Zaawansowane metody kryptografii i ochrony informacji (14L)

# Porównanie dwóch wybranych testów pierwszości (Solovaya-Strassena i Fermata)

## Prowadzący projekt:

## Mgr inż. Marcin Tunia

# Wprowadzenie

Zagadnienie badania liczb pierwszych od wielu lat znajduje się w centrum zainteresowań matematyków, ze względu na prostotę opisu podstawowych problemów i mocne skorelowanie tych tworów matematycznych ze światem codziennym. W wyniku rozwoju teorii liczb, odkryto szereg własności liczb pierwszych; cały czas jednak mnóstwo zagadnień pozostaje otwartych, jak na przykład problem faktoryzacji liczb całkowitych, problem wyznaczenia wzoru jawnego na liczby pierwsze czy problem znajdowania liczb pierwszych o zadanych właściwościach. Z drugiej strony, na skutek rozwoju kryptografii asymetrycznej w ubiegłym wieku, okazało się, że istnieje szereg zastosowań w kryptografii, w których liczby pierwsze mogą być użyte, aby zrealizować pewne usługi bezpieczeństwa jak poufność czy integralność danych.

Kluczowym problemem w stosowaniu liczb pierwszych, jest stwierdzenie, czy dana liczba jest pierwsza. Z matematycznego punktu widzenia problem jest dość prosty, ponieważ na mocy np. twierdzenia Wilsona [1] wystarczy sprawdzić, czy dana liczba *p* spełnia kongruencję

.

Ze względu na problem obliczania silni, jest to algorytm niepraktyczny dla dużych liczb – w obecnej chwili potrzebne są liczby pierwsze rzędu tysięcy bitów, dla których obliczenie tego typu wyrażeń jest problemem nierozwiązywalnym w sensownym czasie (rzędu kilku sekund) na typowym komputerze. Stąd potrzeba stosowania bardziej wyrafinowanych testów, aby poradzić sobie z ograniczeniami czasowymi (i często też pamięciowymi).

Testy pierwszości można w pierwszej kolejności podzielić na testy deterministyczne i testy niedeterministyczne. W niniejszej pracy skupiono się na tych ostatnich, ze względu na fakt ich powszechnego stosowania w praktyce – posiadają zwykle mniejszą złożoność asymptotyczną niż testy deterministyczne – np. sito Eratostenesa działa w czasie

,

Natomiast najszybszy (asymptotyczne, na chwilę obecną) test deterministyczny AKS działa w czasie

.

Dla porównania badane testy Fermata i Solovaya-Strassena mają złożoność czasową ograniczoną z góry przez

.

Biorąc pod uwagę stałą, jaka kryje się pod notacją duże O, algorytmy niedeterministyczne są dużo szybsze dla „małych” liczb.

# Opracowanie teoretyczne

W wstępie podjęto temat testów pierwszości, dzieląc je w pierwszej kolejności na testy deterministyczne i testy niedeterministyczne, na których skupia się niniejsza praca. Możliwe są jeszcze inne kryteria podziału (generalnie ortogolnane do powyższego), na przykład kryterium ze względu na niepewność odpowiedzi, jaka jest udzielana przez test. W przypadku analizowanych testów, odpowiedź na pytanie „*Czy liczba p jest liczbą pierwszą?”* jest pewna, jeśli udzielono odpowiedzi negatywnej (nie istnieją takie przypadki, że testy stwierdzą złożoność liczby, gdy była ona faktycznie pierwsza). Jednocześnie te testy nie udzielają (w ogólności) pewnej odpowiedzi w przypadku, gdy odpowiedź była pozytywna. Wynika to z budowy algorytmicznej (i podstawy matematycznej) samych testów; dokładne kryteria, w jakich przypadkach dany test może udzielić odpowiedzi nieprawdziwej podane są w dalszej części tekstu.

## Wykorzystane algorytmy

Do implementacji omawianych testów posłużono się (bezpośrednio lub pośrednio) algorytmem szybkiego potęgowania modularnego (jest implementowany przez standardową funkcję podnoszenia do potęgi liczby zapisanej w klasie *BigInteger*) oraz algorytmem szybkiego obliczania wartości symbolu Jacobiego, przy pomocy *prawa wzajemności reszt kwadratowych.*

## Implementacja

Implementację algorytmów przeprowadzono z użyciem języka *Java* (w wersji 8), korzystając ze standardowych bibliotek tego języka, w tym bibliotek odpowiedzialnych za obliczenia na dużych liczbach całkowitych. W celu bieżącego sprawdzania poprawności tworzonego kodu, stworzono zestaw testów jednostkowych z użyciem biblioteki *JUnit* (wersja 4). Klasy zostały zaimplementowane z wykorzystaniem techniki *dependency injection*, dzięki czemu możliwe jest dostosowanie sposobu działania kodu do wymagań użytkownika.

