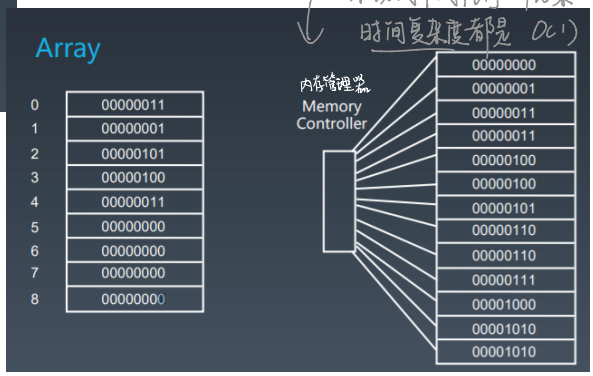
Array
数组

Java, C++: int a[100];

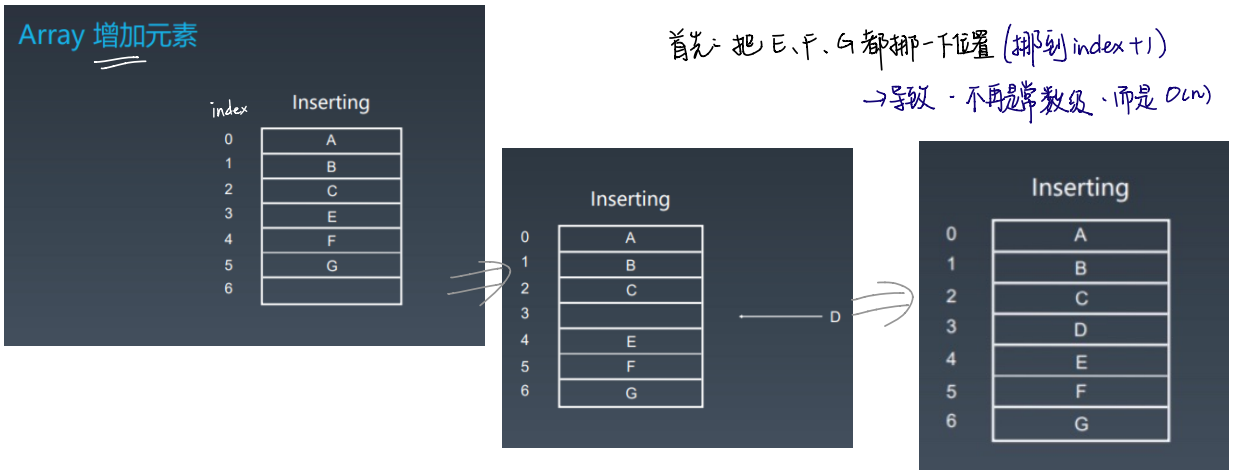
Python: list = []

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

泛型：任何一个单元类型的话都可以放进去

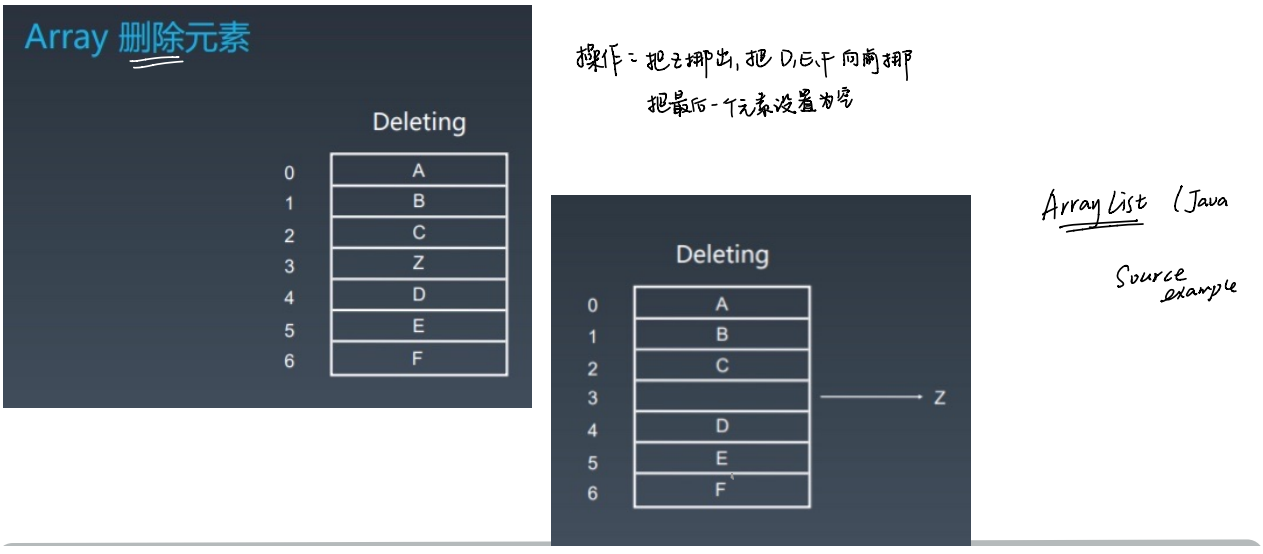


有要访问的话（访问第一个元素）
和访问中间任何一个元素
时间复杂度都是 O(1)



