Uniwersytet Warszawski
Wydział Matematyki, Informatyki i Mechaniki

Grzegorz Chojnacki
Nr albumu: 159128

Deterministyczne testy pierwszości

Praca magisterska
na kierunku INFORMATYKA

Praca wykonana pod kierunkiem
dr Jacek Pomykała

Maj 2002

Pracę przedkładam do oceny

Data Podpis autora pracy:

Praca jest gotowa do oceny przez recenzenta

Data Podpis kierującego pracą:

Streszczenie

W pracy przedstawiono ogólny stan wiedzy o problemie testowania pierwszości. Wskazane są różne paradygmaty tworzenia takich testów wraz z konkretnymi przykładami. Głównym celem pracy jest przedstawienie istotnych argumentów w dyskusji, dlaczego projektowania efektywnych testów deterministycznych stanowi tak skomplikowane i nierozwiązane do dziś zagadnienie. Naszkicowane są pewne ogólne pomysły, których dopracowanie, bądź powiązanie ze sobą, może przyczynić się do postępu w tej dziedzinie. Zagadnienie obliczania funkcji charakterystycznej zbioru PIERWSZE rozważane jest w dwóch aspektach: od strony algorytmiki i jej osiągnięć w tym zakresie, oraz od strony teorii złożoności, która zajmuje się głównie badaniem trudności i wzajemnych zależności konkretnych problemów.

Słowa kluczowe

liczby pierwsze, testowanie pierwszości, dowodzenie pierwszości, dowodzenie złożoności, certyfikaty pierwszości, generowanie liczb pierwszych, problem dzielników, faktoryzacja, symbole Legendre'a, symbole Jacobiego, sumy Jacobiego

Klasyfikacja tematyczna

11A51 Factorization, primality
11A05 Multiplicative structure, Euclidean algorithm, greatest common divisors
11A07 Congruences, primitive roots, residue systems
11Y40 Algebraic number theory computation
68W05 Nonnumerical Algorithms and Problems

Spis treści

Wprowadzenie

Praca składa się z 5 rozdziałów.

W rozdziale pierwszym opisano autorską notację, która będzie używana w dalszej części pracy. Znajdują się tam definicje podstawowych pojęć oraz kilka faktów teorioliczbowych zapisanych przy użyciu wspomnianej notacji.

Drugi rozdział stanowi przypomnienie zagadnień teorii złożoności obliczeniowej dokładnie w takim zakresie, jaki jest wymagany do czytania tej pracy. W dalszej części podane są szacowania złożoności podstawowych działań arytmetycznych, na które będę dalej powoływać się przy analizie konkretnych algorytmów.

Trzeci rozdział przedstawia znane zagadnienia teorioliczbowe, mające związek z liczbami pierwszymi. W dalszej części rozdziału omówione są funkcje na liczbach naturalnych oraz ich własności. Wiedza na temat zachowania tych funkcji będzie potrzebna i wykorzystywana w kolejnym rozdziale pracy.

Pierwsze trzy rozdziały mają na celu zebranie terminologii, podstawowej wiedzy z teorii liczb, algorytmiki i teorii złożoności. Wyposażeni w tę wiedzę możemy przystąpić do najistotniejszej części tej pracy.

W czwartym rozdziale koncentrujemy się już tylko i wyłacznie na problemie testowania pierwszości. Po krótkim wstępie historycznym omówione są pewne fakty teoretyczne dotyczące języka PIERWSZE. Dalej wskazane są najważniejsze metodologie i najciekawsze pomysły dotyczące konstruowania testów pierwszości. W podrozdziale dotyczącym testów warunkowych przedstawię własne usprawnienia do algorytmu Fellowsa-Koblitza, które poprawiają oszacowanie na złożoność tego algorytmu oraz stanowią bardzo istotne wskazówki przy jego praktycznej implementacji. Na koniec omówione jest istotne w praktyce zagadnienie generowania dużych liczb pierwszych. Przedstawiony jest ciekawy i chyba mało znany rezultat dotyczący istnienia nieskończonego zbioru liczb pierwszych z szybkim testem pierwszości.

Ostatni rozdział ma na celu podsumowanie aktualnego stanu wiedzy na temat omawianego zagadnienia. Jest tam zaprezentowana pewna ogólna idea, na której oparty jest najszybszy(asymptotycznie) znany deterministyczny test pierwszości. Okazuje się, że implementacja tego testu wykorzystuje kilka z pomysłów szczegółowo omawianych w rozdziale 4.

Rozdział 1

Wstęp

W tym rozdziale przedstawię definicje podstawowych pojęć, które bedą wykorzystywane w dalszej części pracy. Udowodnię też kilka znanych twierdzeń teorioliczbowych, które pozwolą lepiej oswoić się z wprowadzoną notacją.

Definicja 1.0.1 *Relację $R \subset \mathcal{N} \times \mathcal{N}$, która jest:*

(1) zwrotna: $a \in \mathcal{N} \Longrightarrow (a,a) \in R$

(2) antysymetryczna: $(a,b) \in R \wedge (b,a) \in R \Longrightarrow a = b$

(3) $(a,b) \in R \wedge (a,c) \in R \wedge b > c \Longrightarrow (a, b \pm c) \in R$

będziemy oznaczać, jako $|$.

Definicja 1.0.2 *Relację* $R = \{(a,b) : a \mid b \vee b \mid a\}$ *będziemy oznaczać* $\|$.

Definicja 1.0.3 $a^{\|} := \{b : a \parallel b\}$

Definicja 1.0.4 $a_{\|} := \{b : b \leq a \wedge a \parallel b\}$

Definicja 1.0.5 $NWD(a,b){:=}min(a_{\|} \cap b_{\|})$

Definicja 1.0.6 *Relację $R \subset \mathcal{N} \times \mathcal{N}$, która jest:*

(1) $a \in \mathcal{N} \iff (1,a) \in R$

(2) symetryczna: $(a,b) \in R \Longrightarrow (b,a) \in R$

(3) $(a,b) \in R, a < b \iff (a, b \pm a) \in R$

(4) jeśli $(a,bc) \in R \Longrightarrow (a,b) \in R$

(5) $(a,a) \in R \Longrightarrow a = 1$

będziemy oznaczać, jako $\perp$.

Definicja 1.0.7 $a^{\perp} := \{b : a \perp b\}$

Definicja 1.0.8 $a_{\perp} := \{b : b \leq a \wedge a \perp b\}$

Definicja 1.0.9 *a jest* ***pierwsze*** $\iff a^{\perp} \cup a^{\|} = \mathcal{N}$

Liczby pierwsze są to więc pewne specyficzne liczby naturalne. One i tylko one wśród liczb naturalnych mają tę własność, że dla ustalonej liczby pierwszej p każda liczba naturalna jest do niej albo równoległa, albo prostopadła.

Twierdzenie 1.1 *Definicja (1.0.1) jest poprawna, tzn. istnieje dokładnie jedna relacja $|$ spełniająca warunki z definicji (1.0.1). Jest to znana z arytmetyki relacja podzielności.*

Dowód:

1. *jeśli $a \mid b$, to $(a, b) \in R$*
   (a) *dla a=b, $(a, b) \in R$ z warunku (1);*

   (b) *dla $a < b$, b=ka i k>1, jeśli więc w warunku (3) przyjmiemy $a = a$, $b = a$, $c = a$, otrzymamy $(a, 2a) \in R$, iterując dalej warunek (3) uzyskamy $(a, ka) \in R$.*
2. *jeśli $(a, b) \in R$, to $a \mid b$*
   (a) *dla $a = b$ podzielność jest oczywista;*

   (b) *dla $a > b$, to a=kb+l gdzie $l \in \{0, 1, \ldots, b-1\}$, z warunku (3) otrzymujemy, że $(a, l) \in R$, jeśli l>0, lub $b \mid a$. Ta druga ewentualność nie jest jednak możliwa, bo uzyskalibyśmy sprzeczność z warunkiem (2), jeśli otrzymaliśmy l>0 to proszę zauważyć, że operację możemy powtarzać uzyskując coraz mniejsze l. Po skonczonej liczbie powtórek uzyskamy dzielnik(w skrajnym przypadku dojdziemy do 1-ki), czyli sprzeczność;*

   (c) *dla $a < b$, mamy b=ka+l, gdzie $l \in \{0, 1, \ldots, a-1\}$, ale l musi być równe 0, bo wpp. analogicznie jak wyżej otrzymamy $(a, l) \in R$, ale a>l, co jak się wcześniej okazało prowadzi do sprzeczności.*

   We wszystkich więc przypadkach jeśli $(a, b) \in R$, to $a \mid b$.•

Relacja podzielności jest więc pewną bardzo specyficzną relacją dwuargumentową na liczbach naturalnych i jak się okazało jedyną spełniającą warunki, które od niej wymagaliśmy. Bardzo łatwo(efektywnie) można odpowiedzieć na pytanie, czy $a \mid b$. Okazuje się jednak, że dla danego n wyliczenie elementów zbioru $n_{||}$ nie jest już tak proste(konkretniej nie jest znany efektywny algorytm wyliczania takiego zbioru). Problem ten z pozoru banalny ma zasadnicze znaczenie przy konstrukcji paru używanych w praktyce systemów kryptograficznych.

Twierdzenie 1.2 *Definicja (1.0.6) jest poprawna, tzn. istnieje dokładnie jedna relacja $|$ spełniająca warunki z definicji (1.0.6). Jest to relacja względnej pierwszości.*

Dowód:

1. *jeśli NWD(a,b)=1, to $(a, b) \in R$*

   Z warunku NWD(a,b)=1 wynika, że istnieją liczby naturalne x,y: ax-by=1. Z warunków (1) i (2) $(b, 1) \in R$, więc także $(b, ax - by) \in R$. Korzystając y razy z warunku (3) otrzymujemy $(b, ax) \in R$. Teraz stosując warunek (4) dostajemy $(b, a) \in R$, czyli także $(a, b) \in R$.

2. *jeśli $(a, b) \in R$, to NWD(a,b)=1*

   Niech x=NWD(a,b), k=a/x, l=b/x, wtedy mamy $(kx, lx) \in R$, jeśli k=l, to z warunku (5) dostajemy x=1, wpp. bez straty ogólności możemy przyjąć k<l(jeśli jest odwrotnie to zamieniamy miejscami, relacja jest symetryczna). Korzystając teraz z (3) dostajemy $(kx, (l - k)x) \in R$. Taki proces kontynuujemy tak długo, aż otrzymamy równe liczby, musi to kiedyś nastąpić, bo przy każdym kroku zmniejszamy jedną liczbę. Na końcu uzyskujemy dwie równe liczby podzielne przez x, z punktu (5) wynika, że muszą to być jedynki, czyli także x=1 •

Jak się okaże w dalszej części pracy łatwo można stwierdzić dla danych a i b, czy $a \perp b$, jednak problem efektywnego wyliczania zbioru $n_\perp$ nie został rozwiązany do dziś dnia. Nie wiadomo nawet, jak szybko policzyć $|n_\perp|$. Tak jak i obie relacje mają ze sobą wiele wspólnego, tak i trudne obliczeniowo problemy ich dotyczące okażą się również ze sobą powiązane.

Twierdzenie 1.3 $a \in \mathcal{N} \wedge a \neq 1 \Longrightarrow a^\perp \cap a^{||} = \{1\}$

Dowód:

jeśli $b > 1 \wedge b \in a^{||} \Longrightarrow b \notin a^\perp$, czyli dla dowolnego b>1 $b \notin (a^\perp \cap a^{||})$
łatwo sprawdzić, że $1 \in a^\perp \wedge 1 \in a^{||}$, co kończy dowód twierdzenia •

Widać więc pewną istotną zależność pomiędzy relacjami równoległości i prostopadłości. Dla danej liczby n, nie istnieje $a > 1$, które jest równocześnie prostopadłe i równolegle do n. Odpowiada to naszej intuicji geometrycznej. Przecież jeśli weźmiemy na płaszczyźnie wektor różny od zerowego, to jedynym wektorem, który będzie do niego równocześnie prostopadły i równoległy będzie właśnie wektor zerowy. Tutaj jego rolę pełni jedynka.

Twierdzenie 1.4 (Eulera) $a \perp n \Longrightarrow n \mid (a^{|n_\perp|} - 1)$

Dowód:

ponieważ $a \perp n$, to $n_\perp =_n \{ab : b \in a_\perp\}$ (gdzie relacja $=_n$ jest zdefiniowana następująco: $\mathcal{A} =_n \mathcal{B} \iff \{a \ MOD \ n : a \in \mathcal{A}\} = \{b \ MOD \ n : b \in \mathcal{B}\}$)

mnożymy elementy obu zbiorów i przyrównujemy do siebie oba iloczyny modulo n, otrzymujemy:

$$n \mid (\prod_{i \in a_\perp} ai - \prod_{i \in a_\perp} i)$$

skąd skracając otrzymujemy $n \mid (a^{|n_\perp|} - 1)$.$\bullet$

Lemat 1 *Liczba* $n \in \mathcal{N}$ *jest pierwsza* $\iff n_\perp = \{1, 2, \ldots, n-1\}$.

Dowód:

$\Longleftarrow$

Z własności (3) relacji $\perp$ *wynika, że* $n^\perp = \{a : (a, n) \notin \|\}$. *Teraz wystarczy zauważyć, że* $n^\| = \{kn : k \in \mathcal{N}\}$*(wynika to z tw. (1.3)). Teraz widać, że*

$$n^\perp \cup n^\| = \mathcal{N},$$

skąd n jest pierwsze.

$\Longrightarrow$

Z definicji liczby pierwszej $n^\perp \cup n^\| = \mathcal{N}$, *weźmy dowolne* $a < n$.
Gdyby teraz $n \mid (n + a)$, *to także* $n \mid a$, *sprzeczność bo* $a < n$. *Wiemy więc, że* $(n + a) \notin n^\|$. *Z definicji liczby pierwszej* $(a + n) \in n^\perp$, *czyli* $n \perp (a + n)$. *Z własności prostopadłości otrzymujemy* $n \perp a$. *Liczba n była dowolna, więc* $n_\perp = \{1, 2, \ldots, n-1\}$. $\bullet$

Lemat 2 *Liczba* $n \in \mathcal{N}$ *jest pierwsza* $\iff n_\| = \{1, n\}$.

Dowód:

$\Longleftarrow$

Jeśli n jest pierwsza, to korzystając z poprzedniego lematu oraz twierdzenia (1.3) $n_\| = \{1, n\}$.

$\Longleftarrow$

Dla liczby złożonej n (z definicji liczby pierwszej) oraz z poprzedniego lematu istnieje liczba naturalna $d < n$, *taka że* $d \notin \|$ *oraz* $d \notin \perp$.
Wynika stąd, że $d > NWD(d, n) > 1$. *Stąd otrzymujemy, że dla pewnego c, spełniającego* $1 < c < d < n$ *zachodzi* $c \mid n$, *czyli zbiór* $n_\|$ *musi mieć co najmniej trzy elementy. Jeśli jest on dwuelementowy, to n musi być pierwsze.*$\bullet$

Twierdzenie 1.5 (małe tw. Fermata) *Jeśli p jest pierwsze i* $(a, p) \notin \| \Longrightarrow (a^{p-1} - 1) \| p$.

