POLITECHNIKA POZNAŃSKA

WYDZIAŁ ELEKTRYCZNY

Instytut Matematyki

PRACA DYPLOMOWA MAGISTERSKA

TESTY PIERWSZOŚCI LICZB I ICH ZASTOSOWANIA W KRYPTOGRAFII

Małgorzata Lipińska

Promotor:
dr hab. Małgorzata Migda

POZNAŃ, 2018

KARTA PRACY DYPLOMOWEJ Z DZIEKANATU
(kserokopia z podpisami)

Podziękowania

Składam serdecznie podziękowania
moim rodzicom,

Spis treści

Wstęp

1. Liczby pierwsze

1.1. Zasadnicze twierdzenie arytmetyki i twierdzenie Euklidesa

Definicja 1.1. *Liczbą pierwszą* nazywa się liczbę naturalną większą od 1, która ma dokładnie dwa dzielniki: jedynkę i samą siebie. Zbiór wszystkich liczb pierwszych oznacza się literą $\mathbb{P}$.

Liczbą złożoną jest więc liczba, która ma więcej niż dwa dzielniki. Warto zauważyć, że 1 nie jest liczbą ani pierwszą, ani złożoną.

Definicja 1.2. *Liczby względnie pierwsze* to liczby całkowite, których największym wspólnym dzielnikiem jest liczba 1. Jeśli dwie liczby a i b są względnie pierwsze to zapisuje się to jako $NWD(a, b) = 1$ lub $(a, b) = 1$.

Definicja 1.3. *Liczba całkowita b jest podzielna przez liczbę całkowitą a*, jeśli istnieje taka liczba całkowita e, że zachodzi $b = ae$. Wtedy liczba a jest dzielnikiem liczby b, z kolei o liczbie b mówi się, że jest wielokrotnością a. Jeśli liczba b jest podzielna przez a (a dzieli b), to używa się oznaczenia $a \mid b$, natomiast w przeciwnym przypadku stosuje się oznaczenie $a \nmid b$ [2, s. 3].

Lemat 1.1. (Euklides) Jeśli liczba pierwsza p dzieli iloczyn ab, to dzieli przynajmniej jeden z tych czynników. Inaczej, jeśli $p \in \mathbb{P}$ oraz $p \mid ab$, to $p \mid a$ lub $p \mid b$.

Dowód. Przyjmijmy, że $p \nmid a$, czyli $NWD(a, p) = 1$, gdyż $p \in \mathbb{P}$. Wtedy $p \mid b$.

■

Lemat 1.2. Jeżeli $p \in \mathbb{P}$ i $p \mid a_1 \cdot ... \cdot a_n$, to istnieje k $(1 \leq k \leq n)$, takie że $p \mid a_k$.

Dowód. Dowód wynika wprost z dowodu Lematu 1.1. Iloczyn $a_1 \cdot ... \cdot a_n$ można przedstawić w postaci $(a_1 \cdot ... \cdot a_{n-1})a_n$. Wtedy $p \mid (a_1 \cdot ... \cdot a_{n-1})a_n$, czyli z poprzedniego dowodu $p \mid a_1 \cdot ... \cdot a_{n-1}$ lub $p \mid a_n$. Zasadę tą można uogólnić dla dowolnego k $(1 \leq k \leq n)$. Wtedy $p \mid a_k \cdot \prod_{i \neq k}^{n} a_i$, a więc $p \mid \prod_{i \neq k}^{n} a_i$ lub $p \mid a_k$.

■

Lemat 1.3. Jeśli liczby $p, q_1, ..., q_n \in \mathbb{P}$ oraz $p \mid q_1 \cdot ... \cdot q_n$, to $p = q_k$ dla pewnego k, gdzie $1 \leqslant k \leqslant n$.

Dowód. Z Lematu 1.2 wiadomo, że istnieje takie k, że $p \mid q_k$, ale $q_k \in \mathbb{P}$ oraz $p > 1$, skąd wynika, że $p = q_k$.

■

Trzy powyższe lematy to lematy pomocnicze, które posłużą do udowodnienia twierdzenia Eulera, mówiącego o tym, że liczby pierwsze są czynnikami, na które można rozłożyć wszystkie złożone liczby naturalne.

Twierdzenie 1.1. *(Zasadnicze twierdzenie arytmetyki) Każdą liczbę naturalną większą od 1, która nie jest liczbą pierwszą, można jednoznacznie przedstawić w postaci iloczynu liczb pierwszych.*

Dowód. Dowód powyższego twierdzenia należy podzielić na dwie części. Pierwsza dotyczyć będzie istnienia rozkładu, a druga jego jednoznaczności.

Istnienie rozkładu. Załóżmy, że liczba n jest liczbą złożoną. Wtedy

$$D = \{d \in N, 1 < d < n, d \mid n\}$$

jest niepustym zbiorem dzielników liczby n. Niech p_1 będzie najmniejszą liczbą zawartą w tym zbiorze. Zatem p_1 jest liczbą pierwszą, gdyż w przeciwnym przypadku istniałoby $q < p_1$ takie, że $q > 1$ i $q \mid p_1$, co byłoby sprzeczne z wyborem p_1. Stąd wynika, że

$$n = p_1 \cdot n_1,$$

gdzie $1 < n_1 < n$. Jeżeli liczba $n_1 \in \mathbb{P}$, to żądany rozkład został osiągnięty, w przeciwnym wypadku rozumowanie należy powtórzyć dla n_1, skąd dostajemy

$$n = p_1 \cdot p_2 \cdot n_1,$$

gdzie $1 < n_2 < n_1$, $p_1, p_2 \in \mathbb{P}$. Na koniec uzyskuje się rozkład

$$n = p_1 \cdot ... \cdot p_k,$$

ponieważ $n_1 > n_2 > ... > 1$ jest malejącym ciągiem liczb naturalnych, czyli jest ciągiem skończonym.

Jednoznaczność rozkładu. Załóżmy przeciwnie, że rozkład nie jest jednoznaczny, czyli $n = p_1 \cdot ... \cdot p_r = q_1 \cdot ... \cdot q_s$, $(r \leqslant s)$, gdzie p_i i q_j dla $i = 1, 2, ..., r$ i $j = 1, 2, ..., s$ są pierwsze i uporządkowane niemalejąco. Ponieważ $p_1 \mid q_1 ... q_s$, zatem z Lematu 1.3 wynika, że $p_1 = q_k$ dla pewnego k, gdzie $1 \leqslant k \leqslant s$, skąd wynika, że $p_1 \geqslant q_1$. Analogicznie $q_1 \mid p_1 ... p_r$, więc $q_1 \geqslant p_1$, czyli $p_1 = q_1$. Dzieląc początkową

równość przez $p_1 = q_1$ i powtarzając to samo rozumowanie dla kolejnych czynników obu rozkładów, otrzymujemy

$$1 = q_{r+l} \cdot ... \cdot q_s,$$

jeśli $r < s$. Jest to sprzeczne dla $q_{r+l}, ..., q_s \in \mathbb{P}$, a więc $r = s$ oraz $p_i = q_i$ dla każdego $i = 1, 2, ..., r$, czyli rozkład jest jednoznaczny [2, s. 10].

■

Z Zasadniczego Twierdzenia Arytmetyki 1.1 wynika bezpośrednio poniższy wniosek.

