一种消息接收并打印的结构设计

【题目】

消息流吐出 2，一种结构接收而不打印 2，因为 1 还没出现。
消息流吐出 1，一种结构接收 1，并且打印：1，2。
消息流吐出 4，一种结构接收而不打印 4，因为 3 还没出现。
消息流吐出 5，一种结构接收而不打印 5，因为 3 还没出现。
消息流吐出 7，一种结构接收而不打印 7，因为 3 还没出现。
消息流吐出 3，一种结构接收 3，并且打印：3，4，5。
消息流吐出 9，一种结构接收而不打印 9，因为 6 还没出现。
消息流吐出 8，一种结构接收而不打印 8，因为 6 还没出现。
消息流吐出 6，一种结构接收 6，并且打印：6，7，8，9。

已知一个消息流会不断地吐出整数 1~N，但不一定按照顺序吐出。如果上次打印的数为 i，那么当 i+1 出现时，请打印 i+1 及其之后接收过的并且连续的所有数，直到 1~N 全部接收并打印完，请设计这种接收并打印的结构。

【要求】

消息流最终会吐出全部的 1~N，当然最终也会打印完所有的 1~N，要求接收和打印 1~N 的整个过程，时间复杂度为 $O(N)$。

【难度】

尉 ★★☆☆

【解答】

本题的设计方法有很多，本书提供一种设计实现供读者参考。结构假设叫 MessageBox，先以一个与题目不同的例子来简单说明过程：

1．消息流吐出 2，MessageBox 接收并生成连续区间{2}，此时不打印，因为 1 没出现。

2．消息流吐出 1，MessageBox 接收并生成连续区间{1}，发现可以与{2}连在一起，所以连成整个连续区间{1,2}。此时 1 出现了，所以打印 1，2，打印后删除连续区间{1,2}。

3．消息流吐出 4，MessageBox 接收并生成连续区间{4}。

4．消息流吐出 5，MessageBox 接收并生成连续区间{5}，发现可以与{4}连在一起，所以连成整个连续区间{4,5}。

5．消息流吐出 7，MessageBox 接收并生成连续区间{7}，此时 MessageBox 中有两个连续区间，分别为{4,5}和{7}。但 3 还没出现，所以不打印。

6．消息流吐出 9，MessageBox 接收并生成连续区间{9}，此时 MessageBox 中有三个连续区间，分别为{4,5}、{7}和{9}。但 3 还没出现，所以不打印。

7．消息流吐出 8，MessageBox 接收并生成连续区间{8}，此时发现{8}的出现可以把{7}和{9}连在一起，所以连成整个连续区间{7,8,9}。此时 MessageBox 中有两个连续区间，分别为{4,5}和{7,8,9}。但 3 还没出现，所以不打印。

8．消息流吐出 6，MessageBox 接收并生成连续区间{6}，此时发现{6}的出现可以把{4,5}和{7,8,9}连在一起，所以连成整个连续区间{4,5,6,7,8,9}。但 3 还没出现，所以不打印。

9．消息流吐出 3，MessageBox 接收并生成连续区间{3}，发现可以与{4,5,6,7,8,9}连在一起，所以连成整个连续区间{3,4,5,6,7,8,9}。此时 3 出现了，所以打印 3,4,5,6,7,8,9。打印后删除连续区间{3,4,5,6,7,8,9}，整个过程结束。

分析如上过程可以知道，如果达到整个过程，其时间复杂度为 $O(N)$，我们需要设计好的连续区间结构，并且在一个数出现时，还要方便地将这个数上下有关的连续区间连接在一起。下面就介绍 MessageBox 结构的具体设计细节：

1．当接收一个数 num 时，先根据 num 生成一个单链表节点的实例，单链表结构记为 Node，具体请参看如下的 Node 类。

```
public class Node {
        public int num;
        public Node next;

        public Node(int num) {
                this.num = num;
        }
}
```

2．连续结构就是一个单链表结构，但这是不够的，为了可以快速合并，MessageBox 中还有三个重要的部分：headMap、tailMap 和 lastPrint。headMap 是一个哈希表，key 为整型，表示一个连续区间开始的数，value 为 Node 类型，表示根据 key 这个数生成的节点，也是连续区间的第一个节点。tailMap 也是一个哈希表，key 为整型，表示一个连续区间结束的数，value 为 Node 类型，表示根据 key 这个数生成的节点，也是连续区间的最后一个