Dowód:

*Liczby, p i a spełniają założenia **tw. Eulera**, bo* $a \notin p^\|$ *więc z definicji pierwszości* $a \in p^\perp$. *Dostajemy w ten sposób* $(a^{|p_\perp|} - 1) \| p$, *ale z lematu (1) wiemy, że* $|p_\perp| = p - 1$, *skąd otrzymujemy* $(a^{p-1} - 1) \| p$. $\bullet$

Kolejne twierdzenie jest o postaci liczb prostopadłych do danej.

Twierdzenie 1.6 *$A \perp B$, $a \perp B$, $b \perp A$, $aA > bB \iff (aA - bB) \perp AB$.*

Dowód:

$\Longleftarrow$

Z warunku (4) na prostopadłość otrzymujemy: $(aA - bB) \perp A$, $(aA - bB) \perp B$. Teraz stosując odpowiednią ilość razy warunek (3) uzyskujemy $aA \perp B$, skąd: $a \perp B$ i $A \perp B$. Gdyby $(b, A) \notin \perp$ wówczas $(bB, A) \notin \perp$ oraz $(aA - bB, A) \notin \perp$, co daje $(aA - bB, AB) \notin \perp$ czyli sprzeczność z założeniem.

$\Longrightarrow$

Gdyby $(aA - bB, AB) \notin \perp$ wówczas $(A, B) \notin \perp$ lub $(aA, B) \notin \perp$ lub $(bB, A) \notin \perp$. Co w każdym przypadku daje sprzeczność z założeniem i kończy dowód.$\bullet$

Okazuje się, że wskazana wyżej postać liczb prostopadłych jest jednoznaczna.

Twierdzenie 1.7 *Jeśli $A \perp B$ i $c \in (AB)_{\perp}$, to istnieje dokładnie jedna para a,b spełniająca: $a \in A_{\perp}$, $b \in B_{\perp}$, $|aA - bB| = c$.*

Definicja 1.0.10 *Wspólną wielokrotnością liczb $a, b \in \mathcal{N}$ będziemy nazywać liczbę $c \in \mathcal{N}$, taką że $a \mid c$ i $b \mid c$.*

Definicja 1.0.11 *Najmniejszą wspólną wielokrotnością liczb $a, b \in \mathcal{N}$, będziemy nazywać najmniejszą liczbę, która jest wspólną wielokrotnością a i b. Oznaczać ją będziemy NWW(a,b).*

Lemat 3 *Dla ustalonych liczb $a, b \in \mathcal{N}$ NWW(a,b) jest dzielnikiem każdej wspólnej wielokrotności liczb a i b.*

Dowód:

Niech W będzie pewną wspólną wielokrotnością a i b, natomiast N=NWW(a,b). Gdyby N nie dzieliło W, wtedy dla pewnych k,r: $k \geq 1$, $1 \leq rN$ mielibyśmy W=kN+r. Ponieważ $a \mid N$ oraz $a \mid W$, to $a \mid (W - kN)$, czyli $a \mid r$. Analogicznie $b \mid r$, skąd otrzymujemy, że r jest wspólną wielokrotnością a i b. Daje to sprzeczność, gdyż r<N i kończy dowód lematu.$\bullet$

Lemat 4 *Jeśli $a \perp b$, to NWW(a,b)=ab.*

Dowód:

Niech W=ab(wspólna wielokrotność a i b), N=NWW(a,b). Ponieważ N jest wielokrotnością a i b, to N=ka, N=rb. Równocześnie z poprzedniego lematu W=tN. Dalej mamy W=tka=trb=ab, skąd b=tk i a=tr. Ale $a \perp b$, więc musi być t=1, czyli W=N.$\bullet$

Twierdzenie 1.8 (zasadnicze twierdzenie arytmetyki) *Jeśi $c \mid ab$ i $c \perp a$, to $c \mid b$.*

Dowód:

Ponieważ $c|ab$, $a|ab$ i $a \perp c$, to z wcześniejszego lematu $ac|ab$(ac=NWW(a,c), ab jest wspólną wielokrotnością a i c). Mamy więc ab=kac, czyli b=kc, skąd $c|b$.$\bullet$

Twierdzenie 1.9 (fundamentalne twierdzenie arytmetyki) *Każdą liczbę naturalną $n>1$ można wyrazić jako iloczyn różnych liczb pierwszych w dodatnich potęgach*

$$n = p_1^{e_1} p_2^{e_2} \dots p_k^{e_k}$$

z dokładnością do kolejności czynników pierwszych, takie przedstawienie jest jednoznaczne.

Dowód:

Niech $m > 1$ będzie najmniejszą liczbą naturalną, która nie jest iloczynem liczb pierwszych. Liczba m musi być w takim razie złożona. Korzystając z lematu (2) musi istnieć $d \mid m$, $1 < d < m$, czyli $m = cd$, gdzie $1 < c < m$. Liczby c i d wobec minimalności m są iloczynami liczb pierwszych, czyli m również. Oznacza to, że dowolna liczba naturalna $n > 1$ rozkłada się na iloczyn liczb pierwszych. Pozostaje wykazać, że taki rozkład jest jednoznaczny. Przyjmijmy, że n ma rozkład:

$$n = p_1^{e_1} p_2^{e_2} \dots p_k^{e_k}.$$

Gdyby istniał inny rozkład na czynniki, wtedy oznaczałoby to, że dla pewnej liczby pierwszej q, takiej że $q \neq p_i$, dla $i \in \{1, 2, \dots, k\}$ $q \mid n$. Równocześnie q nie jest dzielnikiem żadnej z liczb p_i (wszystkie ewentualne dzielniki określa lemat (2)). Stosując twierdzenie (1.8) dla dzielnika q liczby n, który jest jednocześnie względnie pierwszy z każdym z czynników rozkładu, uzyskujemy sprzeczność. Oznacza to, że inny, istotnie różniący się rozkład n na czynniki pierwsze nie istnieje.•

W niniejszej pracy zdecydowałem się na wprowadzenie nieco odmiennego języka niż ten, który można spotkać w literaturze. Mam nadzieję dzieki temu osiągnąć dwa pozytywne efekty. Po pierwsze uproszczą się sformułowania i dowody omawianych twierdzeń. Po drugie język ten w kontekście omawianych dalej zagadnień okazać się może bardziej intuicyjny sprowadzając opisane testy pierwszości do kategorii dualności przestrzeni $n^{\perp}$ i $n^{\parallel}$.

Rozdział 2

Wybrane aspekty teorii złożoności obliczeniowej

Wiekszość z opisanych w tym rozdziale rezultatów jest powszechnie znana, pominę więc dowody w większości przypadków. Można je odnaleźć w niemal każdej książce z algebraicznej teorii liczb. Celem tego rozdziału jest przypomnienie podstawowych faktów, które będą miały zastosowanie przy analizie bardziej zaawansowanych algorytmów w dalszej części pracy.

2.1. Notacja O,Ω,Θ

Do porównywania czasu działania algorytmów będzie nam pomocna notacja opisana poniżej.

Definicja 2.1.1 *Będziemy oznaczać* $f(n) = O(g(n))$ *jeśli istnieje stała* $c > 0$ *i* N*, takie że* $f(n) \leq cg(n)$ *dla wszystkich* $n \geq N$.

Definicja 2.1.2 *Będziemy oznaczać* $f(n) = \Omega(g(n))$ *jeśli istnieje stała* c *i* N*, takie że* $f(n) \geq cg(n)$ *dla wszystkich* $n \geq N$.

Definicja 2.1.3 *Jeśli* $f(n) = O(g(n))$ *i* $g(n) = O(f(n))$ *to będziemy używać oznaczenia* $f(n) = \Theta(g(n))$.

2.2. Języki i klasy języków

Definicja 2.2.1 *Alfabetem nazywamy dowolny skończony zbiór* Σ.

Definicja 2.2.2 *Słowem nad alfabetem* Σ *nazywamy dowolny ciąg elementów zbioru* Σ.

Definicja 2.2.3 *Przez* Σ^* *będziemy oznaczać zbiór wszystkich słów nad alfabetem* Σ.

Definicja 2.2.4 *Językiem nad alfabetem* Σ *nazywamy dowolny podzbiór* Σ^*.

Definicja 2.2.5 *Problem decyzyjny dla języka* L *nad alfabetem* Σ*:*
Dane: słowo $s \in \Sigma^*$
Pytanie: czy $s \in L$*?*

Definicja 2.2.6 *Niech x będzie słowem nad alfabetem $\Sigma (x \in \Sigma^*)$. Przez $\lg x$ oznaczamy długość tego słowa.*

Definicja 2.2.7 *Mówimy, że funkcja f jest obliczalna w czasie wielomianowym, jeśli istnieje algorytm obliczający f, który wymaga co najwyżej $p(\lg x)$ operacji binarnych dla wejścia x dla pewnego wielomianu p.*

Definicja 2.2.8 *Będziemy mówić, że język L jest w klasie złożoności $\mathcal{P}$, jeśli funkcja charakterystyczna L jest obliczalna w czasie wielomianowym.*

Definicja 2.2.9 .
$\mathcal{NP}=\{L | \exists$ *zbiór* $S \in \mathcal{P}$ *i wielomian* p, *taki że* $x \in L \iff \exists y, \lg y \leq p(\lg x)$, *oraz* $(x,y) \in S\}$.

Najlepiej jest patrzeć na słowo y, jak na „wielomianowej długości" dowód, że x∈L.

W tej pracy będziemy się głównie interesować problemami dotyczącymi liczb naturalnych. Za alfabet Σ będziemy przyjmować zbiór składający się z 0 i 1. Każdej liczbie naturalnej będzie odpowiadać słowo reprezentujące zapis binarny takiej liczby. Przykłady:

NATURALNE=$\{0, 1, 10, 11, \ldots\}$ - liczby naturalne
NIENATURALNE=$\Sigma^*\setminus$ NATURALNE= $\{0^k x : x \in \mathcal{N} \wedge k > 1\}$ - słowa, które nie kodują liczb naturalnych
PARZYSTE=$\{0, 10, 100, 110, 1000, \ldots\}$ - liczby parzyste
ZŁOŻONE=$\{100, 110, 1000, 1010, \ldots\}$ - liczby złożone
PIERWSZE=$\{10, 11, 101, 111, \ldots\}$ - liczby pierwsze

Definicja 2.2.10 *Jeśli C jest klasą złożoności, to przez co-C będziemy rozumieć klasę złożoną z dopełnień elementów z C.*

Przykładowo PIERWSZE∈co-$\mathcal{NP}$. Wynika to z faktu, że ZŁOŻONE∈ $\mathcal{NP}$, a PIERWSZE to dopełnienie języka ZŁOŻONE∪NIENATURALNE (proszę zauważyć, że taka suma teoriomnogościowa jest w klasie $\mathcal{NP}$).

2.3. Redukcje i $\mathcal{NP}$-Zupełność

W tym podrozdziale postaram się formalnie zdefiniować pojęcie wielomianowej redukcji dwóch problemów decyzyjnych. Mówienie o algorytmach, jako o problemach decyzyjnych jest tylko pozornym uproszczeniem. Większość z omawianych dalej problemów obliczeniowych można zredukować do pewnego problemu decyzyjnego. Dla przykładu problem faktoryzacji oblicza wszystkie dzielniki pierwsze danej liczby n. Możemy zredukować go do problemu decyzyjnego:

DZIELNIK=$\{$(n,k): n ma nietrywialny dzielnik$\leq$k$\}$.

Każdy algorytm obliczający funkcję charakterystyczną zbioru DZIELNIK może zostać użyty do rozkładu liczby na czynniki.

Definicja 2.3.1 *Niech L_1 i L_2 będą językami nad alfabetem Σ. Obliczalną funkcję $f : \Sigma^* \to \Sigma^*$ będziemy traktować jako redukcję z L_1 do L_2, jeśli dla dowolnego x: $x \in L_1 \iff f(x) \in L_2$. Jeśli f jest obliczalna w czasie wielomianowym, to będziemy mówić o wielomianowej redukcji.*

Będziemy pisać $L_1 \leq_m L_2$ w celu oznaczenia powyższej zależności oraz $L_1 \leq_m^P L_2$ dla redukcji w czasie wielomianowym. Innymi słowami jeśli $L_1 \leq_m^P L_2$, to problem decyzyjny L_1 jest łatwiejszy od L_2. Warto również rozpoznawać klasę języków, które są co najmniej tak trudne, jak dowolny problem z klasy NP.

Definicja 2.3.2 *Język L jest $\mathcal{NP}$-Zupełny, jeśli $L \in \mathcal{NP}$ i dla każdego $L' \in \mathcal{NP}$ mamy $L' \leq_m^P L$.*

2.4. Randomizacyjne klasy złożoności

Zanim przystąpimy do formalnych definicji klas złożoności odwołajmy się na chwilkę do intuicji.

$\mathcal{RP}$ będzie kolekcją języków L, dla których istnieje niedeterministyczny algorytm akceptujący słowa $x \in L$ z wysokim prawdopodobieństwem i zawsze odrzucający słowa $x \notin L$. Dodatkowo wymagamy od takiego algorytmu, żeby działał w czasie wielomianowym.

$\mathcal{BPP}$ to kolekcja jezyków L, dla których istnieje algorytm akceptujący słowa z L z dużym prawdopodobieństwem oraz odrzucający słowa nie należące do L także z dużym prawdopodobieństwem. Podobnie jak poprzednio wymagamy ponadto, aby taki algorytm działał w czasie wielomianowym.

$\mathcal{ZPP}$ to kolekcja języków L, dla których istnieje algorytm akceptujący wszystkie słowa z L oraz odrzucający wszystkie słowa nie należące do L. Tym razem od takiego algorytmu wymagamy, aby działał w oczekiwanym czasie wielomianowym.

Czas na formalne definicje omówionych klas:

Definicja 2.4.1 $\mathcal{RP} = \{L|\ \exists$ *zbiór* $S \in \mathcal{P}$ *oraz wielomian p, taki że*
$X \in L \Rightarrow$ *dla co najmniej* $1/2$ *wszystkich y,* $\lg y \leq p(\lg x)$ *mamy* $(x, y) \in S$;
$X \notin L \Rightarrow$ *dla wszystkich y,* $\lg y \leq p(\lg x)$ *mamy* $(x, y) \notin S\}$.

