Algorytmy Algebry i Teorii Liczb

Notatki

„Jeśli ktoś mi napisze, że tu można rozpatrzyć 64 przypadki, to nie uznam takiej odpowiedzi. Bo to zakłada, że ja mam je rozpatrzyć. A ja nie mam ochoty."

Maciej Mikołajczak

Kraków, 2026

Spis treści

1. Arytmetyka na liczbach całkowitych

2026-03-03

Pomysł. Na tym kursie będziemy typowo rozważać liczby na tyle duże, że operacje na nich nie są jednostkowe. Będziemy uwzględniać wielkość liczb w złożoności algorytmów. Konkretniej, będziemy uwzględniać długość zapisu liczby w złożoności. Zatem na przykład znajdowanie dzielnika liczby N w czasie $O\left(\sqrt{N}\right)$ jest algorytmem wykładniczym. Zwykle przez N oznaczamy liczbę, a $n = \log N$ jej długość.

Algorytm 1 (RSA). Niech p, q będą pierwsze, $N = pq$. Mamy $\varphi(N) = (p-1)(q-1)$. Dla $e, d \in \mathbb{Z}$ takich, że $ed \equiv 1 \pmod{\varphi(N)}$ możemy szyfrować $x \to x^e \pmod N$ i deszyfrować $y \to y^d \pmod N$. Liczba e jest kluczem publicznym, a d kluczem prywatnym.

Uwaga. Dodawać i odejmować liczby umiemy w czasie $O(n)$. Nie da się lepiej. Mnożenie pisemne daje nam algorytm $O\left(n^2\right)$. Mamy lepsze algorytmy: Karatsuba $O\left(n^{1.59}\right)$, Toom-Cook $O\left(n^{1.46}\right)$, Schönhage-Strassen $O(n \log n \log\log n)$, Fürer $O\left(n \log n 2^{O(\log^* n)}\right)$ z 2007 i Harver, van der Hoeven $O(n \log n)$ z 2019.

Algorytm 2 (Karatsuba). Mając do pomnożenia liczby A, B długości n (zakładamy $n = 2^k$ dopełniając zerami) możemy podzielić je na pół $A = A_1 \cdot K + A_0$, $B = B_1 \cdot K + B_0$, gdzie $K = 2^{\frac{n}{2}}$.

Zachodzi $AB = A_1B_1K^2 + (A_0B_1 + A_1B_0)K + A_0B_0$. Do tego możemy przekształcić $A_0B_1 + A_1B_0 = (A_0 + A_1) \cdot (B_0 + B_1) - A_0B_0 - A_1B_1$. Zatem wystarczy wykonać 3 rekurencyjne mnożenia zamiast 4 (i kilka dodawań). Daje nam to rekursję $T(n) = 3T\left(\frac{n}{2}\right) + O(n)$, co daje $T(n) = O\left(n^{\log_2 3}\right) \approx O\left(n^{1.59}\right)$.

Algorytm 3 (Toom-Cook, Toom-3). Mając do pomnożenia liczby A, B długości n (zakładamy $n = 3^k$ dopełniając zerami) możemy podzielić je na trzy części $A = A_2 \cdot K^2 + A_1 \cdot K + A_0$, $B = B_2 \cdot K^2 + B_1 \cdot K + B_0$, gdzie $K = 2^{\frac{n}{3}}$.

Chcemy pomnożyć wielomiany $\mathcal{A}(X) = A_2X^2 + A_1X + A_0$, $\mathcal{B}(X) = B_2X^2 + B_1X + B_0$. Wielomian $\mathcal{AB}$ jest stopnia 4, ewaluujemy więc $\mathcal{A}$ i $\mathcal{B}$ w pięciu punktach (najlepiej $-2, -1, 0, 1, 2$, bo dla nich łatwo policzyć), mnożymy te wartości i interpolujemy $\mathcal{AB}$. Równanie $T(n) = 5T\left(\frac{n}{3}\right) + O(n)$ daje $T(n) = O\left(n^{\log_3 5}\right) \approx O\left(n^{1.46}\right)$.

