```
                } else {
                        return this.setAll.getValue();
                }
            } else {
                return null;
            }
        }
    }
```

最大的 leftMax 与 rightMax 之差的绝对值

【题目】

给定一个长度为 N（N>1）的整型数组 arr，可以划分成左右两个部分，左部分为 arr[0..K]，右部分为 arr[K+1..N-1]，K 可以取值的范围是[0,N-2]。求这么多划分方案中，左部分中的最大值减去右部分最大值的绝对值中，最大是多少？

例如：[2,7,3,1,1]，当左部分为[2,7]，右部分为[3,1,1]时，左部分中的最大值减去右部分最大值的绝对值为 4。当左部分为[2,7,3]，右部分为[1,1]时，左部分中的最大值减去右部分最大值的绝对值为 6。还有很多划分方案，但最终返回 6。

【难度】

校 ★★★☆

【解答】

方法一：时间复杂度为 $O(N^2)$，额外空间复杂度为 $O(1)$。这是最笨的方法，在数组的每个位置 i 都做一次这种划分，找到 arr[0..i]的最大值 maxLeft，找到 arr[i+1..N-1]的最大值 maxRight，然后计算两个值相减的绝对值。每次划分都这样求一次，自然可以得到最大的相减的绝对值。具体请参看如下代码中的 maxABS1 方法。

```
public int maxABS1(int[] arr) {
        int res = Integer.MIN_VALUE;
        int maxLeft = 0;
        int maxRight = 0;
        for (int i = 0; i != arr.length - 1; i++) {
                maxLeft = Integer.MIN_VALUE;
                for (int j = 0; j != i + 1; j++) {
                        maxLeft = Math.max(arr[j], maxLeft);
                }
                maxRight = Integer.MIN_VALUE;
```

```
                for (int j = i + 1; j != arr.length; j++) {
                        maxRight = Math.max(arr[j], maxRight);
                }
                res = Math.max(Math.abs(maxLeft - maxRight), res);
        }
        return res;
}
```

方法二：时间复杂度为 $O(N)$，额外空间复杂度为 $O(N)$。使用预处理数组的方法，先从左到右遍历一次生成 lArr，lArr[i]表示 arr[0..i]中的最大值。再从右到左遍历一次生成 rArr，rArr[i]表示 arr[i..N-1]中的最大值。最后一次遍历看哪种划分的情况下可以得到两部分最大的相减的绝对值，因为预处理数组已经保存了所有划分的 max 值，所以过程得到了加速。具体请参看如下代码中的 maxABS2 方法。

```
public int maxABS2(int[] arr) {
        int[] lArr = new int[arr.length];
        int[] rArr = new int[arr.length];
        lArr[0] = arr[0];
        rArr[arr.length - 1] = arr[arr.length - 1];
        for (int i = 1; i < arr.length; i++) {
                lArr[i] = Math.max(lArr[i - 1], arr[i]);
        }
        for (int i = arr.length - 2; i > -1; i--) {
                rArr[i] = Math.max(rArr[i + 1], arr[i]);
        }
        int max = 0;
        for (int i = 0; i < arr.length - 1; i++) {
                max = Math.max(max, Math.abs(lArr[i] - rArr[i + 1]));
        }
        return max;
}
```

方法三：最优解，时间复杂度为 $O(N)$，额外空间复杂度为 $O(1)$。先求整个 arr 的最大值 max，因为 max 是全局最大值，所以不管怎么划分，max 要么会成为左部分的最大值，要么会成为右部分的最大值。如果 max 作为左部分的最大值，接下来只要让右部分的最大值尽量小就可以。右部分的最大值怎么尽量小呢？右部分只含有 arr[N-1]的时候就是尽量小的时候。同理，如果 max 作为右部分的最大值，只要让左部分的最大值尽量小就可以，左部分只含有 arr[0]的时候就是尽量小的时候。所以整个求解过程会变得异常简单。具体请参看如下代码中的 maxABS3 方法。

```
public int maxABS3(int[] arr) {
        int max = Integer.MIN_VALUE;
        for (int i = 0; i < arr.length; i++) {
                max = Math.max(arr[i], max);
```

```
            }
            return max - Math.min(arr[0], arr[arr.length - 1]);
        }
```

设计可以变更的缓存结构

【题目】

设计一种缓存结构，该结构在构造时确定大小，假设大小为 K，并有两个功能：

- set(key,value)：将记录(key,value)插入该结构。
- get(key)：返回 key 对应的 value 值。

【要求】

1．set 和 get 方法的时间复杂度为 $O(1)$。

2．某个 key 的 set 或 get 操作一旦发生，认为这个 key 的记录成了最经常使用的。

3．当缓存的大小超过 K 时，移除最不经常使用的记录，即 set 或 get 最久远的。

【举例】

假设缓存结构的实例是 cache，大小为 3，并依次发生如下行为：

1．cache.set("A",1)。最经常使用的记录为("A",1)。

2．cache.set("B",2)。最经常使用的记录为("B",2)，("A",1)变为最不经常的。

3．cache.set("C",3)。最经常使用的记录为("C",2)，("A",1)还是最不经常的。

4．cache.get("A")。最经常使用的记录为("A",1)，("B",2)变为最不经常的。

5．cache.set("D",4)。大小超过了 3，所以移除此时最不经常使用的记录("B",2)，加入记录("D",4)，并且为最经常使用的记录，然后("C",2)变为最不经常使用的记录。

【难度】

尉 ★★☆☆

【解答】

这种缓存结构可以由双端队列与哈希表相结合的方式实现。首先实现一个基本的双向链表节点的结构，请参看如下代码中的 Node 类。