Definicja 2.4.2 $\mathcal{BPP} = \{L|\ \exists$ *zbiór* $S \in \mathcal{P}$ *oraz wielomian p, taki że*
$x \in L \Rightarrow$ *dla co najmniej* $3/4$ *wszystkich y,* $\lg y \leq p(\lg x)$ *mamy* $(x, y) \in S$;
$x \notin L \Rightarrow$ *dla co najmniej* $3/4$ *wszystkich y,* $\lg y \leq p(\lg x)$ *mamy* $(x, y) \notin S\}$.

Definicja 2.4.3 $\mathcal{ZPP} = \mathcal{RP} \cap co - \mathcal{RP}$

Pomiędzy omawianymi klasami języków zachodzą następujące zawierania:

$$\mathcal{P} \subset \mathcal{ZPP} \subset \mathcal{RP} \subset (\mathcal{NP} \cap \mathcal{BPP}).$$

Jako ciekawostkę należy w tym miejscu odnotować, że nie znamy żadnej zależności pomiędzy $\mathcal{BPP}$ i $\mathcal{NP}$ oraz nie wiemy, czy którekolwiek z zawierań jest właściwe.

Będziemy mówić, że algorytm randomizowany jest typu „Las Vegas", jeśli jego wyjście jest zawsze poprawne. Jego czas działania może zależeć od konkretnych losowych wyborów w nim dokonywanych.

Przez algorytm typu „Atlantic City” będziemy rozumieć taki, który udziela poprawnej odpowiedzi w co najmniej 3 na 4 razy.

Będziemy mówić, że algorytm jest typu „Monte Carlo”, jeśli rozpoznaje słowa z języka L z prawdopodobieństwem co najmniej $1/2$ i nigdy nie zdarza mu się błędnie uznać dane słowo za należące do L.

Proszę zauważyć, że jeśli $L \in \mathcal{ZPP}$,to istnieje dla L algorytm typu „Las Vegas” działający w oczekiwanym czasie wielomianowym. Podobnie dla każdego języka $L \in \mathcal{RP}$ istnieje wielomianowy algorytm typu „Monte Carlo”. Dla języków z klasy $\mathcal{BPP}$ można zawsze znaleźć wielomianowy algorytm typu „Atlantic City”.

2.5. Podstawowe działania arytmetyczne i ich złożoność

Celem tego podrozdziału będzie skrótowy przegląd algorytmów realizujących podstawowe działania na liczbach naturalnych ze szczególnym naciskiem na ich złożoność obliczeniową. We wszystkich algorytmach, jeśli nie zostanie podane inaczej będziemy milcząco zakładać, że $n = a + b$, gdzie a, b są danymi wejściowymi algorytmu.

2.5.1. Dodawanie

Można wykonać w czasie $O(\lg n)$ operacji na bitach.

2.5.2. Odejmowanie

Można wykonać w czasie $O(\lg n)$ operacji na bitach.

2.5.3. Mnożenie

Zwykły(szkolny) algorytm mnożenia działa w czasie $O((\lg n)^2)$ operacji na bitach. Czas ten można jeszcze nieco poprawić. Używając szybkiej transformaty Fouriera uzyskujemy algorytm o złożoności obliczeniowej $O((\lg n)^{1+\epsilon})$. Będziemy dalej korzystać z obu tych wyników.

2.5.4. Dzielenie całkowite

Można zaimplementować algorytm o złożoności $O((\lg n)^2)$ realizujący to działanie.

2.5.5. Dzielenie modulo

Korzystając z algorytmu na dzielenie całkowite można zaprojektować algorytm dzielenia modulo, który będzie miał złożoność $O((\lg n)^2)$.

2.5.6. Największy wspólny dzielnik

Algorytm 2.5.1 *dane:* $a, b \in \mathcal{N}$

1. *jeśli (a>b) zamień a z b*

   $\{a \leq b\}$

2. *jeśli a=1 zwróć 1*

   $\{(a \leq b) \wedge (a \neq 1)\}$

3. *b:=b MOD a*
4. *jeśli b=0 zwróć a*
5. *idź do 1*

Przedstawiony algorytm działa w czasie $O((\lg a)(\lg b))$ operacji na bitach.

2.5.7. Prostopadłość

dane: $a, b \in \mathcal{N}$
szukamy: czy $a \perp b$?

Zgodnie z twierdzeniem (1.2): $a \perp b \iff NWD(a, b) = 1$.
Wystarczy więc skorzystać z algorytmu na NWD.
Jak widać można testować prostopadłość w tym samym czasie co dzielenie modulo, czyli $O(\ln(a + b))$.

2.5.8. Równoległość

dane: $a, b \in \mathcal{N}$
szukamy: czy $a \parallel b$?

Zgodnie z twierdzeniem (1.1): $a \parallel b \iff$ (a MOD b=0)$\vee$(b MOD a=0).
Mamy więc algorytm działający asymptotycznie w tym samym czasie, co dzielenie modulo, czyli $O((\lg n)^2)$.

2.5.9. Element odwrotny modulo n

Dane: $a, n \in \mathcal{N}$ spełniające $a \perp n$
Szukamy: $b \in \mathcal{N}$, które spełnia $(ab - 1) \parallel n$

Problem sprowadza się więc do znalezienia $b, c \in \mathcal{N}$, które spełniają $ab \pm nc = 1$.
Istnieje algorytm rozwiązujący ten problem w czasie $O((\lg n)^2)$.

2.5.10. Potęgowanie modulo

Dane: $a, e, n \in \mathcal{N}$ $e \geq 0$, $n \geq 2$
Szukamy: a^e MOD n

Algorytm jest oparty na tożsamości:

$$a^e = \begin{cases} (a^{e/2})^2, & \text{dla } 2 \mid e \\ a \cdot a^{e-1}, & \text{wpp.} \end{cases}$$

Algorytm 2.5.2 *POTĘGA(a,e,n)*

1. *jeśli e=0 zwróć (1)*
2. *jeśli (e MOD 2)=0*

   t:=POTĘGA(a,e/2,n); zwróć (t^2)
3. *t:=POTĘGA(a,e-1,n) zwróć (at)*

Powyższy algorytm działa w czasie $O((\lg e)(\lg n)^2)$. Jeśli skorzystamy z szybkiego algorytmu mnożenia uzyskamy złożoność rzędu $O((\lg e)(\lg n)^{1+\epsilon})$.

Zebraliśmy więc podstawową wiedzę o algorytmach realizujących najprostsze operacje na liczbach naturalnych. Znamy też szacowania ich złożoności obliczeniowej. Mamy dzięki temu pewien aparat do badania i projektowania bardziej skomplikowanych algorytmów rozwiązujących bardziej złożone problemy dla liczb naturalnych.

Rozdział 3

Ważne problemy i funkcje teorioliczbowe, ich związek z testowaniem pierwszości

Na początku tego rozdziału przedstawię bardzo znany i ważny problem z teorii liczb. Jest on powszechnie uznawany za trudny obliczeniowo, jego trudność jest wykorzystywana w praktyce w kryptografii klucza publicznego. W dalszej części rozdziału przypomnę znane lub mniej znane funkcje teorioliczbowe oraz kilka prostych, lecz niezmiernie istotnych faktów ich dotyczących. W dalszym ciągu, tak jak w dwóch poprzednich rozdziałach, kolekcjonujemy termninologię i narzędzia, które mamy ambicję wykorzystać przy projektowaniu testów pierwszości w dalszej części pracy.

3.1. Problem dzielników

Dane: $a \leq b \leq n$ $a, b, n \in \mathcal{N}$
Szukamy: czy istnieje d: $a \leq d \leq b$, takie że $d \mid n$?

Tak postawiony problem decyzyjny jest w tej samej klasie złożoności, co problem faktoryzacji, choć nie wiadomo jaka to jest klasa. Nie znaleziono do tej pory wielomianowego algorytmu. Na trudności tego problemu bazuje wiele powszechnie stosowanych systemów kryptograficznych. Problem testowania pierwszości, którym to głównie będziemy się zajmować w tej pracy jest problemem łatwiejszym, tzn. mając do dyspozycji szybki algorytm rozwiązujący problem dzielników łatwo otrzymamy szybki algorytm do testowania pierwszości. Wystarczy sprawdzić istnienie dzielników dla a=2, b=n-1 i uzyskamy jednocześnie odpowiedź na pytanie, czy liczba jest pierwsza.

3.2. Funkcje teorioliczbowe i ich własności

W tym podrozdziale przypomnę definicje kilku interesujących funkcji teorioliczbowych oraz podam te ich własności, które okażą się istotne w dalszej części pracy.

3.2.1. Funkcja $\pi(n)$

Ilość liczb pierwszych nie przekraczających n będziemy oznaczać przez $\pi(n)$. O tym, jak gęsto są rozmieszczone liczby pierwsze wśród liczb naturalnych, mówi następujący fakt:

$$\lim_{x\to\infty} \frac{\pi(x)}{x/\log x} = 1.$$

Wynika stąd, że losując liczby naturalne o s cyfrach, szansa natknięcia się na liczbę pierwszą wynosi około $\frac{1}{s}$. Szybkie generowanie dużych liczb pierwszych ma poważne zastosowania praktyczne chociażby w kryptografii. To szacowanie pokazuje, że łatwo jest natrafić na liczbę pierwszą, inną sprawą jest weryfikacja czy taka losowa liczba jest pierwsza, ale do tego dojdziemy w następnym rozdziale.

3.2.2. Funkcja $\phi(n)$ Eulera

$\phi(n) := \#\{a : a \leq n \wedge a \perp n\}$
Używając notacji z rozdziału (1) $\phi(n) = |n_\perp|$.

Jednym z podejść do testowania pierwszości jest badanie właściwości grupy multiplikatywnej pierścienia $Z/(n)$. Rząd takiej grupy, to właśnie dokładnie $\phi(n)$, a rząd każdego z jej elementów jest dzielnikiem $\phi(n)$. Badając więc rzędy elementów wymienionej grupy zdobywamy informacje o faktoryzacji $\phi(n)$. Jak to się ma do ewentualnych dzielników n będzie się można dowiedzieć już w następnym rozdziale.

Już niebawem będzie nam potrzebne następujące oszacowanie na $\phi(n)$:

Lemat 5 *$\phi(n) = \Omega(n/\log\log n)$.*

3.2.3. Funkcja $\sigma(n)$

Funkcja sumująca dzielniki jest zdefiniowana w następujący sposób:

$$\sigma(n) = \sum_{d|n} d.$$

Uogólnieniem tej funkcji na sumę k-tych potęg dzielników n jest:

$$\sigma_k(n) = \sum_{d|n} d^k.$$

Możemy więc zapisać:

$$\sigma(n) = \sigma_1(n).$$

3.2.4. Funkcja zliczająca dzielniki pierwsze

Jak wiadomo każda liczba naturalna jednoznacznie rozkłada się na iloczyn potęg liczb pierwszych. Do badania pewnych właściwości takiego rozkładu użyjemy funkcji $\omega(n)$ zliczającej ilość różnych dzielników pierwszych liczby n..

Jeśli liczbę naturalną n przedstawimy w postaci rozkładu na czynniki pierwsze:

$$n = p_1^{e_1} \dots p_k^{e_k},$$

wtedy

$$\omega(n) = k.$$

Pewna klasa warunkowych testów pierwszości, o której będzie można przeczytać dalej, opiera się na znajomości dzielników pierwszych pewnej liczby zależnej od liczby badanej. Jednym z kroków takich testów jest wykonywanie pewnych obliczeń dla każdego takiego dzielnika. Szacując złożoność obliczeniową takich algorytmów będziemy więc chcieli wiedzieć, ile maksymalnie takich dzielników pierwszych możemy się spodziewać. Bardzo pomocny będzie do tego poniższy lemat:

Lemat 6 $\omega(n) = O(\log n / \log \log n)$.

Dowód przedstawionego faktu można odnazeźć w [1] (str. 366).

3.2.5. Liczby y-gładkie

Niech y będzie dodatnią liczbą rzeczywistą. Będziemy mówić, że liczba naturalna x jest y-gładka, jeśli każdy dzielnik pierwszy x nie przekracza y. Przykładowo liczba 30=2*3*5 jest 5-gładka, ale nie jest 3-gładka, bo $5 \mid 30$.

Do zliczania liczb y-gładkich będziemy używać funkcji $\Psi(x, y)$, która będzie określać ilość liczb y-gładkich nieprzekraczających x. Przykładowo $\Psi(10, 3) = 7$, bo spośród liczb $1, 2, \dots, 10$ tylko 1,2,3,4,6,8,9 są 3-gładkie.

Nas głównie będą interesować liczby $p(\log x)$-gładkie, gdzie p jest pewnym ustalonym wielomianem. Jeśli więc weźmiemy pewien wielomian p, oraz wybierzemy s-długość interesującej nas liczby, to na pytanie, czy liczba x o długości s jest $p(s)$-gładka potrafimy odpowiedzieć w czasie $O(p(s)q(s))$, gdzie q jest wielomianem szacującym złożoność pojedynczego dzielenia. Takie liczby można oczywiście też faktoryzować w czasie wielomianowym. Badanie, jak dużo jest liczb $p(x)$-gładkich jest więc jednocześnie próbą odpowiedzi na pytanie jak wiele liczb potrafimy na pewno szybko faktoryzować.

Poniżej przedstawię szacowanie funkcji $\Psi(x, y)$, które okaże się istotne dla algorytmu Fellowsa-Koblitza omówionego w dalszej części pracy.

Twierdzenie 3.1 *Dla $x \geq 4$ oraz $2 \leq y \leq x$ mamy*

$$\Psi(x, y) > x^{1-(\log \log x)/(\log y)}.$$

Dowód powyższego twierdzenia znajduje się w [9].

Rozdział 4

Testowanie pierwszości w szerszym ujęciu

W tym rozdziale skoncentrujemy się już tylko i wyłącznie na bardzo istotnych, nie tylko teoretycznie, zagadnieniach testowania pierwszości. Wykorzystamy rezultaty zgromadzone w poprzednich rozdziałach. Opowiemy nie tylko o obliczaniu funkcji charakterystycznej zbioru PIERWSZE, ale także o efektywnych sposobach dowodzenia pierwszości, a także poszukiwania dużych liczb pierwszych na potrzeby systemów kryptograficznych.

4.1. Historia zmagań z liczbami pierwszymi

Problem badania pierwszości angażował wybitne umysły już wiele tysięcy lat temu. Pierwszy znany rezultat pochodzi od Eratostenesa (250 rok p.n.e.), który zaproponował metodę wyznaczania wszystkich liczb pierwszych poniżej pewnej granicy. Na początku XIII wieku znany skądinąd Fibonacci zauważył, że aby stwiedzić pierwszość danej liczby naturalnej wystarczy zbadać wszystkie potencjalne dzielniki $\leq \sqrt{n}$. Euler przedstawił dowód, że jeśli n=$4k+1$ może być zapisane jako suma kwadratów na jeden możliwy sposób (z dokładnością do symetrii) $n = n_1^2 + n_2^2$ oraz $n_1 \perp n_2$, to n jest pierwsze. Korzystając z wymyślonego przez siebie twierdzenia Euler udowodnił, że n=$1000^2 + 3^2$ jest pierwsze.

