06 | 链表（上）：如何实现LRU缓存淘汰算法?

2018-10-03 王争

06 | 链表（上）：如何实现LRU缓存淘汰算法?
朗读人：修阳　17′06″ | 7.85M

今天我们来聊聊“链表（Linked list）”这个数据结构。学习链表有什么用呢？为了回答这个问题，我们先来讨论一个经典的链表应用场景，那就是 LRU 缓存淘汰算法。

缓存是一种提高数据读取性能的技术，在硬件设计、软件开发中都有着非常广泛的应用，比如常见的 CPU 缓存、数据库缓存、浏览器缓存等等。

缓存的大小有限，当缓存被用满时，哪些数据应该被清理出去，哪些数据应该被保留？这就需要缓存淘汰策略来决定。常见的策略有三种：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）。

这些策略你不用死记，我打个比方你很容易就明白了。假如说，你买了很多本技术书，但有一天你发现，这些书太多了，太占书房空间了，你要做个大扫除，扔掉一些书籍。那这个时候，你会选择扔掉哪些书呢？对应一下，你的选择标准是不是和上面的三种策略神似呢？

好了，回到正题，我们今天的开篇问题就是：如何用链表来实现 LRU 缓存淘汰策略呢？ 带着这个问题，我们开始今天的内容吧！

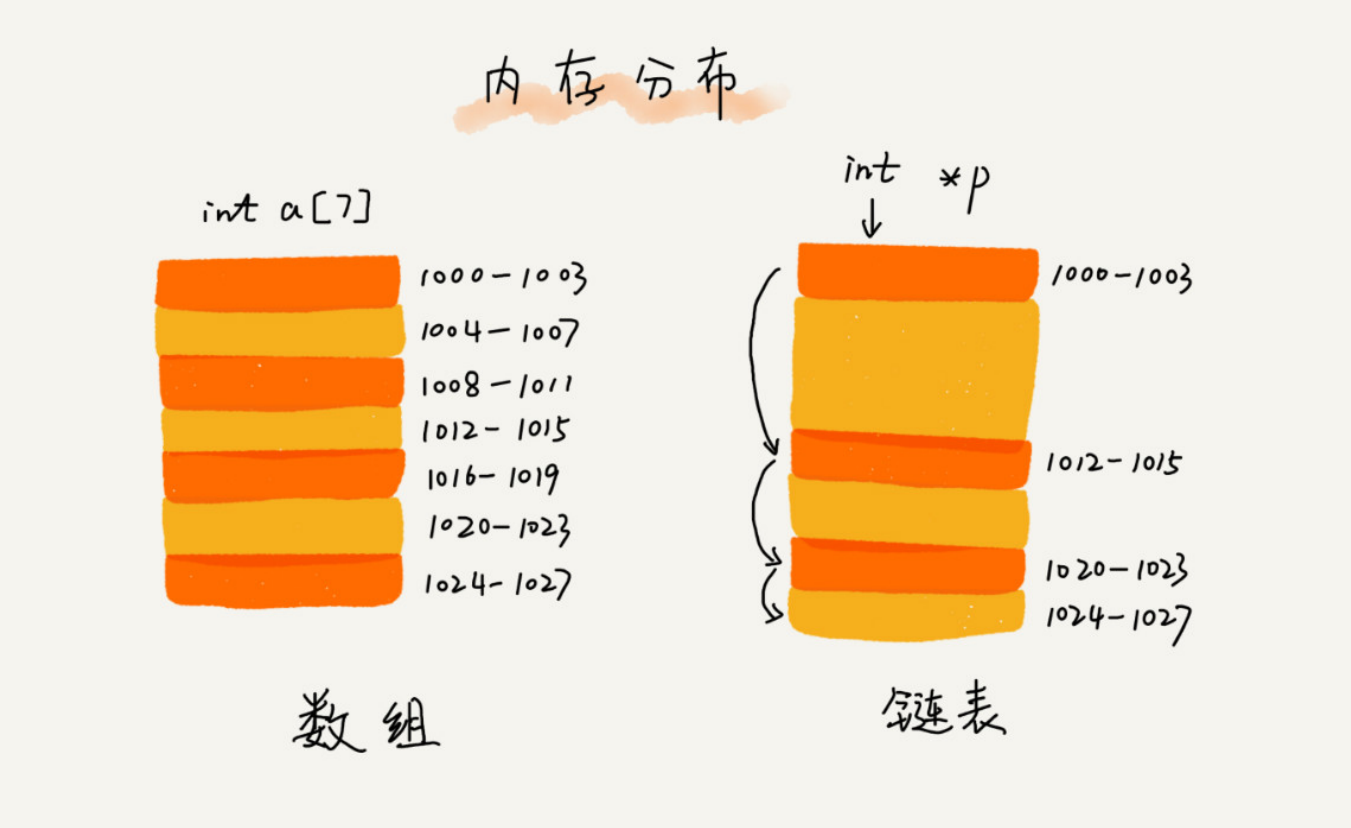
五花八门的链表结构

相比数组，链表是一种稍微复杂一点的数据结构。对于初学者来说，掌握起来也要比数组稍难一些。这两个非常基础、非常常用的数据结构，我们常常将会放到一块儿来比较。所以我们先来看，这两者有什么区别。

我们先从**底层的存储结构**上来看一看。

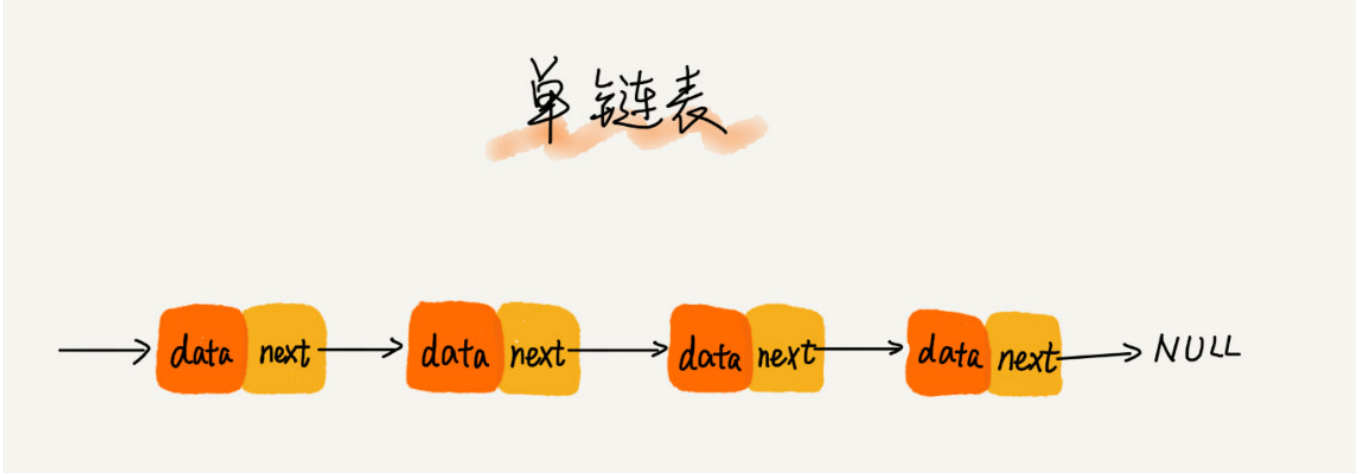
为了直观地对比，我画了一张图。从图中我们看到，数组需要一块**连续的内存空间**来存储，对内存的要求比较高。如果我们申请一个 100MB 大小的数组，当内存中没有连续的、足够大的存储空间时，即便内存的剩余总可用空间大于 100MB，仍然会申请失败。

而链表恰恰相反，它并不需要一块连续的内存空间，它通过“指针”将一组**零散的内存块**串联起来使用，所以如果我们申请的是 100MB 大小的链表，根本不会有问题。



链表结构五花八门，今天我重点给你介绍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**。

我们刚刚讲到，链表通过指针将一组零散的内存块串联在一起。其中，我们把内存块称为链表的“**结点**”。为了将所有的结点串起来，每个链表的结点除了存储数据之外，还需要记录链上的下一个结点的地址。如图所示，我们把这个记录下个结点地址的指针叫作**后继指针 next**。

