??

- □□□□□□□□

????????

?????????????
??????????????????“?”????????????????????????????

??????????
?????O(n)??????O(1)?

???

1. □□□□
   □□
   □□□□□□□□□□□□□ ID□□□□□□□□□□ ID
   □□□□□□□□□□□□□□□□□□□□□□□□□□□□ ID
   □□
   □□□□□□□□□□□□□□□□□
2. □□□□□□□□ A->B->C->D->B->C->D□ □□□□□□ D
   □□□□□□□□□□□□□□□□□ A□ B□ C□□□□□□□□□□□□□□□□□□□□□□□
   B□ B□□□□□ A□ B□ C□ D□□□□□□ B□□□ B□□□□□□□□□□□□□□
3. □□□□□□□□□□□□□□□□ D□□□□□□□ S□□□□□□□□□□□□
   0+1+2+3+…+(D+S-1) = (D+S-1)*(D+S)/2 □ □□□□□□□□ O(N*N)
   □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ O(1)□

?????????

****□□□□□□□□□ ID□□□ HashSet□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□ HashSet□□□□□□□□□□□□□□□□□ HashSet
□□□□□□□□ ID□□□□□□□□□

□□ HashSet□□□□□□□□□□ ID□□□□□□□□ ID□□ HashSet
□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□ HashSet□□□□□□□□

□□□□□□□□□□□□□□□□ D□□□□□□□ S□□□□□ HashSet□□□□□□□□□□□ O(1),
□□□□□□□□□□□ 1*(D+S)=D+S□□□□□□□□ O(N)□□□□□□□□□□□□ D+S-1
□□□□□□□□□ O(N)□

????????

首先创建两个指针 1和 2（在 java里就是两个对象引用），同时指向这个链表的头节点。

然后开始一个大循环，在循环体中，让指针 1每次向下移动一个节点，让指针 2
每次向下移动两个节点，然后比较两个指针指向的节点是否相同。如果相同，则判断出链表有环，如果不同，
则继续下一次循环。

例如链表 A->B->C->D->B->C->D，两个指针最初都指向节点 A，进入第一轮循环，指针 1移动到了节点 B
，指针 2移动到了 C。第二轮循环，指针 1移动到了节点 C，指针 2移动到了节点 B。第三轮循环，指针 1
移动到了节点 D，指针 2移动到了节点 D，此时两指针指向同一节点，判断出链表有环。

此方法也可以用一个更生动的例子来形容：在一个环形跑道上，两个运动员在同一地点起跑，一个运动员速度
快，一个运动员速度慢。当两人跑了一段时间，速度快的运动员必然会从速度慢的运动员身后再次追上并超过
，原因很简单，因为跑道是环形的。

```
/**
 * 判断单链表是否存在环
 * @param head
 * @return
 */
public static <T> boolean isLoopList(ListNode<T> head){
    ListNode<T> slowPointer, fastPointer;

    //使用快慢指针，慢指针每次向前一步，快指针每次两步
    slowPointer = fastPointer = head;
    while(fastPointer != null && fastPointer.next != null){
        slowPointer = slowPointer.next;
        fastPointer = fastPointer.next.next;

        //两指针相遇则有环
        if(slowPointer == fastPointer){
            return true;
        }
    }
    return false;
}
```

假设从链表头节点到入环点的距离是 D，链表的环长是 S。那么循环会进行 S次（为什么是 S
次，有心的同学可以自己揣摩下），可以简单理解为 O（ N
）。除了两个指针以外，没有使用任何额外存储空间，所以空间复杂度是 O（ 1）。

????Set??????

评论中有 @长沙小辣椒 同学指出：还可以用 set 遍历链表，把节点放入 set里，每次访问下个节点时，如果 set [illegible] set[illegible] +1[illegible]

[illegible] O[illegible] N[illegible] O[illegible] n[illegible]

?????????????