Do pracy E.Lucasa, która pojawiła się w roku 1876, nie rozróżniało się praktycznie testowania pierwszości i faktoryzacji, jako dwóch oddzielnych problemów. Lucas pokazał wtedy po raz pierwszy, że w pewnych specjalnych przypadkach może być łatwiej udowodnić, że liczba jest złożona (pierwsza) niż wskazać jej dzielnik (sprawdzić możliwe dzielniki). Pomysły Lucasa zostały dostrzeżone przez M.Kraitchika i D.H.Lehmera. W 1920 roku opracowali oni testy działające wprawdzie dla liczb specjalnej postaci (np. $2^n \pm 1$), ale za to szybko dla dużych liczb.

Pod koniec XX wieku testowanie pierwszości zaczęło coraz bardziej przyciągać uwagę informatyków i kryptografów. Kryptosystemy, takie jak RSA bazują na założeniu, że jest stosunkowo prosto wyprodukować dużą liczbę pierwszą, podczas gdy faktoryzacja iloczynu takich różnych liczb pierwszych jest już bardzo trudna.

W artykule *„Theorie des Nombres”* opublikowanym w 1891 roku Lucas wskazał pięć problemów, które wówczas nie doczekały się jeszcze satysfakcjonujących rozwiązań. Były to:

1. Znaleźć liczbę pierwszą większą od danej liczby pierwszej.

2. Znaleźć nietrywialną funkcję, która przyjmuje tylko liczby pierwsze.

3. Znaleźć kolejną liczbę pierwszą większą od danej.
4. Obliczyć $\pi(n)$.
5. Umieć obliczać p_n(n-ta liczba pierwsza).

Na początku XIX wieku Gauss wyraził do naukowców na całym Świecie takie oto przesłanie:

> „Problem odróżniania liczb pierwszych od złożonych i rozkładania na czynniki pierwsze tych ostatnich jest jednym z najbardziej istotnych i użytecznych w arytmetyce...
> ...Godność samej nauki wymaga, aby użyć wszystkich dostępnych środków do rozwiązaniu problemu tak eleganckiego i tak celebrowanego."

4.2. Grupa $(Z/(n))^*$

W tym podrozdziale zainteresujemy się bliżej pewną strukturą algebraiczną, z której właściwości, jak się okaże, można będzie uzyskać pewne informacje o rozkładzie liczby n.

Na początek krótkie przypomnienie z algebry:

1. Jako $Z/(n)$ będziemy rozumieć pierścień składający się z elementów $\{0, 1, \ldots, n-1\}$ z określonymi działaniami dodawania i mnożenia modulo n.
2. Elementem odracalnym pierścienia R nazywamy taki element $x \in R$, że istnieje $x^{-1} \in R$, takie że $xx^{-1} = 1_R$.
3. Elementy odwracalne pierścienia R tworzą grupę ze względu na mnożenie. Taką multiplikatywną grupę elementów odwracalnych będziemy oznaczać R^*.
4. Grupa $(Z/(n))^*$, czyli grupa elementów odwracalnych pierścienia $Z/(n)$, zawiera dokładnie te same elementy, co zbiór $n_\perp$. Inaczej, $x \in (Z/(n))^* \iff x \perp n$.
5. Rzędem elementu x należącego do grupy G nazywamy najmniejszą liczbę naturalną dodatnią r, taką że $x^r = 1_G$. Będziemy tę sytuację oznaczać $rz_G x = r$. Jeśli $x^r \neq 1_G$ dla każdej liczby naturalnej dodatniej r, to będziemy pisać $rz_G x = \infty$. Bardzo często za G będziemy przyjmować grupę $(Z/(n))^*$ i dla uproszczenia notacji pisać $rz_n x$ zamiast $rz_{(Z/(n))^*} x$.
6. Niepusty podzbiór $H \subseteq G$ nazywamy podrupą jeśli jest grupą ze względu na to samo działanie co G.
7. Indeksem podgrupy H grupy skończonej G będziemy nazywać stosunek $|G|/|H|$.

Twierdzenie 4.1 (Lagrange) *Jeśli G jest grupą o skończonej liczbie elementów, $x \in G$, to $rz_G x \mid |G|$.*

Na potrzeby dalszych rozważań zauważmy w tym miejscu, że jeśli $rz_G x = r$, to $x^y = x^{y \bmod r}$(elementy powtarzają się okresowo, długość okresu wynosi r). Będziemy także wykorzystywać następujący:

Wniosek 1 *Jeśli G jest grupą i $x \in G$, oraz dla pewnego dodatniego e zachodzi $x^e = 1_G$, wówczas $rz_G x \mid e$.*

Zauważmy również, że elementy $x^1, x^2, \ldots, x^{rz_G x}$ nie zawierają powtórzeń, bo gdyby $x^i = x^j$, gdzie $1 \leq i < j \leq rz_G x$, to mnożąc równanie stronami przez $(x^{-1})^i$ otrzymamy $x^{j-i} = 1$ czyli sprzeczność.

Podane twierdzenie wskazuje istnienie zależności między problemem dzielników a problemem obliczania rzędu elementu w grupie $(Z/(n))^*$. Jeśli bowiem potrafimy efektywnie obliczać rząd elementu w grupie $(Z/(n))^*$, to potrafimy także wyznaczać dzielniki właściwe $\phi(n)$. Obliczamy wówczas rzędy $rza_1, \ldots, rza_k$ dla losowo wybranych elementów $a_1, \ldots, a_k$, każdy z tych rzędów jest dzielnikiem $\phi(n)$.

Jak się okazuje pewne właściwości teorioliczbowe można ładnie formułować i szybko dowodzić w języku teorii grup. Dla przykładu twierdzenie Eulera (1.4) wynika wprost tw. Lagrange'a zastosowanego dla grupy $(Z/(n))^*$.

Jeśli w grupie G istnieje element g, taki że $\{g^i : 1 \leq i \leq |G|\} = G$, to mówimy wtedy, że G jest cykliczna, a element g nazywamy generatorem. Generator grupy $(Z/(n))^*$(jeśli istnieje) będziemy nazywać pierwiastkiem pierwotnym.

Na pytanie dla jakich n grupa $(Z/(n))^*$ ma generator odpowiada następujące twierdzenie.

Twierdzenie 4.2 *Niech $n \in \mathcal{N}$, $n>1$. Grupa $(Z/(n))^*$ ma pierwiastek pierwotny wtedy i tylko wtedy, gdy $n = 2, 4, p^e, 2p^e$, gdzie $e \geq 1$ i p jest nieparzystą liczbą pierwszą.*

Dostajemy stąd następującą charakterystykę liczby pierwszych:

Wniosek 2 *Liczba naturalna n jest pierwsza $\iff$ grupa $(Z/(n))^*$ zawiera element rzędu n-1.*

Twierdzenie, choć ładnie charakteryzuje liczby pierwsze w świetle właściwości badanej grupy, nie odpowiada jednak na pytanie w jaki sposób szukać takiego generatora lub sprawdzać, że dany element jest generatorem.
Okazuje się, że jeśli istnieje element pierwotny w grupie $(Z/(n))^*$, to takich elementów pierwotnych jest dosyć dużo. Konkretniej:

Twierdzenie 4.3 *Jeśli liczba naturalna p jest pierwsza, to grupa $(Z/(p))^*$ zawiera dokładnie $|(p-1)_\perp|$ elementów pierwotnych.*

Dowód:

Dla p będącego liczbą pierwszą grupa $G = (Z/(p))^*$ jest cykliczna. Niech g będzie jej generatorem. Zbiór $\{g, g^2, \ldots, g^{p-1}\}$ zawiera wszystkie elementy pierwotne z G, bo po prostu zawiera wszystkie elementy grupy G. Zauważmy, że g^i jest elementem pierwotnym wtw. gdy $i \perp (p-1)$. Aby to pokazać zauważmy, że

$$(g^i)^{\frac{p-1}{NWD(i,p-1)}} = 1_G,$$

ponieważ zapisując i=jNWD(i,p-1) otrzymamy:

$$g^i = g^{jNWD(i,p-1)}$$

$$(g^i)^{\frac{p-1}{NWD(i,p-1)}} = (g^{jNWD(i,p-1)})^{\frac{p-1}{NWD(i,p-1)}} = g^{j(p-1)} = (g^{p-1})^j = 1_G.$$

Widać więc, że jeśli nie zachodzi $i \perp (p-1)$, to g^i nie może być generatorem. Jeśli zaś przyjmiemy $i \perp (p-1)$ i przyjmiemy $rz_G g^i = r$, to $g^{ir} = 1_G$ czyli musi być $rz_G g^i \mid ir$. Z zasadnieczego tw. arytmetyki (1.8) ponieważ $(p-1) \perp i$ wynika $(p-1) \mid r$, ale równocześnie $r \mid (p-1)$, więc r=p-1. Pokazaliśmy więc, że elementów pierwotnych w grupie $(Z/(p))^*$ jest dokładnie $|(p-1)_\perp|$.•

Jeśli przypomnimy sobie szacowanie na $|n_\perp|$ z rozdziału (3) ($|n_\perp| = \Omega(n/\log\log n)$), to okazuje się, że element pierwotny jest łatwo zgadnąć. Cały problem w tym, że nie potrafimy efektywie sprawdzić, że wylosowany element jest rzeczywiście generatorem.

Okazuje się, że posiadając pewne dodatkowe informacje o liczbie n jesteśmy w stanie efektywnie stwierdzić, czy nasz kandydat jest generatorem. Ta dodatkowa wiedza, choć w przypadku ogólnym jest trudna do uzyskania, to w pewnych sytuacjach, bądź dla liczb specjalnej postaci, będzie dostępna. Otóż okazuje się, że wystarczy znać faktoryzację liczby $p-1$ do szybkiego obliczenia $rz_p a$, bądź w ogólniejszym przypadku (n niekoniecznie pierwsze) faktoryzację $|n_\perp|$ do obliczenia $rz_n a$.

Algorytm 4.2.1 (Obliczanie rzędu elementu w $(Z/(n))^*$**)** .

Dane: $n, a \in \mathcal{N}$, $a \perp n$, $|n_\perp| = p_1^{e_1} \ldots p_k^{e_k}$ *(rozkład na czynniki pierwsze)*

1. *r:=1*
2. *Dla i=1,2,...,k wykonuj:*
   - $q := a^{|n_\perp|/p_i^{e_i}}$
   - *dopóki* $(q \neq 1)$ *wykonuj*
     - $q := q^{p_i}$
     - $r := r \cdot p_i$

   $\{a^{r \prod_{j=i+1}^{k} p_j^{e_j}} = 1\}$, $\{r \mid rz_n a\}$
3. *zwróć (r)*

Twierdzenie 4.4 *Algorytm (4.2.1) jest poprawny i działa w czasie* $O((\log n)^4/(\log\log n))$.

Dowód:

Przyjmijmy, że dla danych a,n algorytm zwrócił nam wynik W. Aby wykazać poprawność algorytmu wystarczy zauważyć poprawność niezmienników, gdyż po zakończeniu wynika z nich, że $a^W = 1$ *oraz* $W \mid rz_n a$. *Stąd* $W = rz_n a (rz_n a \mid W \wedge W \mid rz_n a)$.

Pozostaje jeszcze policzyć złożoność obliczeniową powyższego algorytmu. Główną pętlę wykonujemy dla każdego dzielnika pierwszego $|n_\perp|$. *Takich dzielników jest* $O(\log n/\log\log n)$ *(ten wynik się pojawił w sekcji (3.3.4)). Obliczenie potęgi a w pierwszej instrukcji głównej pętli to* $O((\log n)^3)$ *operacji binarnych(2.5.10). Bez pętli „dopóki" mamy więc złożoność* $O((\log n)^4/(\log\log n))$. *Operacje mnożenia nie psują naszego oszacowania, pozostaje więc zbadać, ile czasu zajmuje potęgowanie w pętli wewnętrznej. W i-tej iteracji głównej pętli jest takich potęgowań co najwyżej* e_i. *Każde z potęgowań i-tej iteracji to* $O((\log p_i)(\log n)^2)$ *operacji. W sumie wszystkie wymagają więc*

$$O(\sum_{i=1}^{k} e_i (\log p_i)(\log n)^2) = O((\log n)^2 \sum_{i=1}^{k} (\log p_i^{e_i})) = O((\log n)^3),$$

czyli te operacje także nie psują nam naszego pierwotnego szacowania.•

Podsumowując dotychczasowe rozważania w tym podrozdziale udało nam się sprowadzić problem badania pierwszości do problemu szukania w grupie $(Z/(n))^*$ elementu rzędu $n-1$. Okazuje się, że można ten rezultat jeszcze poprawić.

Otóż jeśli $r = rz_n a$, $r > \sqrt{n}$ oraz znamy rozkład liczby r na czynniki pierwsze, to można sprawdzić pierwszość n w czasie wielomianowym. Sama informacja, że jakiś element $a \in (Z/(n))^*$ ma rząd r daje już informację o ewentualnych dzielnikach pierwszych n.

4.3. Testowanie pierwszości, jako problem decyzyjny w ujęciu teorii obliczeń

4.3.1. Definicja problemu

Będziemy kodować liczby naturalne za pomocą słów nad alfabetem $\Sigma = \{0, 1\}$. Danej liczbie naturalnej będziemy przyporządkowywać jej zapis w dwójkowym systemie pozycyjnym.

Definicja 4.3.1 *PIERWSZE:={x: $x \in \Sigma^* \wedge k^{-1}(x)$ jest liczbą pierwszą}, gdzie $\Sigma = \{0, 1\}$, a k:$\mathcal{N} \rightarrow \Sigma^*$ jest funkcją kodującą przyporządkowującą liczbom naturalnym ich zapisy binarne.*

4.3.2. Dowodzenie pierwszości i złożoności

Jak do tej pory nikomu nie udało się napisać szybkiego algorytmu deterministycznego (tzn. działającego w czasie wielomianowym względem długości liczby) badającego pierwszość. W tym podrozdziale postawimy nieco prostsze pytanie. Brzmi ono: jak pokazać, że liczba jest pierwsza? Na pozór wydaje się, że mamy do czynienia z tym samym zagadnieniem, jednak w tym wypadku pytamy nie jak znaleźć takie uzasadnienie, ale czy takie uzasadnienie (odpowiednio krótkie) zawsze istnieje.

Wracając do języka ZŁOŻONE widać, że dowód faktu $n \in$ZŁOŻONE zawsze istnieje (wystarczy podać dzielnik właściwy n) i może być zweryfikowany w czasie wielomianowym. Ten najbardziej narzucający się sposób dowodzenia złożoności nie jest jednak jedyny.