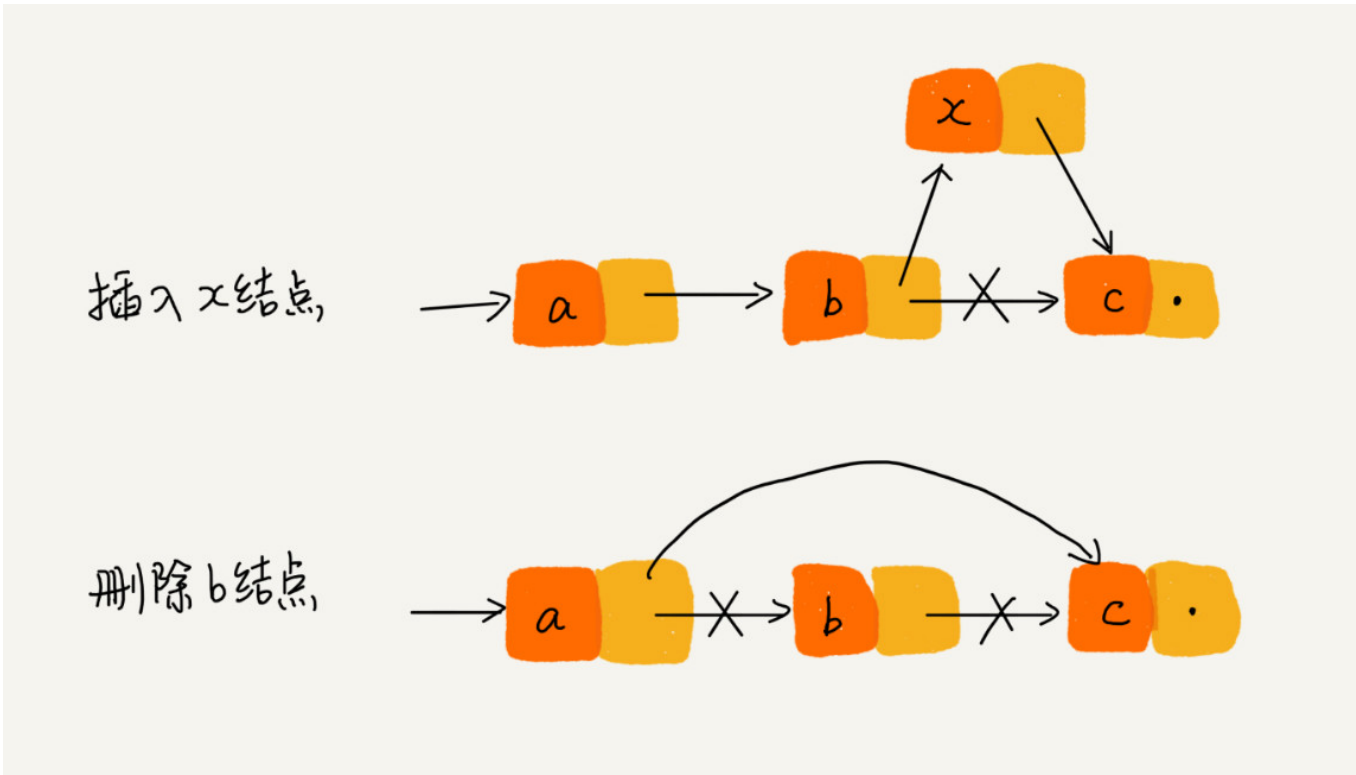


从我画的单链表图中，你应该可以发现，其中有两个结点是比较特殊的，它们分别是第一个结点和最后一个结点。我们习惯性地把第一个结点叫作**头结点**，把最后一个结点叫作**尾结点**。其中，头结点用来记录链表的基地址。有了它，我们就可以遍历得到整条链表。而尾结点特殊的地方是：指针不是指向下一个结点，而是指向一个**空地址 NULL**，表示这是链表上最后一个结点。

与数组一样，链表也支持数据的查找、插入和删除操作。

我们知道，在进行数组的插入、删除操作时，为了保持内存数据的连续性，需要做大量的数据搬移，所以时间复杂度是 O(n)。而在链表中插入或者删除一个数据，我们并不需要为了保持内存的连续性而搬移结点，因为链表的存储空间本身就不是连续的。所以，在链表中插入和删除一个数据是非常快速的。

为了方便你理解，我画了一张图，从图中我们可以看出，针对链表的插入和删除操作，我们只需要考虑相邻结点的指针改变，所以对应的时间复杂度是 O(1)。

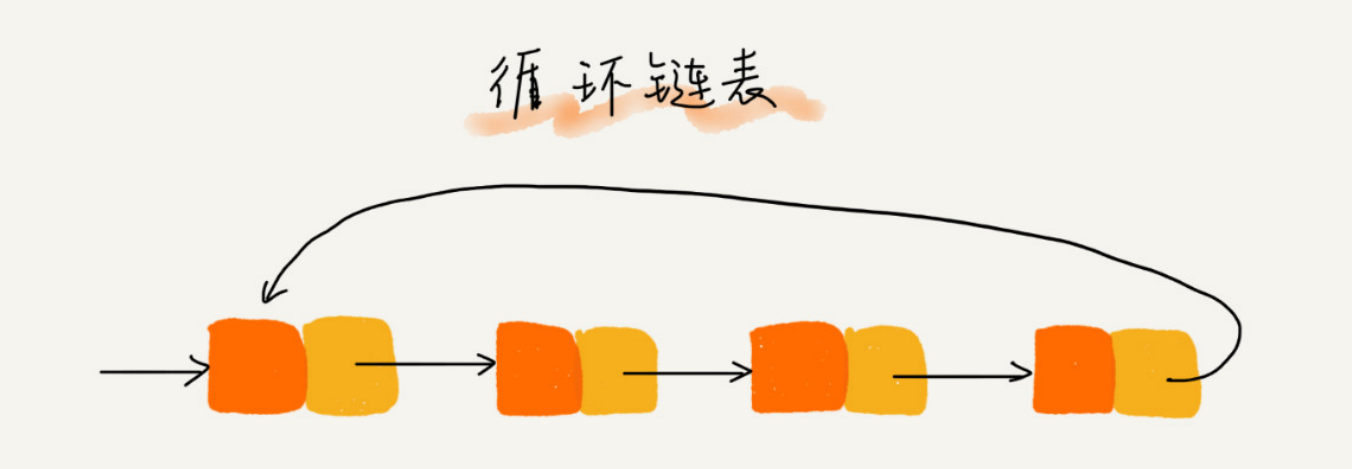


但是，有利就有弊。链表要想随机访问第 k 个元素，就没有数组那么高效了。因为链表中的数据并非连续存储的，所以无法像数组那样，根据首地址和下标，通过寻址公式就能直接计算出对应的内存地址，而是需要根据指针一个结点一个结点地依次遍历，直到找到相应的结点。

你可以把链表想象成一个队伍，队伍中的每个人都只知道自己后面的人是谁，所以当我们希望知道排在第 k 位的人是谁的时候，我们就需要从第一个人开始，一个一个地往下数。所以，链表随机访问的性能没有数组好，需要 O(n) 的时间复杂度。

好了，单链表我们就简单介绍完了，接着来看另外两个复杂的升级版，**循环链表**和**双向链表**。

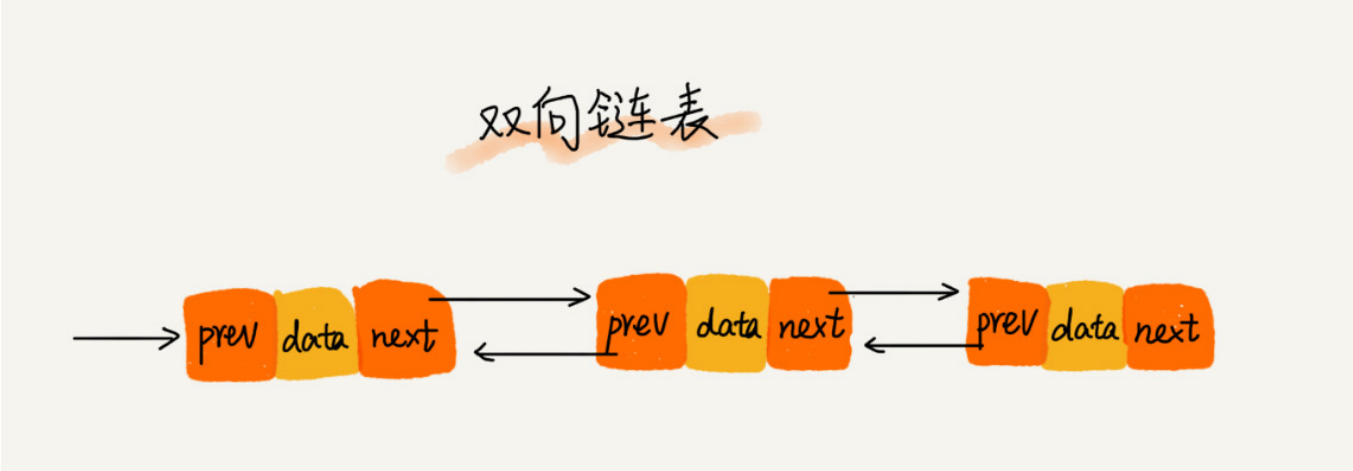
循环链表是一种特殊的单链表。实际上，循环链表也很简单。它跟单链表唯一的区别就在尾结点。我们知道，单链表的尾结点指针指向空地址，表示这就是最后的结点了。而循环链表的尾结点指针是指向链表的头结点。从我画的循环链表图中，你应该可以看出来，它像一个环一样首尾相连，所以叫作“循环”链表。



和单链表相比，**循环链表**的优点是从链尾到链头比较方便。当要处理的数据具有环型结构特点时，就特别适合采用循环链表。比如著名的约瑟夫问题。尽管用单链表也可以实现，但是用循环链表实现的话，代码就会简洁很多。

单链表和循环链表是不是都不难？接下来我们再来看一个稍微复杂的，在实际的软件开发中，也更加常用的链表结构：**双向链表**。

单向链表只有一个方向，结点只有一个后继指针 next 指向后面的结点。而双向链表，顾名思义，它支持两个方向，每个结点不止有一个后继指针 next 指向后面的结点，还有一个前驱指针 prev 指向前面的结点。



从我画的图中可以看出来，双向链表需要额外的两个空间来存储后继结点和前驱结点的地址。所以，如果存储同样多的数据，双向链表要比单链表占用更多的内存空间。虽然两个指针比较浪费存储空间，但可以支持双向遍历，这样也带来了双向链表操作的灵活性。那相比单链表，双向链表适合解决哪种问题呢？

从结构上来看，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。

你可能会说，我刚讲到单链表的插入、删除操作的时间复杂度已经是 O(1) 了，双向链表还能再怎么高效呢？别着急，刚刚的分析比较偏理论，很多数据结构和算法书籍中都会这么讲，但是这种说法实际上是不准确的，或者说是有先决条件的。我再来带你分析一下链表的两个操作。

我们先来看**删除操作**。

在实际的软件开发中，从链表中删除一个数据无外乎这两种情况：

- 删除结点中“值等于某个给定值”的结点；
- 删除给定指针指向的结点。

对于第一种情况，不管是单链表还是双向链表，为了查找到值等于给定值的结点，都需要从头结点开始一个一个依次遍历对比，直到找到值等于给定值的结点，然后再通过我前面讲的指针操作将其删除。

