**1.给定一个单向链表（长度未知），请设计一个既节省时间又节省空间的算法来找出该链表中的倒数第m个元素。实现这个算法，并为可能出现的特例情况安排好处理措施。“倒数第m个元素”是这样规定的：当m=0时，链表的最后一个元素将被返回。**

解决问题方法思路如下：

方法一、如果我们知道链表的长度n，查找倒数第m个元素，也就是查找正序的第（n - m）个元素（这里的序号只是为了分析，可能跟题目不一定正确的符合）。那么这样来说就简单很多。首先遍历链表得到链表长度，然后重新遍历一次，查找正数第（n-m）个元素。时间复杂度大约是O(2n)。

方法二、我们是不是可以提供一个辅助存储空间，是的我们在遍历到链表结束的时候可以回溯到倒数第m个元素。比如用一个支持随机访问的容器记录链表每一个节点的地址。那么这样的就可以只遍历一次链表就能得到结果。时间复杂度大约是O(n)，但是我们是用空间换取时间的，辅助存储空间的大小由m决定，如果m过大也是不可取的。

方法三、头结点指针为当前指针，尾节点指针为拖后指针。开始的时候当前指针和拖后指针初始化为链表的头结点，首先我们让当前指针遍历到第m个元素，拖后指针不变；然后同步更新当前指针和拖后指针；直到当前指针为链表结尾。这样我们就能保证当前指针和拖尾指针之间的距离是m。

代码如下：

Node\* FindMToLastNode(Node\* pHead, int m) {

// 查找到第m个元素

Node\* pCurrent = pHead;

for (int i = 0; i < m; ++i)

{

if (pCurrent)

{

pCurrent = pCurrent->next;

}

else

{

return NULL;

}

}

Node\* pFind = pHead;

while (pCurrent) {

pFind = pFind->next;

pCurrent = pCurrent->next;

}

return pFind;

}

**2.** **给定一个单向链表（长度未知），请遍历一次就找到中间的指针，假设该链表存储在只读存储器，不能被修改**

设置两个指针，一个每次移动两个位置，一个每次移动一个位置，当第一个指针到达尾节点时，第二个指针就达到了中间节点的位置

处理链表问题时，”快行指针“是一种很常见的技巧，快行指针指的是同时用两个指针来迭代访问链表，只不过其中一个比另一个超前一些。快指针往往先行几步，或与慢指针相差固定的步数。

node \*create() {

node \*p1, \*p2, \*head;

int cycle = 1, x;

head = (node\*)malloc(sizeof(node));

p1 = head;

while (cycle)

{

cout << "please input an integer: ";

cin >> x;

if (x != 0)

{

p2 = (node\*)malloc(sizeof(node));

p2->data = x;

p1->next = p2;

p1 = p2;

}

else

{

cycle = 0;

}

}

head = head->next;

p1->next = NULL;

return head;

}

void findmid(node\* head) {

node \*p1, \*p2, \*mid;

p1 = head;

p2 = head;

while (p1->next->next != NULL)

{

p1 = p1->next->next;

p2 = p2->next;

mid = p2;

}

}

**3、输入一个链表的头节点，从尾到头反过来打印出每个节点的值**

有三种思路可以参考：

1. 栈：栈天然是先进后出的，在遍历链表时，把值按顺序放入栈中，最后出栈就是逆序了。
2. 既然想到了栈，那么递归天然是一个栈结构，一样的，访问到一个节点，先递归输出它的后面节点，再输出自己，这样链表就反过来了
3. 头插法：使用头插法可以得到一个逆序的链表。

1->2->3，加入一个头节点head，只用于操作中转，不存储数据

先把head->1，然后继续从2->3中取

变为head->2->1，最后变为head->3->2->1，从head.next开始输出就可以了

**4、给定单身链表的头指针和一个节点指针，定义一个函数在O(1)内删除这个节点。**

**解题思路**

* 如果这个节点不是尾节点，那么可以直接把下一个节点的值赋值给这个节点，再把这个节点指向下下个节点，时间复杂度为O(1)

如，1->2->3->4，要删除的是2，那么先把2的下一个节点值3，赋值给2的节点，变为1-3-3-4，然后把第二个节点的位置，指向下下个节点4，这样，就变为了1-3-4，2这个节点这删除了。

* 如果是尾指针，那么找到前一个节点，把前一个节点指向null，时间复杂度为O(N)

如果进行N次操作，这个算法的平均时间复杂度为O(1)

