LeetCode 281

Description

Given two 1d vectors, implement an iterator to return their elements alternately.

For example, given two 1d vectors:

v1 = [1, 2]
v2 = [3, 4, 5, 6]
By calling next repeatedly until hasNext returns false, the order of elements returned by next should be: [1, 3, 2, 4, 5, 6].

Follow up: What if you are given k 1d vectors? How well can your code be extended to such cases?

Clarification for the follow up question - Update (2015-09-18):
The "Zigzag" order is not clearly defined and is ambiguous for k > 2 cases. If "Zigzag" does not look right to you, replace "Zigzag" with "Cyclic". For example, given the following input:

[1,2,3]
[4,5,6,7]
[8,9]
It should return [1,4,8,2,5,9,3,6,7].

Thought

When we call next, we need to decide which vector to get number. A easy way is to use two pointers for each list, and check if the sum of the two pointers(index) is odd or even, to decide which vector to fetch.

Need to take care of edge cases if one list ended before the other.

Solution

```
public class ZigzagIterator {
  private List<Integer> v1;
  private List<Integer> v2;
  private int c1;
  private int c2;
```

```
    public ZigzagIterator(List <Integer> v1, List <Integer> v2){
      c1 = 0;
      c2 = 0;
      this.v1 = new ArrayList<> (v1); //use arraylist to construct
      this.v2 = new ArrayList<> (v2);
    }

    public int next() {
      //if v1 is finished, get v2
      if (c1 == v1.size()) {
        return getNextV2();
      }
      //if v2 is finished, get v1
      if (c2 == v2.size()) {
        return getNextV1();
      }
      //get v1 or v2 based on odd or even of the indexs
      if ((c1 + c2) % 2 == 0) {
        return getNextV1();
      } else {
        return getNextV2();
      }
    }

    public boolean hasNext(){
      return c1 + c2 < v1.size() + v2.size();
    }

    private int getNextV1(){
      c1 += 1;
      return v1.get(c1 - 1);
    }

    private int getNextV2(){
      c2 += 1;
      return v2.get(c2 - 1);
    }
  }
}
```

Takeaways

- Consider edge cases - one list is empty or ended
- Pointers value are added zigzagly, often useful to check odd or even