Linked List

#G面试准备

Reverse Linked List II

Reverse Linked List II

- 思想：存第一个点的起始和prev 存最后一个点和其next 中间next指针调换 最后头尾next正确链接
- 一开始写复杂了 而且中间指针调换没做好
- 先用prev遍历到第m个点的prev 那第m个点node1就是prev.next
- prev留住 用来最后改指针
- 进入m-n次的循环 2 ➡ 3 ➡ 4 变成 4 ➡ 3 ➡ 2

  curt nkplus temp
- temp = nkplus.next
- nkplus.next = curt
- curt = nkplus
- nkplus = temp
- 循环结束 curt是子序列最后一个点 nkplus是他的下一个点
- 改头尾的next
  - m的prev node的next指向n
  - m的next指向nkplus

Reverse Nodes in k-Group

Reverse Nodes in k-Group - LeetCode

- 用到上一题的解法 解决每一个子问题
- helper函数返回子序列的prev 子序列的node1就是prev.next
- 如果不够k个点就返回null 主函数判断如果有null就退 不做了
- 注意找nodek这回要遍历k次 因为是从prev开始找的
- 退出for循环也要判断当前nk是不是为空 就正好比k少一个的情况
- 如果还是像上题一样定义nkplus在curt的next话 报错null pointer
  - 有点奇怪 如果少点应该直接就退了 不知道为什么nkplus会是null

```
ListNode curt = node1;
ListNode nkplus = curt.next;
while (nkplus != nk.next) {
     ListNode temp = nkplus.next; // null pointer
     nkplus.next = curt;
```

```
        curt = nkplus;
        nkplus = temp;
    }
```

- 所以用curt和cur_prev来 curt从node1开始 cur_prev初始为0 反正之后都会改指针的 curt在node1乱指next也没关系
- 最后改两个指针
- 返回值注意 是返回node1不是nodek 作为下一轮函数的prev指针
  - 因为已经反转了 最后一个node是node1

Copy List with Random Pointer

Copy List with Random Pointer - LeetCode

- hashmap做法 key是原node value是新node
- dummy指向新列表
- pre是新列表的prev指针
- head在原列表里面遍历
- 如果hashmap里已经存在新node 当前newNode指针就直接get
- 如果不存在 newNode就要new一个新的node
- pre.next = newNode
- random pointer可能为null 所以copy之前还要判断当前head.random != null
  - 错在这里
- head.random判断在不在map里 不在就创建 在的话直接把当前newNode.random = map.get(head.random)
- head = head.next
- pre = pre.next
- 返回dummy.next

Linked List Cycle

Linked List Cycle - LeetCode

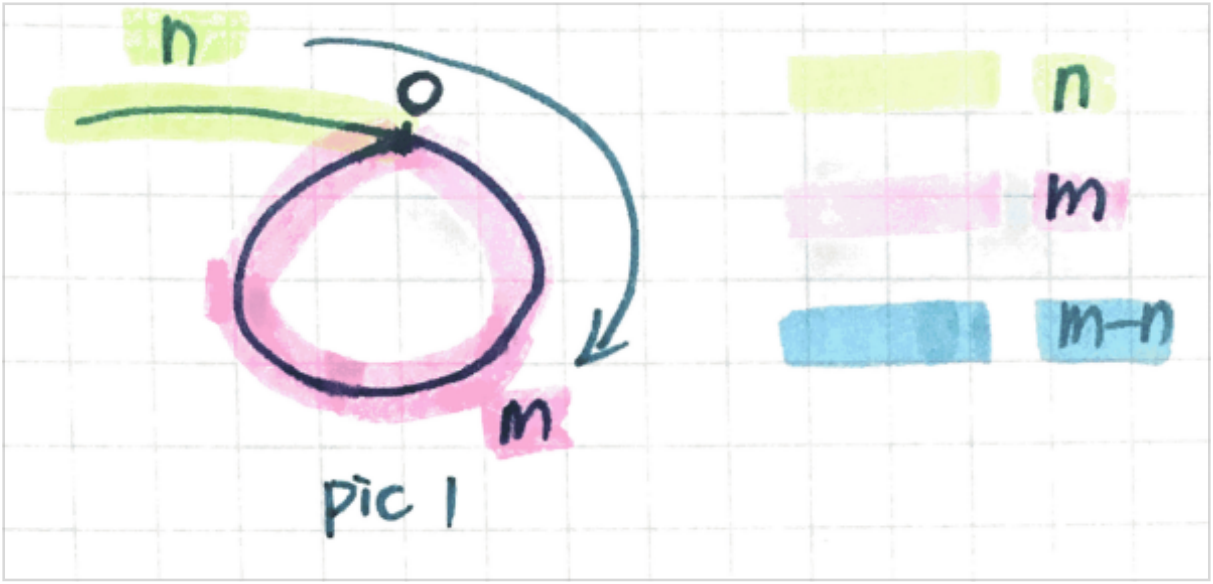
- 快慢指针
- 要注意初始化的时候
  - slow = head
  - fast = head.next 不是 = head 否则只有一个node的情况就会WA
  - while slow != fast
  - 判断fast不空 fast.next也不空