首先：把 E、F、G 都挪一下位置（挪到 index+1）

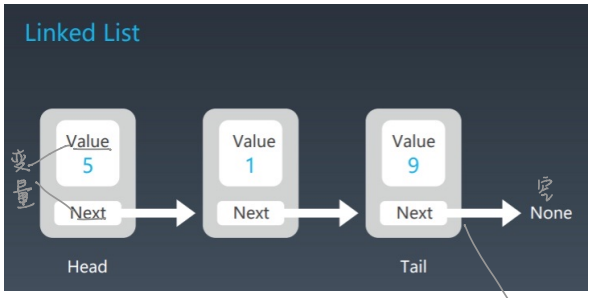
→导致：不再是常数级，而是 O(n)



操作：把 Z 挪出，把 D、E、F 向前挪

把最后一个元素设置为空

ArrayList（Java Source example）



弥补数组缺点

在修改和添加操作、删除操作比较频繁的情况下

数组

Next 指向下一元素

每一个元素一般用 class 来定义

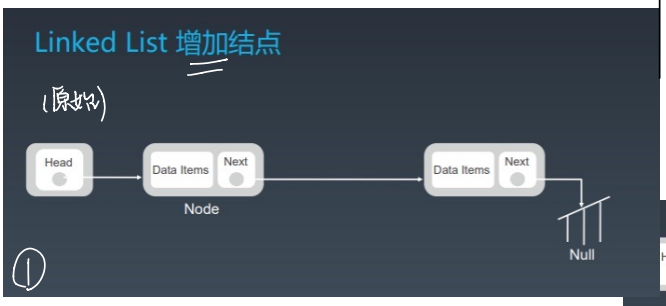
指回 Head → 循环列表

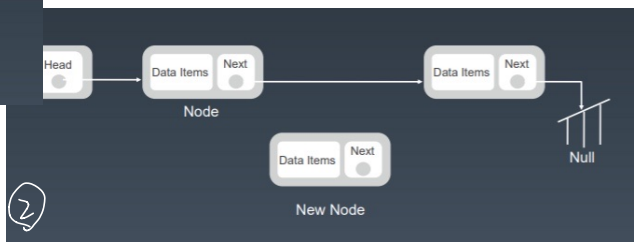
标准实现代码

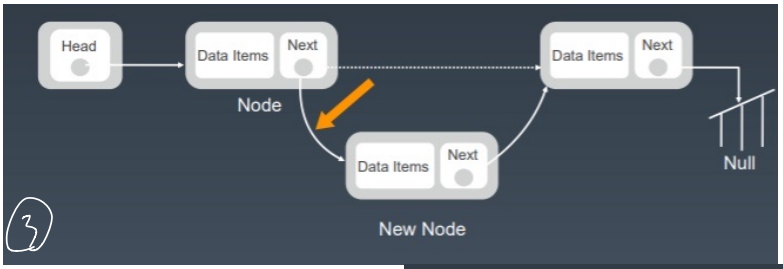
```
class LinkedList {
    Node head; // head of list

    /* Linked list Node*/
    class Node {
        int data;
        Node next;

        // Constructor to create a new node
        // Next is by default initialized
        // as null
        Node(int d) { data = d; }
    }
}
```

