# 题目

给定一个 双链表 的 任意 node，其中的节点具有指向下一个节点的指针和上一个节点的指针。

返回一个 按顺序 包含链表中元素的整数数组。

示例 1：

输入：head = [1,2,3,4,5], node = 5

输出：[1,2,3,4,5]

示例 2：

输入：head = [4,5,6,7,8], node = 8

输出：[4,5,6,7,8]

提示：

给定列表中的节点数在范围 [1, 500] 内。

1 <= Node.val <= 1000

所有节点的 Node.val 互不相同。

# 分析

要解决这个问题，我们需要从双链表的任意一个节点出发，提取整个链表的所有元素并按顺序组成数组。核心利用双链表的特性——每个节点有prev（前驱）和next（后继）指针，先通过prev找到链表的头节点，再从头部通过next遍历整个链表收集元素。

思路分析

1、找到链表头节点：由于输入的node是任意节点，我们需要先通过prev指针向左移动，直到prev为nullptr（此时到达链表的第一个节点，即头节点）。

2、遍历链表收集元素：从找到的头节点开始，通过next指针向右遍历整个链表，依次将每个节点的val存入结果数组，直到next为nullptr（遍历结束）。

代码：

/\*\*

\* Definition for doubly-linked list.

\* class Node {

\* int val;

\* Node\* prev;

\* Node\* next;

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

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

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

\* };

\*/

class Solution {

public:

vector<int> toArray(Node \*node) {

vector<int> result;

if (node == nullptr) return result; // 极端情况：输入节点为空（题目保证节点数≥1，可省略）

// 步骤1：通过prev指针找到链表的头节点（prev为nullptr的节点）

Node\* curr = node;

while (curr->prev != nullptr) {

curr = curr->prev;

}

// 步骤2：从头部通过next指针遍历，收集所有节点值

while (curr != nullptr) {

result.push\_back(curr->val);

curr = curr->next;

}

return result;

}

};

代码解释：

1、找到头节点：

- 初始化curr为输入的任意节点node。

- 循环移动curr到curr->prev，直到curr->prev为nullptr——此时curr就是链表的头节点（第一个元素），确保后续遍历从链表起点开始。

2、遍历收集元素：

- 从头部节点开始，通过curr->next依次访问每个节点，将节点的val添加到result数组。

- 当curr为nullptr时，说明已遍历到链表末尾，停止循环。

3、边界处理：题目保证链表节点数≥1，输入node非空，因此“输入节点为空”的判断可省略，但保留可增强代码鲁棒性。

复杂度分析

时间复杂度：O(n)，其中n是链表的节点数。

- 找头节点的过程最多遍历n-1个节点（若输入节点是尾节点）。

- 收集元素的过程遍历所有n个节点。

- 总操作次数为O(n)，线性时间高效。

空间复杂度：O(n)，结果数组需要存储所有`n`个节点的值，属于必要空间。

该方法充分利用双链表的双向遍历特性，先定位头部再顺序收集，确保数组元素与链表顺序完全一致，逻辑清晰且适配所有输入场景。