尽管单纯的删除操作时间复杂度是 O(1)，但遍历查找的时间是主要的耗时点，对应的时间复杂度为 O(n)。根据时间复杂度分析中的加法法则，删除值等于给定值的结点对应的链表操作的总时间复杂度为 O(n)。

对于第二种情况，我们已经找到了要删除的结点，但是删除某个结点 q 需要知道其前驱结点，而单链表并不支持直接获取前驱结点，所以，为了找到前驱结点，我们还是要从头结点开始遍历链表，直到 p->next=q，说明 p 是 q 的前驱结点。

但是对于双向链表来说，这种情况就比较有优势了。因为双向链表中的结点已经保存了前驱结点的指针，不需要像单链表那样遍历。所以，针对第二种情况，单链表删除操作需要 O(n) 的时间复杂度，而双向链表只需要在 O(1) 的时间复杂度内就搞定了！

同理，如果我们希望在链表的某个指定结点前面插入一个结点，双向链表比单链表有很大的优势。双向链表可以在 O(1) 时间复杂度搞定，而单向链表需要 O(n) 的时间复杂度。你可以参照我刚刚讲过的删除操作自己分析一下。

除了插入、删除操作有优势之外，对于一个有序链表，双向链表的按值查询的效率也要比单链表高一些。因为，我们可以记录上次查找的位置 p，每次查询时，根据要查找的值与 p 的大小关系，决定是往前还是往后查找，所以平均只需要查找一半的数据。

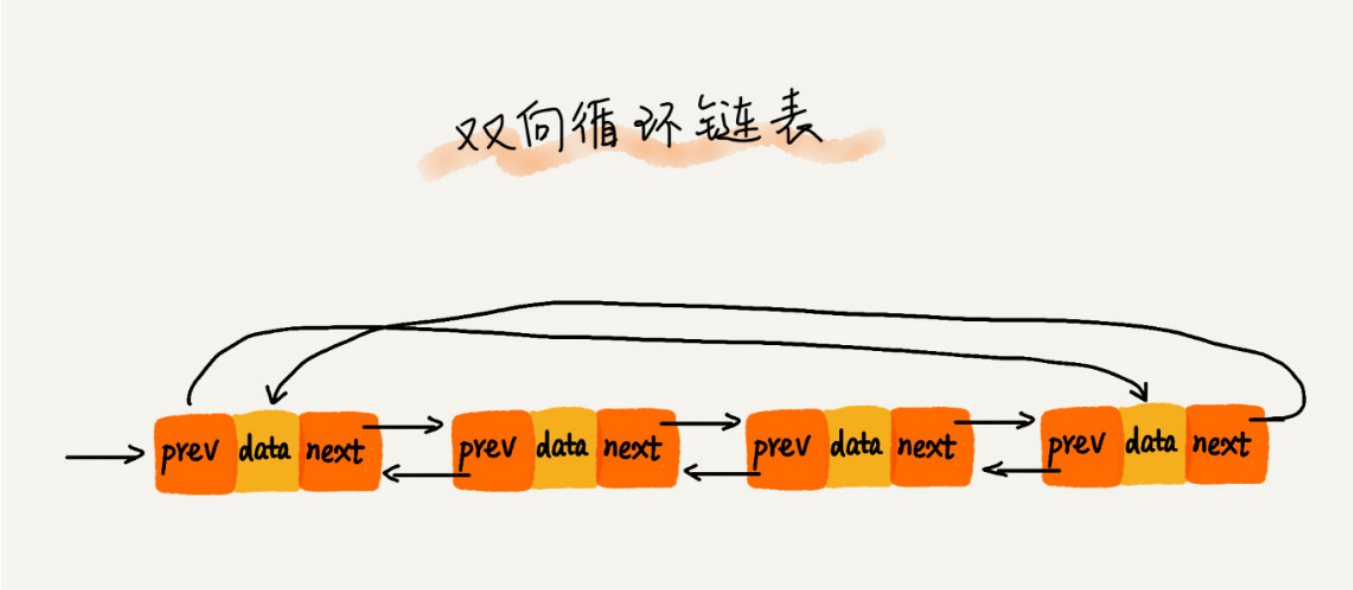
现在，你有没有觉得双向链表要比单链表更加高效呢？这就是为什么在实际的软件开发中，双向链表尽管比较费内存，但还是比单链表的应用更加广泛的原因。如果你熟悉 Java 语言，你肯定用过 LinkedHashMap 这个容器。如果你深入研究 LinkedHashMap 的实现原理，就会发现其中就用到了双向链表这种数据结构。

实际上，这里有一个更加重要的知识点需要你掌握，那就是**用空间换时间**的设计思想。当内存空间充足的时候，如果我们更加追求代码的执行速度，我们就可以选择空间复杂度相对较高、但时间复杂度相对很低的算法或者数据结构。相反，如果内存比较紧缺，比如代码跑在手机或者单片机上，这个时候，就要反过来用时间换空间的设计思路。

还是开篇缓存的例子。缓存实际上就是利用了空间换时间的设计思想。如果我们把数据存储在硬盘上，会比较节省内存，但每次查找数据都要询问一次硬盘，会比较慢。但如果我们通过缓存技术，事先将数据加载在内存中，虽然会比较耗费内存空间，但是每次数据查询的速度就大大提高了。

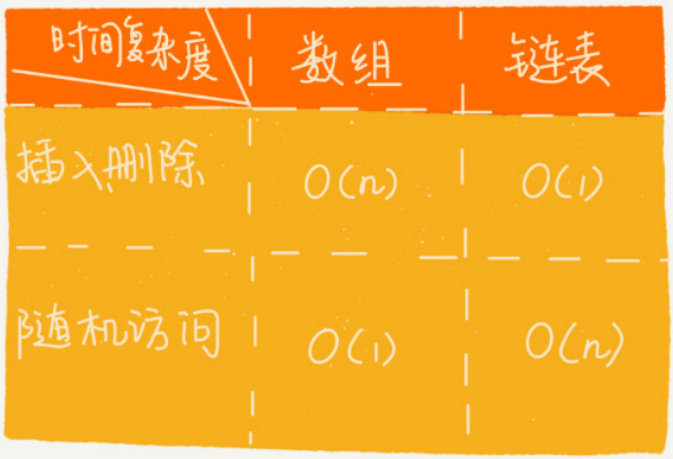
所以我总结一下，对于执行较慢的程序，可以通过消耗更多的内存（空间换时间）来进行优化；而消耗过多内存的程序，可以通过消耗更多的时间（时间换空间）来降低内存的消耗。你还能想到其他时间换空间或者空间换时间的例子吗？

了解了循环链表和双向链表，如果把这两种链表整合在一起就是一个新的版本：**双向循环链表**。我想不用我多讲，你应该知道双向循环链表长什么样子了吧？你可以自己试着在纸上画一画。



链表 VS 数组性能大比拼

通过前面内容的学习，你应该已经知道，数组和链表是两种截然不同的内存组织方式。正是因为内存存储的区别，它们插入、删除、随机访问操作的时间复杂度正好相反。



不过，数组和链表的对比，并不能局限于时间复杂度。而且，在实际的软件开发中，不能仅仅利用复杂度分析就决定使用哪个数据结构来存储数据。

数组简单易用，在实现上使用的是连续的内存空间，可以借助 CPU 的缓存机制，预读数组中的数据，所以访问效率更高。而链表在内存中并不是连续存储，所以对 CPU 缓存不友好，没办法有效预读。

数组的缺点是大小固定，一经声明就要占用整块连续内存空间。如果声明的数组过大，系统可能没有足够的连续内存空间分配给它，导致“内存不足（out of memory）”。如果声明的数组过小，则可能出现不够用的情况。这时只能再申请一个更大的内存空间，把原数组拷贝进去，非常费时。链表本身没有大小的限制，天然地支持动态扩容，我觉得这也是它与数组最大的区别。

你可能会说，我们 Java 中的 ArrayList 容器，也可以支持动态扩容啊？我们上一节课讲过，当我们往支持动态扩容的数组中插入一个数据时，如果数组中没有空闲空间了，就会申请一个更大的空间，将数据拷贝过去，而数据拷贝的操作是非常耗时的。

我举一个稍微极端的例子。如果我们用 ArrayList 存储了了 1GB 大小的数据，这个时候已经没有空闲空间了，当我们再插入数据的时候，ArrayList 会申请一个 1.5GB 大小的存储空间，并且把原来那 1GB 的数据拷贝到新申请的空间上。听起来是不是就很耗时？

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）。

所以，在我们实际的开发中，针对不同类型的项目，要根据具体情况，权衡究竟是选择数组还是链表。

解答开篇

好了，关于链表的知识我们就讲完了。我们现在回过头来看下开篇留给你的思考题。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 m 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。

除了基于链表的实现思路，实际上还可以用数组来实现 LRU 缓存淘汰策略。如何利用数组实现 LRU 缓存淘汰策略呢？我把这个问题留给你思考。

内容小结

今天我们讲了一种跟数组“相反”的数据结构，链表。它跟数组一样，也是非常基础、非常常用的数据结构。不过链表要比数组稍微复杂，从普通的单链表衍生出来好几种链表结构，比如双向链表、循环链表、双向循环链表。

和数组相比，链表更适合插入、删除操作频繁的场景，查询的时间复杂度较高。不过，在具体软件开发中，要对数组和链表的各种性能进行对比，综合来选择使用两者中的哪一个。

课后思考

如何判断一个字符串是否是回文字符串的问题，我想你应该听过，我们今天的思题目就是基于这个问题的改造版本。如果字符串是通过单链表来存储的，那该如何来判断是一个回文串呢？你有什么好的解决思路呢？相应的时间空间复杂度又是多少呢？

欢迎留言和我分享，我会第一时间给你反馈。

戳此查看本节内容相关的详细代码

写留言

精选留言

姜威

34

五、应用

1.如何分别用链表和数组实现LRU缓冲淘汰策略?

