题目地址（278. 第一个错误的版本）

https://leetcode-cn.com/problems/first-bad-version

入选理由

1. 仍然是一个简单二分。 出这个题目的是为了让大家抽离问题本质，然后使用合适算法。简单来说就是换个皮大家也要会

标签

- 二分

难度

- 简单

题目描述

你是产品经理，目前正在带领一个团队开发新的产品。不幸的是，你的产品的最新版本没有通过质量检测。由于每个版本都是基于之前的版本开发的，所以错误的版

假设你有 n 个版本 [1, 2, ..., n]，你想找出导致之后所有版本出错的第一个错误的版本。

你可以通过调用 bool isBadVersion(version) 接口来判断版本号 version 是否在单元测试中出错。实现一个函数来查找第一个错误的版本。你应该尽量减

示例:

给定 n = 5，并且 version = 4 是第一个错误的版本。

```
调用 isBadVersion(3) -> false
调用 isBadVersion(5) -> true
调用 isBadVersion(4) -> true

所以，4 是第一个错误的版本。
```

前置知识

- 二分法

思路

典型的**二分寻找最左边的满足条件的值**， 具体看我的讲义。 一句话概括就是：**寻找最左边和寻找指定值的差别就是碰到等于号的处理情况。** 如果是寻找最左边那么碰到等于继续收缩右边界（寻找最右边就是收缩左边界）， 查找指定值则是直接返回。

我们直接套模板即可。

代码

Python3:

```
class Solution:
    def firstBadVersion(self, n):
        l, r = 1, n
        while l <= r:
            mid = (l + r) // 2
            if isBadVersion(mid):
                # 收缩
                r = mid - 1
            else:
                l = mid + 1
        return l
```

CPP:

```
// The API isBadVersion is defined for you.
// bool isBadVersion(int version);

class Solution {
public:
    int firstBadVersion(int n) {
    int l=1,r=n;
    while(l<=r){
        int mid=l+(r-l)/2;
        if(isBadVersion(mid)){
            r=mid-1;
        }
        else
            l=mid+1;
    }

    return l;
}
};
```

Java:

```java
public class Solution extends VersionControl {
    public int firstBadVersion(int n) {
        int l = 1;
        int r = n;
        while (l <= r) {
            int mid = l + ((r - l) >> 1);
            if (isBadVersion(mid)) {
                r = mid -1;
            } else {
                l = mid + 1;
            }
        }

        return l;
    }
}
```

JS:

```javascript
var solution = function (isBadVersion) {
  /**
   * @param {integer} n Total versions
   * @return {integer} The first bad version
   */
  return function (n) {
    let left = 1;
    let right = n;

    while (left <= right) {
      let mid = Math.floor((right + left) / 2);

      if (isBadVersion(mid)) {
        right = mid - 1;
      } else {
        left = mid + 1;
      }
    }

    return left;
  };
};
```

复杂度分析

- 时间复杂度：$O(logN)$
- 空间复杂度：$O(1)$

上一页 下一页