Wniosek 1.1. Każda liczba naturalna $n > l$ może być jednoznacznie zapisana w postaci kanonicznej

$$n = p_1^{\alpha_1} \cdot p_2^{\alpha_2} \cdot ... \cdot p_r^{\alpha_r},$$

gdzie $p_i \in \mathbb{P}$, $\alpha_i \in \mathbb{N}$ dla każdego $i = 1, 2, ..., r$ oraz $p_1 < p_2 < ... < p_r$.

Twierdzenie 1.2. *(Euklides) Istnieje nieskończenie wiele liczb pierwszych.*

Powstało kilkanaście dowodów twierdzenia 1.2. Poniżej przedstawiono pierwszy z nich, który został sformułowany w IV wieku p.n.e. przez Euklidesa w dziele pod tytułem *Elementy*.

Dowód. Załóżmy, że istnieje skończony zbiór liczb pierwszych ponumerowanych kolejno $p_1, p_2, ..., p_k$. Rozważmy liczbę

$$n = p_1 p_2 ... p_k + 1.$$

Z Twierdzenia 1.1 wynika, że n może być liczbą pierwszą różną od wszystkich p_i lub ma rozkład na czynniki pierwsze. Załóżmy, że jednym z tych czynników jest p. Liczba n przy dzieleniu przez każde z p_i daje resztę 1, więc p jest różne od wszystkich p_i. Z tego wynika, że albo p jest pierwsze lub samo n jest kolejną liczbą pierwszą, większą od każdego p_i, co jest sprzeczne z założeniem [3, s. 5].

■

1.2. Sito Eratostenesa i sito Atkina

Definicja 1.4. Najprostszym i jednocześnie najmniej wydajnym algorytmem wyszukiwania kolejnych liczb pierwszych z przedziału $[2, n]$ jest tak zwane *Sito Eratostenesa*, które zostało opracowane przez greckiego matematyka Eratostenesa z Cyreny (III-II w. p.n.e.). Algorytm polega na usuwaniu liczb złożonych z zadanego zbioru

liczb naturalnych w taki sposób, że ze zbioru wybieramy najmniejszą liczbę i usuwamy wszystkie jej wielokrotności, następnie wybieramy najmniejszą z pozostałych i postępujemy tak jak z pierwszą. Na koniec procesu w zbiorze pozostaną tylko liczby pierwsze.

Przykład 1.1. Dla przykładu pokazano sito Eratostenesa dla zbioru $P = \{2, 3, ..., 40\}$.

I. Usunięto wszystkie wielokrotności liczby 2.

$$P = \{2, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39\}.$$

II. Usunięto wszystkie wielokrotności liczby 3.

$$P = \{2, 3, 5, 7, 11, 13, 17, 19, 23, 25, 29, 31, 35, 37\}.$$

III. Usunięto wszystkie wielokrotności liczby 5.

$$P = \{2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37\}.$$

Po trzecim kroku w zbiorze P pozostały jedynie liczby pierwsze.

Sito Eratostenesa można teoretycznie zapisać w postaci funkcji, na przykład w programie Matlab. Poniżej przedstawiono kod programu oraz tabelę z wynikami dla różnych wielkości przedziałów.

```
function [n,p] = sitoEratostenesa(N)
   sito=1:N;
   a=2;
   while a^2<=N
      sito(a^2:a:N)=0;
      a=find(sito>a,1);
   end
   p = sito(sito>1);
   n = length(p)
end
```

Lp.	Koniec przedziału $\{2, n\}$	Czas działania programu $[s]$	Ilość liczb pierwszych
1	100	0.002345	25
2	1000	0.00219	168
3	10 000	0.107349	1229
4	100 000	0.191775	9592
5	1 000 000	0.253845	78 498
6	10 000 000	6.684239	664 579
7	100 000 000	174.164354	5 761 455

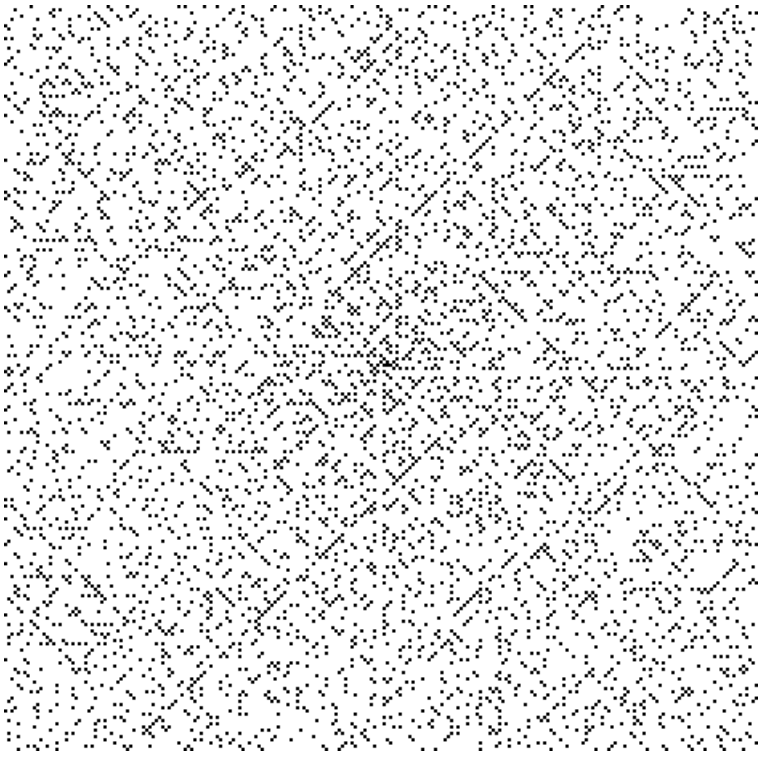
Tabela 1.1. Wyniki działania funkcji szukającej liczb pierwszych sitem Eratostenesa.

Niestety w przypadku większego zakresu liczb czas oczekiwania na wynik jest bardzo długi. Na przykład przeszukiwanie zbioru liczb naturalnych $[2, 1\,000\,000\,000]$ może trwać już kilka godzin. Co więcej, największa znana obecnie liczba pierwsza jest o wiele większa od liczb, które można znaleźć za pomocą tego algorytmu nawet na nowoczesnych komputerach o dużej wydajności [1, s. 48-49].

Definicja 1.5. *Sito Atkina* lub *sito Atkina-Bernsteina* to algorytm autorstwa dwóch matematyków - Arthura Atkina i D.J. Bernsteina, który służy do wyszukiwania liczb pierwszych w dużych przedziałach. Metoda ta działa podobnie do sita Eratostenesa, jednak dzięki wykorzystaniu pewnych zależności jest efektywniejsza i wymaga znacznie mniej pamięci. Na przykład algorytm całkowicie pomija wszystkie liczby podzielne przez 2, 3 lub 5, liczby z parzystą resztą z dzielenia przez 60, liczby, które mają resztę z dzielenia przez 60 podzielną przez 3 oraz liczby z resztą z dzielenia przez 60 podzielną przez 5 [4].

