REPORT

Prime Number Generator and Checker

Name : Siya Kapoor

Date: March 11, 2025

Roll No.: 202401100300248

CSE-AI - D

INTRODUCTION

One very important aspect in number theory is study of prime numbers. They are classified as natural numbers above one that can only be divided by themselves and one. Along with 1, prime numbers make up the building blocks of natural numbers. They have a great importance in the field of cryptology, computer security, and optimization algorithms. As an example, RSA encryption makes use of large prime numbers to transmit information securely.

Determining if a number is prime and generating a list of all the prime numbers are difficult computational problems. In principle, a prime number can be established with trial division, but that becomes difficult for larger numbers. This project deals with the optimization of prime number detection that uses the 6k -/+ 1 technique that reduces steps to be completed and Eratosthenes' Sieve which is the sieve used to find all prime numbers up to a specified integer.

The current report includes the problem statement, the approach taken for computational optimization, the implementation, and results from the program.

METHODOLOGY

In order to verify whether an input number is prime, and also generate a list of prime numbers, two distinct methods are employed:

1. Prime Number Checker (is_prime(n)):

- This function verifies if the given number n is prime through an optimized approach:
-
- Eliminating Small Cases: If n is less than 2, then n is not prime, 2 and 3 are considered prime.
-
- Divisibility Check: Any number that can be divided by either 2 or 3 should be eliminated.
-
- $6k \pm 1$ Optimization: Unlike checking all numbers till square root of n, the function checks all potential factors that would fit the forms $6k + 1$ and $6k – 1$. This is because all prime numbers beyond 3 can be formulated that way. This greatly reduces the iterative process and enhances speed.
-
- The time complexity at its worst depends on the square root of n, $O(\sqrt{n})$. However, this makes it significantly faster than rudimentary trial subdivision.

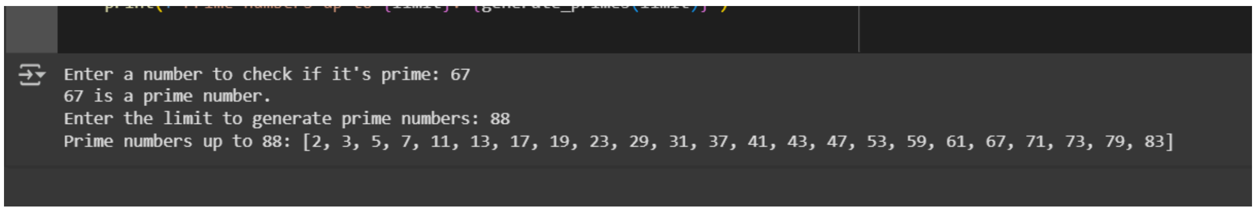
2. Prime Number Generator (Generate_primes(limit)):

- This function efficiently generates every prime number up to a certain limit using the Sieve of Eratosthenes, one of the fastest methods for determining prime numbers.
- Initialization: A boolean list is created, with each index representing a number, assuming that all of the numbers are prime.
- Marking Non-Primes: By iterating from 2 to $\sqrt{\text{limit}}$, multiples of each prime are identified as non-prime.
- Result Extraction: The last prime list in the boolean list is extracted.
- Time Complexity: The algorithm runs in $O(n \log \log n)$, which is significantly faster than evaluating each number independently.

CODE

```
def is_prime(n):
    """
    Check if a number is prime using optimized divisibility rules.

    :param n: The number to check
    :return: True if n is prime, False otherwise
    """
    if n < 2:
        return False
    if n in (2, 3):  # 2 and 3 are prime
        return True
    if n % 2 == 0 or n % 3 == 0:  # Eliminate even numbers & multiples of 3
        return False

    # Check factors from 5 to sqrt(n), skipping multiples of 2 and 3
    for i in range(5, int(n ** 0.5) + 1, 6):
        if n % i == 0 or n % (i + 2) == 0:
            return False

    return True


def generate_primes(limit):
    """
    Generate a list of prime numbers up to a given limit using the Sieve of
Eratosthenes.

    :param limit: The upper limit (inclusive)
    :return: A list of prime numbers up to the limit
    """
    if limit < 2:
        return []

    primes = [True] * (limit + 1)
    primes[0] = primes[1] = False  # 0 and 1 are not prime

    for num in range(2, int(limit ** 0.5) + 1):
        if primes[num]:  # If num is prime, mark its multiples as non-prime
            for multiple in range(num * num, limit + 1, num):
                primes[multiple] = False

    return [i for i, is_prime in enumerate(primes) if is_prime]


# Example usage
if __name__ == "__main__":
    # Check if a number is prime
    num = int(input("Enter a number to check if it's prime: "))
    print(f"{num} is {'a prime' if is_prime(num) else 'not a prime'} number.")

    # Generate prime numbers up to a limit
    limit = int(input("Enter the limit to generate prime numbers: "))
    print(f"Prime numbers up to {limit}: {generate_primes(limit)}")
```

OUTPUT



```
Enter a number to check if it's prime: 67
67 is a prime number.
Enter the limit to generate prime numbers: 88
Prime numbers up to 88: [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83]
```

CONCLUSION

This project successfully implements a prime number generator and checker using optimized techniques. The 6k ± 1 method improves efficiency in checking prime numbers, while the Sieve of Eratosthenes ensures fast generation of prime numbers. These methods significantly reduce computational complexity, making them ideal for handling larger numbers efficiently. Prime number algorithms play a vital role in cryptography, security, and mathematical applications, proving the importance of their optimization.

REFERENCES

1. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). The MIT Press.

2. Python Documentation - https://docs.python.org

3. Knuth, D. E. (1998). *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*. Addison-Wesley.

4. Rosser, J. B., & Schoenfeld, L. (1962). Approximate formulas for some functions of prime numbers. *Illinois Journal of Mathematics, 6*(1), 64-94.