Zaawansowane metody kryptografii i ochrony informacji (14L)

Porównanie testów pierwszości Solovaya-Strassena i Fermata

Prowadzący projekt:

Mgr inż. Marcin Tunia

Tomasz Nowak 236552 Tomasz Barszcz 236424

T.W.Nowak@stud.elka.pw.edu.pl T.Barszcz@stud.elka.pw.edu.pl

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

$$(p-1)! \equiv -1 \ (mod\ p).$$

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

$$O(\sqrt{p}),$$

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

$$O(log^{12}p)^{1}.$$

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

[1] Istnieją lepsze oszacowania, zmniejszające wielkość wykładnika, jednak mimo tego algorytm ten jest wolniejszy niż znane testy niedeterministyczne.

$$O(log^3 p).$$

Znane algorytmy niedeterministyczne są z natury szybsze, chociaż mogą czasem podać błędny wynik, jednak przy stosowaniu ich w odpowiedni sposób, prawdopodobieństwo błędnego wyniku można zaniedbać, ze względu na to, że może być ono dowolnie małe.

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[2]. 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.

[2] Są to tzw. negatywnie nastawione algorytmy Monte-Carlo, biorąc pod uwagę tak postawione pytanie [2].

Tomasz Nowak 236552 Tomasz Barszcz 236424

T.W.Nowak@stud.elka.pw.edu.pl T.Barszcz@stud.elka.pw.edu.pl

Opis algorytmów

Test Fermata

Podstawą działania testu jest *Małe Twierdzenie Fermata* oraz zasady obowiązujące w arytmetyce modularnej. Podstawowa wersja twierdzenia wygląda następująco:

$$a^p - p \ (mod\ p) \equiv 0$$

Gdzie *a, p* są to kolejno dowolna liczna natualna oraz dowolna liczba pierwsza. Po dokonaniu odpowiednich przekształceń, otrzymujemy równoważną mu wersję (zakładając, że te dwie liczby są względnie pierwsze):

$$a^{p-1} \equiv 1 \ (mod\ p)$$

Przedstawione twierdzenie można wykorzystać do badania pierwszości liczb poprzez sprawdzanie powyższego warunku dla losowo wybranej podstawy *a*. Korzystając z własności liczb pierwszych wystarczy sprawdzać wynik potęgowania modulo i podejmować decyzję o pierwszości.

Na podstawie [3] zdefiniujmy prosty test pierwszości Fermata. Będziemy mówić, że badana liczba jest pierwsza, jeśli dla losowo wybranej podstawy *a* z zakresu [*2,p-1*] wynik potęgowania a^{p-1} jest równy jedności (modulo *p). Dla* pozostałych wyników dzielenia różnych od 1 liczba będzie określana jako złożona.

Niestety nie oznacza to, że nie ma innych liczb złożonych spełniających powyższy warunek – są nimi m.in. liczby *Carmichael'a*. Są nimi liczby złożone spełniające warunki *Małego Twierdzenia Fermata*, tzn. dające identyczny rezultat testu jak liczby pierwsze. Jednak występują one na tyle rzadko[3], że badany test z bardzo dużym prawdopodobieństwem pozwala potwierdzić pierwszość liczby. Natomiast liczby określone jako niepierwsze są takimi ze 100% pewnością.

Znacznym ulepszeniem prostego testu Fermata jest kilkukrotne powtórzenie testu dla różnych wartości podstawy *a*. Natomiast w celu eliminacji liczb *Carmichael'a* sprawdza się podzielność badanej liczby przez początkowe liczby pierwsze[4].

Poprawność uproszczonego testu Fermata jest dosyć dobra – prawdopodobieństwo popełnienia błędu (tzn określenia liczby złożonej jako pierwszej) jest szacunkowo mniejsza od 10^{-6} dla badanej liczby 50 – cyfrowej dla warunku o podstawie 2, natomiast dla liczby 100 cyfrowej jest mniejsza od 10^{-13} [3]. Złożoność obliczeniowa jest rzędu $O(m * log^3 n)$,

[3] Nie mniej jednak jest ich nieskończenie wiele.
[4] Im więcej czynników pierwszych, tym więcej tych liczb uda się odnaleźć w ten sposób.

ze względu na potęgowanie modularne, które występuje w tym teście, gdzie *m* jest liczbą prób.