1.3. Spirala Ulama

Definicja 1.6. *Spirala Ulama* lub *spirala liczb pierwszych* to graficzna metoda przedstawiania rozkładu liczb pierwszych, która została przedstawiona w 1963 roku przez polskiego matematyka Stanisława Ulama. Sposób zapisu polega na tym, że na kwadratowej tablicy zaczynając od 1 w środku spiralnie wypisuje się kolejne liczby naturalne. Okazuje się, że na pewnych przekątnych liczby pierwsze pojawiają się o wiele częściej niż na innych. Co więcej, zjawisko nie zależy od wyboru pierwszej liczby w spirali. Na Rysunku 1.6 przedstawiono spiralę Ulama o rozmiarze 200x200.



Rysunek 1.1. Spirala Ulama o rozmiarze 200x200 [8].

1.4. Twierdzenie Wilsona i małe twierdzenie Fermata

Definicja 1.7. Niech n będzie dowolną liczbą naturalną. Liczby całkowite a i b *przystają modulo* n, jeżeli ich różnica $a - b$ jest podzielna przez n. Symbolicznie zapisuje się to jako

$$a \equiv b \; mod \; n.$$

Zapis ten nazywany jest kongruencją [3, s. 13].

Definicja 1.8. Dla każdej liczby naturalnej m zbiór

$$\phi(m) := \{k : 1 \leq k \leq m, (k, m) = 1\}$$

tworzy *zbiór reszt modulo m.*

Definicja 1.9. *Odwrotnością elementu a modulo n* jest taki element, oznaczany przez a^{-1}, że

$$aa^{-1} \equiv 1 \; mod \; n.$$

Liczba naturalna a jest odwracalna modulo n wtedy i tylko wtedy, gdy a i n są względnie pierwsze.

Twierdzenie 1.3. *(twierdzenie Wilsona, 1770 r.) Liczba naturalna $p > 1$ jest liczbą pierwszą wtedy i tylko wtedy, gdy*

$$(p-1)! + 1 \equiv 0 \ mod \ p.$$

Dowód. Na początek należy pokazać, że jeśli p jest liczbą pierwszą, to zachodzi powyższa kongruencja. Dla $p = 2$ implikacja jest spełniona, więc ograniczono się do liczb pierwszych nieparzystych. Ponieważ p jest liczbą pierwszą, więc każda z liczb $1, 2, 3, ..., p-2, p-1$ ma swoją odwrotność modulo p. Poszukuje się, dla jakich a spośród tych liczb zachodzi

$$a \equiv a^{-1} \ mod \ p,$$

tzn.

$$a^2 \equiv 1 \ mod \ p.$$

Ostatni warunek oznacza, że liczba p musi dzielić

$$a^2 - 1 = (a+1)(a-1).$$

Z Lematu 1.1 (Euklidesa) wynika, że p dzieli $a+1$ lub $a-1$, więc $a = p-1$ lub $a = 1$. Te dwie liczby na razie pominięto. Wówczas liczby $2, 3, ..., p-2$ dzielą się na pary, liczba i jej odwrotność modulo p, skąd wynika, że

$$2 \cdot 3 \cdot ... \cdot (p-2) \equiv 1 \ mod \ p.$$

Mnożąc obie strony powyższej kongruencji przez $p-1$ otrzymano

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

co kończy dowód pierwszej implikacji.

Teraz wystarczy pokazać implikację w drugą stronę. Należy zatem udowodnić, że jeżeli n jest liczbą złożoną to zachodzi

$$(n-1)! + 1 \not\equiv 0 \ mod \ n.$$

Jeśli n jest liczbą złożoną, to zachodzi $n = pq$. Założono, że $p < q$. Wówczas

$$(n-1)! = 1 \cdot 2 \cdot 3 \cdot ... \cdot (p-1) \cdot p \cdot ... \cdot (q-1) \cdot q \cdot ... \cdot (n-1),$$

więc $(n-1)!$ jest podzielne przez n. Stąd $(n-1)! + 1$ nie może być podzielne przez n. Wynika stąd, że

$$(n-1)! + 1 \not\equiv 0 \ mod \ n,$$

co kończy dowód.

■

Twierdzenie 1.4. *(małe twierdzenie Fermata, 1640 r.) Jeśli p jest liczą pierwszą, to dla każdej liczby całkowitej a niepodzielnej przez p zachodzi*

$$a^{p-1} \equiv 1 \ mod \ p.$$

Dowód. Jeśli liczba pierwsza p nie dzieli a, to ciąg reszt z dzielenia a przez p $1, 2, 3, ..., p-1$ oraz ciąg liczb

$$a, 2a, 3a, ..., (p-1)a \ mod \ p \tag{1.1}$$

różnią się jedynie kolejnością. Łatwo zauważyć, że jeśli p nie dzieli a, to niemożliwe jest, aby $ka \equiv 0 \ mod \ p$ dla $k = 1, 2, ..., p-1$. Z tego wynika, że wszystkie reszty $ka \ mod \ p$ są liczbami naturalnymi od 1 do $p-1$. Ponieważ liczb tych jest dokładnie $p-1$, więc należy pokazać, że są one parami różne.

Założono, że dla $i, j < p$, gdzie $i \neq j$ zachodzi $ai \equiv aj \ mod \ p$. Z treści twierdzenia wiadomo, że a nie jest podzielne przez p, więc obie strony można podzielić przez a, skąd otrzymano $i \equiv j \ mod \ p$. Obie liczby są z założenia mniejsze od p, więc $i = j$, co prowadzi do sprzeczności.

Jak wcześniej zauważono, ciąg liczb 1.1 modulo p to ciąg $1, 2, ..., p-1$, więc

$$a \cdot 2a \cdot 3a \cdot ... \cdot (p-1)a \equiv 1 \cdot 2 \cdot 3 \cdot ... \cdot (p-1) \ mod \ p,$$

a po przekształceniu

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

Z Twierdzenia 1.3 wiadomo, że liczba $(p-1)!$ jest względnie pierwsza z p, więc obie strony kongruencji można podzielić przez $(p-1)!$, co kończy dowód.

■

Definicja 1.10. Jeżeli n jest nieparzystą liczbą złożoną, a liczba b, względnie pierwsza z n, spełnia warunek

$$b^{n-1} \equiv 1 \mod n,$$

to n jest *liczbą pseudopierwszą* przy podstawie b.

1.5. Własności liczb pierwszych

Twierdzenie 1.5. *Istnieje nieskończenie wiele liczb pierwszych postaci $4k-1$.*

Dowód. Przypuśćmy przeciwnie, że $\{q_1, q_2, ..., q_r\}$ jest zbiorem wszystkich liczb pierwszych postaci $4k-1$. Zdefiniujmy $N = 4q_1q_2...q_r - 1$.

Ponieważ N jest liczbą nieparzystą, więc wszystkie dzielniki liczby N również są nieparzyste. Z kolei każda liczba nieparzysta ma postać $4k-1$ lub $4k+1$. Wiadomo także, że iloczyn liczb postaci $4k+1$ jest też liczbą postaci $4k+1$, więc N musi mieć dzielnik pierwszy q postaci $4k-1$. Jak wiadomo, $q_1 \nmid N$, $q_2 \nmid N$, ..., $q_r \nmid N$, skąd wynika, że $q \notin \{q_1, q_2, ..., q_r\}$, co jest sprzeczne z założeniem.

■

Twierdzenie 1.6. *Istnieje nieskończenie wiele liczb pierwszych postaci* $4k+1$.

