# 题目

给出两个整数向量 v1 和 v2，请你实现一个迭代器，交替返回它们的元素。

实现 ZigzagIterator 类：

ZigzagIterator(List<int> v1, List<int> v2) 用两个向量 v1 和 v2 初始化对象。

boolean hasNext() 如果迭代器还有元素返回 true，否则返回 false。

int next() 返回迭代器的当前元素并将迭代器移动到下一个元素。

示例 1:

输入：v1 = [1,2], v2 = [3,4,5,6]

输出：[1,3,2,4,5,6]

解释：通过重复调用 next 直到 hasNext 返回 false，那么 next 返回的元素的顺序应该是：[1,3,2,4,5,6]。

示例 2:

输入：v1 = [1], v2 = []

输出：[1]

示例 3:

输入：v1 = [], v2 = [1]

输出：[1]

拓展：假如给你 k 个向量呢？你的代码在这种情况下的扩展性又会如何呢?

拓展声明：

“锯齿” 顺序对于 k > 2 的情况定义可能会有些歧义。所以，假如你觉得 “锯齿” 这个表述不妥，也可以认为这是一种“循环”。例如：

输入：v1 = [1,2,3], v2 = [4,5,6,7], v3 = [8,9]

输出：[1,4,8,2,5,9,3,6,7]

# 分析

要解决这个问题，我们需要实现一个迭代器，交替返回两个整数向量的元素。核心思路是通过维护两个向量的当前索引，按“先取v1当前元素、再取v2当前元素”的顺序循环获取，直到所有元素都被遍历完毕。

思路：

1、索引维护：用两个指针（索引）分别记录v1和v2当前待访问的位置，初始值均为0。

2、交替取元素：通过一个“开关”变量（turn）控制当前该取哪个向量的元素：

- turn为true时，优先取v1的当前元素（若v1还有未访问元素）；

- turn为false时，优先取v2的当前元素（若v2还有未访问元素）；

- 每次成功取到元素后，切换`turn`的状态，并将对应向量的索引+1。

3、边界处理：若当前向量已无未访问元素（索引超出向量长度），则直接切换turn，跳过当前向量，尝试取另一个向量的元素。

代码：

class ZigzagIterator {

private:

vector<int> vec1; // 存储第一个向量

vector<int> vec2; // 存储第二个向量

int idx1; // vec1的当前索引

int idx2; // vec2的当前索引

bool turn; // 控制交替：true->取vec1，false->取vec2

public:

// 构造函数：初始化向量和索引、交替开关

ZigzagIterator(vector<int>& v1, vector<int>& v2) {

vec1 = v1;

vec2 = v2;

idx1 = 0;

idx2 = 0;

turn = true; // 初始先尝试取vec1的元素

}

// 返回当前元素并移动迭代器

int next() {

// 若当前该取vec1，且vec1还有元素

if (turn && idx1 < vec1.size()) {

int val = vec1[idx1++]; // 取元素并更新索引

turn = false; // 切换到下一轮取vec2

return val;

}

// 若当前该取vec2，且vec2还有元素

else if (!turn && idx2 < vec2.size()) {

int val = vec2[idx2++]; // 取元素并更新索引

turn = true; // 切换到下一轮取vec1

return val;

}

// 若当前向量已无元素，尝试取另一个向量

else {

// 若vec1还有元素，取vec1（此时turn可能为false，但vec2已空）

if (idx1 < vec1.size()) {

return vec1[idx1++];

}

// 否则取vec2（此时vec1已空）

else {

return vec2[idx2++];

}

}

}

// 判断是否还有下一个元素

bool hasNext() {

// 只要任一向量还有未访问元素，就返回true

return idx1 < vec1.size() || idx2 < vec2.size();

}

};

/\*\*

\* Your ZigzagIterator object will be instantiated and called as such:

\* ZigzagIterator i(v1, v2);

\* while (i.hasNext()) cout << i.next();

\*/

代码解释：

1、构造函数：将输入的v1和v2分别存储到成员变量vec1和vec2中，初始化索引idx1和idx2为0，设置turn为true（初始优先取v1）。

2、next()方法：

- 按turn的状态尝试取对应向量的元素，取到后更新索引并切换turn；

- 若当前向量已空（索引超出长度），则直接取另一个非空向量的元素（无需切换turn，因为下一轮仍需优先取非空向量）。

3、hasNext()方法：简单判断两个向量的索引是否均超出长度，只要有一个未超出，就表示还有元素可访问。

拓展思考（k个向量的情况）

若题目拓展为k个向量，上述双向量的思路可通过以下方式扩展，保证良好的扩展性：

1、用队列维护待访问的向量信息：队列中存储“向量索引 + 当前元素索引”的 pair，初始时将所有非空向量的（向量下标，0）加入队列。

2、循环取元素：每次从队列头部取出一个向量信息，返回其当前元素，然后将“该向量的下一个元素索引”重新加入队列（若下一个索引未超出向量长度）。

3、示例：对于v1=[1,2,3]、v2=[4,5,6,7]、v3=[8,9]，队列初始为[(0,0), (1,0), (2,0)]，每次取队头后更新并重新入队，最终顺序为1→4→8→2→5→9→3→6→7。

这种拓展方案的时间复杂度为O(n)（n为所有向量的总元素数），空间复杂度为O(k)（k为向量个数），适合任意k个向量的场景。