REPORT

과목	컴퓨터보안
과제명	Miller-Rabin algorithm
학과	컴퓨터공학과
학번	12161633
이름	이영주
연락처	01055021072
이메일	clleo97@naver.com

1. 개요

Miller-Rabin 알고리즘은 n과 s가 주어질 때, n이 소수인지 아닌지를 판별하는 알고리즘이다. 이때 s번 판별하여 2^{-s}의 오차를 가지는 확률적인 알고리즘이다. 반복을 여러 번 진행하여 오차를 줄여나간다.

Miller-Rabin 알고리즘은 소수가 가지는 특별한 성질을 이용한다. n이 홀수라고 할 때, n-1을 $2^t u$라고 할 수 있다. 이를 이용하여 NSR test와 Fermat test를 함께 진행한다.

본 프로젝트에서는 NST test와 Fermat test를 진행하는 Subroutine인 함수 Test를 구현하고자 한다.

2. 입력 양식

```
primes = [7879, 7883, 7901, 7907, 7919, 7927, 7933, 7937, 7949, 7951,
          7963, 7993, 8009, 8011, 8017, 8039, 8053, 8059, 8069, 8081,
          8087, 8089, 8093, 8101, 8111, 8117, 8123, 8147, 8161, 8167]
```

3. 프로그램 동작 원리

구현하고자 하는 부분의 수도코드과 함께 살펴보자.

Subroutine **Test** (a, n)
1: Let t and u be s.t. $t \geq 1$, u is odd, and $n - 1 = 2^t u$.
2: $x_0 = a^u mod\ n$.
3: For i = 1 to t
4: $x_i = x_{i-1}^2 mod\ n$.
5: if $x_i = 1$ and $x_{i-1} \neq 1$ and $x_{i-1} \neq n - 1$, then return **True**. // NSR test
6: End For
7: If $x_t \neq 1$, then return **True**. // Fermat test
8: Return **False**.

[1] Cormen, Leiserson, Rivest, and Stein, Introduction to Algorithms, 3rd ed., MIT Press

우선 주어진 n에 대하여 홀수 u와 2의 지수인 t로 분리한다.

```
# t,u --> t >=1, u : odd & n-1 = 2^t * u
c = 0
t = n-1
while t % 2 == 0 :
    c += 1
    t = t/2
t = c
u = (n-1) // (2 ** t)
```

2로 나눈 횟수를 저장할 변수를 c로 선언한다. 처음에 n-1값을 t에 저장하여 t를 2로 나눈 나머지가 0이 되지 않을 때까지 해당 반복문을 진행한다. 반복문을 진행하면 횟수인 c를 1 증가시키고 t에 t/2한 값을 대입한다. 반복문을 더는 진행할 수 없다면 t에 c를 대입하고 u는 (n-1)을 2^t로 나눈 몫을 대입한다.

수도코드의 2번째 줄을 살펴보자.

$$2: x_0 = a^u mod\ n.$$

x_0을 $a^u \bmod n$한 값을 대입하면 된다.

```
# x0 = a^u mod n --> modular exponentiation
x0 = exp(a, u, n)
```

modular 연산을 하는 것은 어렵지 않으나 a나 u가 큰 수라면 이 값을 구하는 시간은 굉장히 오래 걸릴 것이다. 시간을 단축하기 위해 exponentiation.py 파일을 import 한다. u를 bit로 변환 후 $a^u \bmod n$를 빠르게 계산할 수 있는 exp 함수를 호출하여 해당 수식의 답을 쉽게 구할 수 있다. 나온 결과를 x0에 대입한다.

3: For $i = 1$ to t
4: $x_i = x_{i-1}^2 mod\ n.$
5: if $x_i = 1$ and $x_{i-1} \neq 1$ and $x_{i-1} \neq n-1$, then return **True**. // NSR test
6: End For

다음은 NSR test를 진행하기 위한 수도코드이다. t번의 반복문을 수행한다. x_i에 이전의 x값을 제곱한 후 mod n을 한 값을 대입한다. 이 수가 1이고 이전의 x가 1이 아니고 n-1도 아니라면 True 반환한다.

```
Prime = 0
Composite = 1
```

```
    elif result == Composite:
        print("Composite")
```

```
# for문
for _ in range(t):
    x1 = (x0**2) % n
    if x1 == 1:
        if x0 != 1:
            if x0 != n-1:
                return True          # NSR test
    x0 = x1
    x1 = 0
```

이를 코드로 구현하였다. x0를 제곱한 값에 n으로 나눈 나머지를 x1에 대입한다. 만약 x1이 1이고 x0가 1이 아니고 n-1도 아니라면 True를 반환하여 main함수에서 “Composite”를 출력하게 된다. 만약 위의 조건을 충족시키지 못했다면 반복문을 이어나가야 하므로 x1의 값을 x0에 대입한다.

7: If $x_t \neq 1$, then return True. // Fermat test

NSR test를 통과했다면 Fermat test를 진행한다. t번의 반복문을 통해 나온 마지막 x_t가 1이 아니라면 True를 반환한다.

```
# Fermat test
if x0 != 1 :
    return True
```

해당 코드를 통해 Fermat test를 구현하였다.

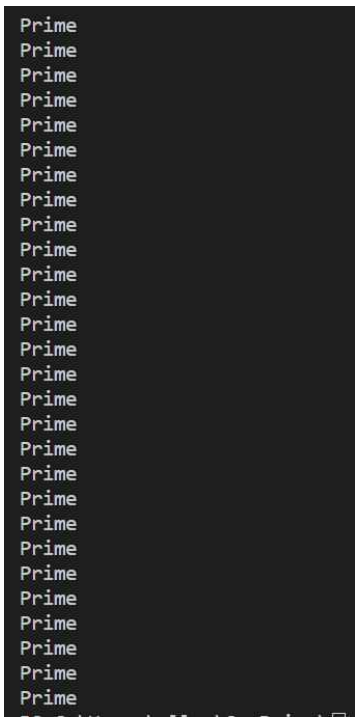
8: Return False.

모든 과정을 통과한다면 False를 반환한다.

```
return False
```

코드 또한 마찬가지이다.

4. 결과


Prime
Prime
Prime
Prime
Prime
Prime
Prime
Prime
Prime
Prime
Prime
Prime
Prime
Prime
Prime
Prime
Prime
Prime
Prime
Prime
Prime
Prime
Prime
Prime
Prime
Prime