Wróćmy do Małego Twierdzenia Fermata(1.5). Jest ono prawdziwe na pewno dla liczb pierwszych, jeśli więc dla pewnego a, takiego że $a \perp n$ nie zachodzi $a^{n-1} \equiv 1(\text{mod n})$, to n nie może być pierwsze. Mamy więc dowód złożoności bez wskazania konkretnego dzielnika właściwego n.

Dowody na pierwszość najlepiej opierać na pewnych charakteryzacjach liczb pierwszych. Chodzi tu o twierdzenia postaci: ($n \in \mathcal{N}$ jest pierwsza $\iff$ warunek(n)). Taki dowód polegałby więc na poprawnym obliczeniu wyrażenia warunek(n), które da wynik 1. Oczywiście nie każde twierdzenie takiej postaci pokazuje w jaki sposób szybko obliczać warunek(n). Weźmy np. słynne:

Twierdzenie 4.5 (Wilsona) *Liczba n jest pierwsza* $\iff$ $(n-1)! \equiv -1$*(mod n).*

Niby takie proste, wystarczy policzyć ((n-1)! mod n) i już wszystko wiemy. Okazuje się, że obliczenie funkcji silnia w czasie wielomianowym stanowi nadal wyzwanie dla algorytmiki.

Nie poddawajmy się tak łatwo, przecież dysponujemy pewną wiedzą o ewentualnych generatorach grupy $(Z/(n))^*$, przypomnę ciekawy wniosek z poprzedniego rozdziału:

Wniosek 3 *Liczba naturalna n jest pierwsza* $\iff$ *grupa* $(Z/(n))^*$ *zawiera element rzędu n-1.*

Tu uzbrojeni w wiedzę o strukturze grupy $(Z/(n))^*$ jesteśmy w zdecydowanie lepszej sytuacji.

Udowodnię dla zilustrowania ogólnej idei, że 31 jest pierwsze. Wystarczy pokazać, że 3 jest elementem pierwotnym, oto dowód:

- $3^{30/2} \equiv 30 (\mathrm{mod}\ 31)$,
- $3^{30/3} \equiv 25 (\mathrm{mod}\ 31)$,
- $3^{30/5} \equiv 16 (\mathrm{mod}\ 31)$ i oczywiście
- $3^{30} \equiv 1 (\mathrm{mod}\ 31)$.

Zastosowałem algorytm do obliczania rzędu elementu w $(Z/(n))^*$. W naszym przypadku $n = 31$, natomiast $n - 1 = 2 * 3 * 5$ to rozkład na czynniki pierwsze.

Naturalnym wydaje się więc pytanie, czy dla każdej liczby pierwszej istnieje taki dowód, który daje się przeprowadzić w czasie wielomianowym. Zauważmy, że dla dowolnej liczby pierwszej p jej generator oraz rozkład $p - 1$ na czynniki pierwsze można zakodować przy pomocy $O(\log p)$ bitów. Przy weryfikacji takiego dowodu pojawia się jednak pewien dosyć istotny problem. Otóż musimy sprawdzić czy dany w dowodzie rozkład n jest rzeczywiście rozkładem, co sprowadza się do:

- zbadania, czy iloczyn elementów rozkładu w odpowiednich potęgach wynosi $p - 1$
- zweryfikowania, czy czynniki rozkładu są rzeczywiście liczbami pierwszymi.

Reasumując: nasz certyfikat pierwszości dla liczby n będzie składać się z następujących informacji:

1. element pierwotny grupy $(Z/(n))^*$
2. certyfikaty pierwszości dla wszystkich dzielników pierwszych liczby $p - 1$

Można więc myśleć o certyfikacie pierwszości dla n, jak o drzewie, którego korzeń zawiera liczbę n oraz element pierwotny grupy $(Z/(n))^*$, natomiast synowie są korzeniami drzew certyfikatów dla liczb pierwszych będących dzielnikami $n-1$. Za certyfikat pierwszości dla liczby 2 przyjmiemy drzewo składające się z jednego węzła etykietowanego dwójką. Oznacza to, że liśćmi dowolnego certyfikatu pierwszości mogą być jedynie certyfikaty dwójki.

W rozdziale (2) pokazaliśmy, że ZŁOŻONE$\in \mathcal{NP}$. Sprawa była prosta, bo dowolny dzielnik właściwy liczby jest dowodem złożoności, który można zweryfikować w czasie wielomianowym. Czy uda się pokazać, że certyfikat dla danej liczby n można zweryfikować w czasie wielomianowym, co oznaczałoby, że PIERWSZE$\in \mathcal{NP}$?

Oto odpowiedź twierdząca na to pytanie:

4.3.3. Tw. Pratta

Twierdzenie 4.6 (PRATT) *PIERWSZE$\in \mathcal{NP}$.*

Dowód:

Mając liczbę pierwszą n możemy zgadnąć element a rzędu n-1 oraz faktoryzację n-1. Znając taką faktoryzację możemy w czasie wielomianowym sprawdzić, że istotnie a jest wskazanego rzędu, co oznacza, że n jest pierwsze. Pozostaje do rozwiązania jeden istotny problem. Otóż *skąd wiadomo, że faktoryzacja n-1 jest poprawna? Możemy oczywiście sprawdzić czy są to dzielniki n, ale jak sprawdzić czy zostały tam użyte tylko liczby pierwsze? Żeby poradzić sobie z tym problemem musimy rekurencyjnie przeprowadzić dowody pierwszości dla wszystkich liczb z rozkładu n-1. Rekursja kończy się dla liczby 2. Przykładowe drzewo dowodu pierwszości dla liczby 617 znajduje się poniżej.*

Dzieci węzła o etykiecie n są etykietowane dzielnikami pierwszymi n-1. Liczby w nawiasach to elementy rzędu m-1 w $(Z/(m))^*$.

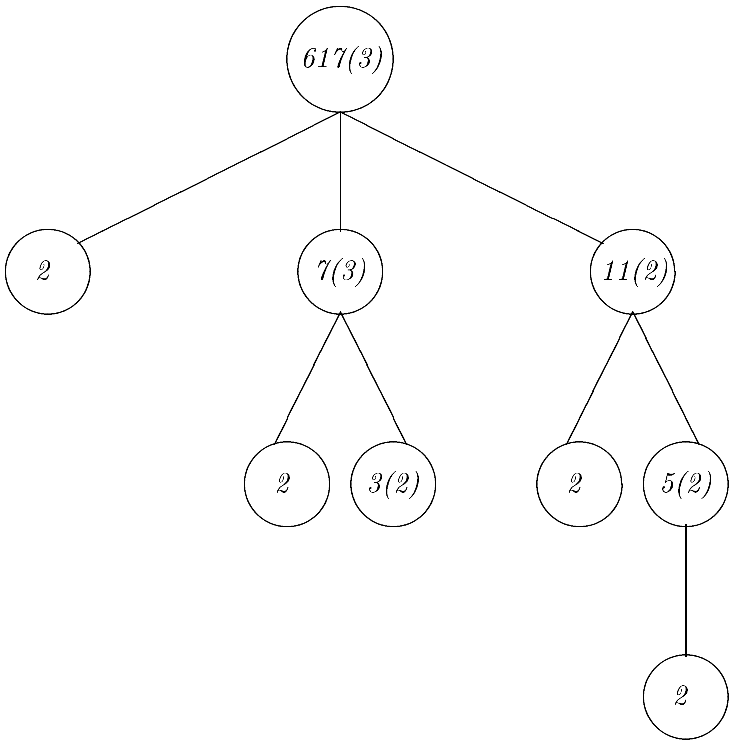
Mamy więc dowody pierwszości. Żeby kontynuować dowód potrzebujemy jeszcze wielomianowe oszacowanie na T(n), gdzie T(n) to ilość wierzchołków certyfikatu liczby pierwszej n. Zauważmy, że:

$$T(2) = 1$$

oraz dla $n \geq 3$

$$T(n) = 1 + \sum_{p|n-1} T(p).$$

Udowodnimy indukcyjnie, że $T(n) < 2\log n$. *Dla n=2 jest to prawda. Teraz załóżmy, że ograniczenie jest prawdziwe dla wszystkich liczb pierwszych mniejszych od n. Zapiszmy:* $n - 1 = p_1 p_2 p_3 \dots p_k$, *gdzie* $p_1 = 2$ *i* p_i *są liczbami pierwszymi (mogą się także powtarzać). Drzewo ceryfikatu dla n ma korzeń, węzeł dla* $p_1 = 2$ *oraz poddrzewa będące drzewami certyfikatów dla pozostałych dzielników pierwszych n-1.*



Rysunek 4.3.1 *Certyfikat pierwszości liczby 617*

$$T(n) \leq 1 + T(2) + \sum_{2\leq i\leq k} T(p_i) \leq 2 + \sum_{2\leq i\leq k} 2\log p_i = 2 + 2\log((n-1)/2) < 2\log n.$$

Każdy wierzchołek o etykiecie m naszego drzewa zawiera $O(\lg n)$ bitów informacji z nim związanej(chodzi o reprezentacje liczby m i generatora oraz $O(\lg\lg n)$ bitów na wskaźniki do synów). Całe drzewo może być więc reprezentowane przy użyciu $O((\lg n)^2)$ bitów.

Pozostaje jeszcze odpowiedzieć na pytanie, jak efektywnie sprawdzić, że dla danego n i danego certyfikatu jest to rzeczywiście certyfikat pierwszości n. Niech m będzie węzłem z ojcem r. Dla każdego wierzchołka o etykiecie m musimy:

i *sprawdzić, czy jego synowie rzeczywiście zawierają rozkład m-1 na czynniki*

ii *sprawdzić, że* $a^{m-1} \equiv 1 (mod\ m) (a \in (Z/(m))^*)$

iii *sprawdzić, że* $a^{(r-1)/m} \neq 1 (mod\ r)$

Operację (i) potrafimy wykonać w czasie $O((\log n)^2)$, a (ii) i (iii) możemy zrealizować w $O((\log n)^3)$ krokach. Jak wcześniej pokazaliśmy drzewo certyfikatu ma $O(\log n)$ węzłów, łączny koszt weryfikacji wyniesie więc $O((\log n)^4)$, co kończy dowód, że PIERWSZE$\in \mathcal{NP}$.$\bullet$

4.4. Testy pierwszości, różne podejścia

Wymyślenie działającego w praktyce algorytmu rozwiązującego ten niezwykle prosto formułujący się problem nie jest rzeczą specjalnie trudną i każdy kto zna definicje liczby pierwszej jest w stanie go wymyśleć w ciągu paru minut.

Od porządnego algorytmu testowania pierwszości chcemy wymagać jednak następujących cech:

- musi działać dla wszystkich $n \in \mathcal{N}$(ogólność)
- musi zawsze dawać poprawny wynik (poprawność)
- musi być efektywny (działać w czasie wielomianowym)(szybkość)
- powinien dostarczać dowodu pierwszości (złożoności), który da się zweryfikować w czasie wielomianowym

Ogólnie rzecz ujmując różne podejścia do badania pierwszości polegają na osłabianiu któregoś z wymienionych warunków.

4.4.1. Determinizm

Jest to podejście zmierzające do skonstruowania algorytmu, który będzie spełniał wszystkie wymienione wyżej cechy, poza być może efektywnością. Dąży się więc do otrzymania algorytmu, który da poprawną odpowiedź dla dowolnej liczby i zawsze się zatrzyma. W praktyce raczej nie wykorzystuje się algorytmów deterministycznych do testowania pierwszości, gdyż działają one przeraźliwie wolno, bądź też dlatego, że zachowują się gorzej od algorytmów randomizacyjnych. Aby dobrze zilustrować trudności związane ze stworzeniem dobrego algorytmu deterministycznego przyjrzymy się kilku innym podejściom, które nie spełniają któregoś z przedstawionych wyżej wymagań i zastanowimy się na czym polegają trudności z równoczesnym spełnieniem przedstawionych postulatów.

4.4.2. Niedeterminizm

Tutaj rezygnuje się z poprawności algorytmu zachowując szybkość i ogólność.

W praktyce często stosujemy algorytmy typu „Monte Carlo". Zazwyczaj przeprowadzamy kilka iteracji takiego algorytmu i albo otrzymujemy informację, że badana liczba jest złożona, albo zadawalamy się dosyć dużym prawdopodobieństwem, że jest pierwsza.

Dla zilustrowania tego podejścia przypomnę dla przykładu jeden dobrze znany wszystkim algorytm. Pomysł w nim zawarty polega na wykorzystaniu symboli Jacobiego.

Przypuśćmy, że $a \perp n$. Mówimy, że a jest resztą kwadratową modulo n jeśli kongruencja $x^2 \equiv a(mod\ n)$ ma rozwiązanie.

Definicja 4.4.1 (Symbol Legendre'a) *Niech p będzie liczbą pierwszą nieparzystą*
$(\frac{a}{p}) = \begin{cases} 1, & \textit{jeśli } a \textit{ jest resztą kwadratową (mod } p\textit{);} \\ -1, & \textit{jeśli } a \textit{ jest nieresztą kwadratową (mod } p\textit{);} \\ 0, & \textit{jeśli } p \mid a. \end{cases}$

Dla symboli Legendre'a zachodzi poniższe twiedzenie:

Twierdzenie 4.7 *Jeśli p jest nieparzystą liczbą pierwszą, to $(\frac{a}{p}) \equiv a^{(p-1)/2}(mod\ p)$*

Uogólnieniem pojęcia symbolu Legendre'a jest symbol Jacobiego.

Definicja 4.4.2 *Niech $n \in \mathcal{N}$, n nieparzyste, $n = p_1^{e_1} p_2^{e_2} \dots p_k^{e_k}$, wtedy*

$$(\tfrac{a}{n}) = (\tfrac{a}{p_1})^{e_1} (\tfrac{a}{p_2})^{e_2} \cdots (\tfrac{a}{p_k})^{e_k},$$

gdzie wyrażenia w nawiasach z prawej strony to symbole Legendre'a.

Jak widać z powyższej definicji dla liczb pierwszych nieparzystych symbol Jacobiego pokrywa się z symbolem Legendre'a. Do naszych zastosowań będzie przydatne jeszcze następujące twierdzenie:

Twierdzenie 4.8 *Niech n będzie nieparzystą liczbą naturalną, natomiast a dowolną liczbą całkowitą. Symbol Jacobiego $(\frac{a}{n})$ może być obliczony przy użyciu $O((\log a)(\log n))$ operacji na bitach.*

Definicja 4.4.3 *Liczbę nieparzystą złożoną n, taką że $a \perp n$, spełniającą*

$$a^{(n-1)/2} \equiv (\frac{a}{n})(mod\ n)$$

będziemy nazywać pseudopierwszą Eulera przy podstawie a.