**5、在一排序的链表中，如何删除重复的节点**

1->2->2->3->3->3->4

1->4

**解题思路**

* 头节点一样有可能重复
* 遍历整个链表，如果当前节点与next的下一个节点相同，则都可以删除，然后把当前节点的前一个节点与下下个节点相连，我们必须保证前一个节点是无一个无重复的节点相连，这一想就是递归~~

**6、输入一个单向链表，输出这个链表中倒数第k个节点**

1->2->3->4->5->6

倒数第三个点为4

**解题思路**

这是一个单向链表，如果我们使用遍历，很明显不方便从尾开始遍历的，这时候我们可以想到的方法是双指针法。

* 定义两个指针p1和p2，p2先不动，p1走k-1步，这时p1指向3，p2指向1
* 现在p1和p2同时向后遍历，当p1遍历到6，它的next为null时，p2所指向的值就是我们所要的值

**7、输入两个链表，找出它们的第一个公共节点**

1-2-3-6-7

4-5-6-7

**解题思路**

* 暴力破解法

在第一个链表上顺序遍历到一个节点，就到第二个链表上遍历每一个节点，如果有一个节点与这个值相等，就找到了。但是时间复杂度为O(mn)

* 栈

我们发现，公共部分一定是在尾部，如果我们从后面开始找会更快，最后一个相同点就是我们要找的点，不过这是单向链表，所以我们想到了栈的结构——后进先出

* 求差法

其实我们用栈，只是为了想同时能到达两个链表的尾节点。当两链表长度不同时，我们从头开始遍历，到后面的时间就不一致。**我们可以先遍历两个链表，得到它们的长度，然后，链表长的先走先走多的几个节点，接着同时在两个链表遍历，找到的第一个相同点就是我们要找的第一个公共点**

**也可以转化:**

**8、链表求和**

Input: (7 -> 2 -> 4 -> 3) + (5 -> 6 -> 4)

Output: 7 -> 8 -> 0 -> 7

复制代码

**解题思路**

我们可以发现，其实就是倒序对应相加，有进位的话要保存到下次相加上，想到了栈这样的数据结构，后进先出

**9、回文链表**

给定一个单链表，判断它是否是回文链表，即正着遍历和反着遍历得到的序列是相同的。

要求，0(1)空间复杂度

**解题思路**

因为要求的空间复杂度为O(1)，所以栈啊什么的数据结构来存储的就不能用了，这时候需要用到分治的思想

我们可以把链切成两半，把后半段反转，再比较这两半是否相等就可以了。

具体可以这么来实现，使用快慢两个指针，快指针每次两步，慢指针每次一步，当快指针的next或者next.next为空时，慢指针就在链表的中间（偶数个时，慢指针是靠近头那一侧的节点）。然后从慢指针的下一个点开始把后面的链表反转，然后分别从头和尾（这个时候尾已经转到了中间）前进，这可以判断是否是一个回文

**10、链表元素按奇偶聚集**

给定一个单链表，请将所有的偶数节点拿出来，放在奇数节点后面，注意，偶数指的是节点的位置编号，而不是值。

你需要使用O(N)的时间和O(N)的空间

**解题思路**

这个还是可以通过双指针来解决，用两个指针一起遍历，每次遍历两个节点，然后再将偶数链表头接在奇数尾上。

11、**如何判断一个链表有环**

**问题描述：**

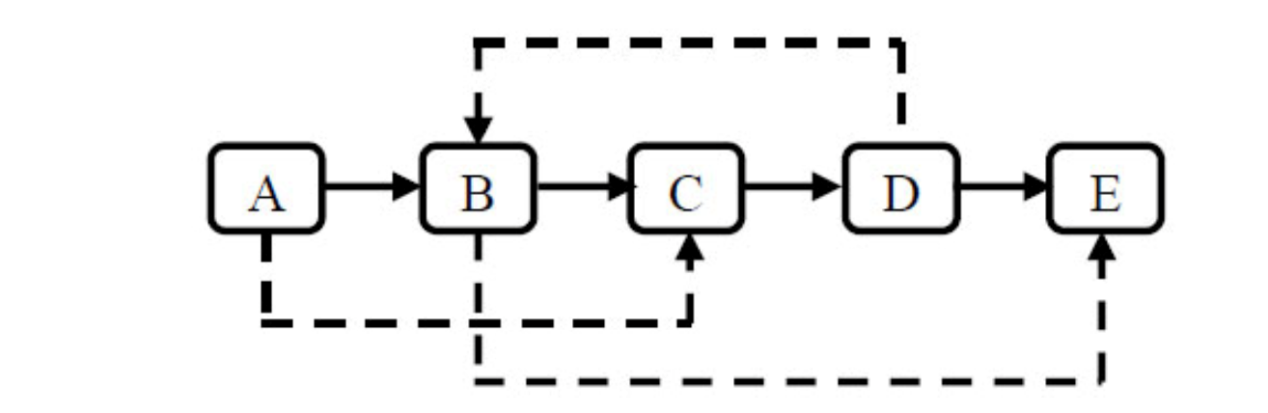
有一个单向链表，链表当中有可能出现“环”，如何用程序判断出这个链表是有环链表？  
不允许修改链表结构。时间复杂度O(n)，空间复杂度O(1)。

