Optimization of Prime Number Checking Algorithm

Bobur Yusupov
Bachelor student in Computer Science at Eötvös Loránd University

March 16, 2025

Abstract

Prime number detection is a fundamental problem in number theory and computer science. This paper explores optimizations for naive primality test, reducing its computational complexity. We implement and analyze an improved algorithm using Python, leveraging mathematical properties to enhance efficiency.

1 Introduction

Optimization of algorithms is a critical component of computational efficiency. This paper discusses a basic prime number checking algorithm and explores various techniques for improving its performance. The initial algorithm, generated by ChatGPT, is as follows:

```python
def is_prime(n):
    if n <= 1:
        return False

    for i in range(2, n):
        if n % i == 0:
            return False

    return True
```

Listing 1: Naive Prime Number Checking Algorithm

2 Algorithm Description

The given algorithm determines whether a number n is prime through the following steps:

- If n is less than or equal to 1, return `False`.
- Iterate from 2 to *n-1*, checking divisibility using the modulo operator.
- If n is divisible by any i, return `False` immediately; otherwise, return `True`.

This approach has a worst-case time complexity of $O(n)$.

3 Optimization Overview

The primary inefficiency in the naive approach lies in its $O(n)$ complexity. We explored several optimization techniques and implemented the most effective ones:

- **Checking divisibility by 2 and 3**: First we check whether the number is even or not. If it is an even number we can immediately return **False**. This reduces the number of iterations, as every even number (except 2) can we skipped.

- **Reducing the range:** The concept of **Trial Division Method** is checking divisibility of n from 2 to $\sqrt{n} + 1$. If no divisor is found, we can conclude that n is prime.
- **Skipping even numbers:** Except for 2, all prime numbers are odd. We can skip even numbers beyond 2.

4 Optimized Algorithm

Applying these optimizations, we derive the following improved algorithm:

```python
import math

def is_prime_optimized(n):
    if n <= 1:
        return False

    if n <= 3:
        return True

    if n % 2 == 0 or n % 3 == 0:
        return False

    limit = int(math.sqrt(n)) + 1

    for i in range(5, limit, 2):
        if n % i == 0:
            return False

    return True
```

Listing 2: Optimized Prime Number Checking Algorithm

This optimization reduces the number of iterations significantly and improves the efficiency of prime number detection. The time complexity of the optimized algorithm is $O(\sqrt{n}/2)$

5 Conclusion

Optimizing algorithms is crucial in computer science. It leads to decrease usage of resources like memory, storage and time. Our algorithm might be simple and might not take too much time for smaller numbers, however for large numbers it requires use to wait.

Algorithm optimization is crucial in computer science, as it reduces resource usage, including memory, storage, and computation time. While naive approach might be sufficient for small numbers, however larger inputs require more resource usage. We can use more efficient methods to tackle with this issue. The optimized algorithm significantly reduces resource usage and execution time, making it more suitable for large-scale computations.