**一、用数组实现大小固定的栈和队列**  1、实现栈

首先在栈初始化的时候传入大小值，创建该长度的数组。定义index指针，初始化为0，其代表的含义是指向下一个要存放的位置，因此每次加入就是存到arr[index++],取出就是arr[--index],当然每次存入和取出要判断数组满了或者空的情况，index==arr.length和index==0。

2、实现队列

定义队列头head和队列尾tail,然后定义size变量来记录当前长度，这样解耦了head和tail，否则要通过判断他们两个的距离来确定是不是满了或者空了。进队列都是从tail进入arr[tail],tail的自增不能简单加1，要判断一下是不是到了数组末尾了arr.length-1处，是的话直接返回0处（类似于环状）。同理出队列从head出，也要进行相应判断。

**二、实现带有返回最小值的栈，时间复杂度为O(1)** 基本思路1：增加一个辅助栈用来存放当前的最小值，每当有更小的值进栈时，就同时将其压入最小值栈，更新最小值栈顶。当大于和等于最小值栈顶的数进来时候就把自己的栈顶元素重复压一次。出栈的时候就不用做任何判断，两个栈同时都出一个。

基本思路2：也是增加辅助栈，和上面基本相同，只是入栈的时候当出现大于最小值栈顶的元素的时候不作任何操作。而在出栈的时候判断一下，如果要出栈的数<=最小值栈顶，那就跟着出栈，其他不对最小值栈作改变。

**三、用栈实现队列和用队列实现栈** 1、栈实现队列：定义两个栈，一个只管入队，一个只管出队。当出队的时候执行以下 “倒数据的操作”---convert（）函数，convert函数有两个准则：等出队栈为空时才倒、每次倒就必须倒完。这样就一定不会出错，还要注意的是出队的时候要判断是不是为空，为空的条件是两个栈都为空的情况。

2、队实现栈：定义两个队列，其中一个为存数据的队列queue,另一个为辅助队列help，每次入栈的时候直接入queue,出栈的时候先判断queue是不是为空，出栈的时候把queue中的元素全部都出队入到辅助队help中，只保留最后一个，然后出队。出队之后要把help指针和queue指针互换，意思是让queue指针重新指向有值得那个队列。还要注意一点是如果是peek(),则步骤和出栈一样，但是在获取queue中最后一个元素后记得把该元素又入到help中，最后交换指针，这样就能保证queue报持不变。

**四、猫狗队列**

思路：创建两个队列，一个用来放猫一个用来放狗。而因为放进去的猫和狗还要知道先后顺序，因此创建一个新的类来包装宠物类，就可以添加入队标签count，然后把包装后的类入队。在实现猫狗队列的大类中也添加一个count,初始化为0，每当添加一个宠物就自增一，然后给猫狗包装类中，这样每个入队的宠物都有自己的而唯一count，而且跟先后顺序相关，出队是后进行比较就可以知道谁是队列头。

**五、转圈打印矩阵**

思路：首先思考给定了一个矩阵，如果给了左上角的坐标和右下角的坐标，让打印最外面一圈。这样打印就很容易了，整个转圈分为打印上边，右边，下边，左边。四个循环就能做到。这是常规情况，还需要考虑两个特殊情况只有一行和只有一列。因此上述问题的打印就很简单了，用if判断，只有一行只有一列分别单独来写，最后剩下的常规打印一圈。

打印了一圈之后剩下的就是把左上角行列都自加，右下角行列都自减，只要左上角的行列都小于等于下角就一直循环下去。

**六、正方向矩阵向右旋转90度**

思路：和转圈打印一样，也是将问题先换成最外圈。先确定循环次数为eC-sC，然后找每次要交换的四个点，和循环中递增的i变量配合就能实现外圈旋转。剩下的就是把左上角行列都自加，右下角行列都自减，只要左上角的行列都小于下角就一直循环下去。（注意这里小于就行，因为只要正方形矩阵元素个数大于1的就不会出现等于的情况，而元素个数为1的是等于情况但是根本不需要旋转）。

**七、链表反转**

1单链表：需要两个辅助Node，用来记录前一个节点pre和下一个节点next。以当前点head为基础，首先做的是将下一个节点获取到，用next记录，然后调整next指针指向前一个pre。该过程完成，更新到下一个状态，将head给pre，next给head,一直循环下去直到head==null，反转结束，注意此时的链表头不是head而是pre,返回pre.

2双链表;双链表和单链表思路一模一样，只是多了last节点的操作。以当前点head为基础，首先做的是将下一个节点获取到，用next记录，然后调整next指针指向前一个pre,调整last指正给next节点。该过程完成，更新到下一个状态，将head给pre，next给head,一直循环下去直到head==null，反转结束，注意此时的链表头不是head而是pre,返回pre.

**八、之字形打印矩阵：**

思路：利用宏观思路，和转圈打印一样，从全局去考虑，将问题细化。发现“之字形”打印其实就是由每条斜着的打印路线组成，而只要给出斜线的两个端点，那么打印斜线是很简单的。于是定义两个点a,b.这两个点一开始都处于左上角，a点向左走，碰到右边界就转为向下走，b点向下走，碰到下边界就转为向右走，每次两个点都走一步，这样就能保证两点连线的路径是符合要求的，定义布尔型变量用来确定是由a到b打印还是反过来。最后当两个点相会于右下角时候整个矩阵就打印完毕。

**九、行和列都排好序的矩阵找数**

思路：根据数据特征来做，每行都从左到右递增，每列都从上到下递增。那么可以从左下角开始遍历，定义大循环，条件是这个点在矩阵中，然后开始遍历，如果要找到数比当前位置相等就返回找到，如果大了那么肯定不在这一列上，因此列数加一次，如果小于，那么可以排除当前行，行数减一次，直到遍历完还没找到就是没有。