1）什么是缓存?

缓存是一种提高数据读取性能的技术，在硬件设计、软件开发中都有着非广泛的应用，比如常见的CPU缓存、数据库缓存、浏览器缓存等等。

2）为什么使用缓存？即缓存的特点

缓存的大小是有限的，当缓存被用满时，哪些数据应该被清理出去，哪些数据应该被保留？就需要用到缓存淘汰策略。

3）什么是缓存淘汰策略?

指的是当缓存被用满时清理数据的优先顺序。

4）有哪些缓存淘汰策略?

常见的3种包括先进先出策略FIFO（First In，First Out）、最少使用策略LFU（Least Frenquently Used）、最近最少使用策略LRU（Least Recently Used）。

5）链表实现LRU缓存淘汰策略

当访问的数据没有存储在缓存的链表中时，直接将数据插入链表表头，时间复杂度为O(1)；当访问的数据存在于存储的链表中时，将该数据对应的节点，插入到链表表头,时间复杂度为O(n)。如果缓存被占满，则从链表尾部的数据开始清理，时间复杂度为O(1)。

6）数组实现LRU缓存淘汰策略

方式一：首位置保存最新访问数据，末尾位置优先清理

当访问的数据未存在于缓存的数组中时，直接将数据插入数组第一个元素位置，此时数组所有元素需要向后移动1个位置，时间复杂度为O(n)；当访问的数据存在于缓存的数组中时，查找到数据并将其插入数组的第一个位置，此时亦需移动数组元素，时间复杂度为O(n)。缓存用满时，则清理掉末尾的数据，时间复杂度为O(1)。

方式二：首位置优先清理，末尾位置保存最新访问数据

当访问的数据未存在于缓存的数组中时，直接将数据添加进数组作为当前最有一个元素时间复杂度为O(1)；当访问的数据存在于缓存的数组中时，查找到数据并将其插入当前数组最后一个元素的位置，此时亦需移动数组元素，时间复杂度为O(n)。缓存用满时，则清理掉数组首位置的元素，且剩余数组元素需整体前移一位，时间复杂度为O(n)。（优化：清理的时候可以考虑一次性清理一定数量，从而降低清理次数，提高性能。）

2.如何通过单链表实现“判断某个字符串是否为水仙花字符串”？（比如 上海自来水来自海上）

1）前提：字符串以单个字符的形式存储在单链表中。

2）遍历链表，判断字符个数是否为奇数，若为偶数，则不是。

3）将链表中的字符倒序存储一份在另一个链表中。

4）同步遍历2个链表，比较对应的字符是否相等，若相等，则是水仙花字串，否则，不是。

六、设计思想

时空替换思想：“用空间换时间”与“用时间换空间”

当内存空间充足的时候，如果我们更加追求代码的执行速度，我们就可以选择空间复杂度相对较高，时间复杂度小相对较低的算法和数据结构，缓存就是空间换时间的例子。如果内存比较紧缺，比如代码跑在手机或者单片机上，这时，就要反过来用时间换空间的思路。

2018-10-03

作者回复

👍

2018-10-03

JK David 14

思考题：

使用快慢两个指针找到链表中点，慢指针每次前进一步，快指针每次前进两步。在慢指针前进的过程中，同时修改其 next 指针，使得链表前半部分反序。最后比较中点两侧的链表是否相等。

时间复杂度：O(n)
空间复杂度：O(1)

https://github.com/andavid/leetcode-java/blob/master/note/234/README.md

2018-10-03

作者回复

思路正确，不过空间复杂度计算的不对，应该是O(1)，不是O(n)。我们要看额外的内存消耗，不是看链表本身存储需要多少空间。

2018-10-04

_stuView 14

双向链表存储，两个指针分别从头节点和尾节点开始遍历，依次比较节点value，判断是否为回文序列

2018-10-03

雨山 14

果然有程序员风格，放假还更新，昨天临睡前就看完了，但是没有评价，总之这个课绝对物有所值。

2018-10-03

姜威 12

总结

一、什么是链表?

1.和数组一样，链表也是一种线性表。

2.从内存结构来看，链表的内存结构是不连续的内存空间，是将一组零散的内存块串联起来，从而进行数据存储的数据结构。

3.链表中的每一个内存块被称为节点Node。节点除了存储数据外，还需记录链上下一个节点的地址，即后继指针next。

二、为什么使用链表？即链表的特点

1.插入、删除数据效率高O(1)级别（只需更改指针指向即可），随机访问效率低O(n)级别（需要从链头至链尾进行遍历）。

2.和数组相比，内存空间消耗更大，因为每个存储数据的节点都需要额外的空间存储后继指针。

三、常用链表：单链表、循环链表和双向链表

1.单链表

1）每个节点只包含一个指针，即后继指针。

2）单链表有两个特殊的节点，即首节点和尾节点。为什么特殊？用首节点地址表示整条链表，尾节点的后继指针指向空地址null。

3）性能特点：插入和删除节点的时间复杂度为O（1），查找的时间复杂度为O(n)。

2.循环链表

1）除了尾节点的后继指针指向首节点的地址外均与单链表一致。

2）适用于存储有循环特点的数据，比如约瑟夫问题。

3.双向链表

1）节点除了存储数据外，还有两个指针分别指向前一个节点地址（前驱指针prev）和下一个节点地址（后继指针next）。

2）首节点的前驱指针prev和尾节点的后继指针均指向空地址。

3）性能特点：

和单链表相比，存储相同的数据，需要消耗更多的存储空间。

插入、删除操作比单链表效率更高O(1)级别。以删除操作为例，删除操作分为2种情况：给定数据值删除对应节点和给定节点地址删除节点。对于前一种情况，单链表和双向链表都需要从头到尾进行遍历从而找到对应节点进行删除，时间复杂度为O(n)。对于第二种情况，要进行删除操作必须找到前驱节点，单链表需要从头到尾进行遍历直到p->next = q，时间复杂度为O(n)，而双向链表可以直接找到前驱节点，时间复杂度为O(1)。

对于一个有序链表，双向链表的按值查询效率要比单链表高一些。因为我们可以记录上次查找的位置p，每一次查询时，根据要查找的值与p的大小关系，决定是往前还是往后查找，所以平均只需要查找一半的数据。

4.双向循环链表：首节点的前驱指针指向尾节点，尾节点的后继指针指向首节点。

四、选择数组还是链表?

1.插入、删除和随机访问的时间复杂度

数组：插入、删除的时间复杂度是O(n)，随机访问的时间复杂度是O(1)。

链表：插入、删除的时间复杂度是O(1)，随机访问的时间复杂端是O(n)。

2.数组缺点

1）若申请内存空间很大，比如100M，但若内存空间没有100M的连续空间时，则会申请失败，尽管内存可用空间超过100M。

2）大小固定，若存储空间不足，需进行扩容，一旦扩容就要进行数据复制，而这时非常费时的。

3.链表缺点

1）内存空间消耗更大，因为需要额外的空间存储指针信息。

2）对链表进行频繁的插入和删除操作，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，还可能会造成频繁的GC（自动垃圾回收器）操作。

4.如何选择?

数组简单易用，在实现上使用连续的内存空间，可以借助CPU的缓冲机制预读数组中的数据，所以访问效率更高，而链表在内存中并不是连续存储，所以对CPU缓存不友好，没办法预读。

如果代码对内存的使用非常苛刻，那数组就更适合。

2018-10-03

作者回复

👍

2018-10-03

Joshua 兆甲 12

习题解答

1.快进慢进法[两组指针，从头开始，a组一次进一，b组一次进二，b组到终点时，a组位置即为链表中间结点，循环次数为链表除去中间结点后前后两组的长度] 求得单向链表“中间”节点。并计算遍历次数，经过验证，遍历次数为‘’半链表‘’长度

2.从中间结点开始，以动态步长[每第i次步长是半链表长度-i+1]遍历链表，同时，从头节点开始，以1步长遍历。比较两组对应元素是否相同，相同继续，不同退出，返回不是回文字符串的结论。

3.返回是回文字符串的结论，退出.

空间复杂度O(n). 不用连续内存，可以磁盘操作

时间复杂度度O(n). 主要费时操作遍历

算了，不够直观，不易别人看懂。还是先把单项链表转存为线性表。

1.单向遍历，获得对应的线性表Arr，求线性表长度为L

2.运用线性表可以任意访问的性质，遍历Arr，令下标i从0。比较Arr[i]和Arr[L－i]是否相等 相等继续，不等报告不是回文字符串结论，退出

3.报告是回文字符串结论，结束。

空间复杂度O(n)

时间复杂度O(n)

看起来一样，这个就需要字符串不太大，有足够的连续内存可以分配，而且，预先不知道链表多长，可能还会遇到扩容问题。

2018-10-03

落叶飞逝的恋 7

老师，关于解答开篇那边，能不能附加一些代码示例，这样配合代码跟思路讲解，可能更好的理解呢。

2018-10-03

Liam 4

1 快慢指针定位中间节点

2 从中间节点对后半部分逆序

3 前后半部分比较，判断是否为回文

4 后半部分逆序复原

时间复杂度On, 空间复杂度O1

把LRU和回文都实现了一遍~~

如果是双向链表，时间效率更高，看了下LinkedList，底层也是用双向链表实现

2018-10-03

作者回复

回答的很好！👍

2018-10-03

Smallfly 4

思考题：根据原有单链表回文创建一个逆向的单链表回文，while 循环遍历比较，复杂度为 O(N)。

2018-10-03

青柠 3

用数组解决Lru缓存问题：
维护一个有序的数组，越靠近数组首位置的数据越是最早访问的。
1.如果这个数据已经存在于数组中，把对应位置的数据删掉，直接把这个数据加到数组的最后一位。时间复杂度为o(n)
2.如果数据不存在这个数组中，数据还有空间的话，就把数据直接插到最后一位。没有的话，就把第一个数据删掉，然后把数据插入到数组最后一个。这样的时间复杂度为o(n) 。

第一个小伙伴的留言有点问题。判断是否为回文串和奇偶数没关系吧，偶数个字符串也可以是哈，比如abccba。

2018-10-05

六六六 3

判断单链表是否是回文，只想到了这种low一些的做法，时间复杂度为O(n^2)：

```
public static boolean isHuiwen(LinkedList linkedList) {
Node first = linkedList.getFirst();
int size = linkedList.getSize();
Node head = null;
Node foot = null;
for (int a = 0; a < size / 2; a++) {
head = head == null ? first : head.next;
foot = head;
for (int i = a; i < size – 1 – a; i++) {
foot = foot.next;
}
if (!head.getData().equals(foot.getData())) {
return false;
}
}
return true;
}
```

2018-10-03

James Scott 3

回复一下CaiBird的提问：应该是写错了，应该是越靠近链表头部的节点是越早被访问的。

2018-10-03

作者回复

谢谢你🙏 不过我没有写错。已经回复给他。不过，你为什么觉得我写错了呢?

