LeetCode 链表类总结

链表是LeetCode里面比较容易解决的一种问题，它的解题套路比较少，归纳起来也不困难。链表结构分为单链表和双链表。leetcode里面单链表操作比较多。双链表相对较少。

1，链表的创建
链表创建一般分为两种，头插法和尾插法，分别是每次将新的节点插入链表头部或者尾部。头插法得到的链表为倒序，而尾插法得到的为顺序。

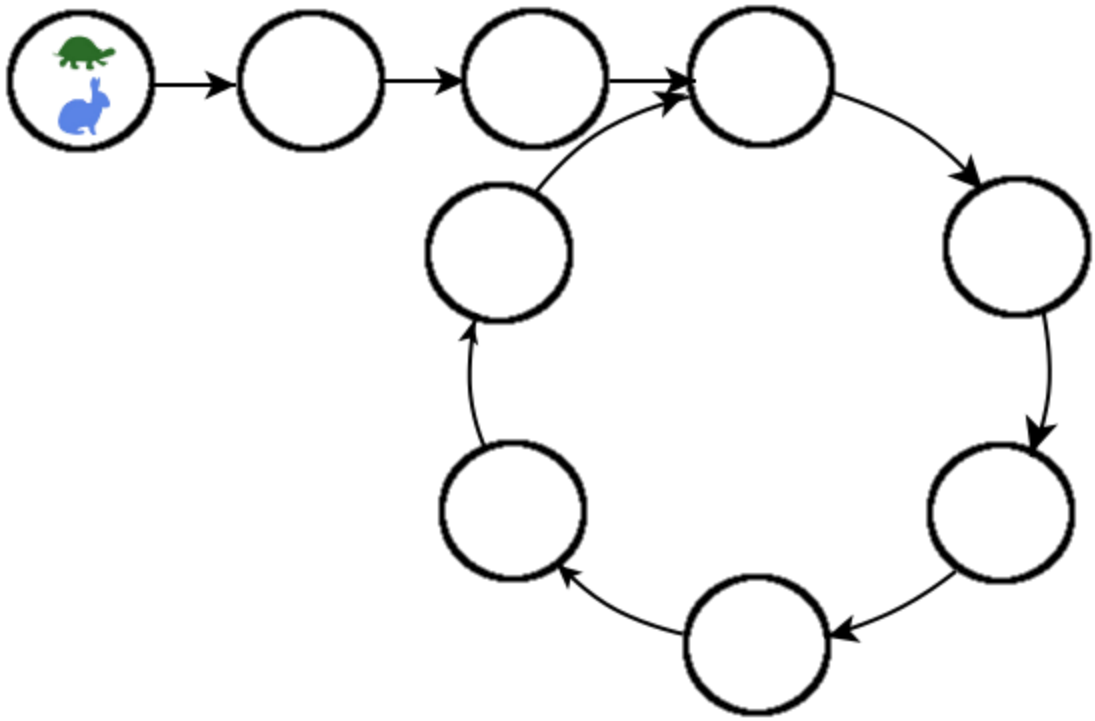
顺序
```
Node head = new Node();
Node node = head;
for(int i : arr)
{
        node.next = new Node(i);
        node = node.next;
}
```
逆序
```
Node head = new Node();
Node node = null;
for(int i : arr)
{
        node = head.next;
        head.next = new Node(i);
        Head.next.next = node;
}
```

2，链表的操作
链表加法 lc 2
合并链表 lc 21，lc23
交换节点 lc24，lc25
拷贝链表 lc 138,
反转 lc206

3，链表中间点
我们可以使用快慢指针遍历链表来找到链表的中间点：
```
slowpointer = head;
fastpointer = head;
while(fastpointer!=null && fastpointer.next != null)
{
        slowpointer = slowpointer.next;
        fastpointer = fastpointer.next.next;
}
```

例子：lc 143，lc 234

4，链表环问题



简单来说，可以使用快慢指针法测试一个链表是否存在环。快指针如果和慢指针相遇而非到达末尾的话，那可以确定**存在环**。到达环后，把慢指针重新定向到头部，然后再做一次移动，这次的交点是**环入口**。这个方法也叫龟兔追击算法。

1，如何寻找环 lc 141
2，如何寻找环入口lc 142

Floyd cycle detection （tortoise and hare algorithm）
from wiki
算法描述
如果有限状态机、迭代函数或者链表存在环，那么一定存在一个起点可以到达某个环的某处(这个起点也可以在某个环上)。

初始状态下，假设已知某个起点节点为节点S。现设两个指针t和h，将它们均指向S。

接着，同时让t和h往前推进，但是二者的速度不同：t每前进1步，h前进2步。只要二者都可以前进而且没有相遇，就如此保持二者的推进。当h无法前进，即到达某个没有后继的节点时，就可以确定从S出发不会遇到环。反之当t与h再次相遇时，就可以确定从S出发一定会进入某个环，设其为环C。

如果确定了存在某个环，就可以求此环的起点与长度。

上述算法刚判断出存在环C时，显然t和h位于同一节点，设其为节点M。显然，仅需令h不动，而t不断推进，最终又会返回节点M，统计这一次t推进的步数，显然这就是环C的长度。

为了求出环C的起点，只要令h仍均位于节点M，而令t返回起点节点S，此时h与t之间距为环C长度的整数倍。随后，同时让t和h往前推进，且保持二者的速度相同：t每前进1步，h前进1步。持续该过程直至t与h再一次相遇，设此次相遇时位于同一节点P，则节点P即为从节点S出发所到达的环C的第一个节点，即环C的一个起点。

阅读材料
https://math.stackexchange.com/questions/913499/proof-of-floyd-cycle-chasing-tortoise-and-hare
https://stackoverflow.com/questions/3952805/proof-of-detecting-the-start-of-cycle-in-linked-list?rq=1

习题
lc 2，lc23，lc24，lc25，lc 138，lc206，lc 143，lc 234，lc 141， lc142