算法学习心得

20210421

## 数组：为什么很多编程语言数组都从0开始编号？

一、数组（Array）是一种线性表数据结构。它用一组连续的内存空间，来存储一组具有相同类型的数据。

二、第一是线性表（Linear List）。顾名思义，线性表就是数据排成像一条线一样的结构。每个线性表上的数据最多只有前和后两个方向。其实除了数组，链表、队列、栈等也是线性表结构。

三、非线性表，比如二叉树、堆、图等。之所以叫非线性，是因为，在非线性表中，数据之间并不是简单的前后关系。

四、你用二分查找，时间复杂度也是 O(logn)。所以，正确的表述应该是，数组支持随机访问，根据下标随机访问的时间复杂度为 O(1)。

五、ArrayList 最大的优势就是可以将很多数组操作的细节封装起来。比如前面提到的数组插入、删除数据时需要搬移其他数据等。另外，它还有一个优势，就是支持动态扩容。

六、对于业务开发，直接使用容器就足够了，省时省力。毕竟损耗一丢丢性能，完全不会影响到系统整体的性能。但如果你是做一些非常底层的开发，比如开发网络框架，性能的优化需要做到极致，这个时候数组就会优于容器，成为首选。

七、我们今天学习了数组。它可以说是最基础、最简单的数据结构了。数组用一块连续的内存空间，来存储相同类型的一组数据，最大的特点就是支持随机访问，但插入、删除操作也因此变得比较低效，平均情况时间复杂度为 O(n)。在平时的业务开发中，我们可以直接使用编程语言提供的容器类，但是，如果是特别底层的开发，直接使用数组可能会更合适。

八、JVM标记清除算法：

大多数主流虚拟机采用可达性分析算法来判断对象是否存活，在标记阶段，会遍历所有 GC ROOTS，将所有 GC ROOTS 可达的对象标记为存活。只有当标记工作完成后，清理工作才会开始。  
不足：1.效率问题。标记和清理效率都不高，但是当知道只有少量垃圾产生时会很高效。2.空间问题。会产生不连续的内存空间碎片。  
二维数组内存寻址：  
对于 m \* n 的数组，a [ i ][ j ] (i < m,j < n)的地址为：  
address = base\_address + ( i \* n + j) \* type\_size  
另外，对于数组访问越界造成无限循环，我理解是编译器的问题，对于不同的编译器，在内存分配时，会按照内存地址递增或递减的方式进行分配。老师的程序，如果是内存地址递减的方式，就会造成无限循环。

## 链表（上）：如何实现LRU缓存淘汰算法？

一、LRU 缓存淘汰算法

先进先出策略 FIFO（First In，First Out）、最少使用策略 LFU（Least Frequently Used）、最近最少使用策略 LRU（Least Recently Used）。

二、三种最常见的链表结构，它们分别是：单链表、双向链表和循环链表。我们首先来看最简单、最常用的单链表

三、双向链表可以支持 O(1) 时间复杂度的情况下找到前驱结点，正是这样的特点，也使双向链表在某些情况下的插入、删除等操作都要比单链表简单、高效。

四、除此之外，如果你的代码对内存的使用非常苛刻，那数组就更适合你。因为链表中的每个结点都需要消耗额外的存储空间去存储一份指向下一个结点的指针，所以内存消耗会翻倍。而且，对链表进行频繁的插入、删除操作，还会导致频繁的内存申请和释放，容易造成内存碎片，如果是 Java 语言，就有可能会导致频繁的 GC（Garbage Collection，垃圾回收）。

五、

如何基于链表实现 LRU 缓存淘汰算法？我的思路是这样的：我们维护一个有序单链表，越靠近链表尾部的结点是越早之前访问的。当有一个新的数据被访问时，我们从链表头开始顺序遍历链表。

1. 如果此数据之前已经被缓存在链表中了，我们遍历得到这个数据对应的结点，并将其从原来的位置删除，然后再插入到链表的头部。

2. 如果此数据没有在缓存链表中，又可以分为两种情况：如果此时缓存未满，则将此结点直接插入到链表的头部；如果此时缓存已满，则链表尾结点删除，将新的数据结点插入链表的头部。

这样我们就用链表实现了一个 LRU 缓存，是不是很简单？现在我们来看下缓存访问的时间复杂度是多少。因为不管缓存有没有满，我们都需要遍历一遍链表，所以这种基于链表的实现思路，缓存访问的时间复杂度为 O(n)。

实际上，我们可以继续优化这个实现思路，比如引入散列表（Hash table）来记录每个数据的位置，将缓存访问的时间复杂度降到 O(1)。因为要涉及我们还没有讲到的数据结构，所以这个优化方案，我现在就不详细说了，等讲到散列表的时候，我会再拿出来讲。

六、

## 链表（下）：如何轻松写出正确的链表代码？

一、

单链表反转

链表中环的检测

两个有序的链表合并

删除链表倒数第 n 个结点

求链表的中间结点

二、

写链表代码是最考验逻辑思维能力的

三、

我经常用来检查链表代码是否正确的边界条件有这样几个：

如果链表为空时，代码是否能正常工作？

如果链表只包含一个结点时，代码是否能正常工作？

如果链表只包含两个结点时，代码是否能正常工作？

代码逻辑在处理头结点和尾结点的时候，是否能正常工作？

四、

这节我主要和你讲了写出正确链表代码的六个技巧。

分别是理解指针或引用的含义、

警惕指针丢失和内存泄漏、

利用哨兵简化实现难度、

重点留意边界条件处理，

以及举例画图、辅助思考，

还有多写多练。

五、