2018-10-04

yaya 3

思考题:先用快慢指针得中点，然后将后段链表逆序，比较是否是回文串。再将链表逆序回来，时间复杂度o（n）,空间复杂度o（1）.
利用数组实现lru,想法是利用堆的思想，根据时间最远最少用的规则建立堆，利用结构体维护最近的访问时间，根据其建堆，如果一个新的元素来，遍历数组，如果能找到，更新他的参数，调整堆。如果不能就将堆的头元素删除，插入该元素

2018-10-03

徐凯 3

通过一个栈 遍历整个链表 然后再从栈中弹出 如果元素都匹配则为回文

2018-10-03

Rain 2

Re Ydyhm:

“数组简单易用，在实现上使用的是连续的内存空间，可以借助 CPU 的缓存机制，预读数组中的数据，所以访问效率更高。而链表在内存中并不是连续存储，所以对 CPU 缓存不友好，没办法有效预读。”这里的CPU缓存机制指的是什么？为什么就数组更好了？

我没有百度也没有Google。之前开发时遇到过，我斗胆说下。
CPU在从内存读取数据的时候，会先把读取到的数据加载到CPU的缓存中。而CPU每次从内存读取数据并不是只读取那个特定要访问的地址，而是读取一个数据块(这个大小我不太确定。。)并保存到CPU缓存中，然后下次访问内存数据的时候就会先从CPU缓存开始查找，如果找到就不需要再从内存中取。这样就实现了比内存访问速度更快的机制，也就是CPU缓存存在的意义:为了弥补内存访问速度过慢与CPU执行速度快之间的差异而引入。

对于数组来说，存储空间是连续的，所以在加载某个下标的时候可以把以后的几个下标元素也加载到CPU缓存这样执行速度会快于存储空间不连续的链表存储。

大牛请指正哈!

2018-10-04

作者回复

同学，太爱你了。写的太好了！就喜欢你这样的，减轻了我很多回复留言的工作量。👍

2018-10-04

港

2

创建一个与原来链表相同的反向链表，然后一个一个结点比对，时间复杂度是O(n)，空间复杂度是O(n)。

2018-10-03

fiseasky

2

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。

1。LRU算法这点不太理解，为什么都在cache中，还有删除掉重新插入呢?

2. 这个LRU算法，cache过期了，如何更新呢？请老师指点一下。

2018-10-03

作者回复

1. 因为链表是有序的，按照访问时间排序；

2. cache怎么过期的呢？麻烦说详细点。

2018-10-03

Kudo

2

>> 思考题：用数组来实现 LRU 缓存淘汰策略。

为了使访问更有效率，我们用数组尾部的结点存储最近访问的数据。

1. 如果此数据之前已经被缓存在数组中了，反向遍历得到这个数据对应的结点，将其放在数组尾部，并将原先该结果后面的结点分别向前移动一个单位。

2. 如果此数据没有在缓存数组中，又可分两种情况。

（1）如果此时缓存未满，则将此结点插入到数组的尾部；

（2）如果此时缓存已满，则将数组头部结点删除，将新的数据结点插入数组尾部，并将其它所有节点分别前移一个单位。

该操作的时间复杂度为O(n)，相对链表实现方式涉及更多的数据移动动操作，因此访问效率要低一些。

2018-10-03

🤔

1

理论大概明白，但是还有个问题，用js实现了个链表类，并且模拟Array.prototype.splice实现了链表的在指定下标插入操作，因为要循环遍历找到指定下标前的节点所以时间复杂度应该是O(n)，splice应该也是O(n)，但用console.time测了下循环10000次插入操作，链表插入比数组splice快了6ms，仅仅是因为splice执行了其他额外操作吗？怎么理解这种相同复杂度下两种不同结构造成的差异呢?

2018-10-04

阳仔

1

学习反馈：
链表也是一种基础的线性表结构。由于它的很多特点跟数组是相反的，因此可以与数组一起对比着学习。
数组的存储空间是连续，而链表不是；数组可以通过寻址公式计算通过下标来访问，而链表访问元素需要遍历。
常见的链表有：
单链表、双向链表、循环链表、双向循环链表。
链表擅长插入、删除操作，时间复杂度为O(1)；查询的效率不高，时间复杂度为O(n)。
数组擅长通过下标随机访问元素，时间复杂度为O(1)；插入、删除的效率不高，时间复杂度为O(n)。
在实际项目开发中，选择数组或者链表不能只关注时间复杂度，还需要考虑具体业务，综合考虑选择数组还是链表。
了解了链表的数据结构，那么实现一个机遇链表数据结构的LRU算法就比较简单了：
从链表中查询此缓存数据是否存在：
1、如果存在，则删除该缓存数据节点，并把数据插入到链表头部的位置；
1、如果不存在，则也考虑两种情况：
1、如果缓存充足，则把数据插入到链表头部的位置；
2、如果缓存不足，则把链表中的末尾节点删除，再把缓存数据插入到头部。
思考题：
如果是只使用单链表的话，假设存储回文的链表是L1，再用一个链表L2来存储逆文；
我的思路是这样：
1、循环这个回文链表L1，在遍历到一半之前把逆文存在一个L2中；
例如L1 为A->B->C->B->A，那么遍历到一半时，L2为：B->A；
偶数和奇数的区别在与中间的节点要不要放在L2中。
2、继续遍历比较L1,L2两个链表各个元素是否相等，如果不相等则立即返回；如果比较到最后遍历结束，则说明是回文；
因此通过一次遍历就知道这个链表是否为回文。时间复杂度为O(n)。

2018-10-03

作者回复

👍

2018-10-03

wistbean 1

--------总结一下--------

常见的链表结构
1.单链表：
每个节点除了存储数据之外，还有记录下一个节点的地址，这样才能串联起来，记录下一个节点地址的指针就是「后续指针next」。
其中有两个特殊的节点为「头结点」和「尾节点」，头结点记录链表的基地址，尾节点记录下一个节点地址是NULL，代表链表的最后一个节点。

链表增删效率高
对于链表（存储数据无需连续性）的增删，只需要考虑指针的改变，复杂度为O（1）。

链表访问慢
由于不像数组那样可以根据首选地址和下标计算出内存地址，只能通过遍历节点获取地址。时间复杂度为0（n）。

2.循环链表
是特殊的单链表，尾节点指针指向链表的头结点。

3.双向链表
支持双向，除了「后续指针next」还有「前驱指针prev」

相比单链表和双链表，增删查效率高，内存消耗高。

空间换时间，时间换空间
对于执行慢的程序-->空间换时间
对于消耗内存多的程序-->时间换空间

链表和数组的区别

底层存储结构：
数组需要一块连续的内存空间存储
链表通过“指针”将一组零散的内存块串联起来使用

性能：
1.链表和数组的（增删查）时间复杂度正好相反。

2.数组使用连续的内存空间，可以借助缓存机制提高效率。
链表不连续，所以无法借助缓存机制。

3.数组大小固定，当要申请更大的空间，需要拷贝数据，很耗时。
链表则支持动态扩容。

4.相对来说链表比较耗内存，因为需要记录节点指针，内存消耗翻倍。

2018-10-03

作者回复

👍

2018-10-03

A漩 1

最近最少使用策略 LRU（Least Recently Used）
老师此处的例子是着重体现了”最近“的场景吧！

如果想体现”最少“是不是还得为LinkedList加个”访问次数“的属性?