Java里的 LinkedList 不是单链表，而是双向链表

Source example

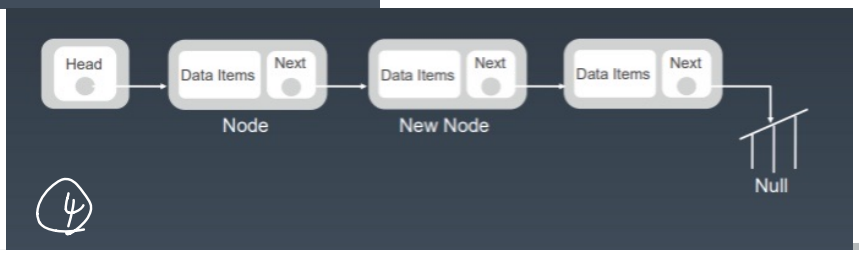


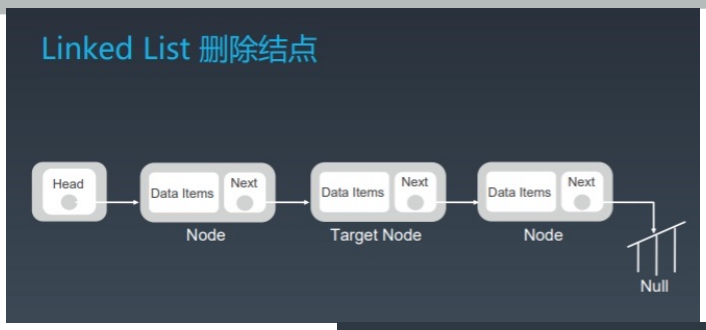




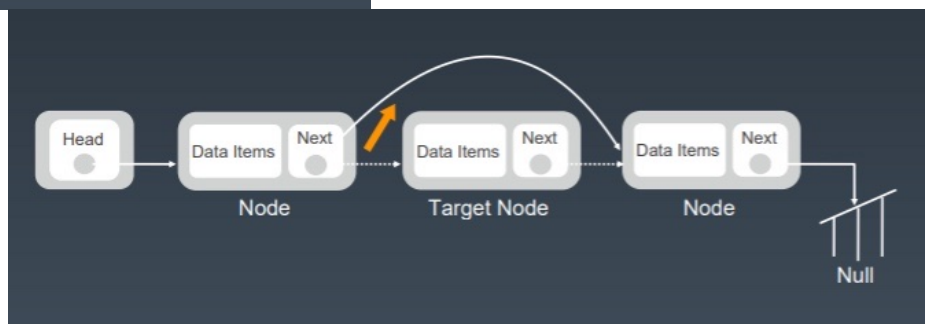
总共操作两次，常数次

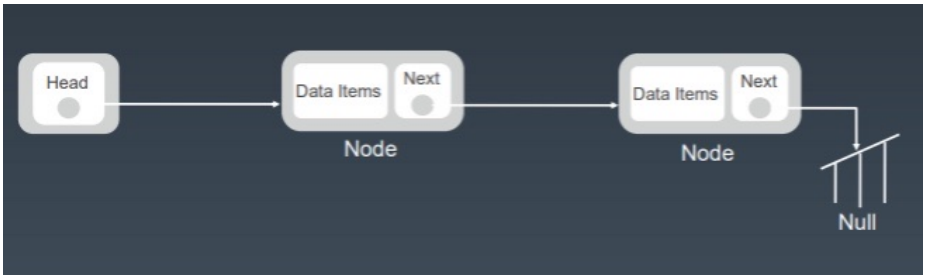
∴ O(1)





操作：
把前驱的节点的next指向
移到后续的节点去

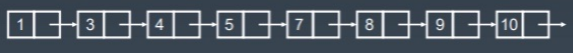




增加、删除节点的话
没有引起整个链表的群移操作
也不需要复制元素
也不需要挪动大量元素
O(1)

但、访问节点 头、尾 O(1)
中间 O(n)

链表元素有序的时候



普通的数组可以进行二分查找

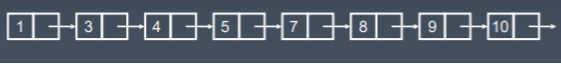
给有序的链表加速

跳表的特点

注意：只能用于元素有序的情况。

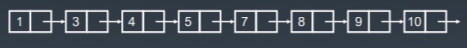
所以，跳表（skip list）对标的是平衡树（AVL Tree）和二分查找，是一种 插入/删除/搜索 都是 O(log n) 的数据结构。1989 年出现。

它最大的优势是原理简单、容易实现、方便扩展、效率更高。因此在一些热门的项目里用来替代平衡树，如 Redis、LevelDB 等。



如何给有序的链表加速

原始链表



时间复杂度：查询 O(n)