[illegible]

[illegible] fast[illegible] slow[illegible] slow[illegible] ([illegible])[illegible] ([illegible] 1[illegible])
[illegible] ([illegible] n[illegible] c[illegible] n=s+p[illegible] c[illegible] n[illegible] c [illegible] CB[illegible] n-p=s[illegible] s[illegible] A [illegible] B(AB=CB=s))[illegible]

```
/**
 * [illegible]
 * @param head
 * @return
 */
public static <T> ListNode<T> findEntranceInLoopList(ListNode<T> head){
    ListNode<T> slowPointer, fastPointer;

    //[illegible]
    boolean isLoop = false;
    slowPointer = fastPointer = head;
    while(fastPointer != null && fastPointer.next != null){
        slowPointer = slowPointer.next;
        fastPointer = fastPointer.next.next;

        //[illegible]
        if(slowPointer == fastPointer){
            isLoop = true;
            break;
        }
    }

    //[illegible]
    if(isLoop){
        slowPointer = head;
        while(fastPointer != null && fastPointer.next != null){
            //[illegible]
            if(slowPointer == fastPointer){
```

```
                return slowPointer;
            }

            slowPointer = slowPointer.next;
            fastPointer = fastPointer.next;
        }
    }
    return null;
}
```

[illegible]

[illegible]
[illegible] O(len1*len2)[illegible]

[illegible]
[illegible]
[illegible] [illegible] hash[illegible] hash
[illegible] hash[illegible] hash[illegible]
[illegible] O(max(len1+len2)[illegible] O(len1)
[illegible] [illegible]

[illegible] Hash
[illegible] Hash[illegible]

[illegible] Hash[illegible]
[illegible] O(m+n)[illegible] m[illegible] n
[illegible]
[illegible] Hash[illegible]

[illegible]
[illegible]
[illegible] O(len1+len2)
[illegible] K
[illegible] K[illegible]

[illegible]
[illegible]

[illegible]
[illegible]
[illegible]
[illegible] [illegible] [illegible] [illegible]
[illegible]
[illegible] [illegible]
[illegible]

????????

[illegible] N[illegible] n[illegible]

[illegible] n>=N/2[illegible] n[illegible]

[illegible] n[illegible] 2n[illegible] n [illegible] n[illegible]

[illegible] n<N/2[illegible] [illegible] N-N%n[illegible]

[illegible] N%n<n[illegible] N-N%n[illegible] N-N%n[illegible] n [illegible] [illegible] [illegible]

[illegible] N[illegible] n [illegible] n [illegible] N-N%n[illegible] m[illegible]

[illegible] m[illegible] m[illegible] m[illegible] m [illegible] m [illegible] m[illegible] m [illegible]

[illegible]

```
Node *getCircleHead(Node *node){
    //[illegible]
    if (hasCircle(node)) {

        Node *fast=node;
        Node *slow=node;

        //[illegible] m
        do{
            fast=fast->next->next;
            slow=slow->next;
        }while (fast!=slow);

        //[illegible]
        fast=node;
```

```
        while (fast!=slow) {
            fast=fast->next;
            slow=slow->next;
        }
        return fast;
    }
    return NULL;
}
```

[illegible]

[illegible] n[illegible] N-N%n[illegible]

[illegible] N[illegible] n[illegible] m=N-n[illegible]

[illegible] m[illegible] 2m[illegible]

[illegible] 2m-m=m[illegible] n[illegible] m[illegible] n[illegible] m%n[illegible]

[illegible] (n-m%n)%n [illegible] 2*((n-m%n)%n)[illegible]

OK[illegible] d=m+(n-m%n)%n[illegible]

1. [illegible] n-m%n[illegible] 0<n-m%n<=n[illegible] m==n[illegible] n-m%n=n[illegible] m=n[illegible] d=m+(n-m%n)%n[illegible] d=n;

2. [illegible] m!=n[illegible] 0<n-m%n<n [illegible] (n-m%n)%n=n-m%n [illegible] d= m+n-m%n =m+n-m%n [illegible]

[illegible] 1[illegible] m<n[illegible] d=m+n-m=n;

[illegible] 2[illegible] m>n[illegible] d=m+n-m%n=N-(N-n)%n=N-N%n;

[illegible]

[illegible] 1[illegible] m<=n[illegible] d=n;

[illegible] 2[illegible] m> n[illegible] d=N-N%n;

[illegible]

[illegible] d [illegible] d [illegible] d
[illegible] d [illegible]

[illegible] $m <= n$ [illegible] $d = n$ [illegible] n [illegible] d [illegible]

[illegible] $m > n$ [illegible] [illegible] $d = m + n - m\%n$,

$d\%n = m\%n + n\%n - m\%n = 0$ [illegible]

————————————————