Zadanie 3 - probabilistyczne testy pierwszości

Zaimplementuj dwa algorytmy probabilistycznego testowania pierwszości\ • test Fermata,\ • test Millera-Rabina.\ Wykorzystaj szybki algorytm potęgowania modulo.\ **Test Fermata** dla liczby p polega na wielokrotnym losowaniu liczby q takiej, że $q \in [1, 2, 3, \ldots, p)$, oraz q, p są względnie pierwsze. Następnie sprawdzamy, czy $q^{p-1} \ mod \ p \ = \ 1$. Jeśli nie, to liczba p nie jest pierwsza. Jeśli natomiast test wyjdzie pozytywnie dla wielu q, to liczba p prawdopodobnie jest pierwsza (im więcej testów, tym większe prawdopodobieństwo).

In [2]:
```
import numpy as np
```

In [3]:
```
# Algorytm Euklidesa dla 2 liczb x, y zwraca NWD
def euclides(x, y):
    if y == 0:
        return x
    else:
        return euclides(y, x%y)
```

In [4]:
```
# Funkcja kontrolna sprawdza czy liczba x jest pierwsza
def is_prime(x):
    if x <= 1:
        return False
    for n in range(2, int(x/2)):
        if x % n == 0:
            return False
    return True
```

In [5]:
```
# Test Fermata
def fermat(p):
    t = 0  # licznik przypadków prawdziwych (prawdopodobnie pierwsza)
    f = 0  # licznik przypadków fałszywych prawdopodobnie złożona

    for q in range(2, p+1):
        # sprawdzamy czy q, p są względnie pierwsze ( NWD = 1)
        if euclides(q, p) == 1:
            # test pierwszości
            if q ** (p - 1) % p == 1:
                t+=1
            else:
                f+=1

    # jeśli w 75% przypadków q^(p-1) mod p = 1 zwracamy (prawdopodobnie pierwsza)
    if t / (f+t) > 0.75:
        print(int(t / (f+t) * 100), '%')
        return 'Prawdopodobnie pierwsza'
    else:
        print(int(t / (f+t) * 100), '%')
        return 'Złożona'

fermat(997)
```

Out[5]:
```
100 %
'Prawdopodobnie pierwsza'
```

Test Millera-Rabina dla nieparzystej liczby p > 1 zaczyna się od przedstawienia tej liczby w postaci $p = 2^r \times q + 1$ (czyli od znalezienia r, a w konsekwencji q). Następnie wykonujemy:

1. losuj $a \in [2, p-2]$
2. niech $x = a^q \ mod \ p$
3. jeśli $x \in [1, p-1]$ to wracamy do pkt. 1
4. Powtarzaj $r-1$ razy
5. $x := x^2 \ mod \ p$
6. Jeśli $x = \ p-1$ to zwróć: prawdopodobnie piewsza
7. zwróć: złożona\ \ Przedstawioną procedurę potwarzamy wielokrotnie. Każde powtórzenie, które nie stwierdza, że liczba p jest złożona zwiększa prawdopodobieństwo, że jest ona piewsza.\

write n as 2r·d + 1 with d odd (by factoring out powers of 2 from n − 1)\ WitnessLoop: repeat k times:\ pick a random integer a in the range [2, n − 2]\ x ← ad mod n\ if x = 1 or x = n − 1 then\ continue WitnessLoop\ repeat r − 1 times:\ $x \leftarrow x^2 mod n$ if x = n − 1 then\ continue WitnessLoop\ return “composite”\ return “probably prime”\

In [35]:
```
from time import sleep

def miller_rabin(p, k):
    """
    Funkcja wykonująca test Millera-Rabina
    :param int p: p>3 nieparzysta liczba do przetestowania
    :param int k: liczba pętli do wykonania
    :return string: informacj o złożoności liczby 'złożona' lub 'prawdopodobnie pierwsza'
    """

    # test wykonujemy dla nieparzystych p większych od 1
    if p <= 1 or p % 2 == 0:
        return


    # ustalenie q i r tak że p = 2^r * q + 1
    for i in range(1, p):
        for j in range(p):
            if 2 ** i * j + 1 == p:
                r = i
                q = j

    for _ in range(k): # wykonujemy pętle tyle ile zadaliśmy w parametrze
        a = np.random.randint(2, p-1)
        x = a ** q % p
        if x == 1 or x == p - 1:
            pass
        for _ in range(r - 1):
            x = x ** 2 % p
            if x == p - 1:
                continue
        else:
            return 'Złożona'

    return 'Prawdopodobnie pierwsza'




print(miller_rabin(85, 10))
```

```
Złożona
```

In [15]:
```
is_prime(101)
```

Out[15]:
```
True
```