O rozpoznawaniu liczb pierwszych

Marek ZAKRZEWSKI, Wrocław

Czym różni się liczba pierwsza od złożonej wie każdy, kto wie co to jest liczba pierwsza. Jednak znana ze szkoły metoda w przypadku dużych liczb nie daje się w praktyce zastosować. Komputer wykonujący miliard operacji na sekundę (założenie bardzo optymistyczne!) dla wyjaśnienia charakteru liczby 50-cyfrowej musiałby pracować około 5 miliardów lat. Zasadniczy postęp uzyskano dopiero w połowie lat siedemdziesiątych. Dobry superkomputer potrafi dziś rozwiązać zadanie tego rodzaju w ciągu kilkunastu minut nawet dla liczby kilkusetcyfrowej. Ciekawe, że te świetne wyniki uzyskuje się przy pomocy technik *losowych*. Oznacza to, że w trakcie obliczeń odwołujemy się do procedur odpowiadających, z grubsza biorąc, rzucaniu monetą.

W artykule mowa jest wyłącznie o algorytmach uniwersalnych tzn. takich, które stosują się do dowolnej liczby naturalnej. Od dawna znane są szybkie algorytmy stosujące się do liczb szczególnego kształtu, jak np. liczb postaci $2^p - 1$ (liczby pierwsze tej postaci zwane są liczbami Mersenne'a). Stąd odnotowywane w Księdze Guinessa największe znane liczby pierwsze mają postać zawsze bardzo charakterystyczną: przez wiele lat były to na ogół liczby Mersenne'a; obecnie rekordzistą jest chyba wciąż liczba $391582 \cdot 2^{216193} - 1$ mająca 65087 cyfr.

1.

Algorytm, o którym opowiem, jest połączeniem wnikliwej analizy Małego Twierdzenia Fermata z szybkim potęgowaniem.

Jak obliczyć x^n przy użyciu niewielkiej liczby mnożeń? Dla ustalenia uwagi, niech $n = 91 = (1011011)_2$. Przepisujemy postać binarną, zastępując każde 0 symbolem K, każdą zaś 1 symbolem KX, po czym pierwszy symbol K zamieniamy na 1.

1	0	1	1	0	1	1
KX	K	KX	KX	K	KX	KX
1X	K	KX	KX	K	KX	KX

Zapis w ostatnim wierszu koduje ciąg instrukcji: zaczynając od 1 mnóż przez x (instrukcja X) lub podnoś do kwadratu (instrukcja K). W naszym przykładzie otrzymujemy zatem:

$$1\ x\ x^2\ x^4\ x^5\ x^{10} x^{11}\ x^{22}\ x^{44}\ x^{45}\ x^{90}\ x^{91}.$$

Mamy tu więc tylko 11 mnożeń zamiast 90; w ogólnym przypadku obliczenie x^n może wymagać $2 \cdot \log_2 n$ mnożeń.

Dalej okaże się dla nas istotną umiejętność szybkiego obliczania wartości wyrażeń typu $c^{p-1} \bmod p$. W trakcie obliczeń można, oczywiście, kolejne potęgi redukować do przedziału $[0, p-1]$, zatem wyniki pośrednie będą miały co najwyżej $\log_2 p$ bitów. Mnożenie dwu takich liczb sprowadza się do *około* $(\log_2 p)^2$ operacji na bitach. Tak więc wartość wyrażenia $c^{p-1} \bmod p$ potrafimy znaleźć przy użyciu około $2 \cdot (\log_2 p)^3$ operacji.

Imponującą oszczędność czasu, jaką daje ta technika, ilustruje następujący przykład. Przypuśćmy, że mamy obliczyć $3^{p-1} \bmod p$ dla $p = 2^{127} - 1$. Wykonując potęgowanie krok po kroku musielibyśmy operację mnożenia powtórzyć prawie $2^{127} > 10^{39}$ razy! Ponieważ

$$(2^{127} - 1) - 1 = \underbrace{1111\ldots110}_{126\text{ razy}},$$

więc szybkie potęgowanie redukuje problem do 128 mnożeń. Każde z tych mnożeń wymaga około 16000 operacji na bitach – łącznie około 2 milionów operacji.