**十、打印两个有序链表的相同部分**

思路：利用外排的思路来做。在一个大的循环中进行，循环条件是两个链表都还没到尾。从两个链表的头指针开始比较大小，如果相等就打印，如果head1大了，那么head2=head2.next表示向后移动，然后再对比。如果head1小了同样向后移动，最终总会有一个先结束，那么循环结束。

同样类型的：两个有序链表合并成一个有序链表（剑指offer）,常规方法和递归的思路。

**十一、判断链表是不是回文结构**

思路1（空间复杂度为O(N)）：直接遍历链表然后存到栈中，然后又一次遍历链表，每次都和栈顶弹出的元素比较，只要有一次不相等就是false,如果遍历完了后都是相等的就是回文结构。

思路2（空间复杂度为O(N/2）：定义快慢指针，快指针一次两步慢指针一次一步，这样快指针走完后慢指针到达中点，接下来慢指针继续遍历，把得到的元素入栈，直到走完。

然后和思路一一样继续遍历对比，这次的终止条件设为栈为空。（上面也是可以的）

思路3：同样定义快慢指针，迅速找到链表中点，记录下这个位置，一位后面判断完后还要恢复链表。然后慢指继续向前进行链表反转的操作，反转完成后指针一从头出发，指针而从尾部出发，相互对比前进，如果到达中点出还是相等那就视为回文结构。

这里主要注意这里的中点分奇偶两种情况，如果为奇数那么指的就是正中间，如果是偶数那么应该到中间偏左那个点。

**十二、链表的荷兰国旗问题**

思路1：将链表读入数组中，然后用荷兰国旗问题排好三个区域，然后再讲数组转换成链表返回，缺点空间复杂度O(N)

思路2：创建三个链表，分别存储小于等于大于三个区域，遍历链表，将遍历到的链表节点（注意遍历的时候取得一个节点应该将其和后面断开再加入相应的链表中）与指定的值比较然后存入相应的链表中（这个过程和数组创建链表的过程（规范过程要看）很相似），最后将三个链表串起来（串起来的逻辑比较复杂）。

**十三、复杂链表的复制问题**

问题的关键在于如何根据现有的链表元素找到其对应复制的那个元素思路1通过map映射来做，思路2通过插到其后面来做。

思路1：利用map的一对一映射关系来做，第一次遍历，将原始链表的元素作为key，对其值进行复制放在对应的value中.然后重新遍历复制指针，由于一对一的映射。通过原始链表元素找到复制的链表的元素。

思路2：不适用额外空间的思路。第一次遍历链表，将每个元素复制并将复制后的分别放在其对应的后面，形成一个复制链表与原链表相交错的情况。第二次遍历复制随机指针，最后将整个长的链表解开成原始链表和复制链表。

**十四、判断两个单链表是否相交**

**1、判断单链表有没有环** 思路1：占用额外空间Set。遍历链表，首先盘算set中是否包含该元素，有的话这个就是公共节点，如果没有继续遍历最后如果碰到null跳出循环说明没有。

思路2:不占用外围空间，是重点。利用快慢指针实现，首先快指针走两步，慢指着走一步，如果快指针碰到null说明无环，如果不是的话一直走，慢指着和快指着最终会走到一起，这个时候让快指针回到head,两个指针每次都各走一步，第一次相遇就是公共节点。

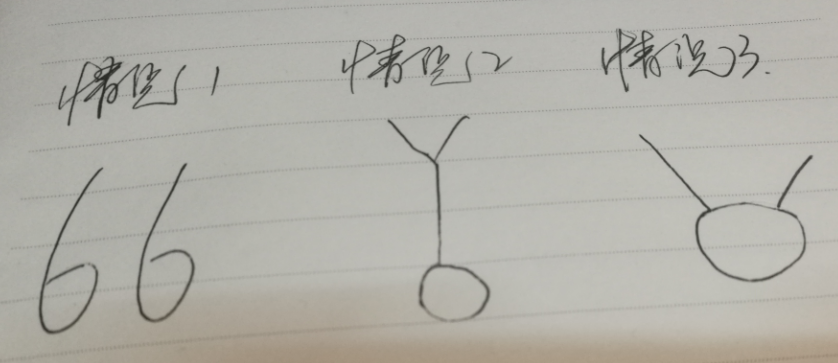
**2、判断无两个环单链表是否相交**

思路1：占用额外空间Set。遍历链表1，将每个元素都放到set中，然后遍历链表2，每个元素都看下set中有没有，有的话就是相交，该点就是相交点，如果遍历完都没有就是不相交。

思路2:不占用外围空间，是重点。定义计数变量初始值0，遍历链表1，没走一步计数加一次，直到走到最后一个节点。遍历链表2，每次走一步计数减一次，直到走到最后一个节点（目的为了看这两个链表的长度差）。先判断两个的最后一个节点是不是的同一个，不是的话就不相交，是的话判断一下两个链表的长度关系，长的那个先走两者计数值差步，然后一起走，走到相遇就是相交的点

**3、判断两个有环单链表是否相交（一个无环一个有环必定不相交）**

两个有环链表就是三种情况如图：



首先判断两个入环节点是不是一样，是的话（情况2）就是相交，找到第一个相交点的方法就和上面第2点一样，不在赘述。然后从loop1开始一直往后走，过中间碰到了loop2那么两个相交（情况3），返回loop1和loop2都行，如果走了一圈回到loop1还没遇到loop2说明两个不相交）（情况1）。