Zigzag Iterator

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]`.

Solution 1

```
class ZigzagIterator {
public:
    ZigzagIterator(vector<int>& v1, vector<int>& v2) {
        if (v1.size() != 0)
            Q.push(make_pair(v1.begin(), v1.end()));
        if (v2.size() != 0)
            Q.push(make_pair(v2.begin(), v2.end()));
    }

    int next() {
        vector<int>::iterator it = Q.front().first;
        vector<int>::iterator endIt = Q.front().second;
        Q.pop();
        if (it + 1 != endIt)
            Q.push(make_pair(it+1, endIt));
        return *it;
    }

    bool hasNext() {
        return !Q.empty();
    }
private:
    queue<pair<vector<int>::iterator, vector<int>::iterator>> Q;
};
```

somehow similar to BFS.

written by lightmark original link here

Solution 2

Two iterators, one for each list. Switching them *before* reading the next number instead of afterwards saves a bit of code, I think.

```
public class ZigzagIterator {

    private Iterator<Integer> i, j, tmp;

    public ZigzagIterator(List<Integer> v1, List<Integer> v2) {
        i = v2.iterator();
        j = v1.iterator();
    }

    public int next() {
        if (j.hasNext()) { tmp = j; j = i; i = tmp; }
        return i.next();
    }

    public boolean hasNext() {
        return i.hasNext() || j.hasNext();
    }
}
```

written by StefanPochmann original link here

Solution 3

Uses a linkedlist to store the iterators in different vectors. Every time we call next(), we pop an element from the list, and re-add it to the end to cycle through the lists.

```
public class ZigzagIterator {
    LinkedList<Iterator> list;
    public ZigzagIterator(List<Integer> v1, List<Integer> v2) {
        list = new LinkedList<Iterator>();
        if(!v1.isEmpty()) list.add(v1.iterator());
        if(!v2.isEmpty()) list.add(v2.iterator());
    }

    public int next() {
        Iterator poll = list.remove();
        int result = (Integer)poll.next();
        if(poll.hasNext()) list.add(poll);
        return result;
    }

    public boolean hasNext() {
        return !list.isEmpty();
    }
}
```

written by kevinhsu original link here