Twierdzenie 1.7. *(Dirichlet) Jeśli* $m \in \mathbb{N}$, $a \in \mathbb{Z}$ *oraz* $(a, m) = 1$, *to w ciągu arytmetycznym* $a + mk$, *gdzie* $k = 1, 2, ...$, *istnieje nieskończenie wiele liczb pierwszych [2, s. 47].*

Definicja 1.11. Niech $n \in \mathbb{N}$. Liczbę w postaci

$$F_n = 2^{2^n} + 1, \tag{1.2}$$

gdzie $n \geq 1$, nazywa się *n-tą liczbą Fermata.*

Fermat obliczył, że $F_1 = 5$, $F_2 = 17$, $F_3 = 2575$, $F_4 = 65537$ są liczbami pierwszymi i założył, że każda liczba postaci 1.2 jest liczbą pierwszą. Hipotezę tą obalił Euler, który udowodnił, że F_5 jest podzielne przez 641.

Definicja 1.12. Liczby pierwsze p i $q = p + 2$ nazywane są *liczbami pierwszymi bliźniaczymi.*

Liczbami pierwszymi bliźniaczymi są np.: (3, 5), (11, 13), (1949, 1951), (4967, 4969). Największe znane dziś liczby bliźniacze, każda składająca się z 200 700 cyfr, to

$$3\,756\,801\,695\,685 \cdot 2^{666\,669} \pm 1,$$

znalezione w 2011 roku [5].

Definicja 1.13. *Liczba pierwsza Sophie Germain* to dowolna liczba pierwsza p, dla której liczba $2p + 1$ także jest liczbą pierwszą (np. 23, gdyż $2 \cdot 23 + 1 = 47$).

Największą znaną obecnie liczbą pierwszą Sophie Germain jest

$$18\,543\,637\,900\,515 \cdot 2^{666\,667} - 1,$$

a jej zapis dziesiętny wymaga 200701 cyfr. Została ona odkryta w 2012 przez Philipa Bliedunga.

Definicja 1.14. *Liczbami Mersenne'a* są liczby postaci $M_n = 2^n - 1$, $n \geq 1$. Jeżeli liczba Mersenne'a jest pierwsza to nazywa się *liczbą pierwszą Mersenne'a*.

Nie wiadomo, czy istnieje nieskończenie wiele liczb pierwszych Mersenne'a. Do 2017 roku znanych ich 49. Co więcej, dziesięć największych odkrytych do tej pory liczb pierwszych to właśnie liczby Mersenne'a. Zostały one przedstawione w Tabeli 1.2. Ich poszukiwaniem zajmuje się projekt GIMPS (ang. Great Internet Mersenne Prime Search), w którym mogą wziąć udział ochotnicy z całego świata.

Nr	Liczba pierwsza	Liczba cyfr	Data odkrycia	Odkrywcy
1	$2^{74\,207\,281} - 1$	22 338 618	2016-01-07	Cooper, GIMPS
2	$2^{57\,885\,161} - 1$	17 425 170	2013-01-25	Cooper, GIMPS
3	$2^{43\,112\,609} - 1$	12 978 189	2008-08-23	Edson Smith, GIMPS
4	$2^{42\,643\,801} - 1$	12 837 064	2009-06-04	Strindmo, GIMPS
5	$2^{37\,156\,667} - 1$	11 185 272	2008-09-06	H. Elvenich, GIMPS
6	$2^{32\,582\,657} - 1$	9 808 358	2006-09-04	Boone, Cooper, GIMPS
7	$2^{30\,402\,457} - 1$	9 152 052	2005-12-15	Boone, Cooper, GIMPS
8	$2^{25\,964\,951} - 1$	7 816 230	2005-02-18	Nowak, GIMPS
9	$2^{24\,036\,583} - 1$	7 235 733	2004-05-15	Findley, GIMPS
10	$2^{20\,996\,011} - 1$	6 320 430	2003-11-17	Michael Shafer, GIMPS

Tabela 1.2. Dziesięć największych znanych liczb pierwszych na 2017 rok [7].

1.6. Funkcja Eulera i "porządek mnożący"

Definicja 1.15. *Funkcja φ (Eulera)* lub tocjent to funkcja, która każdej liczbie naturalnej przypisuje liczbę liczb względnie pierwszych z nią i nie większych od niej. Jeśli $n = p_1^{m_1} \cdot ... \cdot p_k^{m_k}$ jest rozkładem liczby $n \in \mathbb{N}$ na czynniki pierwsze (gdzie $p_i \neq p_j$ dla różnych $i, j \in K = \{1, 2, ..., k\}$), to wtedy

$$\varphi(n) = \prod_{i \in K} \left(1 - \frac{1}{p_i}\right).$$

Funkcja Eulera jest bardzo prosta do zaimplementowania w środowisku Matlab, co zostało przedstawione poniżej.

```
1 function [m]=euler_phi(n)
2 m=0;
```

```
for  i=1:n
     if  gcd(i,n)==1
         m=m+1;
     end
end
end
```

Poniżej przedstawiono tabelę z wynikami i czasem działania powyższej funkcji dla liczb o różnej wielkości.

Lp.	Liczba n	Wartość funkcji $\varphi(n)$	Czas działania funkcji $[s]$
1	23	22	0.023398
2	100	40	0.029472
3	567	324	0.047793
4	1000	400	0.051363
5	12345	6576	0.496064
6	123456	41088	5.326313
7	1234567	1224720	65.953887

Tabela 1.3. Wartość funkcji $\varphi(n)$ dla liczb o różnej wielkości.

Definicja 1.16. W teorii liczb, dla danej liczby całkowitej a i dodatniej liczby całkowitej n, gdzie $NWD(a,n) = 1$, *porządkiem mnożącym (ang. multiplicative order)* modulo n jest najmniejsza całkowita liczba dodatnia k, dla której zachodzi

$$a^k \equiv 1 \; mod \; n.$$

Porządek modulo n oznacza się $ord_n(a)$ lub $o_n(a)$ [10].

Multiplicative order jest również bardzo prosty do zaimplementowania w środowisku Matlab, a kod tej funkcji został przedstawiony poniżej.

```
function [k] = mult_order(n,a)

if gcd(a,n)~=1
    disp('liczby nie są względnie pierwsze')
    return
end
```

```
7
8  k=1;
9  while  powermod(a,k,n) ~= 1
10       k=k+1;
11 end
12 end
```

Poniżej przedstawiono tabelę z wynikami i czasem działania powyższej funkcji.

Lp.	Liczba n	Liczba a	Wartość funkcji $o_n(a)$	Czas działania funkcji $[s]$
1	7	4	3	0.033575
2	4	7	2	0.025657
3	100	387	20	0.031364
4	167	379	166	0.036658
5	1000	37	100	0.035297

Tabela 1.4. Wartość funkcji $o_n(a)$ dla liczb o różnej wielkości.

2. Testy pierwszości liczb

Definicja 2.1. *Test pierwszości* jest to algorytm, który określa, czy zadana liczba jest liczbą pierwszą, czy złożoną.

Poniższy rozdział ma na celu przedstawienie algorytmów, które testują pierwszość zadanej liczby.

2.1. Testy deterministyczne

Testy deterministyczne to testy, które wydają certyfikat pierwszości, czyli ich wynik jest prawidłowy z prawdopodobieństwem 1. Zwracają odpowiedź, że podana na wejściu liczba n jest pierwsza wtedy i tylko wtedy, gdy naprawdę tak jest, czyli wynik działania testu jest jednoznaczny.