# Opis algorytmów

## Test Fermata

### Pseudokod

## Test Solovaya-Strassena

W poniższym algorytmie wykorzystywane są elementarne pojęcia z teorii liczb jak reszty kwadratowe oraz przystawanie modulo *p*. Poniżej będziemy używać symbolu Legendre’a, zdefiniowanego następująco:

*Niech p będzie nieparzystą liczbą pierwszą. Wówczas*

Symbol Jacobiego jest definiowany jako uogólnienie symbolu Legendre’a na nieparzyste liczby całkowite, w następujący sposób:

Niech liczba *n* ma następujący rozkład na czynniki pierwsze:

Wówczas dla nieujemnej liczby całkowitej *a* zachodzi:

Test opiera się na następującym *twierdzeniu Eulera*:

*Jeżeli liczba p jest liczbą pierwszą, oraz (a,p) = 1, to*

Gdy liczba *p* nie jest liczbą pierwszą, to liczbę *a*, dla której nie zachodzi powyższa kongruencja, nazywamy *świadkiem złożoności liczby p*. Test polega na *m*-krotnym losowaniu liczby *a* i sprawdzaniu czy zachodzi powyższe twierdzenie. Można udowodnić, że prawdopodobieństwo dania złej odpowiedzi (liczba jest złożona, ale test tego nie potwierdził) przez ten test przy *m* próbach daje się oszacować wyrażeniem:

Złożoność obliczeniową testu *Solovaya-Strassena* można oszacować przez wyrażenie: (na podstawie [2, str. 133)]), ponieważ potęgowanie modularne działa w czasie , natomiast algorytm obliczania symbolu Jacobiego można zrealizować w czasie [2, str. 136]. Poniżej przedstawiono własności symbolu Jacobiego, wraz ze schematem postępowania, aby obliczyć jego wartość w tym teście.

1. Jeśli jest nieparzystą liczbą całkowitą oraz , to:
2. Jeśli jest nieparzystą liczbą całkowitą, to:
3. Jeśli jest nieparzystą liczbą całkowitą to:
4. Jeśli i są nieparzystymi liczbami całkowitymi to:

Powyższe twierdzenia są podane np. w książce [2] i należą do kanonu twierdzeń powiązanych z symbolem Jacobiego. Nie będziemy oczywiście rozpatrywać parzystych podstaw (algorytm powinien sprawdzić, czy testowana liczba jest nieparzysta albo równa 2, zanim przystąpi się do dalszych kroków: obliczania potęg modulo n czy obliczania symbolu Jacobiego). Możemy wobec tego napisać następujący ciąg równości, zakładając, że *n* jest liczbą nieparzystą oraz , gdzie jest liczbą nieparzystą:

Obliczenie i jest proste, ponieważ wystarczy sprawdzić ile najmłodszych bitów liczby *m* jest zerowych (jest ich *k*), . Przy obliczaniu symbolu powyższą metodą rekurencyjną, należy pamiętać o następujących własnościach symbolu Jacobiego, aby rekurencja mogła się zakończyć:

W powyższym rozumowaniu należy jeszcze rozpatrzyć przypadek, gdy . Wtedy należy poprzestać na obliczaniu symbolu Jacobiego na drugiej równości, ponieważ 1 nie jest liczbą pierwszą; przyjmuje się jednak dla uogólnienia wzorów, że .

### Pseudokod

Na podstawie [2] można zdefiniować powyższy algorytm w następujący sposób:

Solovay-Strassen(n, m)

Begin

if( n% 2 == 0 )

if( n != 2 )

return “n jest złożona”

else

return “n jest pierwsza”

for( i = 0: i<m;++i)

{

a = rand(n); //losowa liczba całkowita z zakresu 1…n-1

left = resolveOfJacobiSymbol(a, n); // oblicz a nad n

right = pow(a, (n-1)/2) % n; //potęgowanie

if( left != right)

return “n jest złożona”

}

return “n jest pierwsza”

end

W projekcie zaimplementowano obliczanie wartości symbolu Jacobiego, na podstawie *prawa wzajemności reszt kwadratowych*. Poniżej znajduje się pseudokod dla wyprowadzonej powyżej zależności rekurencyjnej:

resolveOfJacobiSymbol(a, n)

begin

if( a == 0 )

return 0;

if( a == 1 )

return 1;

k = firstOnePosition(a); // k ze wzoru rekurencyjnego

tmp = a >> k;

if( k % 2 == 1 && (n % 8 == 3 || n % 8 == 5) )

result = -1;

else

result = 1;

if ( n % 4 == 3 && y % 4 == 3 )

result = - result;

if tmp == 1

return result;

else

return result\* resolveOfJacobiSymbol(n mod tmp, tmp);

end

# Zastosowania

Biorąc pod uwagę zastosowania czysto kryptograficzne (są one w głównym kręgu zainteresowań na tym przedmiocie), powyższe algorytmy mogą być wykorzystane w algorytmach i protokołach wymagających użycia liczb pierwszych (lub z dużym prawdopodobieństwem pierwszych). Do nich zaliczają się algorytmy i protokoły oparte na RSA czy algorytmie ElGamal – w klasycznym ujęciu tych algorytmów operacje wykonywane są z użyciem liczb, które są niemożliwe do rozłożenia na czynniki pierwsze – ze względu na relatywnie duże wartości tych czynników ( porównywalne z pierwiastkiem z danej liczby, aby wykluczyć użycie małych czynników pierwszych).

Ze względu na istnienie liczb Carmicheala, test Fermata nie jest szeroko wykorzystywany w praktyce – jest ich nieskończenie wiele, stąd też nie można testu poprawić, na przykład tablicując wszystkie takie liczby. Z drugiej strony jest to bardzo szybki test – wymaga podniesienia danej liczby do zadanej z góry potęgi, co można zrealizować w czasie wielomianowym od długości bitowej liczby, której pierwszość się bada.

Test Solovaya-Strassena jest prekursorem testu Millera-Rabina, który jest szeroko stosowany w systemach informatycznych – np. biblioteka standardowa języka Java używa testu Millera-Rabina do generowania liczb potencjalnie pierwszych w klasie BigInteger.

# Testy

# Porównanie i wnioski

# Instrukcja obsługi aplikacji

# Bibliografia

1. Dowód Twierdzenia Wilsona - <http://en.wikipedia.org/wiki/Wilson%27s_theorem>
2. Douglas R.Stinson – Kryptografia w teorii i praktyce, WNT 2005