Pseudokod

Na podstawie powyższego algorytmu zaimplementowano *m*-krotnie powtarzany uproszczony test Fermata w następującej postaci:

```
Wejście:
n – testowana liczba, n ∈ N, n > 2
m – liczba powtórzeń algorytmu, m ∈ N
Wyjście:
„n jest złożona” – jeśli n nie jest liczbą pierwszą
„n jest pierwsza” – jeśli n z bardzo dużym prawdopodobieństwem jest
liczbą pierwszą lub jest liczbą Carmichaela

Fermat(n, m)
begin
  for( i = 0; i<m ; ++i )
  {
      a = rand(n); //losowa  liczba całkowita z zakresu 1…n-1
      powResult = pow(a, (n-1)) % n; //potęgowanie modulo
      if( powResult != 1 )
            return “n jest złożona”
   }
return “n jest pierwsza”
end
```

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

$$\left(\frac{a}{p}\right) = \begin{cases} 1 \iff a\ jest\ resztą\ kwadratową\ mod\ p \\ -1 \iff a\ jest\ nieresztą\ kwadratową\ mod\ p \\ 0 \iff (a,p) > 1 \end{cases}$$

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:

$$n = \prod_i {p_i}^{e_i}$$

Wówczas dla nieujemnej liczby całkowitej *a* zachodzi:

$$\left(\frac{a}{n}\right) = \prod_i \left(\frac{a}{p_i}\right)^{e_i}$$

Test opiera się na następującym *twierdzeniu(kryterium) Eulera [4]*:

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

$$\left(\frac{a}{p}\right) \equiv a^{\frac{p-1}{2}} \ (mod\ p)$$

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:

$$P(T(n) = pierwsza \mid n\ jest\ złożona\ ) \approx \frac{\ln n - 2}{\ln n - 2 + 2^{m+1}}$$

Złożoność obliczeniową testu *Solovaya-Strassena* można oszacować przez $O(m * log^3 n)$ (na podstawie [2, str. 133)]) - ponieważ potęgowanie modularne działa w czasie $O(log^3 n)$, natomiast algorytm obliczania symbolu Jacobiego można zrealizować w czasie $O(log^2 n)$ [2, str. 136]. Poniżej przedstawiono własności symbolu Jacobiego, wraz ze schematem postępowania, aby obliczyć jego wartość w tym teście.

1. Jeśli n jest nieparzystą liczbą całkowitą oraz $m_1 \equiv m_2\ (mod\ n)$, to:
$$\left(\frac{m_1}{n}\right) = \left(\frac{m_2}{n}\right)$$
2. Jeśli n jest nieparzystą liczbą całkowitą, to:
$$\left(\frac{2}{n}\right) = \begin{cases} 1 & \Leftrightarrow n \equiv \pm 1\ (mod\ 8) \\ -1 & \Leftrightarrow \equiv \pm 3\ (mod\ 8) \end{cases}$$
3. Jeśli n jest nieparzystą liczbą całkowitą to:
$$\left(\frac{m_1 m_2}{n}\right) = \left(\frac{m_1}{n}\right)\left(\frac{m_2}{n}\right)$$
4. Jeśli n i m są nieparzystymi liczbami całkowitymi to:

$$\left(\frac{m}{n}\right) = \left(\frac{n}{m}\right)(-1)^{\frac{(n-1)(m-1)}{4}} = \begin{cases} -\left(\frac{n}{m}\right) & \Leftrightarrow n \equiv m \equiv 3\ (mod\ 4) \\ \left(\frac{n}{m}\right) & \Leftrightarrow w\ p.p. \end{cases}$$

Powyższe twierdzenia są podane np. w książce [2] i należą do kanonu twierdzeń powiązanych z symbolem Jacobiego. Nie będziemy oczywiście rozpatrywać parzystych podstaw n (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 $m = 2^k s$, gdzie s jest liczbą nieparzystą:

$$\left(\frac{m}{n}\right) = \left(\frac{2}{n}\right)^k \left(\frac{s}{n}\right) = \left(\frac{2}{n}\right)^k \left(\frac{n}{s}\right)(-1)^{\frac{(n-1)(s-1)}{4}} = \left(\frac{2}{n}\right)^k \left(\frac{n \bmod s}{s}\right)(-1)^{\frac{(n-1)(s-1)}{4}}$$

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

$$\left(\frac{0}{n}\right) = 0$$

$$\left(\frac{1}{n}\right) = 1$$

W powyższym rozumowaniu należy jeszcze rozpatrzyć przypadek, gdy $s = 1$. Wtedy należy poprzestać na obliczaniu symbolu Jacobiego na drugiej równości, ponieważ 1 nie jest liczbą pierwszą; przyjmuje się jednak dla uogólnienia wzorów, że $\left(\frac{n}{1}\right) = 1$.

