# 141. Linked List Cycle

**Given a linked list, determine if it has a cycle in it.**

Follow up: Can you solve it without using extra space?

/\*\*

\* Definition for singly-linked list.

\* class ListNode {

\* int val;

\* ListNode next;

\* ListNode(int x) {

\* val = x;

\* next = null;

\* }

\* }

\*/

## 算法1：自己想的。

算法思路：递归思想。记录next的次数，每次递归向下next一次，然后将该current 节点与之前的所有节点遍历判断一遍。若存在相等，则直接返回true，否则继续递归下一节点。

递归的终止条件：遇到节点为null，则返回false。

这种算法:性能不好，建议采用第二种算法。

// //递归方法

public class Solution {

public boolean hasCycle(ListNode head) {

if(head == null) return false;

return isCycle(head,head.next,2);

}

private boolean isCycle(ListNode head,ListNode current,int level){

ListNode temp = head;

if(current == null) return false;

for(int i = 1;i < level;i++){//遍历当前节点是否和前面的节点相同

if(current == temp) return true;

temp = temp.next;

}

return isCycle(head,current.next,level+1);

}

}

## 算法2：性能较好

算法思路：

目前的理解：slow是一步一步走，fast是两步两步地走。当不存在循环链表时，slow和fast肯定会汇聚到一个节点上，所以该算法虽然理解困难，但是效果很好。

public class Solution {

public boolean hasCycle(ListNode head) {

if(head == null) return false;

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

while(fast!=null && fast.next!=null)

{

slow = slow.next;

fast = fast.next.next;

if(slow == fast){

return true;

}

}

return false;

}

}