Linked List Cycle II

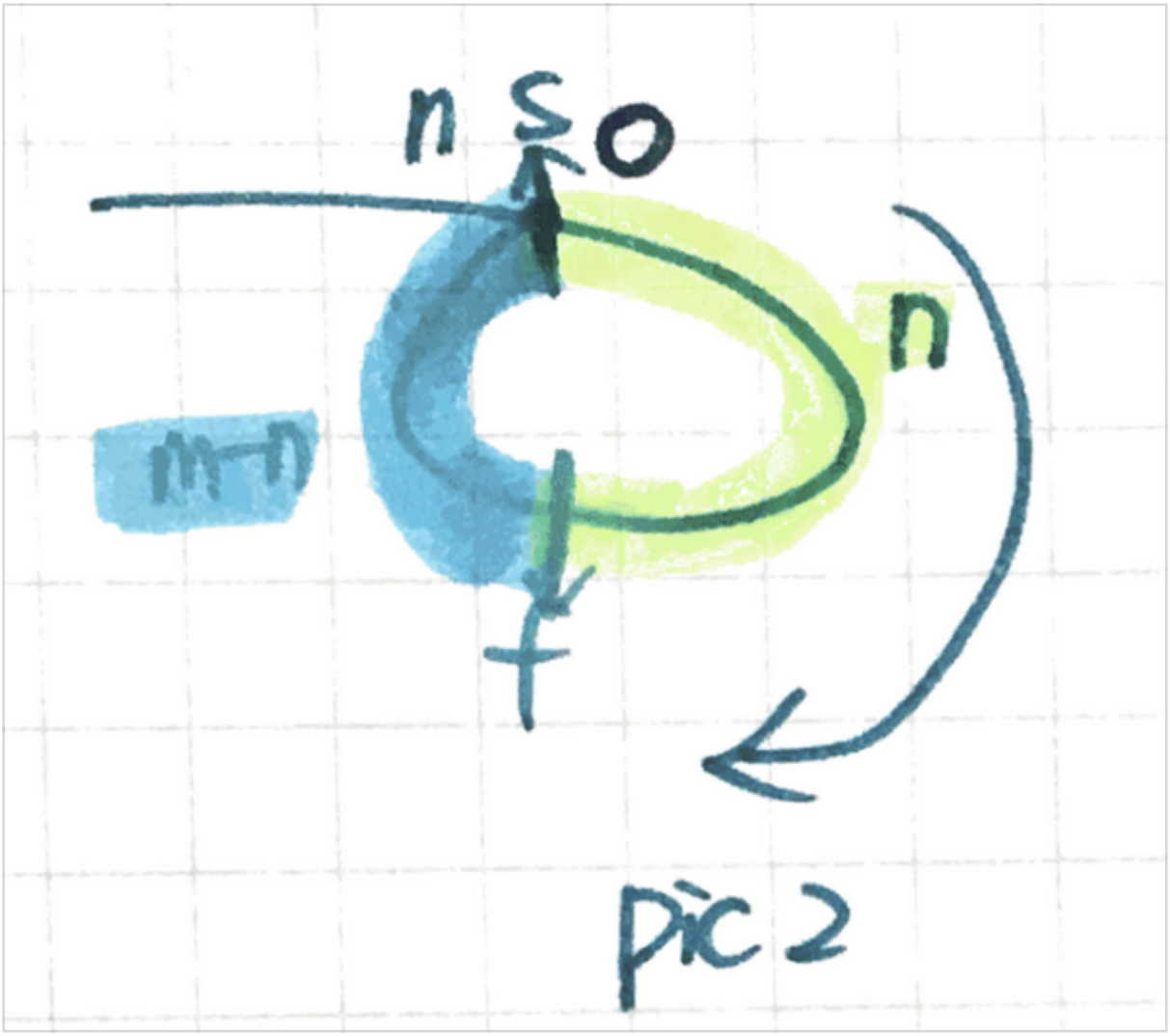
看图理解

九章算法笔记 6.链表与数组 Linked List & Array – Stomachache007

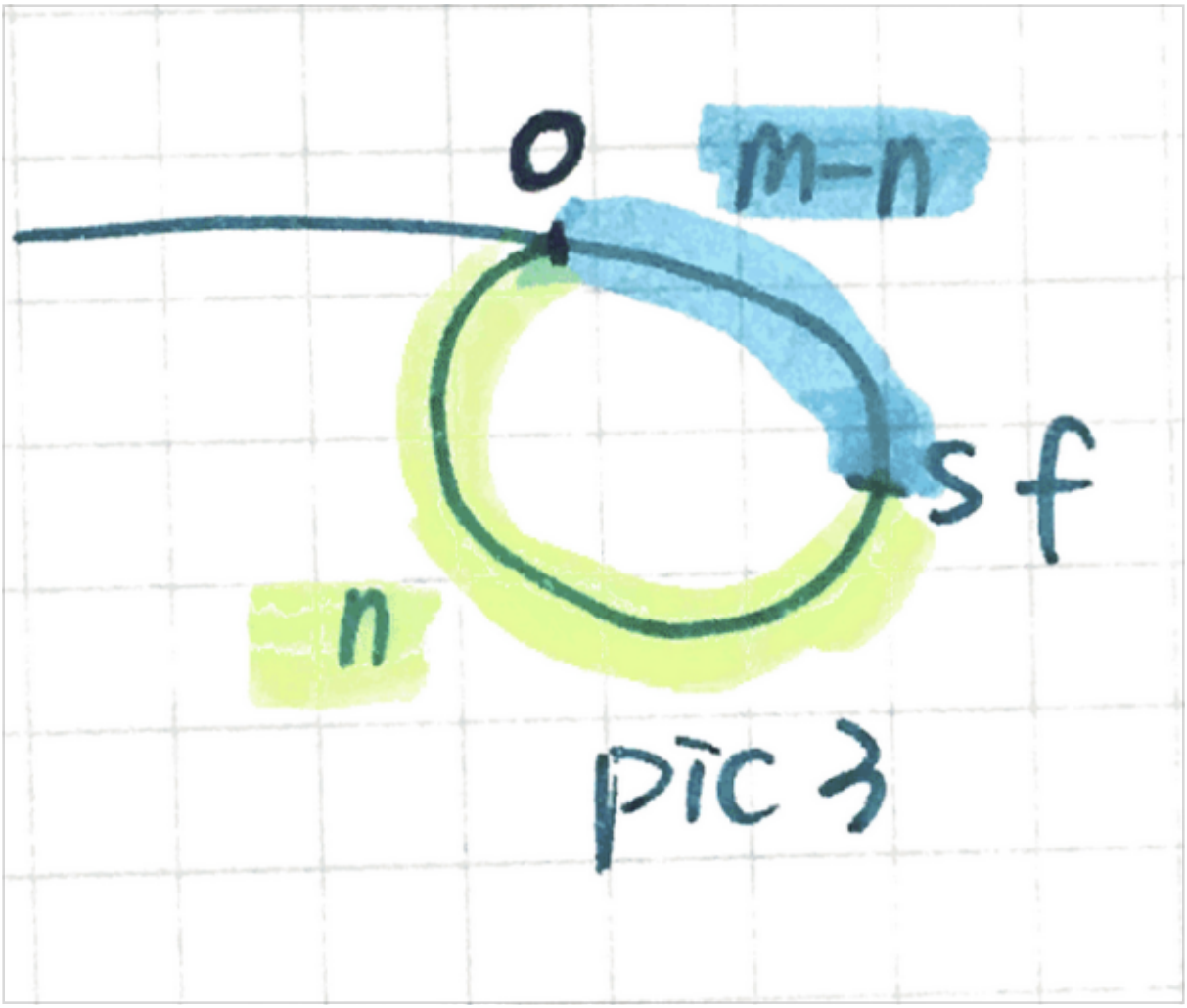
1.环如图，环外长度n用黄绿色表示，环长度m，用粉色表示如pic1。



