**难度: 简单**

编写一个函数，检查输入的链表是否是回文的。

示例 1：

输入： 1->2

输出： false

示例 2：

输入： 1->2->2->1

输出： true

进阶：

你能否用 O(n) 时间复杂度和 O(1) 空间复杂度解决此题？

来源：力扣（LeetCode）

链接：https://leetcode-cn.com/problems/palindrome-linked-list-lcci

著作权归领扣网络所有。商业转载请联系官方授权，非商业转载请注明出处。

**思路:**

1).由于回文数的特性,我们直接把链表中的所以值全部加入到栈中

2).再次循环链表中的值,与栈中抛出的值相互比较

1. 如果不同,直接返回false

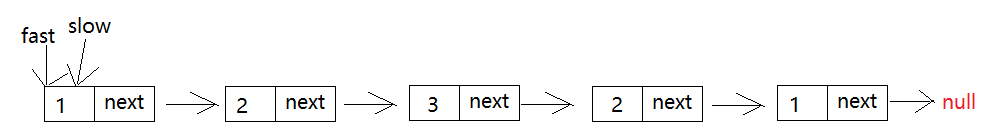
3). 如果能结束循环,则直接返回true

|  |
| --- |
| public boolean isPalindrome(ListNode head) {  if (head == null) {  return true;  }  //获取节点存入栈  Stack<Integer> stack = setStack(head);  //比较  while(head.next != null) {  if (head.val != stack.pop()) {  return false;  }  head = head.next;  }  return true;  }  //存入到栈中  private Stack<Integer> setStack(ListNode head) {  ListNode temp = head;  Stack<Integer> stack = new Stack<>();  while(temp != null) {  stack.push(temp.val);  temp = temp.next;  }  return stack;  } |

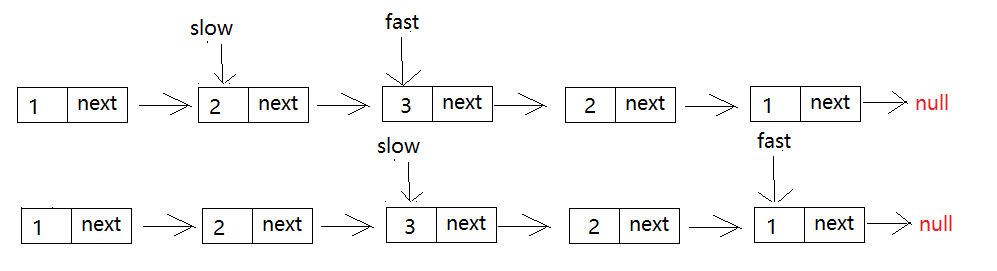
**思路2(使用快慢指针和反转):**

1). 先通过快慢指针找到中间节点

1.一开始

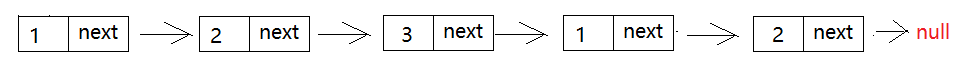


2.持续往下走

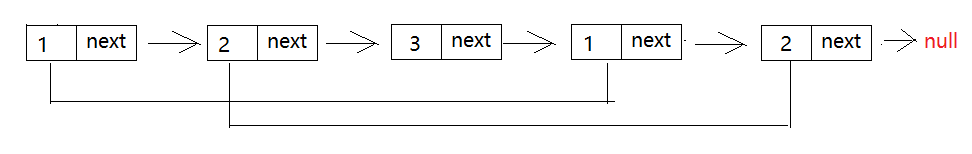


3. 此时slow就是中间节点

2). 反转链表



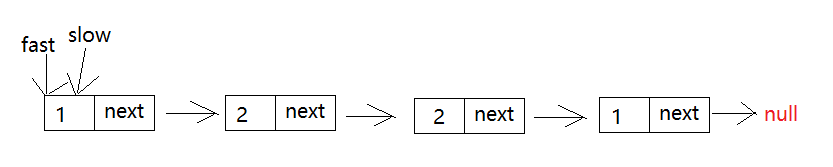
3). 比较节点

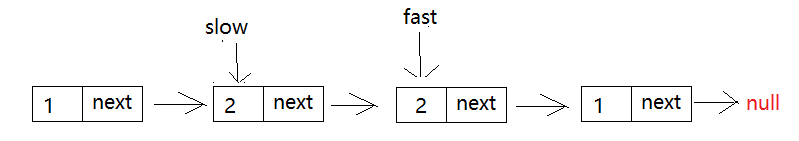


**除了考虑奇数还要考虑偶遇的情况**

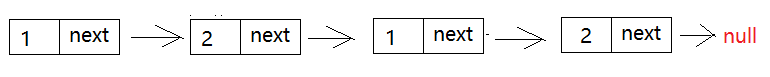
1). 先通过快慢指针找到中间节点

1. 一开始

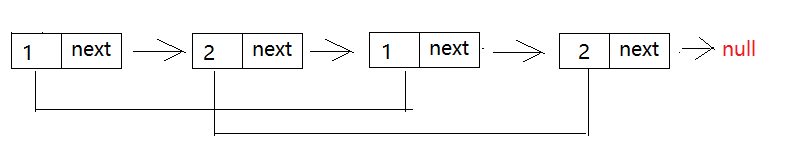
 2.持续往下走



2). 反转链表



3. 比较节点



|  |
| --- |
| public boolean isPalindrome2(ListNode head) {  if (head == null) {  return true;  }  ListNode midNode = findMidNode(head);  ListNode reverseList = reverseLinked(midNode);  ListNode curr = head;  ListNode curr2 = reverseList;  while(curr2.next != null) {  if (curr.val != curr2.val) {  return false;  }  curr = curr.next;  curr2 = curr2.next;  }  return true;  }  //寻找中间指针  private ListNode findMidNode(ListNode head) {  //快指针  ListNode fast = head;  //慢指针  ListNode slow = head;  while(fast.next != null && fast.next.next != null) {  slow = slow.next;  fast = fast.next.next;  }  return slow;  }  //反转指针  private ListNode reverseLinked(ListNode head) {  //临时结点  ListNode temp = head;  //反转的结点  ListNode prev = null;  while(temp != null) {  //下一个结点  ListNode nextNode = temp.next;  temp.next = prev;  prev = temp;  temp = nextNode;  }  return prev;  } |