首先创建两个指针1和2，同时指向这个链表的头节点。然后开始一个大循环，在循环体中，让指针1每次向下移动一个节点，让指针2每次向下移动两个节点，然后比较两个指针指向的节点是否相同。如果相同，则判断出链表有环，如果不同，则继续下一次循环。

**12、复杂的链表复制**

**问题描述**

题目：请实现函数ComplexListNode Clone(ComplexListNode head)，复制一个复杂链表。在复杂链表中，每个结点除了有一个Next指针指向下一个结点外，还有一个Sibling指向链表中的任意结点或者NULL。

　下图是一个含有5个结点的复杂链表。图中实线箭头表示m\_pNext指针，虚线箭头表示m\_pSibling指针。为简单起见，指向NULL的指针没有画出。  


**算法思路**

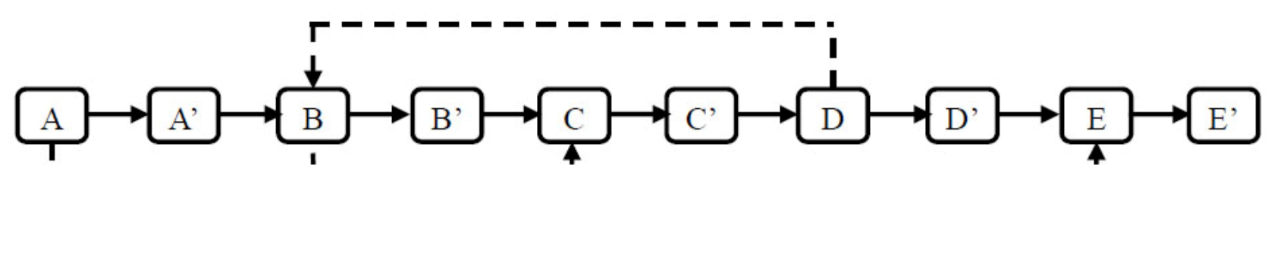
**第一种：O(n2)的普通解法**

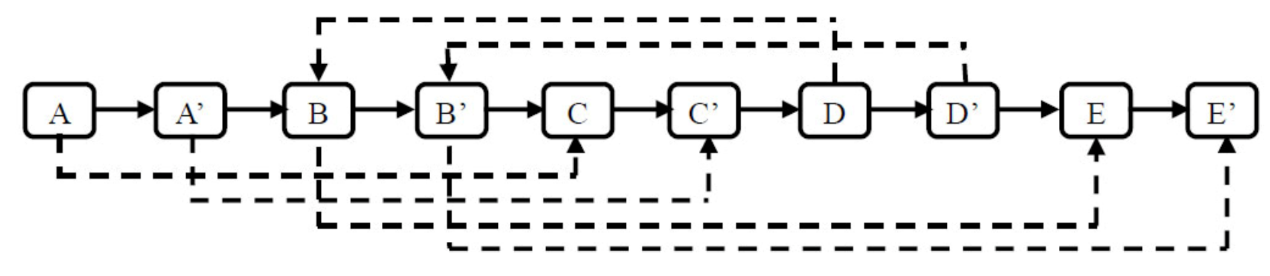
　　第一步是复制原始链表上的每一个结点，并用Next节点链接起来；  
　　第二步是设置每个结点的Sibling节点指针。

**第二种 ：借助辅助空间的O(n)解法**

　　第一步仍然是复制原始链表上的每个结点N创建N'，然后把这些创建出来的结点用Next链接起来。同时我们把<N,N'>的配对信息放到一个哈希表中。  
　　第二步还是设置复制链表上每个结点的m\_pSibling。由于有了哈希表，我们可以用O(1)的时间根据S找到S'。

**第三种：不借助辅助空间的O(n)解法**

　　第一步仍然是根据原始链表的每个结点N创建对应的N'。（把N'链接在N的后面）  
　　

　　第二步设置复制出来的结点的Sibling。（把N'的Sibling指向N的Sibling）  
　

　　第三步把这个长链表拆分成两个链表：把奇数位置的结点用Next链接起来就是原始链表，偶数数值的则是复制链表。