Interpolację możemy przeprowadzić rozpisując układ równań liniowych, gdzie niewiadomymi są współczynniki wielomianu, a współczynniki to podstawiane wartości (które zawsze są takie same, więc można rozwiązać układ raz i podstawić w implementacji).

Algorytm 4 (Metoda Newtona-Raphsona). Obliczenia odwrotności liczby a możemy dokonać stosując wzór iteracyjny $x_{n+1} = x_n(2 - ax_n)$. Ten wzór ma zbieżność kwadratową, czyli liczba poprawnych cyfr podwaja się z każdą iteracją.

Jeśli chcemy obliczyć $\frac{b}{a}$, to zapisujemy $a = 2^k \cdot a'$, gdzie $a' \in [1, 2)$, odwracamy a' i mnożymy $b \cdot 2^{-k} \cdot \frac{1}{a'}$. Złożoność $\log n \cdot M(n)$, gdzie $M(n)$ to złożoność mnożenia.

Uwaga. Lepszą stałą niż metoda Newtona osiąga algorytm Burnikela-Zieglera, który działa na zasadzie dość skomplikowanej rekursji (coś jak dzielenie pisemne).

Algorytm 5 (Euklides). Dla danych $a, b > 0$ chcemy znaleźć $d = \gcd(a, b)$ oraz takie $s, t \in \mathbb{Z}$, że $d = sa + tb$. Robimy to tak:

1. Jeśli $a < b$, zamień a i b.
2. Jeśli $b = 0$, zwróć $d = a$ i parę $(1, 0)$.
3. Podziel z resztą $a = qb + r$.
4. Wywołaj $\gcd(b, r)$, dostajemy d i parę s', t'.
5. Zwróć d i parę $(t', s' - t' \cdot q)$.

Uwaga. Algorytm Euklidesa wykonuje co najwyżej $\log_\varphi a$ iteracji, gdzie $\varphi = \frac{1+\sqrt{5}}{2}$. Zatem złożoność to około $O\left(\log(a)^2\right)$. Do tego $|s| \leq b$ i $|t| \leq a$ (dowód indukcyjny).

Jeśli rozważane liczby są względnie pierwsze, to ten algorytm daje nam sposób wyznaczania odwrotności modularnej.

Algorytm 6 (Binarny Euklidesa). Dla danych $a, b > 0$ chcemy znaleźć $d = \gcd(a, b)$. Robimy to tak:

1. Zakładamy $a < b$.
2. Jeśli $2 \mid a$ i $2 \nmid b$, zwróć $\gcd\left(\frac{a}{2}, b\right)$.
3. Jeśli $2 \nmid a$ i $2 \mid b$, zwróć $\gcd\left(a, \frac{b}{2}\right)$.
4. Jeśli $2 \mid a$ i $2 \mid b$, zwróć $2\gcd\left(\frac{a}{2}, \frac{b}{2}\right)$.
5. Jeśli $2 \nmid a$ i $2 \nmid b$, zwróć $\gcd(b, a-b)$.

Złożoność jest logarytmiczna, bo każda operacja poza ostatnią zmiesza liczby dwa razy, a ostatnia nie może się wykonać dwa razy z rzędu.

Uwaga. Powyższy algorytm jest w praktyce szybszy od klasycznego algorytmu Euklidesa, bo nie korzysta z dzielenie i brania reszty. Można go zmodyfikować, by zwracał s, t takie, jak klasyczny algorytm.

Istnieje szybka wersja algorytmu Euklidesa działająca w czasie $O(M(n)\log n)$.

2. Liczby pierwsze

2026-03-17

Notacja. Przez $\mathbb{P}$ oznaczamy zbiór wszystkich liczb pierwszych.

Definicja 1. Przez $\pi(n)$ oznaczamy ilość liczb pierwszych mniejszych od n.

