# 题目

链表中的临界点定义为一个局部极大值点或局部极小值点。

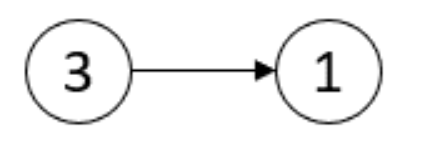
如果当前节点的值严格大于前一个节点和后一个节点，那么这个节点就是一个局部极大值点。

如果当前节点的值严格小于前一个节点和后一个节点，那么这个节点就是一个局部极小值点 。

注意：节点只有在同时存在前一个节点和后一个节点的情况下，才能成为一个局部极大值点/极小值点 。

给你一个链表head，返回一个长度为2的数组[minDistance, maxDistance]，其中minDistance是任意两个不同临界点之间的最小距离，maxDistance是任意两个不同临界点之间的最大距离。如果临界点少于两个，则返回[-1，-1]。

示例 1：

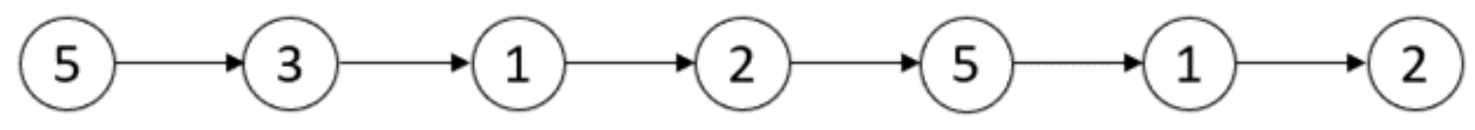


输入：head = [3,1]

输出：[-1,-1]

解释：链表 [3,1] 中不存在临界点。

示例 2：



输入：head = [5,3,1,2,5,1,2]

输出：[1,3]

解释：存在三个临界点：

- [5,3,1,2,5,1,2]：第三个节点是一个局部极小值点，因为 1 比 3 和 2 小。

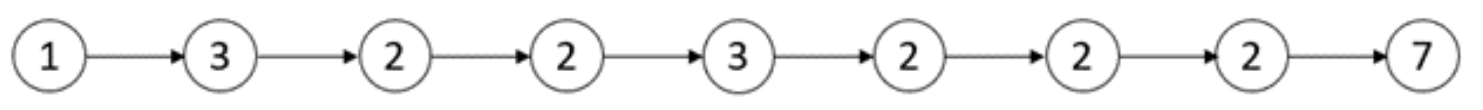
- [5,3,1,2,5,1,2]：第五个节点是一个局部极大值点，因为 5 比 2 和 1 大。

- [5,3,1,2,5,1,2]：第六个节点是一个局部极小值点，因为 1 比 5 和 2 小。

第五个节点和第六个节点之间距离最小。minDistance = 6 - 5 = 1 。

第三个节点和第六个节点之间距离最大。maxDistance = 6 - 3 = 3 。

示例 3：



输入：head = [1,3,2,2,3,2,2,2,7]

输出：[3,3]

解释：存在两个临界点：

- [1,3,2,2,3,2,2,2,7]：第二个节点是一个局部极大值点，因为 3 比 1 和 2 大。

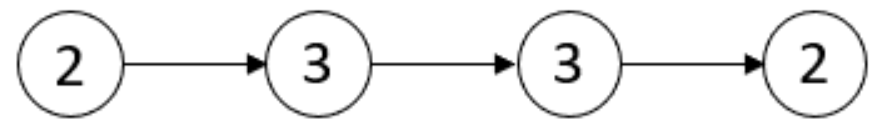
- [1,3,2,2,3,2,2,2,7]：第五个节点是一个局部极大值点，因为 3 比 2 和 2 大。

最小和最大距离都存在于第二个节点和第五个节点之间。

因此，minDistance 和 maxDistance 是 5 - 2 = 3 。

注意，最后一个节点不算一个局部极大值点，因为它之后就没有节点了。

示例 4：



输入：head = [2,3,3,2]

输出：[-1,-1]

解释：链表 [2,3,3,2] 中不存在临界点。

提示：

链表中节点的数量在范围 [2, 105] 内

1 <= Node.val <= 105

# 分析

分析：

我们可以对链表进行一次遍历。

当我们遍历到节点cur时，可以记cur的值、cur后一个节点的值、cur后两个节点的值，分别为x,y,z。如果y严格大于x和z，或者y严格小于x和z，那么cur的后一个节点就是临界点。

由于我们需要得到任意两个临界点之间的最小距离和最大距离，而我们可以发现：

1、最小距离一定出现在两个相邻的临界点之间；

2、最大距离一定出现在第一个和最后一个临界点之间。

因此，在遍历的过程中，我们可以维护上一个临界点的位置以及第一个临界点的位置。这样一来，每当我们找到一个临界点，就可以更新最小距离和最大距离。

代码：

/\*\*

\* 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:

vector<int> nodesBetweenCriticalPoints(ListNode\* head) {

int minDist = -1, maxDist = -1;

int first = -1, last = -1, pos = 0;

for (ListNode \*cur = head; cur && cur->next && cur->next->next; cur = cur->next) {

// 获取连续的三个节点的值

int x = cur->val;

int y = cur->next->val;

int z = cur->next->next->val;

// 判断临界点

if (y > max(x,z) || y < min(x, z)) {

// 找到已经更新的last节点，然后计算Distance信息

if (last != -1) {

// 最小值:当前位置 - 最后一个临界点last(但是需要处理第一次的情况)

if (minDist == -1) {

// 第一次初始化,更新当前位置的最小距离

minDist = pos - last;

} else {

minDist = min(minDist, pos - last);

}

// 最大值:当前位置 - 第一个临界点

maxDist = max(maxDist, pos - first);

}

// 只有第一次找到第一个临界点的时候，才需要更新first

if (first == -1) first = pos;

// last节点要不断更新为pos

last = pos;

}

pos++;

}

return {minDist, maxDist};

}

};

**复杂度分析：**

时间复杂度：O(n)，其中n是给定的链表的长度。

空间复杂度：O(1)。