0204. 计数质数

ITCharge 大约 3 分钟

- 标签：数组、数学、枚举、数论
- 难度：中等

题目链接

- 0204. 计数质数 - 力扣

题目大意

描述：给定 一个非负整数 n。

要求：统计小于 n 的质数数量。

说明：

- $0 \leq n \leq 5 * 10^6$。

示例：

- 示例 1：

```py
输入 n = 10
输出 4
解释 小于 10 的质数一共有 4 个，它们是 2，3，5，7。
```

- 示例 2：

```py
输入：n = 1
输出：0
```

解题思路

思路 1：枚举算法（超时）

对于小于 n 的每一个数 x，我们可以枚举区间 $[2, x-1]$ 上的数是否是 x 的因数，即是否存在能被 x 整数的数。如果存在，则该数 x 不是质数。如果不存在，则该数 x 是质数。

这样我们就可以通过枚举 $[2, n-1]$ 上的所有数 x，并判断 x 是否为质数。

在遍历枚举的同时，我们维护一个用于统计小于 n 的质数数量的变量 `cnt`。如果符合要求，则将计数 `cnt` 加 1。最终返回该数目作为答案。

考虑到如果 i 是 x 的因数，则 $\frac{x}{i}$ 也必然是 x 的因数，则我们只需要检验这两个因数中的较小数即可。而较小数一定会落在 $[2, \sqrt{x}]$ 上。因此我们在检验 x 是否为质数时，只需要枚举 $[2, \sqrt{x}]$ 中的所有数即可。

利用枚举算法单次检查单个数的时间复杂度为 $O(\sqrt{n})$，检查 n 个数的整体时间复杂度为 $O(n\sqrt{n})$。

思路 1：代码

py

```py
class Solution:
    def isPrime(self, x):
        for i in range(2, int(pow(x, 0.5)) + 1):
            if x % i == 0:
                return False
        return True

    def countPrimes(self, n: int) -> int:
        cnt = 0
        for x in range(2, n):
            if self.isPrime(x):
                cnt += 1
        return cnt
```

思路 1：复杂度分析

- **时间复杂度**：$O(n \times \sqrt{n})$。
- **空间复杂度**：$O(1)$。

思路 2：埃氏筛法

可以用「埃氏筛」进行求解。这种方法是由古希腊数学家埃拉托斯尼斯提出的，具体步骤如下：

- 使用长度为 n 的数组 `is_prime` 来判断一个数是否是质数。如果 `is_prime[i] == True`，则表示 i 是质数，如果 `is_prime[i] == False`，则表示 i 不是质数。并使用变量 `count` 标记质数个数。
- 然后从 $[2, n-1]$ 的第一个质数（即数字 2）开始，令 `count` 加 1，并将该质数在 $[2, n-1]$ 范围内所有倍数（即 4、6、8、...）都标记为非质数。
- 然后根据数组 `is_prime` 中的信息，找到下一个没有标记为非质数的质数（即数字 3），令 `count` 加 1，然后将该质数在 $[2, n-1]$ 范围内的所有倍数（即 6、9、12、...）都标记为非质数。
- 以此类推，直到所有小于或等于 n 的质数和质数的倍数都标记完毕时，输出 `count`。

优化：对于一个质数 x，我们可以直接从 $x \times x$ 开始标记，这是因为 $2 \times x$、$3 \times x$、... 这些数已经在 x 之前就被其他数的倍数标记过了，例如 2 的所有倍数、3 的所有倍数等等。

思路 2：代码

```py
class Solution:
    def countPrimes(self, n: int) -> int:
        is_prime = [True] * n
        count = 0
        for i in range(2, n):
            if is_prime[i]:
                count += 1
                for j in range(i * i, n, i):
                    is_prime[j] = False
        return count
```

思路 2：复杂度分析

- **时间复杂度**：$O(n \times \log_2 log_2 n)$。
- **空间复杂度**：$O(n)$。