2.

Przypomnijmy Małe Twierdzenie Fermata:

Jeżeli p jest liczbą pierwszą nie będącą dzielnikiem liczby całkowitej c, to $c^{p-1} \equiv 1 \bmod p$.

Gdyby kongruencja $c^{p-1} \equiv 1 \bmod p$ zachodziła wyłącznie dla liczb pierwszych, mielibyśmy doskonałe kryterium rozpoznawania liczb pierwszych. Niestety, tak nie jest, co pokazuje poniższy przykład:

$2^{10} \equiv 1 \bmod 341$, więc $2^{340} \equiv 1 \bmod 341$, chociaż $341 = 11 \cdot 31$.

Co więcej, istnieją liczby złożone, takie jak $561 = 3 \cdot 11 \cdot 17$, dla których kongruencja ta zachodzi dla wszystkich c względnie pierwszych z daną liczbą. Wydaje się zatem, że Małe Twierdzenie Fermata nie może pełnić funkcji takiego kryterium. Głębsza analiza pozwala jednak odnieść częściowy sukces.

Powróćmy do przykładu liczby 341. Pokażemy, jak stwierdzić, że 341 jest liczbą złożoną, ograniczając się wyłącznie do dokładnej analizy kongruencji $2^{340} \equiv 1 \bmod 341$. Kongruencja ta oznacza, że 341 dzieli $2^{340} - 1$. Korzystając dwukrotnie ze wzoru $a^2 - 1 = (a+1)(a-1)$ otrzymujemy:

$$2^{340} - 1 = (2^{170} + 1)(2^{170} - 1) = (2^{170} + 1)(2^{85} + 1)(2^{85} - 1).$$

Gdyby 341 była liczbą pierwszą, to z faktu, że dzieli ona iloczyn, wynikałoby, że dzieli ona jeden z czynników. Mielibyśmy zatem:

$$2^{85} \equiv 1 \bmod 341, \text{ lub } 2^{85} \equiv -1 \bmod 341, \text{ lub } 2^{170} \equiv -1 \bmod 341.$$

Okazuje się, że żaden z tych warunków nie zachodzi.

Uogólniając przeprowadzoną wyżej analizę możemy wprowadzić pojęcie c-próby. Niech n będzie liczbą nieparzystą, c zaś liczbą całkowitą z przedziału $[2, n-1]$. Liczbę $n-1$ można przedstawić w postaci $n - 1 = 2^k \cdot m$, gdzie m nieparzysta (w przykładzie $341 - 1 = 2^2 \cdot 85$). Powiemy, że n wytrzymuje próbę c, jeżeli zachodzi jedna z kongruencji:

$$2^m \equiv 1 \bmod n \quad \text{lub} \quad 2^{m \cdot 2} \equiv -1 \bmod n, \text{ dla pewnego } m < n.$$

Test taki polega zatem na sprawdzeniu $k \leq \log_2 n$ kongruencji. Łatwo zauważyć, że wymaga to nie więcej niż $2 \cdot (\log_2 n)^4$ operacji. Chwila zastanowienia pokazuje, że wystarczy nawet $2 \cdot (\log_2 n)^3$ operacji.

Liczba, która nie wytrzyma jakiejkolwiek c-próby jest *na pewno* liczbą złożoną. Pomyślne przejście przez próbę pozwala *przypuszczać*, że badana liczba jest pierwsza. Co możemy zrobić, aby przypuszczenie takie uczynić wiarygodnym?

3.

W roku 1975 Michael Rabin wykazał, że liczba złożona może wyjść zwycięsko co najwyżej z 25% prób przeprowadzanych dla $c \in [2, n-1]$. Oznacza to, że jeśli liczbę n poddamy próbie generowanej losowo ze wskazanego przedziału, to pomyślne przejście testu oznacza, że n jest liczbą pierwszą z prawdopodobieństwem 75%. Zakończone sukcesem k-krotne powtórzenie testu zwiększa to prawdopodobieństwo do $1 - (1/4)^k$.