2018-10-03

作者回复

字面上是你的理解 不过请百度百科一下LRU 我讲的没错呢

2018-10-03

lovetechlovelife 1

1，基于内存计算的Spark相比基于磁盘计算的Hadoop应该就是空间换时间吧
2，数组实现LRU
分析了一下，数组实现LRU会造成大量数据迁移带来的耗时，而且缓存不满时，数组满了，这时数组要扩容，扩容还得考虑扩容是否能请求到足够的缓存问题，会变得很复杂.....
3，思考题
(1)遍历单链表并取出数据放入数组中
(2)正向遍历数组取出每个字符并生产一个字符串s1
(3)逆向遍历数组取出每个字符并生产一个字符串s2
(4)判断s1的值跟s2的值是否相同，相同就是回文串，否则不是
时间复杂度：n+n+n=3n推断时间复杂度为O(n)
空间复杂度：多申请了一个数组的大小

2018-10-03

Niulx 1

```
public static boolean isPalindrome(Node root) {
if (root == null) {
return false;
}

Node head = root;
Node reserve = null;

while (head != null) {
Node next = head.next;
head.next = reserve;
reserve = head;
head = next;
}

while (root != null && reserve != null) {
if (root.c == reserve.c) {
root = root.next;
reserve = reserve.next;
} else {
return false;
}
```

```
}
return true;
}
```

我是这样想的，先反转单链表，在while循环判断两个链表的字符是否相等

2018-10-03

Astrian 🦊 1

简单写了一下 JS 下的链表实现，https://gist.github.com/Astrian/0fab823f41e288cc2fc324ba3991d1b8，希望可以有帮助+求大佬指点 (=ﾟωﾟ)ﾉ

2018-10-03

熊先生ロ 1

打卡学习！

2018-10-03

CaiBird 1

越靠近链表尾部的结点是越早之前访问的

这里不懂 😳老师可以解答一下吗，谢谢

2018-10-03

作者回复

链表是有序链表，按照访问时间排序，访问时间值最小的放到链表尾部，最大的放到链表头。

当要淘汰一个数据时，直接删除链表尾部的结点，当要加入一个数据时，直接插入链表头部。

当然，你要想按照相反的方式组织，也是可以的啊，只要是满足LRU就可以了。

2018-10-04

Wy 0

这一章内容配上leetcode里LRU那道设计题，以及题解就更直白啦

2018-10-06

勤劳的小胖子-libo 0

乍一想来，直接双向链表，二个指针，一个指头向后，一个指尾向前，对指向的值一一比较，到都指向null，表示相同。时间复杂度，O(n)遍历，空间复杂度是O(1)，保存二个指针。

但应该有优化的方法，结束的条件应该可以优化。

2018-10-05

张飞online 0

其实链表还好，一个链表加多线程操作，再加性能优化，也就是你加锁的粒度不能太大，要到链表里面加

2018-10-05

明翼 0

几个问题请教下：1.单项链表用一个变量保存前一个节点，删除时候应该和双向链表时间复杂度一样啊。
2."当访问的数据没有存储在缓存的链表中时，直接将数据插入链表表头，时间复杂度为O(1)；"这个是有问题的，如何知道访问数据不在链表中，这本身也需要一次遍历。

至于回文问题，用一个双向链表表示，从头到尾遍历再返回来遍历得到的字符串比较即可

2018-10-05

智慧树叶 0

课后思考:可以用双链表，先将字符串单个写入双链表，然后取中间节点向两头发散比较是否相等。核心算法的时间复杂度的话是O(n)。

2018-10-05

wean 0

数组简单易用，在实现上是使用连续的内存空间，可以借助 CPU 的缓存机制，预读数组中的数据，所以访问效率更高。而链表在内存中不是连续存储，对 CPU 缓存不友好，没办法有效预读。

但数组的特点也是它的不足，他的内存空间是固定的，如果声明的数组过大，系统可能没有足够的连续内存空间分配给它，导致内存不足(out of memory)，例如如果现在系统由 100M 不连续的内存空间，声明 100M 数组就会失败，另外到数组扩容时，复制原数组的内容到新数组也很费时。这就是数组和链表最大的区别。

另外如果我们的代码对内存使用非常苛刻，那应该使用数组，比如安卓之类的，因为链表需要维护额外的空间去存储指针。而且对链表进行频繁的插入、删除操作，还对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，就可能导致频繁的 gc。

2018-10-05

liu 0

##数据结构与算法

###第五课，链表上（Linked List）

开篇题目，如何实现LRU的缓存淘汰算法
1、缓存算法：一种高性能的数据读取技术，如CPU缓存，数据库缓存，浏览器缓存
2、三种缓存淘汰策略：先进先出（FIFO），最少使用策略（LFU），最近最少使用车策略（LRU）

链表和数组的底层存储结构
1、数组需要一组连续的内存空间存储，如果连续的内存不够，则申请失败，对内存要求较高
2、链表则与数组相反，它不需要连续的内存空间，它是将一组零散的内存块组合起来使用，只要内存足够，都可以申请
支持动态扩容。

链表结构

单链表
1、内存块，即链表节点，用来存储数据和下一个节点指针next
2、头结点和尾节点（NULL）比较特殊
3、支持插入（O(1)）、删除（O(1)）、查找（O（n））等操作

循环链表
1、循环链表是一种特殊的单链表，首尾相连的
2、优点：从尾到头遍历方便，在处理的数据有环形结构特点的时候适用，如约瑟夫问题

双向链表
1、不仅包含后继指针next，还有前驱指针prev
2、支持双向遍历，灵活性高
3、删除操作
1、删除节点中“某个值等于指定值“的节点，算法时间复杂度为O(n)
2、删除某个指针指向的节点，可以根据前驱节点直接获取前一个指定，无需从头遍历，性能要比单链表高，
算法时间复杂度为O(1)
4、查询操作，对于无序的数据来说，效率跟单链表一样为O(n)，如果数据为有序的，平均只需要查找一半的数据

双向循环链表
1、双向链表差不多，区别在于首尾节点相连

设计思想
1、用空间换取时间的设计思想，如果内存足够，但追求代码的执行速度的时候，可选择空间复杂度高、时间复杂度低的
的算法或数据结构。
2、用时间换取空间的设计思想，如果内存吃紧，则选择空间复杂度低，时间复杂度高的算法或数据结构
3、缓存技术就是用了空间换取时间的设计思想

链表数组性能对比
1、数组，插入删除（O（n）），随机访问（O（1））
2、链表，插入删除（O（1）），随机访问（O（n））

开篇解答，基于链表实现LRU缓存算法
1、维护一个单链表，越靠近尾节点则表示越早之前访问，如果有数据来，则先遍历后插入
2、判断此数据是否已经在链表中
1、如果已经在缓存链表中了，先把数据取出来然后放到链表头
2、如果不在在缓存链表中，则判断缓存链表是否已满

1、如果已满，则把最后一个剔除，然后把新数据放到头结点
2、如果未满，则直接把新数据放到头结点
3、此缓存算法时间复杂度O(n)
4、优化思路，使用散列表记录每个数据的位置，降低缓存访问时间

开篇解答，基于数组的LRU缓存算法的实现
1、维护一个数组，越靠近尾部则表示越早之前访问，如果有数据来，则先遍历后插入
2、判断此数据是否已经在数组中
1、如果已经在缓存链表中，先把数据取出来，然后把之前的数据往后搬移一位，最后把刚才取出的数据放到头
2、如果不在在缓存链表中，则判断缓存链表是否已满
1、如果已满，则把最后一个剔除，然后把之前的数据往后搬移一位，最后把数据放到头
2、如果未满，先把所有数据往后搬移一位，然后把新数据放到头结点
3、此缓存算法时间复杂度O(n)（查找O(n)，数据搬移O(n)），因为涉及数据搬移，所以该算法比基于链表的效率低

课后思考，基于单链表的回文字符串判断
1、声明两个指针a，b，指向链表头结点
2、a，b指针向前移动，a每次移动一格，b每次移动两个，直到b为null或者b.next为null，此时a的位置恰好在链表中间
3、使用栈c存储每次a指针指向的节点
4、循环结束
5、接着b指针继续向前移动，栈c弹栈并且跟b节点值比较
6、如果不相等则循环结束，说明不是回文字符串返回false
7、否则继续5、6步骤，直到b.next为null
8、是回文字符串，返回true

2018-10-05

章光辉 0

判断访问的数据是否存在于缓存里的时候，无论是链表还是数组，时间复杂度都是O(n)对吧？那么在此基础上做的增删改，最高的复杂度也只是O(n)。根据加法法则，总的复杂度都是O(n)。不知道我理解得对不对？

2018-10-05

作者回复

对的

2018-10-05

学渣！！！ 0

思考题采用快慢指针：

```
public static boolean isPalindrome(Node head){
Node pre = null;
Node slow = head;
Node fast = head;
```

```
while(fast != null && fast.next != null){
fast = fast.next.next;
Node next = slow.next;
slow.next = pre; //把slow的下一个节点指向pre,使前半段节点反序
pre = slow;
slow = next;
}

//当是奇数个元素时，slow要跳过最中间过素
if(fast != null){
slow = slow.next;
}

while(slow != null){
if(slow.data != pre.data){
return false;
}
slow = slow.next;
pre = pre.next;
}
return true;
}

public static void main(String[] args) {
Node node1_1 = new Node("n");
Node node1_2 = new Node("o");
Node node1_3 = new Node("o");
Node node1_4 = new Node("n");
node1_1.next = node1_2;
node1_2.next = node1_3;
node1_3.next = node1_4;
node1_4.next = null;
System.out.println(isPalindrome(node1_1));


Node node2_1 = new Node("l");
Node node2_2 = new Node("e");
Node node2_3 = new Node("v");
Node node2_4 = new Node("e");
Node node2_5 = new Node("l");
node2_1.next = node2_2;
node2_2.next = node2_3;
node2_3.next = node2_4;
```

```
node2_4.next = node2_5;
node2_5.next = null;
System.out.println(isPalindrome(node2_1));
}

static class Node {
private String data;
private Node next;

public Node(String data) {
this.data = data;
}
}
```

因为没有额外增加内存所以空间复杂度为O(1),时间复杂度为O(n)
老师，想问下你文章里的图用什么画图工具完成的

2018-10-05

我瑟瑟的方法 0

什么是回文串

2018-10-05

作者回复

麻烦自己百度一下吧 亲 百度一下好几十页都是讲回文串的

2018-10-05

上发条 0

将单链表反转，时间O（n），在对两条链表遍历比较，时间O（n）

2018-10-05

刘十一 0

思考题：利用链表判断一个字符串是否是回文，基本思想就是首尾进行对比，时间复杂度是O(n)，老师可不可以往后将思考题的答案或者说您的思路，在下一篇文章中提供出来呢?