Pseudokod

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

```
Wejście:
n – testowana liczba, n ∈ N, n > 2
m – liczba powtórzeń algorytmu, m ∈ N

Wyjście:
„n jest złożona” – jeśli n nie jest liczbą pierwszą
„n jest pierwsza” – jeśli n jest liczbą pierwszą
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
```

Tomasz Nowak 236552 Tomasz Barszcz 236424

T.W.Nowak@stud.elka.pw.edu.pl T.Barszcz@stud.elka.pw.edu.pl

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

```
Wejście:
a,b – argumenty symbolu Jacobiego, a,b ∈ N, n > 2
Wyjście:
-1,0,1 – wynik zwracany przez symbol Jacobiego

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 np. 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 *Carmicheal'a*, 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-

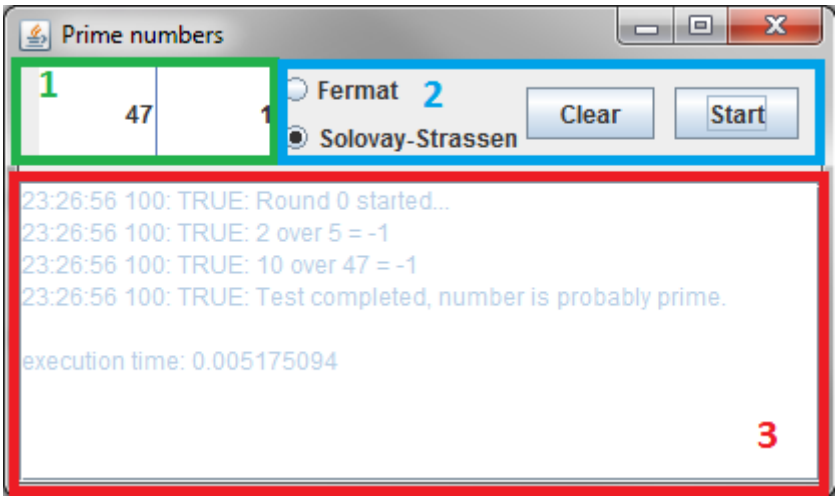
Tomasz Nowak 236552 Tomasz Barszcz 236424

T.W.Nowak@stud.elka.pw.edu.pl T.Barszcz@stud.elka.pw.edu.pl

Rabina do generowania liczb potencjalnie pierwszych w klasie *BigInteger*. Zdecydowano się na to, ponieważ daje on mniejsze prawdopodobieństwo popełnienia błędu, przy danej ilości testów, zachowując jednocześnie asymptotycznie taką samą złożoność obliczeniową.

Instrukcja obsługi aplikacji

Główne okno programu przedstawia Rys. 1. Zostało ono podzielone na 3 części.



Rys. 1 Główne okno programu

Pierwsze odpowiedzialne za wprowadzenie wartości liczbowych – odpowiednio liczby, której pierwszość ma być zbadana oraz ilość wywołań algorytmu (1). Ponieważ zaimplementowane algorytmy do swych obliczeń korzystają z wartości losowych, dlatego większa ilość prób dla badanej liczby zwiększa prawdopodobieństwo poprawności wyniku. Niestety również zwiększa czas jego wykonania, o czym można się przekonać porównując wartość *execution time* dla różnych wartości drugiego parametru. Konieczne jest podanie badanej liczby całkowitej większej od 2 oraz ilości powtórzeń większej od 0.

W drugiej części (2) możliwe jest wybranie jednej z dwóch metod w celu sprawdzenia pierwszości liczby a następnie uruchomienie procedury testującej przy pomocy przycisku *Start*. Przycisk *Clear* odpowiada za czyszczenie zawartości okna logów oraz zerowanie wartości liczbowych podanych przez użytkownika w części 1. Konieczne jest wybranie jednego z dwóch dostępnych algorytmów.

Trzecia część (3) umożliwia użytkownikowi oglądanie rezultatów działania testu. W zależności od metody prezentowane są kolejne kroki działania algorytmu dla poszczególnych powtórzeń oraz podsumowanie w postaci rezultatu i czasu działania całej operacji. Uzytkownik może wyczyścić zawartość okna bądź rozpocząć nowy test korzystając z przycisków obszaru 2.

Tomasz Nowak 236552 Tomasz Barszcz 236424

T.W.Nowak@stud.elka.pw.edu.pl T.Barszcz@stud.elka.pw.edu.pl

Bibliografia

[1] *Dowód Twierdzenia Wilsona* - http://en.wikipedia.org/wiki/Wilson%27s_theorem
[2] Douglas R.Stinson – *Kryptografia w teorii i praktyce*, WNT 2005
[3] T. Cormen, C. Leiserson, R. Rivest – *Wprowadzenie do algorytmów*, WNT 1994
[4] A. Menezes, P. van Oorschot,S. Vanstone - *Handbook of Applied Cryptography* (http://cacr.uwaterloo.ca/hac/about/chap4.pdf)