M. Rabin i V. Pratt sprawdzali eksperymentalnie wartość tej metody dla liczb stukilkudziesięciocyfrowych. Każdą liczbę poddawali serii 30 prób, uzyskując prawdopodobieństwo błędu rzędu 10^{-18}. W szczególności, na komputerze średniej klasy (w rozumieniu matematyka pracującego w Stanach Zjednoczonych w roku 1975!) uzyskali w ciągu 10 minut bezbłędną listę wszystkich liczb pierwszych postaci $2^p - 1$ dla $p < 500$. Znaleźli także największe liczby pierwsze poniżej 2^{300} i poniżej 2^{400}, a także największą znaną parę liczb pierwszych bliźniaczych (tzn. liczb pierwszych różniących się o 2).

Prawdopodobieństwo błędu rzędu 10^{-18} daje stopień pewności w pełni zaspokajający aspiracje kryptografów zainteresowanych generowaniem dużych tzn. kilkusetcyfrowych liczb pierwszych. Ten stopień pewności nie jest chyba nigdy osiągany w przypadku zaawansowanych dowodów współczesnej matematyki. Różnica pojęciowa jest jednak zasadnicza: algorytm Rabina

dostarcza potężnego *argumentu*, że dana liczba jest pierwsza, podczas gdy tradycyjny matematyk oczekuje *dowodu*. Tę teoretyczną ułomność usunęli w roku 1986 L.M. Adleman i M.D.A. Huang.

Przypomnijmy, że ilekroć zastosowanie algorytmu Rabina wykazuje, że liczba jest złożona – odpowiedź jest na pewno poprawna; tylko wówczas, gdy dowiadujemy się, że badana liczba jest pierwsza odpowiedź może być błędna. Algorytm Adlemana–Huanga działa odwrotnie: ilekroć stwierdza, że n jest liczbą pierwszą – działa bezbłędnie, mylić się może natomiast deklarując złożoność n. Algorytm ten oparty jest na zaawansowanych technikach geometrii algebraicznej.

Aby określić charakter liczby n, możemy poddawać ją jednocześnie obu testom, przy czym funkcję obu testów ograniczyć tak, by działały zawsze bezbłędnie. Tzn. algorytm Rabina będzie udzielać wyłącznie odpowiedzi „n jest liczbą złożoną" lub „nie wiem", natomiast algorytm Adlemana–Huanga odpowiedzi „n jest liczbą pierwszą" lub „nie wiem". Powtarzając to postępowanie otrzymamy ostatecznie konkretną, zawsze poprawną odpowiedź. Niekiedy może to być bardzo czasochłonne, ale średni czas realizacji algorytmu jest, przynajmniej z teoretycznego punktu widzenia, zadowalający (co to znaczy – wyjaśnimy niżej).

4.

Pojęcie „algorytmu szybkiego" jest oczywiście bardzo nieprecyzyjne. W teorii złożoności obliczeniowej dopracowano się jednak pewnej klasyfikacji, która pozwala dokonać podziału na algorytmy szybkie i wolne. Podział ten jest w miarę zgodny z intuicją i zarazem całkowicie precyzyjny. W rozumieniu tej klasyfikacji algorytm rozpoznawania liczb pierwszych uznajemy za szybki, o ile istnieją takie stałe A oraz m, że zachodzi nierówność

$$OP(n) \leq A \cdot (\log_2 n)^m ,$$

gdzie $OP(n)$ oznacza liczbę operacji na bitach wykonywanych przez komputer w trakcie rozpoznawania charakteru liczby n.

Pytanie „Czy istnieje szybki algorytm rozpoznawania liczb pierwszych?" jest dziś zapewne najważniejszym otwartym problemem pogranicza teorii liczb z teorią złożoności obliczeniowej. Przy ustalonej liczbie k powtórzeń testu, algorytm Rabina spełnia nierówność $OP(n) \leq 2k \cdot (\log_2 n)^3$, ale jak pamiętamy, może czasem dawać błędną odpowiedź. W algorytmie Rabina–Adlemana–Huanga liczba operacji zależy od tego, jakie konkretnie liczby posłużą do testowania; średnia liczba operacji spełnia żądaną nierówność, przy odpowiednich A oraz m, ale pechowe realizacje algorytmu mogą okazać się znacznie dłuższe.