2.1.1. Metoda naiwna

Najprostszą deterministyczną metodą testowania pierwszości jest *metoda naiwna*. Algorytm działania jest bardzo prosty: dla danej liczby $n > 2$ wystarczy sprawdzić, czy dzieli się ona kolejno przez $2, 3, ..., n-1$. Jeżeli wśród tych liczb nie ma dzielnika liczby n, wtedy n jest pierwsza. Jeśli chociaż przez jedną z tych liczb n dzieli się bez reszty, wtedy n jest złożona.

Jak się okazuje, nie trzeba testować podzielności przez wszystkie liczby od 2 do $n-1$, wystarczy ograniczyć się do przedziału $2, 3, ..., \sqrt{n}$.

Co więcej, algorytm można ulepszyć dzieląc liczbę n jedynie przez wszystkie liczby pierwsze mniejsze lub równe $\sqrt{n}$. Listę takich liczb możemy wyznaczyć sitem Eratostenesa 1.4.

Warto zauważyć, że jeśli liczba n okaże się być złożona, to jednocześnie wyznaczany jest jej nietrywialny dzielnik.

Algorytm metody naiwnej w wersji udoskonalonej działa następująco:

1. Pobierz liczbę n.
2. Wyznacz zbiór p wszystkich liczb pierwszych mniejszych lub równych n.
3. Dziel liczbę n przez kolejne liczby ze zbioru p.
4. Jeśli n dzieli się przez przynajmniej jedną z liczb ze zbioru p bez reszty, wtedy n jest złożona.

5. Jeśli wśród liczb p nie ma dzielnika liczby n, wtedy n jest złożona.

Jest to algorytm deterministyczny, który uznaje pierwszość liczby ze stuprocentową pewnością. W związku z tym, że metoda jest bardzo dokładna, wymaga aż $O(\sqrt{n}\lg^2 n)$ kroków, więc jej efektywność spada dla bardzo dużych liczb.

Algorytm metody naiwnej w programie Matlab został przedstawiony poniżej.

```
function MetodaNaiwna(N)
tic;
  X=['Liczba ', num2str(N), ' jest liczbą złożoną.'];
  Y=['Liczba ', num2str(N), ' jest liczbą pierwszą.'];
  Z=['Liczba ', num2str(N) ,...
      ' nie jest ani pierwsza, ani złożona.'];
  if (N==2 || N==3)
      disp(Y)
      return
  elseif (N==1)
      disp(Z)
      return
  else
  [n,p] = sitoEratostenesa(N);
  lp = p(p<=sqrt(N));
  l = length(lp);
  m = ones(l,1);

  for i=1:l
      if mod(N, lp(i))==0
          disp(X)
          break
      else
          m(i)=0;
          i=i+1;
      end
  end
  if sum(m)==0
      disp(Y)
  end
```

```
31  toc;
32  end
```

Tabela 2.1 przedstawia wyniki i czas działania algorytmu dla liczb o różnej wielkości.

Lp.	Testowana liczba	Czas działania programu [s]	Liczba pierwsza/złożona
1	7	0.005050	pierwsza
2	23	0.004529	pierwsza
3	365	0.002450	złożona
4	5737	0.002264	pierwsza
5	99487	0.015185	pierwsza
6	99489	0.014734	złożona
7	100001	0.013290	złożona
8	12192109	9.409341	pierwsza
9	14476001	11.304216	pierwsza
10	100000001	215.084406	złożona

Tabela 2.1. Wyniki działania funkcji testującej pierwszość liczb metodą naiwną z sitem Eratostenesa.

Łatwo zauważyć, że dla niewielkich liczb czas działania jest bardzo krótki, ale dla liczb rzędu 10^8 sprawdzanie pierwszości trwa już kilka minut.

2.1.2. Test pierwszości AKS

Test pierwszości AKS (lub inaczej test pierwszości Agrawal-Kayal-Saxena lub cyklotomiczny test AKS) jest deterministycznym testem pierwszości liczb opracowanym przez trójkę indyjskich naukowców - Manindra Agrawala, Neeraja Kayala i Nitina Saxena z Indyjskiego Instytutu Technologii w Kanpurze. Test został po raz pierwszy opublikowany 6 sierpnia 2002 roku w artykule zatytułowanym *PRIMES is in P*. W 2006 roku autorzy otrzymali za to odkrycie dwie nagrody - Nagrodę Gödla, którą przyznaje się za osiągnięcia w dziedzinie informatyki teoretycznej, oraz nagrodę Fulkersona, przyznawaną za wybitne prace z zakresu matematyki dyskretnej.

Test AKS jest pierwszym opublikowanym algorytmem testującym czy dana liczba jest pierwsza, który jest jednocześnie szybki, jednoznaczny i bezwarunkowy.

Oznacza to, że algorytm ten zawsze zwróci poprawną odpowiedź (w przeciwieństwie do testów probabilistycznych), a jego działanie nie bazuje na żadnych nieudowodnionych hipotezach.

Test pierwszości AKS opiera się na pewnej tożsamości liczb pierwszych, która jest uogólnieniem małego twierdzenia Fermata 1.4.

Lemat 2.1. (małe twierdzenie Fermata dla wielomianów) Niech $a, n \in \mathbb{N}$ i $n \geq 2$ oraz $(a, n) = 1$. Wówczas n jest liczbą pierwszą wtedy i tylko wtedy, gdy

$$(X + a)^n = X^n + a \ mod \ n. \tag{2.1}$$

Dowód. Równanie 2.1 można zapisać w postaci

$$(X + a)^n - (X^n + a) = 0 \ mod \ n.$$

Dla $0 \leq i \leq n$ współczynnik przy X^i w wyrażeniu $((X + a)^n - (X^n + a))$ wynosi $\binom{n}{i} \ a^{n-i}$.

Załóżmy, że n jest liczbą pierwszą. Wtedy $\binom{n}{i} = 0 \ mod \ n$, a więc wszystkie współczynniki są zerowe.

Załóżmy, że n jest liczbą złożoną. Rozważmy liczbę pierwszą q, która jest dzielnikiem n i dla pewnego k niech $q^k \mid n$. Wtedy q^k nie dzieli $\binom{n}{q}$ i jest względnie pierwsza z a^{n-q}. Stąd współczynnik przy X^q jest niezerowy (mod n). Zatem wyrażenie $((X + a)^n - (X^n + a))$ nie jest tożsamościowo równe zero (mod n).

■

Powyższa tożsamość sugeruje prosty test pierwszości: pobieramy badane n, wybieramy a i sprawdzamy, czy kongruencja 2.1 jest spełniona. Jednak wymaga to czasu, ponieważ w najgorszym przypadku należy oszacować n współczynników po lewej stronie równania. Prostym sposobem na zmniejszenie liczby tych współczynników jest oszacowanie obu stron kongruencji 2.1 modulo wielomian postaci $X^r - 1$ dla odpowiednio wybranego małego r. Innymi słowy, sprawdzić, czy spełnione jest równanie

$$(X + a)^n = X^n + a \ (mod \ X^r - 1, n). \tag{2.2}$$

Z Lematu 2.1 wynika natychmiast, że wszystkie liczby pierwsze n spełniają równanie 2.2 dla wszystkich wartości a i r. Problem polega na tym, że niektóre liczby złożone n mogą również spełnić równanie dla kilku wartości a i r. Jednak możemy pokazać, że dla odpowiednio dobranego r, jeśli równanie 2.2 jest spełnione dla kilku a, to n musi być potęgą liczby pierwszej [9].