2018-10-05

code047 0

打卡

2018-10-05

SunshInW 0

数组实现LRU：可以采用“空间换时间”策略，申请足够大内存的数组，然后定义一个变量count记录位置，count位置的数字代表最近最久未使用的数据，定义一个标志Nan，表示数据为Nan的这个位置被置换过。从数组起始位置开始存放数据。
1）当缓存区没有满时，直接放入数组，需要置换的数据被置为Nan，被置换的数据加入数组。

2）当缓存区满的时候，利用count记录最近最久未使用的数据，如果该位置数据被置换，则count后移一位。如果不是该位置被置换，则被置换的数据被置为Nan，被置换的数据放加入数组。
思考：
https://wizardforcel.gitbooks.io/the-art-of-programming-by-july/content/01.04.html
1、首先利用快慢指针找到单链表中间结点点，同时将链表的前半部分逆序
2、利用中间结点指针和头结点指针判断链表前半部分和后半部分是否相等。

2018-10-05

妮可 0

LRU这个缓存淘汰算法能不能用优先队列呢?

2018-10-04

作者回复

这个貌似不行，你可以说说你的思路：）

2018-10-05

blacksmith 0

用数组实现LRU缓存淘汰算法思路：
维护一个数组，在数组中越靠前位置的数据是越早之前访问的。当有一个新的数据被访问时，我们从数组的0位置开始循环遍历数组。
1.如果此数据已经在缓存的数组中，我们遍历得到这个数据对应的位置，将其从原位置删除，后续位置的数组往前移动一个内存单元，然后将被访问的数据存储在数组的末尾。
2.如果此数据没有在缓存数组中，将其添加到数组的末尾，如果缓存已满，则将此数据存储在数组的开始位置（0位置）。

2018-10-04

dead_lock 0

不理解链表删除为什么是O(1)，因为单向链表删除需要找到前驱节点复杂度为O(n)，谢谢老师解答

2018-10-04

作者回复

删除是O(1)是有前提的，就是已经知道前驱结点的情况下，删除是O(1)。

2018-10-05

Flying 0

『在实际的软件开发中，从链表中删除一个数据无外乎这两种情况：
删除结点中“值等于某个给定值”的结点；
删除给定指针指向的结点。』
问题1（删除结点中值等于某个给定值的结点）是什么意思，是用空值替代吗，还是其他操作。
问题2:（删除给定指针指向的结点），比如删除中间的一个结点，是不是应该这样操作:把前一个的next指向后一个的previous，这样就相当于删除了中间的结点了。
希望老师解答一下。

2018-10-04

Snail 0

如果链表中3个连续的元素A,B,C，删除元素B，只需要将A的next设置为C即可，对吗？B会自动被垃圾回收吗？

2018-10-04

作者回复

java语言可以的，但是其他语言，比如C语言，要程序员自己free这块内存空间

2018-10-04

仲毅 0

雖然要花點時間了解，但是對準備考研也是很有用處的!

2018-10-04

作者回复

嗯嗯

2018-10-04

yaxin 0

数组，查找方便，插入删除不方便。
链表，插入删除方便，查找不方便。
链表是以空间换时间，因为链表指针也要存储。如果对空间要求比较高的程序，优先考虑数组。

2018-10-04

作者回复

是的

2018-10-04

A..... 0

老师,快点更啊,等不及了

2018-10-04

作者回复

每周三篇 已经有同学跟不上了！你这种尖子生就迁就一下吧：）

2018-10-04

煜 0

单链表删除指针指向的节点的时候，为什么不用p->next->next来判断是不是为要删除的节点呢？这样就不用找到相应的前驱节，时间复杂度和双链表一样的

2018-10-04

作者回复

我这里的前提是指针已经指向要删除的结点了。所以没法通过p->next来判定是否是要删除的结点了。你可能会问，指针是怎么指向要删除结点的呢？这个等到我们讲到散列表的时候，会有相应的一个例子。

2018-10-04

煜 0

单链表删除指针指向的节点的时候，为什么不用p->next->next来判断是不是为要删除的节点呢？这样就不用找到相应的前驱节，时间复杂度和双链表一样的

2018-10-04

作者回复

我这里的前提是指针已经指向要删除的结点了。所以没法通过p->next来判定是否是要删除的结点了。你可能会问，指针是怎么指向要删除结点的呢？这个等到我们讲到散列表的时候，会有相应的一个例子。

2018-10-04

起点·终站 0

回文的那个，for循环倒序插入一个新数组在判断相等，所以是O(n).....有现成的方法reverse()

2018-10-04

假装在火星 0

单链表判断是否为回文字符串，最直接的思路就是从字符串两端开始判断，首先是第一个字符与最后一个字符，依次判断所有字符串，如果出现一次不相等，那么就能得出结论。
Python代码实现如下：
(篇幅限制，没有注释，另外代码只给出了用到的方法，完整代码见https://github.com/linsanityHuang/python_data_and_algorithms/blob/master/Reverse_String.py)
单向链表实现：

```
class Node(object):
def __init__(self, value=None, next=None):
self.value, self.next = value, next
class LinkedList(object):
def __init__(self, maxsize=None):
self.maxsize = maxsize
self.root = Node()
self.tailnode = None
self.length = 0

def __len__(self):
return self.length

def append(self, value):
if self.maxsize is not None and len(self) >= self.maxsize:
raise Exception("LinkedList is Full")
```

```
node = Node(value)
tailnode = self.tailnode
# 还没有append过，length=0，追加到root后
if tailnode is None:
self.root.next = node
# 否则追加到最后一个节点的后边，并且更新最后一个节点是刚append的节点
else:
tailnode.next = node
self.tailnode = node
self.length += 1

# 删除尾节点
def pop(self):
if self.root.next is None:
raise Exception('pop from empty LinkedList')
prevnode = self.root
for curnode in self.iter_node():
if curnode is self.tailnode and curnode is not self.root:
value = curnode.value
# 删除当前尾节点
del curnode
prevnode.next = None
self.tailnode = prevnode
self.length -= 1
return value
else:
prevnode = curnode

def popleft(self):
if self.root.next is None:
raise Exception('pop from empty LinkedList')
headnode = self.root.next
self.root.next = headnode.next
self.length -= 1
value = headnode.value

# 如果删除的是尾节点，需要把尾节点置为None
if self.tailnode is headnode:
self.tailnode = None
del headnode
return value
```

判断是否会问字符串：

```
def is_reversed_str(word):
if not isinstance(word, str):
return
str_linkedlist = LinkedList(maxsize=len(word))
# O(n)
for item in word:
str_linkedlist.append(item)

size = len(str_linkedlist)
flag = True
#O(n^2)
for _ in range(size // 2):
if str_linkedlist.pop() != str_linkedlist.popleft():
flag = False
break
return flag
```

复杂度分析：
O(n^2)

2018-10-04

dead_lock 0

针对链表的插入和删除操作，我们只需要考虑相邻结点的指针改变，所以对应的时间复杂度是O(1)。

老师您好，文中单向链表删除和插入操作时间复杂度为O(1)，后面讲解中有提到 单向链表在删除指定指针节点或者删除值相等节点的时候，需要找到前驱节点，复杂度为O(n)，
我理解的是这两个时间复杂度都是在说明单向链表的删除，为什么一个是O(1)一个是O(n)

2018-10-04

作者回复

同学，你觉得应该是O(1)还是O(n)呢？或者你觉得其中哪一个不理解呢？你可以自己试着分析一下时间复杂度的。如果还不清楚，再给我留言吧。

2018-10-04

途 0

链表+hash实现lru不就是jdk的LinkedHashmap么

2018-10-04

作者回复

不是 我后面讲到散列那节课会讲

2018-10-04

Vincen 0

老师，我用go语言实现了一个单链表，为什么除了增加一个数据的时间复杂度为O(1)，删改查的时间复杂度都为O(n)?

2018-10-04

作者回复

你说的稍微有点笼统，要具体看代码怎么写的。比如增加一个数据，是在p指针之后增加、还是之前增加、还是直接增加到链表的头部，这个时间复杂度都是不一样的。删除操作也是类似的，是删除某个指针指向的结点，还是删除值等于某个值的结点等等，要具体看的。

2018-10-04

猫头鹰爱拿铁 0

判断字符串是否为回文字符串？遍历单链表同时放到一个栈里面，然后再遍历一遍单链表，遍历的同时和栈pop出的数据对比，全都一致则说明为回文字符串。

2018-10-04

作者回复

方法很巧妙，不过借助了栈，空间复杂度就是O(n)了。可以想想有没有空间复杂度是O(1)的算法。

2018-10-04

阿伟 0

https://github.com/sunpengwei1992/go_common.git
跟随作者讲解，所有练习会在里面用代码实现，有兴趣的可以探讨，互相学习。已经有一些实现了，大家可以看看，有不对的地方，望指正，邮箱：sunpengwei1992@aliyun.com

2018-10-04

作者回复

👍

2018-10-04

JStFs 0

LRU：活在当下。比如在公司中，一个新员工做出新业绩，马上会得到重用。

LFU：以史为镜。还是比如在公司中，新员工必须做出比那些功勋卓著的老员工更多更好的业绩才可以受到老板重视，这样的方式比较尊重“前辈”。

2018-10-04

作者回复

哈哈 形象！

2018-10-04

James Scott 0

王老师：您好！起初我是这样考虑的，访问单链表从链表头部开始访问的，那么越靠近链表尾部的节点应该被访问的时间越晚，刚刚看了您回复给CaiBird的留言就明白了。谢谢王老师的指点

2018-10-04

作者回复

嗯嗯 理解就好 🤝

2018-10-04

缓哟 0

一个数组，头部是最久没有访问的。当访问一个新数据的时候，遍历数组，如果存在，且内存空间没满，就删除，重新插入到后面。如果不存在，且内存空间没满，直接插入，如果满了，删除数组第一个数据，1～ n-1位置的数据前移，再插入到n-1的位置。
数组实现LRU，不知道对不对;-)