Częściową odpowiedź na to zasadnicze pytanie dał w roku 1976 G. L. Miller. Zaproponował, by dla określenia charakteru liczby n wykonywać wszystkie c-próby dla $c \leq 70 \cdot (\log_2 n)^2$. Miller udowodnił, że jeżeli n jest liczbą złożoną, to przynajmniej jedna z tych prób skończy się porażką. Ponieważ pojedynczy test składa się co najwyżej z $2 \cdot (\log_2 n)^3$ operacji, więc dla algorytmu Millera zachodzi nierówność

$$OP(n) \leq 140 \cdot (\log_2 n)^5 .$$

Niestety, w dowodzie swego twierdzenia Miller korzystał z tzw. Rozszerzonej Hipotezy Riemanna. Panuje, co prawda, głębokie przeświadczenie o jej prawdziwości, ale na razie nie udało się udowodnić nawet jej bardzo szczególnych przypadków.

Można mieć nadzieję, że w końcu znajdziemy szybki algorytm rozpoznawania liczb pierwszych. W naturalny sposób pojawi się wówczas pytanie, na ile można go poprawić. Oczywiście, poniżej pewnej liczby operacji nigdy nie zejdziemy. W szczególności, można przypuszczać, że nie ma algorytmu rozpoznawania liczb pierwszych, dla którego przy pewnej stałej A spełniona byłaby nierówność

$$OP(n) \leq A \cdot log_2 n.$$

Chociaż przypuszczenie to jest niemal na pewno prawdziwe, jego dowód leży prawdopodobnie poza zasięgiem znanych obecnie metod.

5.

Opisane metody są niekonstruktywne w następującym sensie: wykrywają złożoność liczby, nie wskazując żadnego jej dzielnika. Możemy zatem wiedzieć, że liczba jest złożona, nie wiedząc nic o jej rozkładzie na czynniki pierwsze. Okazuje się, że problem rozkładu na czynniki pierwsze jest wyraźnie trudniejszy, niż problem rozpoznawania samego charakteru liczby (chociaż na gruncie matematyki szkolnej problemy te są w zasadzie równoważne). Dla przykładu, w roku 1984 czas potrzebny dla odróżnienia liczby pierwszej od złożonej szacowano w przypadku liczb 200-cyfrowych na 8 minut; czas rozkładu na czynniki pierwsze przy użyciu najszybszych znanych wówczas algorytmów szacowano na około 4 miliardy lat. Mimo tak drastycznej różnicy pomiędzy problemem rozpoznawania a problemem rozkładu istnieją pewne powody, by przypuszczać, że istnieją szybkie algorytmy również dla problemu rozkładu.

Pisząc ten artykuł w istotny sposób korzystałem ze świetnie napisanych artykułów Pomerance'a oraz Williamsa ([1], [3]). Williams jest też autorem krótkiego przeglądu ([4]) algorytmów rozkładu na czynniki, przy czym niektóre spośród tych algorytmów dają się realizować nawet na małych mikrokomputerach. Riesel ([2]) daje obszerne omówienie problemów rozpoznawania liczb pierwszych, jak i problemów rozkładu. Książka Riesela jest niemal w całości elementarna, a stosowane tam bardziej zaawansowane środki algebry czy teorii liczb wyłożone w serii aneksów. Całkowicie elementarny jest również mój wykład ([5]), gdzie piszę nieco więcej o problemie rozpoznawania liczb pierwszych w kontekście teorii złożoności.

Literatura:

[1] Pomerance, C., *Recent developments in primality testing*, The Mathematical Intelligencer, 3, 1981, No. 3, s. 97–105.

[2] Riesel, H., *Prime Numbers and Computer Methods for Factorization*, Stuttgart, Birkhauser, 1985.

[3] Williams, H., *Primality testing on a computer*, Ars Combinatoria, 5, 1978, s. 127–85 (przekład rosyjski: Kiberneticzeskij Sbornik 23, 1986, s. 51–99).

[4] Williams, H., *Factoring on a Computer*, The Mathematical Intelligencer, 6, 1984, No. 3, s. 29–36.

[5] Zakrzewski, M., *Wprowadzenie w teorię złożoności obliczeniowej*, Wyd. Politechniki Wrocławskiej, Wrocław 1990.