Okazuje się, że dla liczb złożonych nieparzystych takich podstaw pseudopierwszości nie jest zbyt dużo. Dokładniej zachodzi:

Twierdzenie 4.9 *Jeśli n jest nieparzystą liczbą złożoną, to kongruencja:*

$$a^{(n-1)/2} \equiv (\frac{a}{n})(mod\ n)$$

ma wśród liczb $1, 2, \dots, n-1$ co najwyżej (n-1)/2 rozwiązań.

Oznacza to, że mamy kolejny sposób dowodzenia złożoności bez podawania dzielnika. Co więcej na takiego świadka złożoności jest stosunkowo łatwo natrafić. Przy k próbach prawdopodobieństwo znalezienia go przekracza $(1-(\frac{1}{2})^k)$.

Algorytm 4.4.1 (SOLOVAY-STRASSEN) *wybierz losowo a ze zbioru* $\{1, 2, \dots, n-1\}$

1. *jeśli $a \notin n_\perp$ zwróć (złożona)*
2. *jeśli $(\frac{a}{n}) \equiv a^{(n-1)/2} (mod\ n)$ zwróć (pierwsza)*
3. *zwróć (złożona)*

Jeśli powyższy algorytm zwraca, że liczba jest złożona, to mamy pewność, że tak jest istotnie. Jeśli wynikiem działania jest pierwsza, to mamy przynajmniej 50 procent pewności, że testowana liczba była rzeczywiście pierwsza.

Warto także odnotować, że iteracja przedstawionego algorytmu zużywa $O((\log n)^3)$ operacji na bitach.

4.4.3. Testy hipotetyczne

Kolejne podejście wskazuje metody badania pierwszości bez formalnego dowodu poprawności tych metod. Najczęściej takie metody oparte są na pewnych hipotezach teorioliczbowych, których nie dało się do tej pory zweryfikować, czy są prawdziwe, jednak badania doświadczalne dostarczają pewnych argumentów, że tak może być.

Badania empiryczne liczb pierwszych sugerują, że rozkładają się one równomiernie wśród różnych klas reszt modulo ustalona liczba n. Jeśli ustalimy n i dobierzemy $a \in n_\perp$, i policzymy jak liczby pierwsze $\le x$ rozkładają się pomiędzy różne klasy reszt modulo n (wybieramy różne a) to narzuci się przypuszczenie, że zbiory takiego podziału będą miały mniej więcej po równo elementów. Niech $\pi(x, n, a)$ oznacza ilość liczb pierwszych nieprzekraczających x dających resztę a z dzielenia przez n. Okazuje się, że nasza nieformalna hipoteza została udowodniona już w 1896 roku.

Twierdzenie 4.10 (de la Valée) *Jeśli $a \perp n$, to $\pi(x, n, a) \sim \frac{x}{\phi(n) \log x}$.*

To się zgadza z naszymi przypuszczeniami, gdyż w przedziale $[1, x]$ mamy około $fracx/\log x$ różnych liczb pierwszych, natomiast interesujących klas reszt modulo n jest dokładnie $\phi(n)$.

Dla uproszczenia wzorów przyda nam się pojęcie całki logarytmicznej:

$$li(x) = \int_2^x dt/\log t.$$

Przypuszczenie 4.1 (ROZSZERZONA HIPOTEZA RIEMANNA) *Niech $a, n \in \mathcal{N}$, takie że $a \perp n$. Dla każdego $\epsilon > 0$ zachodzi:*

$$\pi(x, n, a) = \frac{li(x)}{\phi(n)} + O(x^{1/2+\epsilon}).$$

Taka hipoteza prowadzi do wielu interesujących wniosków odnośnie poszukiwania świadków pierwszości danej liczby naturalnej n. Implikuje ona następujące twierdzenie:

Twierdzenie 4.11 *Załóżmy, że ERH jest prawdziwa. Dla liczby pierwszej p niech $g(p)$ oznacza najmniejszy pierwiastek pierwotny grupy $(Z/(p))^*$, a $\omega = \omega(p-1)$, wówczas:*

$$g(p) = O(\omega^4(\log \omega)^4(\log p)^2) = O((\log p)^6).$$

Okazuje się, że zakładając ERH możemy tak zmodyfikować algorytm Solovaya-Strassena, aby stał się deterministyczny.

Twierdzenie 4.12 *Załóżmy, że ERH jest prawdziwa. Jeśli n jest nieparzystą liczbą złożoną, wtedy istnieje dodatnia liczba naturalna $a < n$, $a \leq 2(\log n)^2$, dla której zachodzi co najmniej jeden z warunków:*

(i) $(a, n) \notin \perp$*, czyli możemy obliczyć NWD(a,n) i otrzymamy świadka złożoności n*

(ii) $(\frac{a}{n}) \neq a^{(n-1)/2} (mod\ n)$.

We wspomnianym wcześniej teście niedeterministycznym możemy więc zastąpić losowy wybór a serią testów dla a ze zbioru $\{1, 2, \ldots, 2(\log n)^2\}$.

Algorytm 4.4.2 (Deterministyczny algorytm SOLOVAYA-STRASSENA) .
dane: testowana liczba n
dla $a = 2, 3, \ldots, min(n-1, \lfloor 2(\log n)^2 \rfloor)$ *wykonuj*

1. *jeśli* $(a, n) \notin \perp$ *zwróć (złożona)*
2. *jeśli* $(\frac{a}{n}) \neq a^{(n-1)/2} (mod\ n)$ *zwróć (złożona)*

zwróć (pierwsza)

Twierdzenie 4.13 *Załóżmy prawdziwość ERH. Dla $n \geq 3$ algorytm (4.4.2) zwraca (pierwsza) wtedy i tylko wtedy, gdy n jest faktycznie pierwsze. Algorytm (4.4.2) wykonuje $O((\log n)^5)$ operacji na bitach.*

Oczywistym wnioskiem z tych rozważań jest:

Wniosek 4 *Jeśli zachodzi ERH, to PIERWSZE$\in \mathcal{P}$.*

4.4.4. Testy warunkowe

Charakterystykę liczb pierwszych, którą wcześniej zapisywaliśmy przy użyciu pojęć z teorii grup można zapisać więc w sposób następujący:

Twierdzenie 4.14 *Liczba $n \in \mathcal{N}$ jest pierwsza wtw., gdy istnieje $a \in \mathcal{N}$, takie że $(n \mid a^{n-1} - 1)$ oraz nie zachodzi $(n \mid a^{(n-1)/q} - 1)$ dla każdej liczby pierwszej q dzielącej $n - 1$.*

Algorytm 4.4.3 (FELLOWS-KOBLITZ) *FK(n,k,$e_1, e_2, \ldots, e_k, q_1, q_2, \ldots, q_k$)*
Dane: $n, k, e_1, e_2, \ldots, e_k \in \mathcal{N}$, liczby pierwsze $q_1, q_2, \ldots, q_k$ spełniające $n-1 = q_1^{e_1} \ldots q_k^{e_k}$

1 dla $a = 2, 3, \ldots, \lfloor (\log n)^2 \rfloor$ wykonaj

2 jeśli nie zachodzi $a^{n-1} \equiv 1 (mod n)$ zwróć (złożona)

3 oblicz $rz_n a$

4 dla każdej liczby pierwszej $q \mid rz_n a$ wykonaj

5 jeśli $(a^{(rz_n a)/q}, n) \notin \perp$ zwróć (złożona)

6 oblicz $h = NWW_{2 \leq a \leq (\log n)^2} rz_n a$

7 jeśli $h < \sqrt{n}$ zwróć (złożona)

8 zwróć (pierwsza)

Twierdzenie 4.15 *Algorytm (4.4.3) jest poprawny i jego złożoność wynosi*

$$O((\log n)^6 / (\log \log n))$$

operacji na bitach.

Dowód:

Na początek zajmijmy się poprawnością. Jeśli w kroku (2) nie zachodzi $a^{n-1} \equiv 1 (mod\ n)$, to n nie może być pierwsze (byłoby to sprzeczne z tw. Eulera). Jeśli algorytm się nie zatrzymał w tym miejscu oznacza to, że $a^{n-1} \equiv 1 (mod\ n)$. Możemy więc użyć algorytmu (4.2.1) do policzenia $rz_n a$. Jeśli w kroku (5) znajdziemy parę liczby nieprostopadłych, będzie to świadczyło o istnieniu dzielnika właściwego liczby n, czyli o jej złożoności.

Przypuśćmy, że osiągnęliśmy krok (6). Twierdzę, że $rz_n a = rz_p a$ dla każdej liczby pierwszej p, która jest dzielnikiem n. Jest tak ponieważ jeśli $p \mid n$ oraz $a^{rz_n a} \equiv 1 (mod\ n)$, to także $a^{rz_n a} \equiv 1 (mod\ p)$, czyli $rz_p a \mid rz_n a$. Gdyby jednak $rz_p a < rz_n a$ wówczas $a^{(rz_n a)/q} \equiv 1 (mod\ p)$ dla pewnej liczby pierwszej q będącej dzielnikiem $rz_n a$. To jednak nie jest możliwe, bo wykrylibyśmy taką sytuację w kroku (5).

Niech p będzie najmniejszą liczbą pierwszą dzielącą n, wtedy $p \leq \sqrt{n}$. Ponieważ jak wcześniej pokazaliśmy $rz_n a = rz_p a$ dla $a = 2, 3, \ldots, \lfloor (\log n)^2 \rfloor$, to $h \mid p-1$. Jeśli w kroku (7) $h \geq \sqrt{n}$, to $p - 1 \geq \sqrt{n}$, co daje sprzeczność i oznacza, że n w tym przypadku nie może mieć dzielnika właściwego, czyli musi być pierwsza.

Teraz przypuśćmy, że sterowanie znalazło się w punkcie (7) i mamy sytuację $h < \sqrt{n}$. Przyjmijmy, że n jest pierwsze i rozważmy podgrupę G grupy $(Z/(n))^*$ generowaną przez elementy $2, 3, \ldots, \lfloor (\log n)^2 \rfloor$. Ponieważ dla n pierwszego $(Z/(n))^*$ jest cykliczna, to także jej skończenie generowana podgrupa jest cykliczna oraz w naszym przypadku $h=|G|$. Zauważmy dalej, że G zawiera wszystkie $(\log n)^2$-gładkie liczby naturalne $\leq n$. Wynika stąd, że $h \geq \Psi(n, (\log n)^2)$. Z tw. (3.1) wiemy, że $\Psi(n, (\log n)^2) > \sqrt{n}$. Daje to sprzeczność, czyli n musi być złożone.

Pozostało więc już tylko pokazać ograniczenie na czas działania algorytmu. Czas wykonania pętli jest zdominowany przez operacje wykonywane w punktach (3) i (4). Dla każdego a te kroki mogą być wykonane w czasie $O((\log n)^4 / (\log \log n))$ operacji na bitach. Pętla jest wykonywana $O((\log n)^2)$ razy, więc sumaryczny czas wynosi $O((\log n)^6 / (\log \log n))$.$\bullet$

Mamy więc pierwszy poważny warunkowy test deterministyczny, który działa w czasie wielomianowym. Okazuje się, że opisany powyżej algorytm można trochę poprawić.

Twierdzenie 4.16 *Jeśli w głównej pętli algorytmu (4.4.3) będziemy brać tylko liczby pierwsze z wymienionego przedziału, to nie wpłynie to na poprawność i poprawi złożoność do* $O((\log n)^6/(\log\log n)^2)$.

Dowód:

Dowód poprawności tak zmodyfikowanego algorytmu pozostaje bez zmian w stosunku do (4.15). Jedyne wymaganie co do zbioru liczb, po których przebiega a w pętli jest takie, żeby jeśli znajdziemy się w punkcie (6) mieć pewność, że grupa generowana przez te elementy zawiera wszystkie liczby $(\log n)^2$*-gładkie spośród elementów* $(Z/(n))^*$.

Koszt wyznaczenia liczb pierwszych ze zbioru $\{2, 3, \ldots, \lfloor(\log n)^2\rfloor\}$ *jest zaniedbywalny w stosunku do szacowania na działanie całego algorytmu. Ilość kroków pętli zmniejszy się do* $O(\pi((\log n)^2))$*, czyli dokładniej* $O((\log n)^2/(\log\log n))$*. Łączny czas działania wyniesie więc* $O((\log n)^6/(\log\log n)^2)$.

Warto jeszcze w tym miejscu zauważyć, że jeśli po każdej iteracji pętli algorytmu (4.4.3) będziemy na bieżąco wyliczać h, to w momencie, gdy $h \geq \sqrt{n}$ możemy przerwać działanie programu z wynikiem (testowana liczba jest pierwsza). Jest to bardzo ważne w praktycznej implementacji tego algorytmu.

Nie zawsze jednak jest łatwo znaleść rozkład liczby $n-1$, w ogólności przecież problem faktoryzacji jest co najmniej tak trudny jak testowanie pierwszości. Omówiony algorytm może być w praktyce stosowany do testowania liczb specjalnej postaci (np. $2p^k+1$, gdzie p pierwsze). Można na nim także oprzeć pomysł na algorytm do znajdowania dzielników. Niech s będzie wielomianem. Algorytm będzie zwracać najmniejszy dzielnik pierwszy liczby n, lub (nie wiem).

Algorytm 4.4.4 *FK-FACTOR(n).*
Dana testowana liczba $n \in \mathcal{N}$

1. *Dla* $a \in \{p : p \text{ } pierwsze \wedge p < s(\log n)\}$ *wykonuj*

   jeśli $a \mid n$*, to zwróć (a)*

2. *jeśli* $s(\log n) \geq \sqrt{n}$ *zwróć (n)(liczba jest pierwsza)*

3. *w tym miejscu staramy się wykazać, że liczba jest pierwsza, jeśli nam się nie uda to algorytm zwróci (nie wiem) Faktoryzujemy (n-1) wywołując rekurencyjnie FK-FACTOR, jeśli otrzymaliśmy (nie wiem), to także zwracamy (nie wiem)*

4. *wywołujemy FK(n, faktoryzacja n-1), jeśli n pierwsze, to zwracamy (n)*

5. *zwracamy (nie wiem)*

Przedstawiony algorytm, choć działa w czasie wielomianowym jest oczywiście pozbawiony ogólności. Dla liczb będących iloczynami dużych liczb pierwszych np. rzędu $O(n^{1/2})$ otrzymamy odpowiedź (nie wiem).

Naturalnym wydaje się pytanie, czy warunku na faktoryzację $n-1$ nie można jakoś osłabić. Okazuje się, że tak:

Twierdzenie 4.17 *POCKLINGTON*
Niech n będzie liczbą naturalną >1, natomiast s dodatnim dzielnikiem n-1. Jeśli istnieje a, spełniające:

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

ii $(a^{(n-1)/q} - 1) \perp n$ *dla wszystkich dzielników pierwszych q liczby n-1*

wówczas każdy dzielnik pierwszy p liczby n spełnia: $p \equiv 1 (mod\ s)$.