回文
如果是用循环单链表存储，那么维护一个尾指针，当尾结点和头结点相等的时候，删除这两个结点，一直这样子，当链表为空或者只有一个结点的时候，就是回文。
这样子行不行嘿;-)

评论里有讲双链表和栈，看题目说是用单链表□□□

2018-10-04

作者回复

1. 可行

2. 不可行

2018-10-04

天宇星旋 0

删除给定指针指向的结点。对于这个问题，老师你文中说错了，无论是单向链表还是双向链表，其删除的时间复杂度都是O(1)。单向链表只需要将待删除结点的指针只想next→next，并将下一个结点的值赋值给待删除结点，这样就可以达到删除指定结点。替换的思想。

2018-10-03

作者回复

我没说错，我说的前提是有一个指针指向了p结点，我现在要删除p结点，如何做？你可以看我github上的代码！同学 再认真看遍我写的文章吧

2018-10-04

ydyhm 0

“数组简单易用，在实现上使用的是连续的内存空间，可以借助 CPU 的缓存机制，预读数组中的数据，所以访问效率更高。而链表在内存中并不是连续存储，所以对 CPU 缓存不友好，没办法有效预读。”这里的CPU缓存机制指的是什么？为什么就数组更好了？谢谢

2018-10-03

作者回复

自己百度一下吧 很基础的内容 网上一搜一大堆

2018-10-03

飞羽 0

我的思路是与栈结合，循环遍历一次链表，并将数据压栈，由于栈先进后出的特性，相当于反转了链表的数据。
之后再遍历一次，依次出栈并与链表节点上的数据进行比较，如果完全相同，就说明是回文。

实现语言是JavaScript（直接用js的数组模拟栈的行为）
时间复杂度是O(n)
空间复杂度是O(n)

```
class Link {
constructor(data, next){
this.data = data;
this.next = next;
}
}

class Stack {
constructor(){
this.datas = []
}
add(data){
this.datas.push(data);
}
get(){
return this.datas.pop();
}
}

const a = new Link('a', null);
let next = a.next = new Link('b', null);
next = next.next = new Link('c', null);
next = next.next = new Link('b', null);
next = next.next = new Link('a', null);

function isRev(link){
if(!link.data || !link.next) return false;
const cache = new Stack();
let _link = link
let i = 0;
while (_link) {
cache.add(_link.data);
_link = _link.next;
}
while (link) {
```

```
const data = cache.get()
if(link.data !== data){
return false;
}
link = link.next;
}
return true;
}

console.log(isRev(a));
```

2018-10-03

小老鼠 0

1,可否总结下什么时候用数组？什么时候用链表？
2,对于嵌入式系统，空间比较缺乏，是不是应该用数组？但数组又要连续的空间，应该如何处理？

2018-10-03

作者回复

1. 你可以根据数组和链表的特点，结合业务需求来看。比如，你存储的数据需要支持排序，那用数组比较合适了。要支持快速查找，那也是数组比较合适了。如果要做频繁的删除、插入，就像举的LRU的例子，那就比较适合链表。
2. 这个要具体看了。如果数组申请的空间不大，内存可以满足，那就用数组吧。

2018-10-04

DADDYHINS 0

我的思路比较弱智，把原本的单链表逆置，写两个指针，遍历两个链表并比较它们指向的元素是否相同，这样做的时间复杂度是O(n^2)，但是空间需求大🤔。坐等更好的解答

2018-10-03

PoetAndPoem 0

回文问题关键在于两点:1.判断链表的中间位置2.从中间位置向两边开始依次比较是否相等。

2018-10-03

作者回复

注意是单链表

2018-10-03

韩 0

回文字符串用双向链表。
从头结点和尾结点逐个删除结点。头尾相同则删除。
最后判断剩余链表长度是否≤1，是则是回文字符串。
时间复杂度O(n),空间复杂度O(n)

2018-10-03

蝴蝶 0

关于数组做LRU算法,我有一个想法,就是申明一个指定长度(n)的数组,将缓存的对象按照时间顺序一一保存,同时也保存一个缓存对象操作时的索引index.当去数组中查找缓存的对象时,按照从头至尾的方向遍历,时间复杂度O(n),当有找到指定的对象时,那就直接取出.当没有找到对象时,按照index的位置,向后算一位(如果后一位不存在,那就从头开始),并将对象缓存到那个位置,然后index向后移动1位.这样的情况,就保证了index处总是新的那个.

2018-10-03

就酱 0

一个链表节点指针每次都从头节点开始遍历链表，到达尾节点时和头节点比较值，不同则返回结果；相同则把头尾节点都删除；循环直到游走的指针节点和头节点相同。
时间复杂度分析：循环次数 n+(n−2)+...+(n−2k+2) 其中 k 为循环次数，最坏情况 k = n/2，最好情况 1，所以平均时间复杂度 O(n)

2018-10-03

辰陌 0

日常打卡，总结到笔记本了，，就不打出来了，，希望各位大神踊跃发言，可以更好地给我等菜鸟解惑创造有利环境。

2018-10-03

旅途 0

有一些问题
1.如果创建不了100个对象的数组，100个对象的链表也应该创建不了吧?
2.关于单向链表删除的问题，删除时使用遍历找到被删除的元素，被删除的前一个就是前置节点啊，不用需要再遍历一遍了吧

2018-10-03

作者回复

1. 为什么呢？能说一下你的理由吗?

2. 嗯，对的，文章里也并没有要遍历两遍呢

2018-10-03

Rice 0

为什么单链表插入删除的时间复杂度是O(1)呢？如果插入一个数在下标是K的位置，不是要先遍历获取到a(k−1)吗?

2018-10-03

作者回复

同学，能认真看看我的文章吗？我不是讲了的嘛！后面还特意强调的！

2018-10-03

Mr.钧 0

链表，是一种线性的，一堆零散内存组成的数据结构。它删除和插入的时间复杂度是O(1)，查询的时间复杂度是O(n)。和数组正好相反。注意时间换空间，空间换时间的设计思路。

2018-10-03

牵手约定 0

一直跟着老师的步伐走。

2018-10-03

张初炼 0

暂时想到了两个办法。

1. 创建一个新的单向链表。遍历原有的链表，遍历过程中把每一个节点存储的字符拷贝一份并插入到新链表的头部。得到的新链表相当于是把原有链表“倒置”了。最后同时遍历两个链表并比较每个节点，如果都相同则是回文字符串，否则不是。时间复杂度是 O(n)，空间复杂度也是 O(n)。

2. 遍历原有链表从而计算节点总个数 list_len。(list_len / 2) 便是这个链表的“中间位置”。再次遍历链表，进行到“中间位置”后，依次把后面所有节点“转向”，即指向 prev 而非 next。结束后从链表的头和尾同时开始再做一个遍历(只进行到“中间位置”)，比较每次访问到节点字符，如果一直相等则是回文字符串，否则不是。时间复杂度是 O(n)，空间复杂度是 O(1)。

2018-10-03

hope 0

看完了，打卡，希望老师上一些代码，个人觉得只有实践才会理解的更牢固，再就是评论区都是大神，可以多看

2018-10-03

甘远林 0

刚查了一下什么是回文，哈哈。

2018-10-03

H~Z~G 0

回文这个是不是可以判断字符的右边界位置，通过递归每次二分判断呢?

2018-10-03

往事随风，顺其自然 0

LRU-k算法时间复杂度和空间复杂度怎样?

2018-10-03

liangjf 0

习题解答方法

1.快慢指针。通过得到中间结点，同时从开始，尾部向中间遍历并对比，不满足则退出直到全部匹配

2.借助 栈。对比入栈前，出栈后的字符串，相同则满足条件。

3.字符串插入到数组。利用求出数组长度，利用数组下标访问特性，循环判断arr[i] == arr[L-i] ，直到不满足退出。

数组和链表各有各的优点缺点，不能一概而论，同时抛开需求业务谈性能都是耍流氓，比如在嵌入式开发时，一些芯片的栈内存资源是很紧张的，如果申请较大的局部数组会造成栈溢出，这时可以用链表替换。

2018-10-03

luxinfeng 0

单链表存储，设置两组指针A和B，从链表头部开始遍历，指针A每次前进一位，指针B每次前进两位；指针A每到一个位置，就将该位置的字符压入栈中，直到指针B到达链表尾部，此时指针A到达字符串的中间位置。然后，每当指针A前进一步，就将栈中的字符弹出一位，比较指针A所指字符与弹出字符是否相等，如果相等，则继续运行；不等，则退出程序，说明该字符串不是回文序列。
时间渐进复杂度O（n），空间渐进复杂度O（n）。

2018-10-03

刘远通 0

遍历把链表存到数组里面 然后用数组随机访问的特点 判断是否是回文?

2018-10-03

一步 0

请教一个问题啊，就是当数组的内存不够用的时候，去申请新的内存，为什么不先判断是否有靠近已申请内存的连续内存呢？有的话，是不是就可以利用上数组原来申请的内存了呢？在原内存上进行扩容呢?

2018-10-03

张岳文 0

用链表实现栈，很容易匹配这种成对出现的东西。
c++中的栈变量感觉就是这样的。比如int a; int b; 释放的时候就反过来release b; release a; 这不就是回文吗。还有函数调用栈。。还有xml各种成对匹配的文本格式。。

m缓存是什么。?

2018-10-03

涛 0

如果解答需要经过别的数据结构去转换单链表，我不知道这个习题的意义是什么。老师给出的存储就是单链表，难道不是思考单链表的操作去判断回文吗?

2018-10-03

雨山 0

果然有程序员风格，放假也更新，感觉物有所值。

2018-10-03

钢 0

取中间节点，分割为两个链表，并将其中一个链表反序，然后匹配是否相等

2018-10-03

大可可

0

数据库的宽表是不是空间换时间

2018-10-03