链滴

LeetCode #367 有效的完全平方数

作者：matthewhan
原文链接：https://ld246.com/article/1589180513777
来源网站：链滴

Problem Description

给定一个正整数 num，编写一个函数，如果 num 是一个完全平方数，则返回 True，否则返回 False。

note

不要使用任何内置的库函数，如：sqrt。

e.g.

- 示例1：
  - 输入：16
  - 输出：True
- 示例2：
  - 输入：14
  - 输出：False

Solution

暴力法

可以无限优化的方法，其优化的核心就在于循环的起始和终止，效率很低不考虑。

根区间&二分法

原文链接：LeetCode #367 有效的完全平方数

1. Java里int的最大值是0x7fffffff，也就是1 ³²-1（2147483647），大约21亿。最值的平方根约等于46340。那我不是根据整数的位数来得到一个计算的区间，这样不是能有效缩小循的次数了吗?

2. 比如1位数和2位数[10, 99]，平方根一定是落在[1, 10)，比如3位数[100, 999]，平方根一定是落在[0, 32)，比如4位数[1000, 9999]，平方根一定是落在[31, 100)；

3. 当这个数很大的话（最大为2147483647），相应的区间也会变大，但是区间最大的反而是9位数因为10位数最大只到2147483647；

4. 我们根据整数的长度取得平方根的区间后可以根据二分查找法，将平均时间复杂度降低。

```
class Solution {
    public static boolean isPerfectSquare(int num) {
        int[] se = interval(String.valueOf(num).length());
        int left = se[0];
        int right = se[1];
        while (true) {
            int mid = left + (right - left) / 2;
            if (mid * mid < num) {
                if (right - left == 1) {
                    return false;
                } else {
                    left = mid;
                }
            } else if (mid * mid == num) {
                return true;
            } else {
                if (right - left == 1) {
                    return false;
                } else {
                    right = mid;
                }
            }
        }
    }

    public static int[] interval(int len) {
        int start;
        int end;
        switch (len) {
            case 3:
                start = 10;
                end = 32;
                break;
            case 4:
                start = 31;
                end = 100;
                break;
            case 5:
                start = 100;
                end = 317;
                break;
            case 6:
                start = 316;
                end = 1000;
```

```
                break;
            case 7:
                start = 1000;
                end = 3163;
                break;
            case 8:
                start = 3162;
                end = 10000;
                break;
            case 9:
                start = 10000;
                end = 31623;
                break;
            case 10:
                start = 31622;
                end = 46341;
                break;
            default:
                start = 1;
                end = 10;
        }
        return new int[]{start, end};
    }
}
```

二分法有一个很需要注意的点：就是求中间值的计算公式。一般最好不要直接这样做：(left + right) / 2，因为很有可能在 left + right的过程中就溢出了，算出来是负数的。改成long类型或者这样算：left + (right - left) / 2。

两个点优化之后，即使一个很大的数字也不需要经过几次的查找便可知道是否是完全平方数。以上算的时间复杂度为O(1) + O(log₂n) = O(log₂n)。