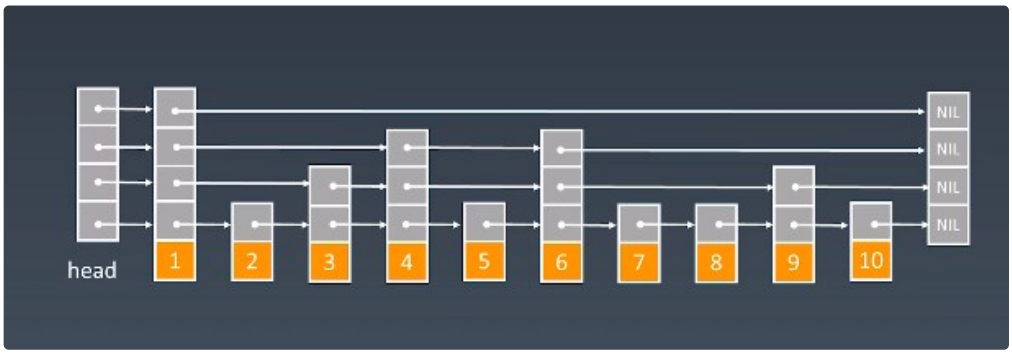


  - 索引的高度：logn;

  - 每层索引遍历的节点个数：3;
  - 在调表中查询任意数据的时间复杂度就是 O(logn).
- 空间复杂度分析：O(n)



- 现实中调表的形态：



  - 由于元素的增加、删除而导致它的索引并不是非常工整的；
  - 经过最后改动后，它最后索引的地方会跨几步、有些地方会少跨、或只跨两步；
  - 维护成本较高，即，当增加或删除一个元素时，需要将它的索引更新一遍。

工程中的应用

  - Linked List： 🔗 **LRU Cache** 、 🔗 **LRU Cache 简介**
  - Skip List： 🔗 **Redis** 、 🔗 **为什么 Redis 使用 Skip List 而不使用 Red - Black?**

笔记时间：2021-03-29 总结《算法训练营 25期》谭超专栏 --- Benjamin Song