Algorytm działania metody AKS jest następujący:

1. Wybierz testowaną liczbę $n > 1$.
2. Jeśli istnieje takie $a \in \mathbb{N}$ i $b > 1$, że $n = a^b$, to liczba n jest złożona - stop.
3. Znajdź najmniejsze r takie, że $o_r(n) > (log_2 n)^2$.
4. Jeśli $1 < NWD(a, n) < n$ dla pewnego $a \leq r$, wtedy n jest złożona - stop.
5. Jeśli $n \leq t$, n jest pierwsza - stop.
6. Dla $a = 1$ do $\lfloor \sqrt{\varphi(r)} log_2 n \rfloor$ rób
   jeśli $((X + a)^n \neq X^n + a \ (mod \ X^r - 1, n))$, n jest złożona - stop.
7. n jest pierwsza.

Poniżej przedstawiono kod metody AKS zaimplementowany w środowisku Matlab.

```
function MetodaAKS(n)

for b=2:n-1
    if round(log(n)/log(b))==log(n)/log(b)
        disp('n jest złożona')
        return
    end
end

r=2;

if gcd(r,n)~=1
    disp('n jest złożona')
    return
end

k=mult_order(r,n);

while k<=(log2(n))^2
    if gcd(r,n)~=1
        disp('n jest złożona')
         return
    end
    k=mult_order(r,n);
    r=r+1;
```

```
26      if r>=n
27          disp('n jest pierwsza')
28          return
29      end
30  end
31
32  for a=2:min(r,n-1)
33      if mod(n,a)==0
34          disp('n jest złożona')
35          return
36      end
37  end
38
39  if n<=r
40      disp('n jest pierwsza')
41      return
42  end
43
44  m=euler_phi(r);
45  syms x
46
47  v=sym2poly(x^r-1);
48
49  for a=1:floor(sqrt(m)*log2(n))
50      [q1,w1]=deconv(sym2poly((x+a)^n),v);
51      [q2,w2]=deconv(sym2poly(x^n+a),v);
52      w1=mod(w1,n); w2=mod(w2,n);
53      if w1~=w2
54          disp('n jest złożona')
55          return
56      end
57  end
58  disp('n jest pierwsza')
59
60  end
```

Tabela poniżej przedstawia wyniki i czas działania testu pierwszości AKS.

Lp.	Liczba n	Pierwsza/złożona	Czas działania funkcji $[s]$
1	3	pierwsza	0.034859
2	23	pierwsza	0.047291
3	25	złożona	0.009026
4	123	złożona	0.045841
5	199	pierwsza	4.284376
6	1000	złożona	0.020794
7	1001	złożona	0.047622
8	1223	pierwsza	35.298198
9	3313	pierwsza	113.642901
10	3315	złożona	0.044053

Tabela 2.2. Wartość funkcji $\varphi(n)$ dla liczb o różnej wielkości.

Łatwo zauważyć, że algorytm radzi sobie bardzo szybko z liczbami złożonymi i niewielkimi liczbami pierwszymi, jednak już dla liczb pierwszych czterocyfrowych potrzebne jest ponad pół minuty na potwierdzenie ich pierwszości. Test AKS ma niestety znikomą wartość praktyczną, ponieważ złożoność tego algorytmu jest duża, a to oznacza, że liczba wykonywanych operacji rośnie bardzo szybko wraz ze wzrostem wielkości badanej liczby.

2.2. Testy probabilistyczne

Testy probabilistyczne (znane także jako testy randomizacyjne) są obecnie najczęściej stosowanymi i najbardziej efektywnymi testami pierwszości liczb, które w swoim działaniu używają losowości. Wykorzystuje się w nich losowo wygenerowane liczby z zadanego przedziału. Zależność od takich elementów powoduje, że algorytm może dać błędny wynik testu pierwszości, lecz prawdopodobieństwo takiego zdarzenia jest bardzo małe.

2.2.1. Test pierwszości Fermata

Zgodnie z małym twierdzeniem Fermata 1.4 dla liczb pierwszych p zachodzi kongruencja

$$a^{p-1} \equiv 1 \; mod \; p. \tag{2.3}$$

Test pierwszości Fermata to probabilistyczny test pierwszości, który opiera się na założeniu, że zachodzi wynikanie odwrotne do Twierdzenia 1.4, czyli jeśli prawdziwa jest kongruencja 2.3, to liczba p jest pierwsza z dużym prawdopodobieństwem.

Algorytm testu pierwszości Fermata dla zadanej liczby p wygląda następująco:

1. Wylosuj $a \in [2, p-2]$.
2. Oblicz $d = NWD(a, p)$. Jeśli $d > 1$ to p jest liczbą złożoną.
3. Sprawdź zachodzenie kongruencji $a^{p-1} \equiv 1 \ mod \ p$.
4. Jeśli zachodzi, to p może być liczbą pierwszą, w innym przypadku p jest liczbą złożoną.

Przy kilkukrotnym powtórzeniu losowania dla różnych liczb test Fermata daje bardzo wiarygodny wynik.

Poniżej przedstawiona została implementacja testu pierwszości Fermata w programie Matlab.

```
function testFermata(p,n)
    X=['Liczba ', num2str(p), ' jest liczbą złożoną.'];
    Y=['Liczba ', num2str(p), ' może być pierwsza.'];
    if (p<=0 || n<=0)
        disp('Liczby p i n muszą być większe od 0')
        return
    end
    if (p==2 || p==3)
        disp(Y)
        return
    end
    if n>(p-3)
        disp('Podaj mniejszą liczbę n')
        return
    end
    pierwsza=zeros(n,1);
    zlozona=zeros(n,1);
    a=randperm(p-3,n)+1;
    for i=1:n
        if gcd(a(i),p)>1
            zlozona(i)=1;
```

```
22            elseif powermod(a(i),p-1,p)~=1
23                zlozona(i)=1;
24            else
25                pierwsza(i)=1;
26            end
27        end
28        if sum(pierwsza)>=sum(zlozona)
29            disp(Y)
30            disp('Prawdopodobienstwo wynosi:')
31            disp(sum(pierwsza)/n*100)
32        else
33            disp(X)
34        end
35    end
```

Tabela 2.3 przedstawia wyniki działania powyższego kodu dla różnych liczb pierwszych i złożonych oraz czas działania algorytmu.

Lp.	Liczba p	Liczba powtórzeń n	Wynik	Czas działania funkcji $[s]$
1	5	2	pierwsza (100%)	0.069366
2	15	8	złożona (100%)	0.097156
3	47	18	pierwsza (100%)	0.102425
4	199	28	pierwsza (100%)	0.090658
5	1999	68	pierwsza (100%)	0.092615
6	6899	190	pierwsza (100%)	0.095254
7	94427	19	pierwsza (100%)	0.080347
8	94428	19	złożona (100%)	0.070723
9	15472147	30	pierwsza (100%)	0.071718
10	15472147	5530	pierwsza (100%)	0.733975
11	15472148	5530	złożona (100%)	0.575719

Tabela 2.3. Wyniki działania testu pierwszości Fermata dla liczb o różnej wielkości.