Dowód:

Niech p będzie dzielnikiem pierwszym n. Niech $r = rz_n a$. *Z warunku (ii) wynika, że* $s \mid r$. *Dalej mamy* $a^r \equiv 1 (mod\ n)$, *czyli także* $a^r \equiv 1 (mod\ p)$. *Gdyby dla pewnego dzielnika pierwszego q liczby s* $a^{r/q} \equiv 1 (mod\ p)$, *wówczas p byłoby wspólnym dzielnikiem liczb* $(a^{(n-1)/q} - 1)$ *i n i mielibyśmy sprzeczność z założeniem(wykorzystaliśmy milcząco fakt, że*

$$(a^{r/q} - 1) \mid (a^{(n-1)/q} - 1),$$

bo $(r/q) \mid (n - 1/q)$*). Wynika stąd, że* $s \mid rz_p a$. *Jednocześnie z tw. Lagrange'a* $rz_p a \mid p - 1$, *czyli* $s \mid p - 1$.•

Jeśli w tw. POCKLINGTONA założymy dodatkowo $s > \sqrt{n} - 1$, to otrzymamy tezę, że każdy dzielnik pierwszy musi być większy od s, czyli n musi być pierwsze.

Załóżmy, że znamy faktoryzację częściową liczby $n - 1 = st$, gdzie $s > \sqrt{n} - 1$, $s \perp t$ oraz potrafimy rozłożyć s na czynniki. Jeśli teraz znajdziemy element $a \in n_\perp$ rzędu s, to badanie pierwszości n jest następujące:

> dla każdej liczby pierwszej $p \mid s$ sprawdzamy, czy $(a^{(n-1)/p} - 1) \perp n$, jeśli tak, to n pierwsza, jeśli nie to n złożona.

Warto jeszcze zauważyć, że element a nie musi być rzędu dokładnie s, wystarcza, aby s dzieliło rząd a i nadal powyższe postępowanie będzie poprawne.

Jeśli zrezygnujemy z założenia, że $s > \sqrt{n} - 1$, wówczas w powyższym schemacie albo otrzymamy dowód złożoności n, albo informację, że każdy dzielnik pierwszy jest postaci $ks+1$, co też może być użyteczne w pewnych przypadkach.

Twierdzenie 4.18 *Niech r,s i n będą liczbami naturalnymi, spełniającymi:*

$$0 \leq r < s < n,\ s > \sqrt[3]{n},\ r \perp s.$$

Wówczas istnieje co najwyżej 11 naturalnych dzielników n przystających do r modulo s. Ponadto można wyznaczyć te dzielniki w czasie wielomianowym.

Dowód można przeczytać w [5].

Przyjmijmy w takim razie, że potrafimy sfaktoryzować $s \mid n - 1$ i $s > \sqrt[3]{n}$. Jeśli ponadto uda nam się znaleźć element a spełniający oba warunki z twierdzenia (4.17), to uzyskamy, że każdy dzielnik pierwszy n musi dawać resztę 1 z dzielenia przez s. Teraz jeśli zastosujemy algorytm, o którym mowa w tw. (4.18) to dostaniemy wszystkie ewentualne dzielniki n. Sprawdzamy, czy któryś z nich dzieli n i możemy już roztrzygnąć, czy n było pierwsze.

Udało się więc po raz kolejny osłabić warunek o faktoryzacji $n-1$. Straciliśmy niestety determinizm, który był cechą algorytmu Fellowsa-Koblitza. Nie mamy przepisu w jaki sposób odnaleźć element a spełniający założenia tw. Pocklingtona poza losowaniem. Jeśli n jest rzeczywiście pierwsze to chociażby wylosowanie elementu pierwotnego załatwi sprawę. Losując liczbę ze zbioru $\{2, 3, \dots, n-1\}$ mamy szansę wyciągnięcia generatora $O(n/(\log\log n))$, czyli oczekiwana ilość losowań do momentu uzyskania generatora wynosi $O(\log\log n)$ prób.

Istnieją również testy warunkowe wykorzystujące znajomość rozkładu na czynniki liczb n+1, n^2+1, $n^2 \pm n+1$. Jednym z podejść jest łączenie takich testów. Jak pokazaliśmy nawet częściowa znajomość rozkładu $n-1$ zawęża krąg poszukiwań ewentualnych dzielników.

Najbardziej znany pomysł polega na połączeniu strategii testowania pierwszości opartej na rozkładzie $n-1$ ze strategią bazującą na rozkładzie $n+1$. Działa to w ten sposób, że próbujemy równocześnie faktoryzować $n-1$ oraz $n+1$ tak długo, aż jakiś kawałek będzie tzw. prawdopodobną liczbą pierwszą (ang. „probable prime"), wtedy stosujemy naszą strategię rekurencyjnie dla tego kawałka i jeśli uda nam się wykazać, że był on rzeczywiście liczbą pierwszą to dalej sobie już poradzimy, bo mamy przecież faktoryzację $n-1$ lub $n+1$.

Nie słyszałem o pomysłach łączenia strategii dla innych liczb z kręgu n^2+1, $n \pm n+1$. Ciekawym kierunkiem badań mogłyby być badania ilościowe, dla jak wielu liczb n jedna z liczb $n \pm 1$, n^2+1, $n \pm n+1$ daje się łatwo faktoryzować. Można by też próbować łączyć ze sobą częściowe faktoryzacje tych wyrażeń i badać do jakich to prowadzi implikacji.

4.5. Generowanie dużych liczb pierwszych

Ważnym zagadnieniem praktycznym jest umiejętność generowania dużych liczb pierwszych. Niech k będzie długością szukanej liczby.

Najczęściej są spotykane dwa podejścia:

i Losujemy dowolną liczbę k-cyfrową. Badamy ją jednym z algorytmów probablistycznych z klasy $\mathcal{RP}$ i albo dowiadujemy się, że jest na pewno złożona albo mamy dostatecznie duże prawdopodobieństwo, że jest pierwsza. Proces powtarzamy tak długo, aż uzyskamy liczbę, która jest pierwsza z zadawalającym nas prawdopodobieństwem.

ii Niech $S \subset \mathcal{N}$, natomiast $f : S \to \{0, 1\}$ będzie funkcją zwracającą 1 wtw., gdy argument jest liczbą pierwszą. Zakładamy ponadto, że potrafimy obliczać funkcje f w czasie wielomianowym względem długości argumentu. Podejście polega więc na losowaniu liczb k-cyfrowych $x \in S$ oraz obliczaniu wartości $f(x)$. Proces powtarzamy tyle razy, aż otrzymamy liczbę pierwszą.

Pierwsze podejście jest w tej chwili najczęściej używane do zastosowań praktycznych. Prawdopodobieństwo, że losowa liczba k-cyfrowa jest pierwsza wynosi w przybliżeniu $1/k$, co oznacza, że średnio po k losowaniach dostaniemy liczbę pierwszą. Będziemy więc potrzebować średnio wielomianowej liczby operacji do znalezienia „prawdopodobnej liczby pierwszej".

Drugie podejście, żeby miało sens, musi wymagać coś od zbioru S. Oto te wymagania wraz z krótką ich motywacją:

- Zbiór S jest nieskończony.
- Zawiera nieskończenie wiele liczb pierwszych.
- Jest dostatecznie gęsty, czyli mówiąc nieformalnie proporcja liczb pierwszych do całości odpowiada mniej więcej proporcjom w zbiorze $\mathcal{N}$.

Ogólnie chodzi o to, aby generować w miarę losowe liczby pierwsze. Gdybyśmy dla przykładu przyjeli, że $S = \{2^p - 1 | \textit{gdzie p jest pierwsze}\}$ wówczas mielibyśmy spełniony na pewno tylko pierwszy warunek. Drugi być może, natomiast trzeci na pewno nie. Jaki mamy pożytek z tego zbioru, jeśli wszystkich liczb pierwszych długości k jest w tym zbiorze co najwyżej k. Jeśli byśmy chcieli wykorzystywać elementy tego zbioru w systemach kryptograficznych do generacji klucza, to jak widać taki klucz byłby łatwy do odgadnięcia.

4.5.1. Nieskończone zbióry liczb pierwszych z szybkim testem pierwszości

W dalszej części tego podrozdziału przedstawię ciekawy pomysł na testowanie pierwszości oraz generowanie dużych liczb pierwszych pochodzący od Pinza, Steigera i Szemerediego.

Udzielili oni twierdzącej odpowiedzi na pytanie o istnienie nieskończonego zbioru liczb pierwszych, którego funkcję charakterystyczną potrafimy obliczać w czasie wielomianowym. Przedstawiony przez nich zbiór wraz z algorytmem testowania pierwszości spełnia również nasze trzecie wymaganie, tzn. jest dostatecznie gęsty. Do testowania pierwszości wystarczające okażą się pomysły z elementarnej teorii liczb, natomiast twierdzenia o gęstości wymagają już silnych narzędzi analitycznych.

Zbiory P i Q

Na początek zdefiniujmy zbiory, których własności będziemy badać:

(1)
$$\begin{aligned} &I_m = \{n : 27^{m-1} < n < 27^m,\, n \equiv 1(\text{mod } 3^m)\},\\ &Q_m = \{p \in I_m : p \textit{ pierwsze}\},\\ &P_m = \{p \in Q_m : \exists l, 1 < l \leq c(\log p)^6, l^{(p-1)/3} \not\equiv 1(\text{mod p})\},\end{aligned}$$

gdzie c jest odpowiednio dobraną stałą większą od 0, będziemy używać następujących oznaczeń

(2)
$$Q = \bigcup_{m>3} Q_m,$$

$$P = \bigcup_{m>3} P_m.$$

Będziemy chcieli pokazać, że P jest nieskończonym zbiorem liczb pierwszych, do którego przynależność można testować w czasie wielomianowym. Zbioru Q będziemy używać do szybkiego generowania liczb pierwszych z danego przedziału.

Pewną motywacją do rozpatrywania liczb przystających do 1 modulo 3^m jest dobrze znane twierdzenie:

Twierdzenie 4.19 (Dirichlet(1837)) *Jeśli $c \perp d$, to postęp arytmetyczny $a_j = c + jd$ zawiera nieskończenie wiele liczb pierwszych.*

Jeśli więc przyjmiemy $c = 1$ i $d = p^k$ będzie potęgą liczby pierwszej, to a_j będziemy w stanie szybko testować na pierwszość przynajmniej tak długo, jak $j \leq p^{2k}$. Wtedy bowiem p^k, jako sfaktoryzowany kawałek $a_j - 1$ będzie spełniać $p^k \geq \sqrt[3]{a_j}$, a to jak już pisaliśmy można szybko testować na pierwszość.

Testowanie przynależności do P i Q

W celu zilustrowania kilku ciekawych właściwości zbiorów P i Q udowodnijmy następujący lemat.

Lemat 7 *Niech $n \geq 27^m$. Jeśli istnieje a, spełniające $a^{3^m} \equiv 1 (mod\ n)$ oraz $(a^{3^{m-1}} - 1) \perp n$ wtedy liczba n jest pierwsza, bądź n jest iloczynem dwóch liczb pierwszych $\equiv 1 (mod\ 3^m)$.*

Dowód:

Jeśli r jest dzielnikiem pierwszym n, to $a^{3^m} \equiv 1 (mod\ r)$ oraz $a^{3^{m-1}} \neq 1 (mod\ r)$. Oznacza to, że $rz_r a = 3^m$, czyli $3^m \mid (r-1)$. Warunek $n \leq 27^m$ implikuje, że czynników pierwszych $\equiv 1 (mod\ 3^m)$ może być co najwyżej 2.

W jaki sposób możemy wykorzystać wiedzę, że n jest iloczynem co najwyżej dwóch liczb pierwszych postaci $k3^m + 1$?

Lemat 8 *Przypuśćmy, że $n = (x3^m + 1)(y3^m + 1) \leq 27^m$, $x, y \geq 1$ oraz zachodzi $n = C9^m + D3^m + 1$, dla $0 < C < 3^m$ i $1 \leq D \leq 3^m$. Wtedy $xy = C$ i $x + y = D$.*

Zauważmy, że jeśli n ma obie reprezentacje wskazane w lemacie, to $D^2 - 4C$ jest kwadratem liczby naturalnej. Zachodzi też implikacja w drugą stronę:

Wniosek 5 *Niech $n = C9^m + D3^m + 1$, gdzie $0 < C < 3^m$, $1 \leq D \leq 3^m$. Istnieją $x, y \in \mathcal{N}$, takie że $n = (x3^m + 1)(y3^m + 1)$ wtedy i tylko wtedy, gdy $D^2 - 4C$ jest kwadratem.*

Zbierzmy więc uzyskane obserwacje:

Twierdzenie 4.20 *Niech $n = C9^m + D3^m + 1$, $0 < C < 3^m$, $1 \leq D \leq 3^m$. Liczba n jest pierwsza wtedy i tylko wtedy, gdy zachodzą oba z poniższych warunków.*

i $D^2 - 4C$ <u>nie</u> jest kwadratem

ii Istnieje l, takie że $l^{n-1} \equiv 1 (mod\ n)$ oraz $(l^{(n-1)/3} - 1) \perp n$.

Mamy więc już wystarczającą wiedzę do konstrukcji algorytmu badającego przynależność do P_m.

Algorytm 4.5.1 .
Dane: $n \in \mathcal{N}$

1. *Znajdź C,D, takie że: $n = C9^m + D3^m + 1$, $0 < C < 3^m$, $1 \leq D \leq 3^m$.*
2. *Jeśli $D^2 - 4C$ jest kwadratem zwróć (złożona)*
3. *Znajdź l, takie że: $1 < l \leq c(\log n)^6$, $l^{n-1} \equiv 1 (mod\ n)$, $(l^{(n-1)/3} - 1) \perp n$.*

   jeśli się udało, to zwróć ($n \in P_m$, l)
4. *Zwróć ($n \notin P_m$).*

Odnotujmy w tym miejscu oszacownie złożoności przedstawionego algorytmu:

Twierdzenie 4.21 *Algorytm (4.5.1) zatrzymuje się w czasie $O((\log n)^9)$ i albo*

(a) stwierdza, że $n \notin P_m$, albo

(b) generuje certyfikat pierwszości l długości $O(\log n)$, który można zweryfikować pozytywnie w czasie $O((\log n)^3)$.

Gęstość zbiorów P i Q

To, że zbiór Q jest nieskończony wynika bezpośrednio z tw. Dirichleta. Okazuje się, że jest prawdziwy następujący lemat:

Lemat 9 $|P_m| = |Q_m|(1 + O(\frac{1}{m}))$.

Przy pomocy silnych twierdzeń z analitycznej teorii liczb można uzyskać następujące:

Twierdzenie 4.22 *Jeśli $x > x_0$, to*

$$|\{p \in Q : p \leq X\}| \geq |\{p \in P : p \leq x\}| > \frac{cx^{2/3}}{\log x}.$$

Jest to bardzo ważny rezultat, gdyż odnosi się do gęstości zbiorów P i Q.

