Report on Primality Test

Sabal Subedi
Master's in Computer Science
Idaho State University
Pocatello, ID, 83209 USA
sabalsubedi@isu.edu

January 23, 2024

Abstract

This project report summarizes the use of the Fermat test and Miller-Rabin test to check whether a positive integer is a prime or composite number. Furthermore, I will discuss the success probability of both the Fermat and Miller-Rabin tests.

1 Introduction

Prime numbers are numbers that have only two factors, 1 and itself, and are greater than 1; i.e. 1 is not a prime number. And all those numbers that are not prime; i.e.have more than two factors are called composite numbers.
There are different methods to determine the primality of an integer and Fermat's little theorem and Miller-Rabin primality test are two of them. Though the Fermat test is an easier and quicker method, it has a huge drawback: indicates Carmichael numbers as prime numbers. Carmichael number is an odd composite number that satisfies Fermat's little theorem. Thus, we need the Miller-Rabin test to further validate the integers that satisfy Fermat's little theorem. Then, I will compute the time and space complexity of both tests as well as their success probability.

2 Pseudocode and Asymptotic Analysis

2.1 Pseudocode for Modular exponential

Algorithm 1 Modular Exponentiation

```
function MODULAREXPONENTIATION(x, y, n)
    if y = 0 then
        return 1
    else
        z ← MODULAREXPONENTIATION(x, ⌊y/2⌋, n)
        if y is even then
            return (z × z) mod n
        else
            return (x × z × z) mod n
```

Time complexity and space complexity
Given, N: a positive integer with n-bits
Space Complexity:

$$S(n) = O(1) + O(N) + O(N) = 2 * O(N) \approx O(N) \approx O(\log n) \tag{1}$$

Time Complexity:

$$T(n) = O(1) + O(N) + O(N) = 2 * O(N) \approx O(N) \approx O(\log n) \tag{2}$$

2.2 Pseudocode for Fermat's little theorem

Algorithm 2 Fermat Primality Test

```
function FERMATTEST(n, k)
    for i ← 1 to k do
        Choose a random base a such that 1 < a < n
        result ← MODULAREXPONENTIATION(a, n − 1, n)
        if result = 1 then
            return Prime
    return Composite
```

Time complexity and space complexity
Given, N: a positive integer with n-bits
k: a positive interger to iterate the test k times
Space Complexity:

$$S(n) = O(1) + O(N) + O(1) = O(N) \approx O(\log n) \tag{3}$$

Time Complexity:

$$T(n) = O(1) + k * ((\log n) + O(1)) \approx O(k * (\log n)) \tag{4}$$

Computing success probability:

The probability of failure: $P_f \;=\; 1/2^k$
The probability of success $P_s \;=\; 1 - 1/2^k$

2.3 Pseudocode for Miller-Rabin Primality Test

Algorithm 3 Recursive Miller-Rabin Primality Test

```
function MillerRabin(n, k)
    if n ≤ 1 then
        return Composite
    if n = 2 or n = 3 then
        return Prime
    epoch, root, flag ← 0, n-1, False
    while ⌊root/2⌋ = 0 do
        epoch ← epoch + 1
        root ← ⌊root/2⌋
    for i ← 1 to k do
        Choose a random base a such that 2 ≤ a ≤ n − 2
        x ← ModularExponentiation(a, n − 1, n)
        if x = 1 or x = n − 1 then
            flag ← True
            Continue
        for j ← 1 to epoch do
            x ← ModularExponentiation(a, (n − 1)/2, n)
            if x = n − 1 then
                flag ← True
                Break
        flag ← False
        return Composite
    if flag is True then
        return Prime
    else
        return Composite
```

Time complexity and space complexity

Given, N: a positive integer with n-bits
k: a positive interger to iterate the test k times
Space Complexity:

$$\begin{aligned} S(n) &= O(1) + O(N/2) + o(1) + O(N) + O(N) = 2 * O(N) + O(N/2) \\ &\approx O(N) \approx O(\log n) \end{aligned} \tag{5}$$

Time Complexity:

$$
\begin{aligned}
T(n) &= O(N/2) + k * ((\log n) + O(1) + (\log n) + O(1) + o(1)) \\
&\approx O(\log n/2) + k * (2 * \log n) \approx O(k * \log n)
\end{aligned}
\tag{6}
$$

Computing success probability:

The probability of failure: $P_f \;=\; 1/4^k$
The probability of success $P_s \;=\; 1 - 1/4^k$

3 Observations and Results

3.1 Observation Table

(Integer, Iteration):(N,k)	Fermat Test Probability	Miller-Rabin Test Probability
312, 10	**False**	**False**
97, 10	True **(0.9990)**	True **(0.9999)**
561, 20	True **(0.0.9999)**	**False**

Both of these test are probability primality test. I used some positive integer along with different values of k to test the primality of N and have listed some of the ressult in table above. And observed the probability of both test.

3.2 Results

I used 11 for primality test (in figure 1) and as expected the program predicts 11 as prime number. Factors of 11 are 1 and 11; i.e. the factors of 11 is one and itself. This matches my defination for prime number. Thus, the program was able to correctly predict 11 as prime number.

Again, I tested 84 for primality test (in figure 2) and as expected the program predicts 84 as not prime (or composite) number. Factors of 84 are 2, 3, 4, 17 and soon; i.e. the factors of 84 is more than 2 which matches my defination for composite number. Therefore, the program was able to correctly predict 84 as composite number.

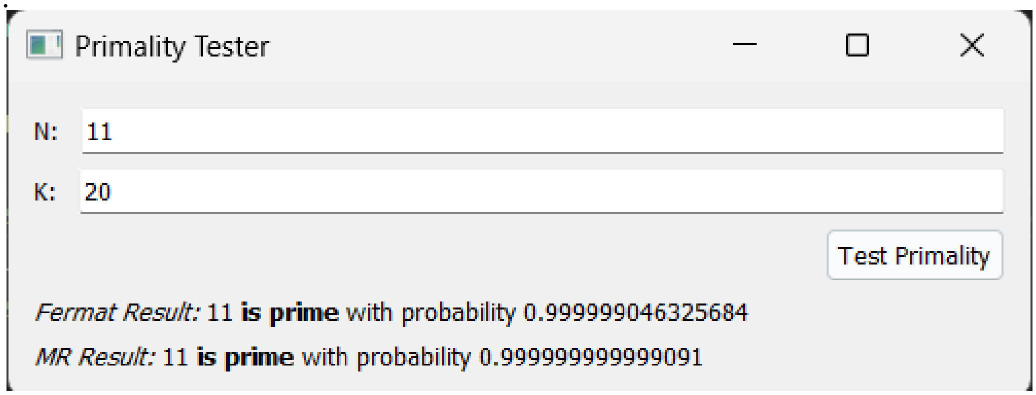

Figure 1: Testing positive interger 11

Figure 2: Testing positive interger 84

4 Conclusion

I tested the both Fermat's little theorem and Miller-Rabin primality test using some postive integers. The program was able to predict the integers with higher probability.