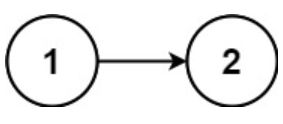
# 题目

请判断一个链表是否为回文链表。

给你一个单链表的头节点head，请你判断该链表是否为回文链表。如果是，返回true；否则，返回false。

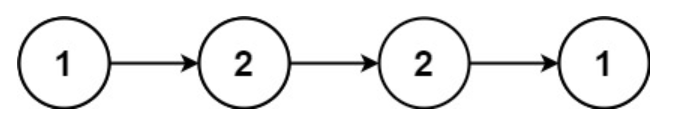
**示例 1:**



输入: 1->2

输出: false

**示例 2:**



输入: 1->2->2->1

输出: true

**提示：**

链表中节点数目在范围[1, 105] 内

0 <= Node.val <= 9

**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？

# 分析

涉及到回文相关的，一般考虑两个思路：

1. 栈
2. 快慢指针/双指针

## 方法一：双指针/快慢指针

**思路：**

用快慢指针遍历的同时翻转前半部分，然后与后半部分比较即可。

**代码：**

class Solution {

public:

bool isPalindrome(ListNode\* head) {

if(!head || !head->next)

return 1;

ListNode \*fast = head, \*slow = head;

ListNode \*p, \*pre = NULL;

while(fast && fast->next){

p = slow;

slow = slow->next; //快慢遍历

fast = fast->next->next;

p->next = pre; //翻转

pre = p;

}

if(fast) //奇数个节点时跳过中间节点

slow = slow->next;

while(p){ //前半部分和后半部分比较

if(p->val != slow->val)

return 0;

p = p->next;

slow = slow->next;

}

return 1;

}

};

## 方法二：栈

class Solution {

public:

bool isPalindrome(ListNode\* head) {

stack<int> s;

ListNode \*p = head;

while(p){

s.push(p->val);

p = p->next;

}

p = head;//再从头开始与栈中的元素（已经逆序）比较

while(p){

if(p->val != s.top()){

return 0;

}

s.pop();

p = p->next;

}

return 1;

}

};

**另一种写法：**

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode() : val(0), next(nullptr) {}

\* ListNode(int x) : val(x), next(nullptr) {}

\* ListNode(int x, ListNode \*next) : val(x), next(next) {}

\* };

\*/

class Solution {

public:

bool isPalindrome(ListNode\* head) {

stack<int> stk;

ListNode \*node = head;

while(node) {

stk.push(node->val);

node = node->next;

}

node = head;

while (node) {

if (node->val != stk.top()) {

return false;

}

stk.pop();

node = node->next;

}

return true;

}

};

或：

/\*\*

 \* Definition for singly-linked list.

 \* struct ListNode {

 \*     int val;

 \*     ListNode \*next;

 \*     ListNode() : val(0), next(nullptr) {}

 \*     ListNode(int x) : val(x), next(nullptr) {}

 \*     ListNode(int x, ListNode \*next) : val(x), next(next) {}

 \* };

 \*/

class Solution {

public:

    bool isPalindrome(ListNode\* head) {

        ListNode \*tmpNode = head;

        stack<int> stk;

        while(tmpNode)

        {

            stk.push(tmpNode->val);

            tmpNode = tmpNode->next;

        }

        tmpNode = head;

        while(tmpNode)

        {

            if(tmpNode->val != stk.top())

                return false;

            stk.pop();

            tmpNode = tmpNode->next;

        }

        return true;

    }

};

## 方法三：递归法