判断链表是否相交的几种算法**

博客分类：

- 算法与数据结构

这个是《编程之美》里面的一个题目，给出两个单项链表的头指针，h1、h2判断这2个链表是否相交？

【解法一】直观的想法

循环遍历h1的每一个节点，判断是否存在一个节点在h2中，由于链表无法随机访问，每次查找需要对链表h2遍历最多length(h1)次，因此算法的时间复杂度是O(length(h1)*length(h2))，显然这个方法很耗时间。

【解法二】利用hash计数

①循环遍历h1，计算每个节点的hash值并存入map中；②循环遍历h2，顺序计算每个节点的hash值v，并用map.get(v)，若返回非空，则算法结束。

第①步算法时间复杂度O(length(h1))，第②步算法时间复杂度O(length(h2))，因此hash计数 算法时间复杂度为O(max(length(h1),length(h2)))，复杂度降低到线性。但是由于使用了额外的map结构，空间复杂度为O(length(h1))。

【解法三】链表连接

我们分析两个单链表相交时，节点的逻辑结构如下：

h1->P1->P2->...->Pi->...->K1->...->Km

h2->Q1->Q2->...->Qi->...->K1->...->Km

可以看到如果把h1链表的尾节点的next指针指向h2链表的第一个节点，那么可以看到如果h1和h2相交，则

h2变成了一个循环单链表，因此只需判断h2是否为循环链表即可。需要遍历h1和h2，因此算法时间负责度O(max(length(h1),length(h2)))，空间负责度为O(1)。但是这个算法的缺点是需要链接h1和h2，改变链表的逻辑结构，在多线程环境下需要上锁，影响一定的性能。

【解法四】更简单的做法，直接判断链表末节点是否相同

同解法三的节点分析，如果h1和h2相交于节点K1，那么根据单链表的定义(任何节点有且仅有唯一的后继和唯一的前驱，null也算作前驱和后继吧)，因此如果h1和h2相交，那么两个链表从K1以后的后继的节点是相同的。因此判断链表是否相交，只用判断尾节点是否相同即可，只需遍历2个链表，时间复杂度为O(max(length(h1),length(h2)))，空间复杂度为O(1)。这个算法很简单优雅，赞一个。

发散思维：

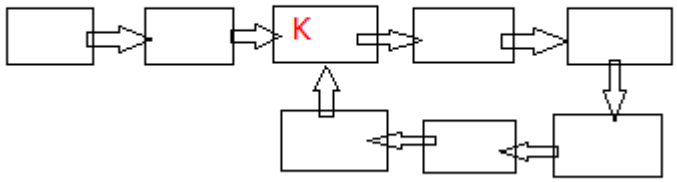
1、如何求解两个相交链表的第一个相交节点？

这个其实也很简单，使用小学数学知识即可解决。

记 L1 = length(h1); L2 = length(h2); h1和h2 由于相交，共有K个节点相同，那么两个链表的长度之差为相交节点之前的节点之差。因此我们让长链表先遍历abs(L1-L2)步，然后让两个链表同时遍历，同时判断2个链表的节点是否相同，若存在相同节点，则返回该节点，算法结束。

2、如果两个链表中本身存在环，该如何解决？

存在环的单链表，类似于这种形式。



如果链表存在环，则上面的算法都无法返回正确结果，陷入死循环，因此首先要把环解开。下一篇日志将介绍几种检测链表是否存在环的算法。