Twierdzenie 1 (O liczbach pierwszych; Hadamard, Vallée Poussin 1896).

$$\lim_{n\to\infty} \frac{\pi(n)\ln n}{n} = 1.$$

Uwaga. Istnieje nawet lepsze przybliżenie $\pi(x) \sim \mathrm{Li}(x)$, gdzie $\mathrm{Li}(x) = \int_2^x \frac{1}{\ln t}\,\mathrm{d}t$ nazywamy logarytmem całkowym. Z obu tych postaci wynika, że średnio potrzebujemy $\ln n$ losowań do wylosowania liczby pierwszej z przedziału $[1, n]$.

Twierdzenie 2 (Postulat Bertranda). Dla każdego k w przedziale $[k, 2k]$ znajduje się liczba pierwsza.

Twierdzenie 3 (Baker, Harman, Pintz 2000). Istnieje takie k_0, że dla każdego $k > k_0$ w przedziale $\left[k, k + k^{0.525}\right]$ jest co najmniej jedna liczba pierwsza.

Uwaga. Istnieje hipoteza Legendre'a mówiąca, że dla każdego $k \in \mathbb{N}$ między k^2 a $(k+1)^2$ istnieje jakaś liczba pierwsza. Mocniejsza hipoteza głosi, że maksymalna różnica między kolejnymi liczbami pierwszymi mniejszymi od n jest rzędu $O\left(\log^2 n\right)$.

Definicja 2. Dla $s > 1$ definiujemy funkcję ζ Riemanna wzorem

$$\zeta(s) = \sum_{n=1}^{\infty} \frac{1}{n^s}.$$

Uwaga. Wartości tej funkcji dla parzystych s są znane. Dla nieparzystych s nie znamy żadnej z wartości. Zachodzi równość

$$\zeta(s) = \prod_{p \in \mathbb{P}} \frac{1}{1 - p^{-s}}.$$

Funkcję ζ można rozszerzyć na liczby zespolone i badać jej własności analityczne, co pozwala nam czasem na powiedzenie czegoś o liczbach pierwszych. Wiadomo, że jeśli $\zeta(z) = 0$, to albo $z = -2k$ dla $k = 1, 2, \ldots$ (trywialne zera), albo $0 < \operatorname{Re}(z) < 1$ (nietrywialne zera). Hipoteza Riemanna mówi, że dla każdego nietrywialnego zera z zachodzi $\operatorname{Re}(z) = \frac{1}{2}$.

Uwaga. Dla dowolnej funkcji $\chi : \mathbb{Z} \to \mathbb{C}$, która jest multiplikatywna i okresowa możemy zdefiniować szereg $L_\chi(s) = \sum_{n=1}^{\infty} \frac{\chi(n)}{n^s}$ i rozszerzyć go na płaszczyznę zespoloną. Uogólniona hipoteza Riemanna mówi, że jeśli z jest nietrywialnym zerem funkcji L_χ, to $\operatorname{Re}(z) = \frac{1}{2}$. Prawdziwość tej hipotezy dałaby nam wniosek, że każda podgrupa $\mathbb{Z}_n^*$ ma zbiór generatorów złożony z liczb mniejszych niż $3(\ln n)^2$.

Propozycja 1.

$$\sum_{p \in \mathbb{P}, p < N} \frac{1}{p} = O(\ln \ln N).$$

Dowód (Szkic). Oznaczmy $f(N) = \sum_{p \in \mathbb{P}, p < N} \frac{1}{p}$. Mamy $e^{f(N)} = e^{\frac{1}{2}} e^{\frac{1}{3}} \ldots$ oraz $e^x \sim 1 + x$ dla małych x. Zatem

$$e^{f(N)} \sim \left(1 + \frac{1}{2}\right)\left(1 + \frac{1}{3}\right) \ldots \sim \sum_{k \text{ bezkwadratowe}} \frac{1}{k} \sim \sum_{k < N} \frac{1}{k} \cdot \text{const} \sim \ln N.$$

Przedostatnie przejście wynika z tego, że możemy przemnożyć przez $\sum_{n=1}^{\infty} \frac{1}{n^2}$ i dostać mniej więcej wszystkie liczby (a nie tylko bezkwadratowe). □

Algorytm 7 (Eratostenes). Chcemy o każdej liczbie z przedziału $[2, N]$ stwierdzić, czy jest pierwsza. Robimy pętlą

```
A[2..N] ← true
for i = 2, ..., N do
    if A[i] then
        for j = 2i, 3i, ... do
            A[j] ← false
        end for
    end if
end for
```

Ten algorytm działa w złożoności $O(N \ln \ln N)$.