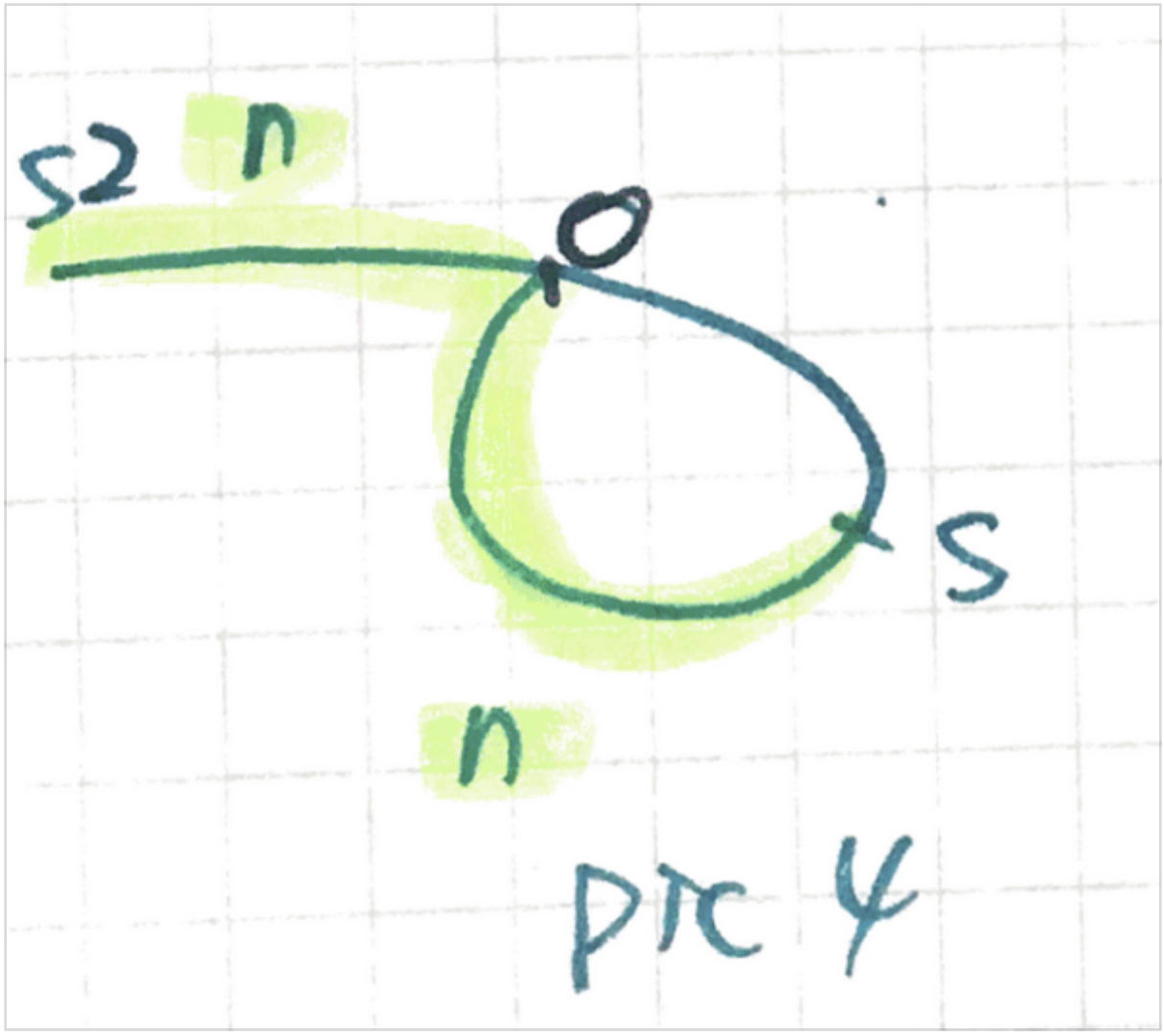
2.因为s一次一步，f一次两步，所以s每走一步，f都比s多走一步。所以当慢指针s走到n的时候，f在他前面n步，如pic2。



3.现阶段f已经在s前面n步了。两个人离口圈还差m-n步，m-n长度用蓝色表示。如果f想追上s，需要比s多走m-n步。又因为f比s多走的步数和s走的一样多（s走1步，f比他多走1步；s走2步，f比它多走2步；以此类推）。所以当s再走m-n步的时候，s和f相遇如pic3.



4.pic3的时候，是s从交点O走出来m-n步的时候，所以s想走到O，还需要m-（m-n）即n步。此时再放一个慢指针s2在起点，当s和s2都一次一步的走的时候，他们同事走完黄绿色的n的长度，在交点O相遇。Bingo