Szybkie generowanie dużych losowych liczb pierwszych

Pozostaje nam więc już tylko wykorzystać zdobytą wiedzę do ułożenia algorytmu generującego liczby pierwsze.

Algorytm 4.5.2 .
Dane: m - ograniczenie na długość szukanej liczby

1. *Wybierz losowo $C, D, 3^{m-3} \leq C < 3^m, 1 \leq D \leq 3^m$.*
2. *Użyj algorytmu (4.5.1) do zbadania pierwszości liczby $n = C9^m + D3^m + 1$.*

   Jeśli n pierwsza, zwróć (n)
   Jeśli nie, to wróć do (1).

Z twierdzeń o gęstości zbioru P wynika, że prawdopodobieństwo natrafienia w nim na liczbę pierwszą m-cyfrową jest rzędu $\frac{1}{O(m)}$. Musimy więc średnio wykonać $O(m)$ testów pierwszości zanim wygenerujemy szukaną liczbę, co daje w sumie oczekiwaną złożoność rzędu $O(m^{10})$. Można ten rezultat poprawić stosując niedeterministyczną wersję algorytmu testowania przynależności do Q, wtedy uzyskuje się czas rzędu $O(m^4)$.

Rozdział 5

Podsumowanie podstawowych faktów, najszybszy (asymptotycznie) znany test pierwszości

W tym rozdziale postaram się zebrać rezultaty i przemyślenia wynikające z tej pracy. Podam kilka wskazówek, co jest istotne przy konstrukcji dobrych algorytmów badających pierwszość. Na koniec opowiem o konstrukcji najszybszego asymptotycznie testu pierwszości i jego możliwościach. Jak się okaże ten potężny algorytm wymyślony i rozwijany przez zespół naukowców na przestrzeni dwudziestu lat wykorzystuje też najbardziej elementarne pomysły, które pojawiły się w poprzednim rozdziale niniejszej pracy.

5.1. Co wiemy o zbiorze PIERWSZE?

Nie znamy jak na razie odpowiedzi na pytanie, czy $PIERWSZE \in \mathcal{P}$. Znane są jednak pewne słabsze rezultaty. Oto one:

1. W 1966 roku Minsky i Papert wykazali, że język PIERWSZE nie jest regularny (nie jest akceptowany przez automat skończony).

2. Dwa lata później Hartmanis i Shank ([10]) oraz niezależnie Schützenberger udowodnili, że $PIERWSZE$ nie jest językiem bezkontekstowym.

3. W roku 1973 Hsia i Yeh opublikowali dowód, że język $\{0^p : p\ pierwsze\}$ jest akceptowany przez deterministyczny, zatrzymujący się „3-marker" automat.

4. W roku 1975 Pratt przedstawia dowód twierdzenia(4.6) $PIERWSZE \in \mathcal{NP}$.

5. Na początku lat dziewięćdziesiątych pojawił się formalny dowód przynależności języka $PIERWSZE$ do klasy ZPP. Algorytm podany wówczas przez Adlemana i Huanga wydaje się jednak bezużyteczny w praktyce.

5.2. Ogólne wskazówki odnośnie projektowania testów pierwszości

W tym podrozdziale postaram się zebrać i podsumować wnioski, które pojawiły się w tej pracy.

Od przyzwoitego testu wymagaliśmy trzech wydaje się sensownych postulatów. Są to:

- ogólność - chcemy, aby test działał dla dowolnych liczb naturalnych
- poprawność - chcemy potrafić udowodnić własność poprawności takiego algorytmu
- efektywność - uzyskany test powinien działać w czasie wielomianowym względem długości liczby na wejściu

Czasem jeszcze dodaje się do tego generowanie ceryfikatu pierwszości, który można zweryfikować w czasie wielomianowym.

Jak się okazuje te trzy na pierwszy rzut oka niezbyt wygórowane wymagania nie jest łatwo spełnić równocześnie. Wszystkie podejścia polegały w praktyce na osłabieniu któregoś z przedstawionych warunków. Rezygnując z ogólności otrzymaliśmy testy warunkowe, które wymagały pewnej dodatkowej wiedzy do poprawnego i efektywnego działania. Uzyskanie tej dodatkowej wiedzy było w ogólności trudniejszym problemem od problemu testowania pierwszości. Rezygnując z poprawności dostawaliśmy testy, które asymptotycznie działały szybko, chociaż nie dało się przeprowadzić formalnego dowodu, że są one poprawne.

W poprzednim rozdziale przedstawiłem krótki przegląd najważniejszych koncepcji i algorytmów testujących pierwszość. Najsilniejsze w praktyce algorytmy testowania pierwszości bazują na umiejętnym połączeniu kilku z przedstawionych pomysłów.

Rezygnując z efektywności dostajemy porządny test, który działa dla wszystkich liczb, ma własność poprawności, ale niestety jest asymptotycznie wolny. Poniżej przedstawię ogólny schemat takiego testu, który mimo tego, że ma on złożoność podwykładniczą jest na tyle dobry, że dla liczb przynajmniej do 1000 cyfr działa wyraźnie szybciej niż większość testów przedstawionych w poprzednim rozdziale. Najlepszym ze znanych algorytmów deterministycznych(kryterium jest złożoność obliczeniowa) jest test Adlemana-Pomerance-Rumely. Działa on w czasie $O((\log n)^{\log\log\log n})$ i przy mocy obliczeniowej współczesnych komputerów jego implementacje powinny już radzić sobie w rozsądnym czasie z liczbami o 1000-cu cyfr dziesiętnych.

5.3. Test sum Jacobiego

Przypomnę, że pomysł na badanie postaci dzielników liczby n z rozdziału (4.4) polegał na przeprowadzeniu pewnej liczby testów dla wszystkich dzielników pierwszych ustalonej liczby s: $n \equiv 1 (\mathrm{mod\ s})$. Jeśli, wszystkie z nich dawały wynik pozytywny, to można było wnioskować, że ewentualne dzielniki pierwsze liczby n muszą być postaci $ks + 1$. Jeśli ponadto $s > \sqrt{n}$, to od razu można było wnioskować pierwszość, dla $s > \sqrt[3]{n}$ istniał szybki algorytm generowania wszystkich dzielników n, spełniających $d \equiv 1 (\mathrm{mod\ s})$.

Można ten pomysł uogólnić na inne liczby s, niekoniecznie dzielące $n - 1$.

Poddajemy liczbę n serii testów o następujących własnościach:

- jeśli n jest pierwsza, to przechodzi przez te testy,
- jeśli n przejdzie przejdzie testy, to mamy informacje o jej możliwych dzielnikach, ta informacja w pewnych przypadkach może być na tyle precyzyjna, że pozwala szybko odpowiedzieć na pytanie, czy n jest pierwsza.

Jeśli więc pewna ilość specjalnie zaprojektowanych testów „typu Fermata" daje wynik pozytywny dla n,s i wszystkich dzielników pierwszych q liczby s, to okazuje się, że każdy dzielnik pierwszy liczby n przystaje do pewnej potęgi n modulo s. Korzystając z takiego ogólniejszego schematu możemy przyjąć za s dowolną liczbę $> \sqrt{n}$. Zakładając, że pojedyncze testy dla $(n, s = q_1^{e_1} \dots q_k^{e_k})$ potrafimy przeprowadzić efektywnie możemy dowodzić pierwszość n weryfikując, że żadna z liczb (n^i mod s) nie jest dzielnikiem n.

Tego typu podejście ma oczywiście sens pod warunkiem, że potrafimy generować wszystkie (n^i mod s) w rozsądnym czasie. Okazuje się, że jeśli odpowiednio dobierzemy s, to będzie to możliwe. Przypuśćmy, że s jest bezkwadratowa (nie ma dzielników pierwszych w potęgach większych od 1), ponadto $n \perp s$. Z małego twierdzenia Fermata(1.5) otrzymujemy, że $n^{q-1} \equiv 1(mod\ q)$ dla wszystkich $q \mid s$, skąd $n^t \equiv 1(\text{mod s})$, gdzie $t = NWW\{q-1 : q \mid s\}$. Wystarczy więc rozpatrywać tylko liczby postaci (n^i mod s), gdzie $i \in \{1, 2, \dots, t-1\}$. Aby taki schemat był efektywny t musi być możliwie małe w stosunku do n, co oznacza, że różne $q-1$, gdzie $q \mid s$ powinny mieć jak najwięcej wspólnych dzielników. Zamiast więc wybierać różne q i na ich podstawie obliczać t sensownym wydaje się zrobić odwrotnie, tzn. dobrać t jako produkt wielu małych liczb pierwszych i zbadać jak wiele liczb pierwszych q ma tę własność, że $q-1 \mid t$.

Weźmy dla przykładu $t = 2^4 \cdot 3^2 \cdot 5 \cdot 7 = 5040$. Dla 27 różnych liczb pierwszych q zachodzi $q-1 \mid t$. Iloczyn tych liczb pierwszych przekracza 10^{48}. Jeśli więc zdecydowalibyśmy się na wykorzystanie $s = \prod_{q-1|t} q$, to otrzymamy test dla liczb do 96 cyfr dziesiętnych. Przy czym druga faza tego testu będzie sprowadzać się do wykonania 5039 próbnych dzieleń.

Twierdzenie 5.1 (Odlyzko, Pomerance) *Dla każdego $n > e^e$ istnieje*

$$t < (\log n)^{c \cdot \log\log\log n},$$

takie że odpowiadające mu s jest $> \sqrt{n}$. Stała c jest oczywiście niezależna od n.

Jeśli chodzi o testy, które nazwaliśmy testami „typu-Fermata" to polegają one w wielkim skrócie na obliczaniu wysokich potęg tzw. sum Gauss-a i ich dokładne sformułowanie wymaga praktycznie napisania osobnej pracy na ten temat. W 1984 roku Lenstra i Cohen w praktyce zaimplementowali test Adlemana-Pomerance-Rumely-ego. Uprościli trochę używane w nim pojęcia algebraiczne używając sum Jacobiego zamiast sum Gaussa, co zdecydowanie uprościło całą teorię i sam algorytm.

W praktycznej implementacji swojego testu Lenstra i Cohen wykorzystują w istocie wszystkie wskazane przez nas w 4 rozdziale pomysły. Sam test sum Jacobiego jest wykonywany dopiero po przejściu przez liczbę n całej masy testów wstępnych. Dopiero, gdy n jest „prawdopodobną liczbą pierwszą" przystępujemy do właściwego testu.

Taka faza wstępna składa się z następujących testów:

- próbne dzielenie przez małe liczby pierwsze wielkości $O(\log n)$,
- wybieramy liczbę m i przeprowadzamy m-razy kilka znanych probablistycznych testów złożoności(min. Millera-Rabina i Solovaya-Strassena),

- kombinowany test Lucasa-Lehmera dla $(n-1)$ i $(n+1)$.

Jeśli liczba n przejdzie przez wszystkie testy wstępne, to staje się tzw. "prawdopodobną liczbą pierwszą"(ang. *probable prime*) i używamy właściwego testu sum Jacobiego do dowiedzenia jej pierwszości. Autorzy implementacji przedstawionego testu podkreślają, że wprawdzie zdarzyło się parę razy, że jakaś liczba przeszła przez fazę wstępną i okazała się potem być złożona, zawsze jednak spowodowane było to błędem w programie.

Poniżej przedstawiam skrótowy opis właściwego testu sum Jacobiego.

Etap 1. Wybierz $t, s \in \mathcal{N}$ o następujących własnościach:

- t jest małe w stosunku do n
- $s > \sqrt{n}$
- $a^t \equiv 1(\text{mod s})$ dla wszystkich $a \perp s$.
- znamy pełną faktoryzację t i s

Etap 2. Przeprowadź dla n serię testów. Jeśli któryś z nich zakończy się porażką to kończymy działanie algorytmu i zwracamy wynik „n jest złożona". Używając informacji uzyskanej z testów próbujemy udowodnić następujące przypuszczenie:

dla każdego $r \mid n$ istnieje $i \in \{0, 1, \ldots, t-1\}$, spełniające $r \equiv n^i(\text{mod s})$.

Etap 3. Sprawdzamy wszystkie dzielniki d liczby n, takie że $d \equiv n^i(\text{mod s})$.

Warunek $s > \sqrt{n}$ można jeszcze poprawić. Okazuje się, że wystarczy, aby $s > \sqrt[3]{n}$, wtedy dla każdego $r_i = (n^i \text{ mod s})$, gdzie $i \in \{0, 1, \ldots, t-1\}$ użyjemy algorytmu z twierdzenia (4.18) do znalezienia wszystkich dzielników d liczby n, spełniających $d \equiv r_i(\text{mod s})$. Takie podejście pozwala nam w podobnym czasie testować liczby o połowę dłuższe. Przykładowo dla $t = 5040$ możemy niewielkim kosztem dodatkowych operacji testować liczby 144 cyfrowe, a przypomnę dla $s > \sqrt{n}$ mogliśmy badać liczby o co najwyżej 96 cyfrach dziesiętnych.

Przedstawiony test sum Jacobiego ma jednak poważną wadę. Nie produkuje on dowodu pierwszości liczby, która przeszła pozytywnie test. Pierwszość przebadanej liczby można udowodnić jedynie powtarzając przeprowadzone przez program obliczenia.

Bibliografia

[1] Eric Bach, Jeffrey Shallit, *Algorithmic Number Theory*, MIT Press, Londyn 1996.

[2] H. Cohen, A. K. Lenstra, „Primality Testing and Jacobi Sums”, Mathematics Of Computation Vol. 42, Num. 165, Styczeń 1984, str. 297-330.

[3] H. Cohen, A. K. Lenstra, „Implementation of a New Primality Test”, Mathematics Of Computation Vol. 48, Num. 177, Styczeń 1987, str. 103-121.

[4] N. Koblitz, *Wykład z teorii liczb i kryptografii*, tłum. W. Guzicki, WNT, Warszawa 1995.

[5] H. W. Lenstra, „Divisors in residue classes”, Mathematics Of Computation, Vol. 41, 1984, str. 331-340.

[6] Arjen K. Lenstra, *Primality Testing* Proceedings of Symposia in Applied Mathematics Vol. 42 1990.

[7] János Pintz, William L. Steiger, Endre Szemerédi, „Infinite Sets of Primes with Fast Primality Tests and Quick Generation of Large Primes”, Mathematics Of Computation Vol. 53, Num. 187, Lipiec 1989, str. 399-406.

[8] Wacław Sierpiński, *Wstęp do teorii liczb*, WSiP, Warszawa 1987.

[9] S. Konyagin, C. Pomerance, „On primes recognizable in deterministic polynomial time”, *The Mathematics of Paul Erdős*, R. L. Graham , J. Nesetril, 1994.

[10] J. Hartmanis, H. Shank, „On the recognition of primes by automata”, J. Assoc. Comput. Mach. 15 (1968), str. 382-389.