节点。比如连续区间{4,5,6,7,8,9}，假设节点值为 4 的节点记为 start，节点值为 9 的节点记为 end，从 start 到 end 是一条单链表，上面有节点值从 4 到 9 的所有节点，而且在 headMap 中还有记录(4,start)，在 tailMap 中还有记录(9,end)。lastPrint 表示上次打印的是什么数。

3．接收 num 之后，假设根据 num 生成的单链表节点实例为 cur。现在的 num 可以自己成为一个连续区间，即在 headMap 中加上记录(num,cur)，在 tailMap 中也加上记录(num,cur)。然后依次进行如下处理：

1）在 tailMap 中查询是否有 key==num-1 的记录。如果有，说明存在一个连续区间以 num-1 结尾，记为连续区间 *A*，那么 *A* 可以和 num 自己的连续区间合并。假设 *A* 最后的数 num-1 对应的节点为 end，那么令 end.next=cur，表示 *A* 的单向链表在最后加了一个节点 cur。然后在 tailMap 中删除记录(num-1,end)，因为以 num-1 结尾的连续区间已经不存在，大的连续区间是以 num 结尾的。最后在 headMap 中删除记录(num,cur)，因为以 num 开始的连续区间已经不存在，大的连续区间的头是合并前连续区间 *A* 的头。如果没有 key==num-1 的记录，则什么也不用做。

2）在 headMap 中查询是否有 key==num+1 的记录。如果有，说明存在一个连续区间以 num+1 开始，记为连续区间 *B*，那么 *B* 可以和以 num 结尾的连续区间合并。假设 *B* 开始的数 num+1 对应的节点为 start，那么令 cur.next=start，表示以 num 结尾的连续区间的链表合和 *B* 的链表合并。然后在 headMap 中删除记录(num+1,start)，因为以 num+1 开始的连续区间已经不存在。最后在 tailMap 中删除记录(num,cur)，因为以 num 结束的连续区间也已经不存在。如果没有 key==num+1 的记录，则什么也不用做。

整个步骤 3 就是做一件事情，看 num 上下的连续区域有没有因为自己的出现可以进行合并，能合并的全部都合并在一起。

4．加入 num 之后，能不能打印。如果能打印，把打印的连续区域一律删除。

如上过程中，连续区域的合并全是 $O(1)$的时间复杂度，因为都是简单的哈希表查询操作或者是把某个节点的 next 指针赋值而已。整体过程的时间复杂度为 $O(N)$，MessageBox 结构的具体实现请参看如下代码中的 MessageBox 类。

```
public class MessageBox {
        private HashMap<Integer, Node> headMap;
        private HashMap<Integer, Node> tailMap;
        private int lastPrint;

        public MessageBox() {
                headMap = new HashMap<Integer, Node>();
                tailMap = new HashMap<Integer, Node>();
                lastPrint = 0;
```

```
        }

        public void receive(int num) {
                if (num < 1) {
                        return;
                }
                Node cur = new Node(num);
                headMap.put(num, cur);
                tailMap.put(num, cur);
                if (tailMap.containsKey(num - 1)) {
                        tailMap.get(num - 1).next = cur;
                        tailMap.remove(num - 1);
                        headMap.remove(num);
                }
                if (headMap.containsKey(num + 1)) {
                        cur.next = headMap.get(num + 1);
                        tailMap.remove(num);
                        headMap.remove(num + 1);
                }
                if (headMap.containsKey(lastPrint + 1)) {
                        print();
                }
        }

        private void print() {
                Node node = headMap.get(++lastPrint);
                headMap.remove(lastPrint);
                while (node != null) {
                        System.out.print(node.num + " ");
                        node = node.next;
                        lastPrint++;
                }
                tailMap.remove(--lastPrint);
                System.out.println();
        }

}
```

设计一个没有扩容负担的堆结构

【题目】

堆结构一般是使用固定长度的数组结构来实现的。这样的实现虽然足够经典，但存在扩容的负担，比如不断向堆中增加元素，使得固定数组快耗尽时，就不得不申请一个更大的固定数组，然后把原来数组中的对象复制到新的数组里完成堆的扩容，所以，如果扩容时堆中的元素个数为 N，那么扩容行为的时间复杂度为 $O(N)$。请设计一种没有扩容负担的