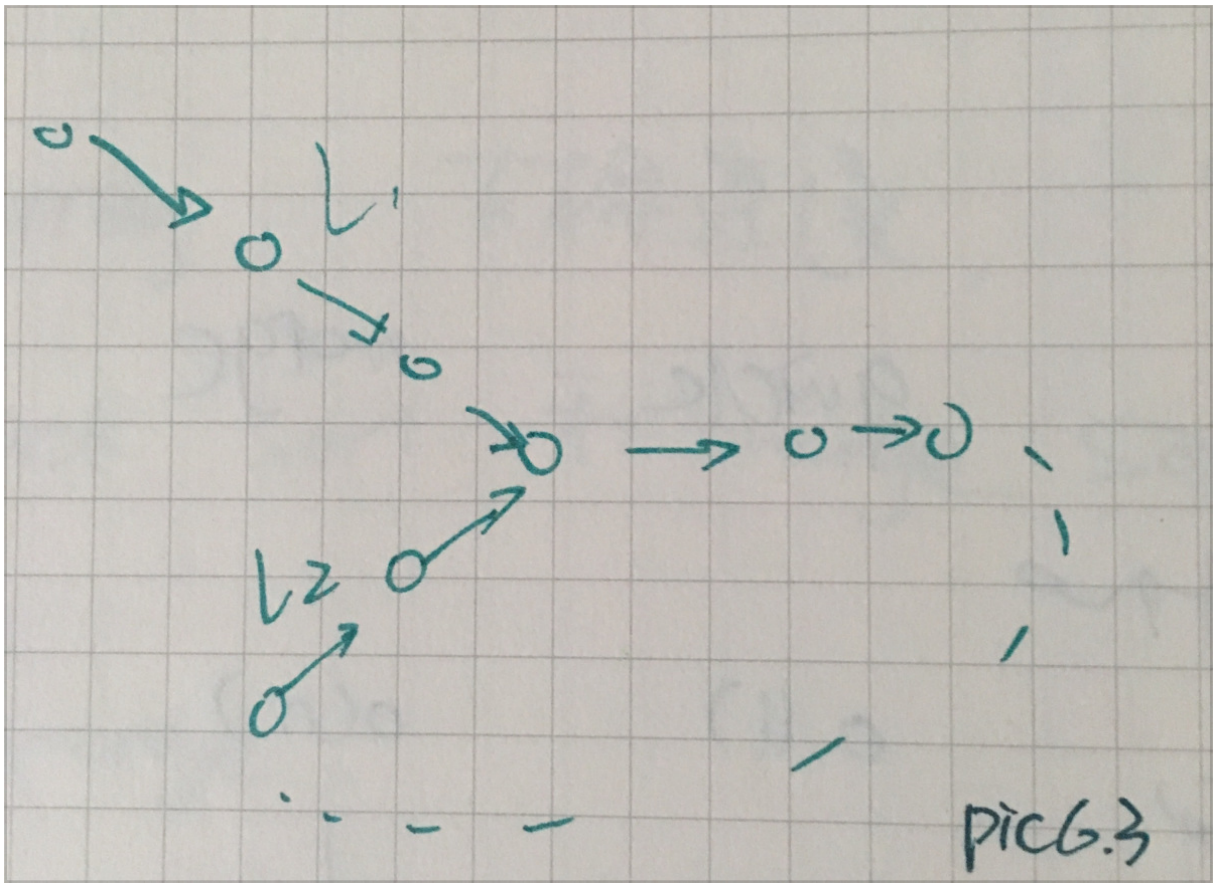


Linked List Cycle II - LeetCode

- s在s和f相遇的点 s2新的慢指针从起点开始走 走到相同n步（从起点到入口的距离）s和s2会相遇
- 找相遇点跟前面一样
- 后面要看了 while的条件是head != slow.next
  - 不是head != slow
  - 因为head其实已经先走在了第一个点
- 返回的结果是head 或者slow.next
  - 不是slow!
  - 错在了这里

Intersection of Two Linked Lists

Intersection of Two Linked Lists - LeetCode



- 先从headA开始找A的尾巴
- 把A的尾巴的next和headB相连 形成一个环
- 就变成了Linked list cycle ii的问题
- 如果两个list没有相交的话 就不会有环
- 结束之前要记得把A尾巴的next改回null 满足题目不更改结构的要求

369. Plus One Linked List

https://leetcode.com/problems/plus-one-linked-list/description/

- 数组plus one题的follow up
- 两种思路: 递归和stack
- 递归做法
- helper往下直到node.next == null 开始处理是否9的情况
- return value是carry
- 从最后一个node开始 如果是9就变成0 返回的carry为1 不是的话加一 返回carry为0
- helper的另一半是node.next != null的情况 在里面先递归调用helper 然后用变量carry接住返回值
- 递归最深层结束 如果返回的是1 就在下面继续处理9或非9的情况 同时判断carry应该继续返回1还是返回0
- 最外层返回0

- 主函数里调用helper 最后还是要判断第一个node是不是0 如果是0的话 加一个值为1的dummy node在前面 返回dummy node 否则返回head