Discrete Structures

LAB II

Number Theory Algorithms

Team members

جون وليم فوزي فهمي 22010701

أحمد راجي حسين 22011690

Problem Statement:

1 Prime Number Checker

Implement a function that determines whether a given positive integer is a prime number or not using Sieve of Eratosthenes

2 Prime Factorization

Create a function that computes the prime factors of a given integer.

3 GCD and LCM Computation

Implement functions to calculate the GCD and LCM of two positive integers.

a) Using the Euclidean algorithm for GCD computation and the relationship between GCD and LCM to find the LCM.

b) Using prime factorization.

Data Structures Used:

1. `boolean[] isComposite`

- Used in: `primeChecker(int n)`
- Purpose:
  This array is used to mark whether a number is composite or not.

2. `Map<Integer, Integer> factors`

1. Used in: **`primeFactors(int n)`**, **`gcdPrime(int a, int b)`**, and **`lcmPrime(int a, int b)`**
2. Purpose:
   A **`Map`** (specifically a **`HashMap`**) is used to store the prime factors of a number and their corresponding exponents.

3. `Set<Integer>` (Implicit)

- Used in: `gcdPrime(int a, int b)` and `lcmPrime(int a, int b)`
- Purpose:
  Although not explicitly defined, the method uses the **keys** of the `factorsA` and `factorsB` maps (which are essentially sets of prime factors) to find the common prime factors (for GCD) and all prime factors (for LCM).

Implementation Details:

Solution Approach

The solution consists of four primary methods, each solving a different aspect of the problem:

1. **Prime Checking (`primeChecker(int n)`):**
    - This method checks whether a given integer `n` is prime.
    - If `n` is less than 2, it returns `false` immediately.
    - The algorithm uses a **Sieve of Eratosthenes** approach, marking non-prime numbers in a boolean array, and then checks whether the number is prime or not.
2. **Prime Factorization (`primeFactors(int n)`):**
    - This method returns a map of prime factors for the number `n`, where the key is the prime number, and the value is its frequency (exponent).
    - The algorithm first handles the factor of `2` (the only even prime) and then checks for all odd numbers starting from `3` up to the square root of `n`.
3. **GCD using Prime Factorization (`gcdPrime(int a, int b)`):**
    - This method calculates the GCD of two numbers by factoring both `a` and `b` into primes, then multiplying the common prime factors with the minimum exponent found in both numbers' prime factorizations.
4. **LCM using Prime Factorization (`lcmPrime(int a, int b)`):**
    - This method calculates the LCM of two numbers using their prime factorizations. It multiplies each prime factor raised to the highest power that appears in either of the numbers' factorizations.
5. **GCD using Euclidean Algorithm (`gcd(int a, int b)`):**
    - This method computes the GCD of two numbers using the Euclidean algorithm, which repeatedly replaces the larger number by the remainder when the larger number is divided by the smaller one.
6. **LCM using GCD (`lcm(int a, int b)`):**
    - This method calculates the LCM of two numbers using the formula: LCM(a,b)= | a×b | GCD(a,b)\LCM(a, b) = frac|a \times b|GCD(a, b)LCM(a,b)=GCD(a,b) | a×b |
    - It uses the previously defined `gcd()` method.

Test Cases

```
int x = 72;
int y = 120;

System.out.println("GCD = " + gcd(x,y));
System.out.println("LCM = " + lcm(x,y));
System.out.println("Prime Factors = " + primeFactors(x));
System.out.println("Prime Factors = " + primeFactors(y));
System.out.println("GCD Prime = " + gcdPrime(x,y));
System.out.println("LCM Prime = " + lcmPrime(x,y));

System.out.print(primeChecker((7)));
```

```
GCD = 24
LCM = 360
Prime Factors = {2=3, 3=2}
Prime Factors = {2=3, 3=1, 5=1}
GCD Prime = 24
LCM Prime = 360
true
Process finished with exit code 0
```

```
int x = 45;
int y = 60;

System.out.println("GCD = " + gcd(x,y));
System.out.println("LCM = " + lcm(x,y));
System.out.println("Prime Factors = " + primeFactors(x));
System.out.println("Prime Factors = " + primeFactors(y));
System.out.println("GCD Prime = " + gcdPrime(x,y));
System.out.println("LCM Prime = " + lcmPrime(x,y));

System.out.print(primeChecker((7)));
```

```
GCD = 15
LCM = 180
Prime Factors = {3=2, 5=1}
Prime Factors = {2=2, 3=1, 5=1}
GCD Prime = 15
LCM Prime = 180
true
Process finished with exit code 0
```

```
int x = 121;
int y = 242;

System.out.println("GCD = " + gcd(x,y));
System.out.println("LCM = " + lcm(x,y));
System.out.println("Prime Factors = " + primeFactors(x));
System.out.println("Prime Factors = " + primeFactors(y));
System.out.println("GCD Prime = " + gcdPrime(x,y));
System.out.println("LCM Prime = " + lcmPrime(x,y));

System.out.print(primeChecker((24)));
```

```
GCD = 121
LCM = 242
Prime Factors = {11=2}
Prime Factors = {2=1, 11=2}
GCD Prime = 121
LCM Prime = 242
false
Process finished with exit code 0
```