简单优化：添加头尾指针

概念：

一维的数据结构要加速的话，经常采用的方式就是升维也就是演变成二维

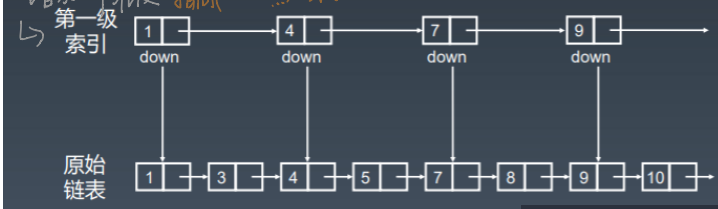
一维 → 二维

why？多一个维度，多一级信息，更快

添加第一级索引

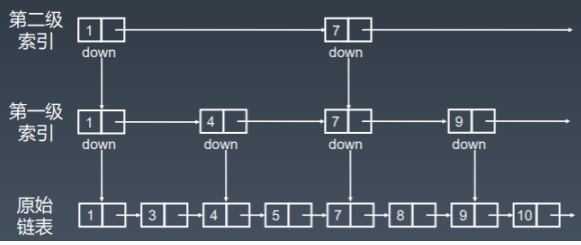
如何提高链表线性查找的效率？

增加一个维度 指向(next+1)元素

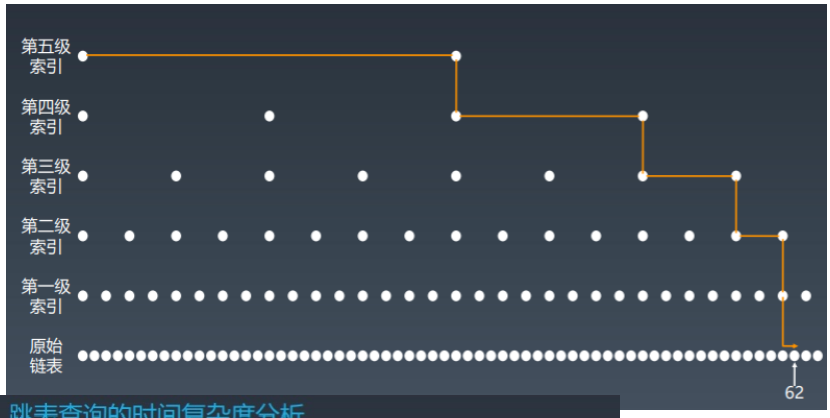


添加第二级索引

如何进一步提高链表查找的效率？



以此类推，增加多级索引

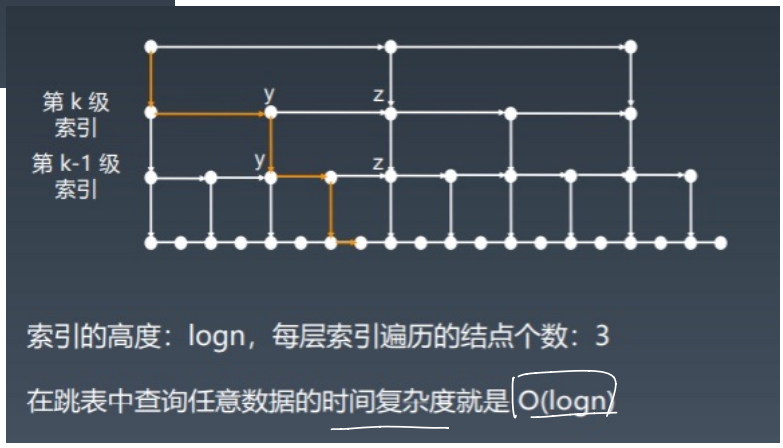


跳表查询的时间复杂度分析

O(logn)

n/2、n/4、n/8、第 k 级索引结点的个数就是 n/(2^k)

假设索引有 h 级，最高级的索引有 2 个结点。n/(2^h) = 2，从而求得 h = log2(n)-1



索引的高度：logn，每层索引遍历的结点个数：3

在跳表中查询任意数据的时间复杂度就是 O(logn)

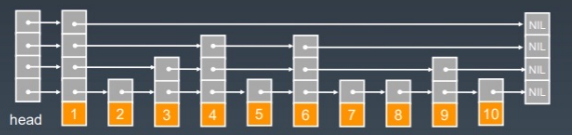
假设查到第8个节点

每层要遍历的节点总共是3个 $\log_2 8$

O(n) → O(logn)

原始链表

现实中跳表的形态



↔ 对元素的增加和删除，

导致有些数并不是完全非常工整的

维护成本较高 ↓

索引更新 x n times

跳表的空间复杂度分析

原始链表大小为 n，每 2 个结点抽 1 个，每层索引的结点数：

$\frac{n}{2}, \frac{n}{4}, \frac{n}{8}, \cdots, 8,4,2$

原始链表大小为 n，每 3 个结点抽 1 个，每层索引的结点数：

$\frac{n}{3}, \frac{n}{9}, \frac{n}{27}, \cdots, 9,3,1$

空间复杂度是 O(n)

工程中的应用

LRU Cache

Redis