Łatwo zauważyć, że algorytm działa niezwykle szybko i zwraca stuprocentowo pewny wynik nawet dla bardzo dużych liczb. Czas działania zależy głównie od wyboru liczby powtórzeń n - im liczba n jest większa, tym czas działania algorytmu się

wydłuża. Jednak już dla niewielkich n wynik jest podawany z prawdopodobieństwem równym 1.

Trzeba zaznaczyć, że kongruencja 2.3 może zachodzić także w pewnych przypadkach, gdy p nie jest liczbą pierwszą. Liczby te omówione zostały poniżej [3, s. 49].

Definicja 2.2. *Liczby Carmichaela* to złożone liczby naturalne, dla których teza małego twierdzenia Fermata 1.4 jest prawdziwa. Liczba naturalna n jest liczbą Carmichaela wtedy i tylko wtedy, gdy jest liczbą złożoną i dla każdej liczby naturalnej a z przedziału $1 < a < n$, względnie pierwszej z n, liczba $a^{n-1} - 1$ jest podzielna przez n [6].

Najmniejszą liczbą Carmichaela jest $561 = 3 \cdot 11 \cdot 17$. Innymi z nich są na przykład $1105 = 5 \cdot 13 \cdot 17$, $1729 = 7 \cdot 13 \cdot 19$ lub $2465 = 5 \cdot 17 \cdot 29$.

Test Fermata zastosowany do liczby Carmichaela jest nieskuteczny, co pokazuje Tabela 2.4.

Lp.	L. Carmichaela	Liczba powtórzeń n	Wynik	Czas działania $[s]$
1	561	5	pierwsza (60%)	0.064905
2	561	15	pierwsza (73%)	0.097156
3	561	150	pierwsza (58%)	0.102425
4	1105	5	pierwsza (100%)	0.067776
5	1105	15	pierwsza (73%)	0.068222
6	1105	150	pierwsza (70.6%)	0.088589
7	1729	5	pierwsza (60%)	0.069128
8	1729	15	złożona (66.67%)	0.078923
9	1729	150	pierwsza (76.67%)	0.082983
10	2465	5	pierwsza (80%)	0.064879
11	2465	15	pierwsza (60%)	0.070719
12	2465	150	pierwsza (75.3%)	0.090890

Tabela 2.4. Wyniki działania testu pierwszości Fermata dla liczb Carmichaela.

Jak wynika z powyższej tabeli, liczby Carmichaela przechodzą test pierwszości Fermata z dużym prawdopodobieństwem.

2.2.2. Test pierwszości Lehmana

Test Lehmana jest kolejnym testem probabilistycznym pierwszości liczb. Algorytm testu Lehmana dla dużej liczby p wygląda następująco:

1. Wylosuj dużą liczbę $a < p$.
2. Oblicz $b = a^{(p-1)/2} \ mod \ p$.
3. Jeśli $b \equiv 1 \ mod \ p$ lub $b \equiv -1 \ mod \ p$ to p jest liczbą pierwszą z prawdopodobieństwem większym lub równym 50%. W przeciwnym wypadku p jest liczbą złożoną [1, s. 60].

Poniższy skrypt przedstawia algorytm Lehmana w programie Matlab.

```
function testLehmana(p)
X = ['Liczba ', num2str(p), ' jest liczbą złożoną.'];
Y = ['Liczba ', num2str(p) ,...
    ' może być pierwsza z prawdopodobieństwem >= 50%.'];
if mod(p,2)==0
    disp(X)
    return
end
a = randi([(p-1)/2,p-1]);
b = powermod(a,(p-1)/2,p);
if (mod(b,p)==1 || mod(b,p)-p==-1)
    disp(Y)
else disp(X)
end
end
```

Wyniki działania programu przedstawione zostały w Tabeli 2.5.

Lp.	Liczba p	Wynik	Czas działania funkcji $[s]$
1	5	pierwsza ($\geq 50\%$)	0.023871
2	47	pierwsza ($\geq 50\%$)	0.032829
3	199	pierwsza ($\geq 50\%$)	0.031084
4	1999	pierwsza ($\geq 50\%$)	0.026158
5	6899	pierwsza ($\geq 50\%$)	0.029798
6	94427	pierwsza ($\geq 50\%$)	0.023563
7	94428	złożona (100%)	0.015226
8	15472145	złożona (100%)	0.027043
9	15472147	pierwsza ($\geq 50\%$)	0.026316
10	15485857	pierwsza ($\geq 50\%$)	0.024070

Tabela 2.5. Wyniki działania testu pierwszości Lehmana dla liczb o różnej wielkości.

Algorytm Lehmana jest bardzo szybki i zwraca prawidłowe wyniki dla liczb o różnej wielkości.

2.2.3. Test pierwszości Solovaya-Strassena

Kolejnym probabilistycznym testem pierwszości jest test Solovaya-Strassena, opracowany przez dwóch matematyków - amerykańskiego profesora Roberta M. Solovaya i profesora Volkera Strassena z Niemiec. Do wprowadzenia algorytmu testowania niezbędna jest definicja symbolu Legendre'a.

Definicja 2.3. Niech p będzie liczbą pierwszą większą od 2, a a liczbą całkowitą. *Symbol Legendre'a* $\left(\frac{a}{p}\right)$ definiuje się następująco

$$\left(\frac{a}{p}\right) = \begin{cases} 0, \text{ jeśli } a \text{ jest wielokrotnością liczby } p, \\ 1, \text{jeśli istnieje } b \text{ takie, że } b^2 = a \ mod \ p, \\ -1, \text{jeśli nie istnieje żadne } b \text{ takie, że } b^2 = a \ mod \ p. \end{cases}$$

Symbol Legendre'a można obliczyć następującym wzorem

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

Symbol Legendre'a może być uogólniony do *symbolu Jacobiego* $\left(\frac{a}{n}\right)$, gdzie n jest dowolną liczbą nieparzystą, która ma rozkład na czynniki pierwsze

$$n = p_1^{m_1} \cdot p_2^{m_2} \cdot ... \cdot p_k^{m_k}.$$

Wtedy symbol Jacobiego można zapisać w postaci

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p_1}\right)^{m_1} \cdot \left(\frac{a}{p_2}\right)^{m_2} \cdot \ldots \cdot \left(\frac{a}{p_k}\right)^{m_k}.$$

Jeżeli n jest liczbą pierwszą to symbol Jacobiego i symbol Legendre'a są sobie równe.

Definicja 2.4. Jeśli n jest liczbą złożoną nieparzystą, a liczba b, względnie pierwsza z n, spełnia warunek

$$b^{\frac{n-1}{2}} \equiv \left(\frac{b}{n}\right) \mod n, \tag{2.4}$$

to n jest *pseudopierwszą liczbą Eulera* przy podstawie b.

Test pierwszości liczby n algorytmem Solovaya-Strassena przebiega następująco

1. Losujemy $b \in \{2, ..., n-2\}$.
2. Obliczamy $r = b^{\frac{n-1}{2}} \mod n$. Jeśli $r \neq \pm 1$, to n jest liczbą złożoną.
3. Obliczamy $s = \left(\frac{b}{n}\right)$. Jeśli $r \neq s$, to n jest złożona.

Jeżeli po zakończeniu algorytmu nie ma odpowiedzi "n jest złożona", to test należy powtórzyć. Po k-krotnym powtórzeniu testu i stwierdzeniu, że n jest liczbą pseudopierwszą Eulera przy każdej z wybranych podstaw b, prawdopodobieństwo, że liczba n jest jednak złożona wynosi $\frac{1}{2^k}$.

Skrypt jednorazowego przejścia algorytmu testu pierwszości Solovaya-Strassena w programie Matlab został przedstawiony poniżej.

```
function testSolovaya(n)
    X = ['Liczba ', num2str(n), ' jest liczbą złożoną.'];
    Y = ['Liczba ', num2str(n), ' może być pierwsza.'];
    if mod(n,2)==0
        disp(X)
        return
    end
    b = randi([2,n-2]);
    r = powermod(b,(n-1)/2,n);
    if (r ~= 1 && r-n ~= -1)
        disp(X)
        return
    else
        s = 1;
```

```
15          f = factor(n);
16          for i = 1:length(f)
17              s = s*powermod(b,(f(i)-1)/2,n);
18          end
19          if r~=s
20              disp(X)
21              return
22          end
23      end
24      disp(Y)
25  end
```

Tabela 2.6 przedstawia wyniki otrzymane za pomocą powyższego algorytmu i czas działania funkcji dla każdej operacji.

Lp.	Liczba n	Wynik	Czas działania funkcji $[s]$
1	5	pierwsza	0.034387
2	47	pierwsza	0.032534
3	199	pierwsza	0.039209
4	1999	pierwsza	0.032661
5	6899	pierwsza	0.036613
6	94427	pierwsza	0.034677
7	94428	złożona	0.017758
8	15472145	złożona	0.027369
9	15472147	pierwsza	0.056256
10	15485857	pierwsza	0.033962

Tabela 2.6. Wyniki działania testu pierwszości Solovaya-Strassena dla liczb o różnej wielkości.

2.2.4. Test pierwszości Millera-Rabina

Test pierwszości Millera-Rabina jest kolejnym testem probabilistycznym, który z pewnym prawdopodobieństwem określa, czy dana liczba jest pierwsza, czy złożona. Oryginalna wersja tego testu, stworzona przez profesora Gary'ego L. Millera, jest testem deterministycznym, lecz jego prawidłowość bazuje na uogólnionej hipotezie

Riemanna, która nie została dotychczas udowodniona. W 1975 roku kryptolog Michael O. Rabin przekształcił algorytm Millera do algorytmu probabilistycznego.

Fakt 2.1. Niech n będzie nieparzystą liczbą pierwszą i niech $n-1 = 2^s \cdot t$, gdzie $2 \nmid t$. Niech $b \in \phi(n)$. Wtedy albo $b^t \equiv 1 \pmod{n}$, albo $b^{2^r t} \equiv -1 \pmod{n}$ dla pewnego $r \in \{0, 1, ..., s-1\}$.

Definicja 2.5. Niech n będzie nieparzystą liczbą złożoną i niech $n-1 = 2^s \cdot t$, gdzie $2 \nmid t$. Niech $b \in \phi(n)$. Jeżeli

- $b^t \equiv 1 \pmod{n}$ lub
- $b^{2^r t} \equiv -1 \pmod{n}$ dla pewnego $r \in \{0, 1, ..., s-1\}$

to liczbę n nazywamy *liczbą silnie pseudopierwszą* przy podstawie b.

Fakt 2.2. Jeśli n jest nieparzystą liczbą złożoną, to jest silnie pseudopierwsza przy podstawie b dla co najwyżej 25% wszystkich podstaw b takich, że $0 < b < n$.

Algorytm działania testu pierwszości Millera-Rabina przebiega następująco:

1. Obliczamy $n-1 = 2^s \cdot t$, gdzie t jest liczbą nieparzystą.
2. Losujemy $b \in \{2, ..., n-2\}$.
3. Obliczamy $a = b^t \bmod n$. Jeśli $a = \pm 1$, to n może być pierwsza.
4. Obliczamy $a = b^{2^r t} \pmod{n}$ dla $0 < r < s$. Jeśli dla pewnego r otrzymamy $a = -1$, to n może być pierwsza, w przeciwnym razie n jest złożona.

Po otrzymaniu odpowiedzi "n jest pierwsza" powtarzamy kroki algorytmu 2-4.

Skrypt testu pierwszości Millera-Rabina przedstawiony został poniżej.

```
function TestMilleraRabina(n)
    X=['Liczba ', num2str(n), ' jest liczbą złożoną.'];
    Y=['Liczba ', num2str(n), ' może być pierwsza.'];
    if (n==2 || n==3)
        disp(Y)
        return
    end
    if mod(n,2) == 0
        disp(X)
        return
    end
    f = factor(n-1);
```

```
13      s = 0;
14      for i = 1:length(f)
15          if f(i) == 2
16              s = s+1;
17          end
18      end
19      t = 1;
20      for j = s+1:length(f)
21         t = t*f(j);
22      end
23      b = randi([2,n-2]);
24      a1 = powermod(b,t,n);
25      if (a1 == 1 || a1 == n-1)
26          disp(Y)
27          return
28      end
29      for r = 1:s
30          a2(r) = powermod(b,2^(r-1)*t,n);
31      end
32      for i = 1:length(a2)
33          if a2(i) == n-1
34              disp(Y)
35              return
36          end
37      end
38      disp(X)
39  end
```

Tabela 2.7 przedstawia wyniki otrzymane za pomocą powyższego algorytmu i czas działania funkcji dla każdej operacji.

Lp.	Liczba n	Wynik	Czas działania funkcji $[s]$
1	5	pierwsza	0.036212
2	47	pierwsza	0.036644
3	199	pierwsza	0.038669
4	1999	pierwsza	0.037960
5	6899	pierwsza	0.037959
6	94427	pierwsza	0.040197
7	94428	złożona	0.018025
8	15472145	złożona	0.049525
9	15472147	pierwsza	0.035766
10	15485857	pierwsza	0.045164

Tabela 2.7. Wyniki działania testu pierwszości Millera-Rabina dla liczb o różnej wielkości.

2.3. Chiński test pierwszości

2.4. Test pierwszości APR

2.5. Test pierwszości ECPP

2.6. Test Lucasa-Lehmera

3. Zastosowania w kryptografii

Bibliografia

[1] M. Karbowski, *Podstawy kryptografii*, Helion, Gliwice, (2006).

[2] W. Marzantowicz, P. Zarzycki, *Elementarna teoria liczb*, Wydawnictwo Naukowe PWN, Warszawa, (2006).

[3] M. Zakrzewski, *Teoria liczb*, Wydanie I. Oficyna Wydawnicza GiS, Wrocław, (2017).

[4] https://en.wikipedia.org/wiki/Sieve_of_Atkin

[5] https://pl.wikipedia.org/wiki/Liczby_bli%C5%BAniacze

[6] https://pl.wikipedia.org/wiki/Liczby_Carmichaela

[7] https://www.mersenne.org/primes/

[8] https://en.wikipedia.org/wiki/Ulam_spiral

[9] https://www.cse.iitk.ac.in/users/manindra/algebra/primality_v6.pdf

[10] https://en.wikipedia.org/wiki/Multiplicative_order