Dowód. Dla każdej znalezionej liczby pierwszej p wykonuje się $\frac{N}{p}$ wykreśleń wielokrotności p. Liczba wykonanych operacji to

$$N \cdot \left(\frac{1}{2} + \frac{1}{3} + \frac{1}{5} + \frac{1}{7} + \ldots\right) = N \sum_{p \in \mathbb{P}, p < N} \frac{1}{p} = O(N \ln \ln N).$$

□

Uwaga. Istnieją techniczne usprawnienia, które powodują, że ten algorytm działa w czasie $O(N)$.

Algorytm 8 (Test Fermata). Na wejściu dostajemy liczbę całkowitą n. Sprawdzamy jej pierwszość:

1. Losujemy $a \in [1, n-1]$.
2. Jeśli $\gcd(a, n) > 1$, to n jest złożona.
3. Jeśli $a^{n-1} \equiv 1 \pmod{n}$ stwierdzamy, że n jest pierwsza, w przeciwnym razie jest złożona.

Uwaga. Jeśli liczba n jest pierwsza, to algorytm da nam prawdziwą odpowiedź. Jeśli jest złożona, to może się zdarzyć, że $a^{n-1} \equiv 1 \pmod{n}$. Wtedy mówimy, że n jest pseudopierwsza przy podstawie a.

Może się zdarzyć, że złożone n jest pseudopierwsze przy każdej podstawie (względnie pierwszej z n). Na przykład dla $n = 561 = 3 \cdot 11 \cdot 17$ z $2, 10, 16 \mid 560$ wynika, że $a^{560} \equiv 1 \pmod{561}$ (małe twierdzenie Fermata na czynnikach pierwszych).

Złożone liczby, które są pseudopierwsze przy każdej względnie pierwszej z nimi podstawie nazywamy liczbami Carmichaela. Wiemy, że takich liczb jest nieskończenie wiele. Dla nich test Fermata nie działa.

Algorytm 9 (Test Rabina-Millera). Na wejściu dostajemy liczbę całkowitą n. Sprawdzamy jej pierwszość:

1. Najpierw sprawdzamy parzystość.
2. Przedstawiamy $n - 1 = k \cdot 2^s$, gdzie k jest nieparzyste.
3. Losujemy $a \in [1, \ldots, n-1]$.
4. Wyznaczamy $(r_1, \ldots, r_s) = \left(a^k, a^{2k}, \ldots, a^{2^{s-1}k}, a^{n-1}\right) \pmod{n}$.
5. Jeśli $r_s \neq 1$ lub $r_i \neq \pm 1$ i $r_{i+1} = 1$ dla pewnego $i < s$, to n jest złożona.
6. W przeciwnym wypadku stwierdzamy, że n jest pierwsza.

Ten test zawsze działa dla liczb pierwszych, a dla złożonych myli się z prawdopodobieństwem co najwyżej $\frac{1}{2}$.

Dowód. Poprawność algorytmu dla liczb pierwszych wynika z małego twierdzenia Fermata i tego, że $x^2 \equiv 1 \pmod{p}$ implikuje $x \equiv \pm 1 \pmod{p}$. Załóżmy teraz, że n jest złożone.

Załóżmy, że $n = p^\alpha$ jest potęgą liczby pierwszej. Załóżmy, że n spełnia test Fermata przy podstawie $p+1$ (oczywiście $p+1 \perp n$), to znaczy $(p+1)^{p^\alpha - 1} \equiv 1 \pmod{p^\alpha}$. Redukując modulo p^2 dostajemy

$$1 \equiv (p+1)^{p^\alpha - 1} = \sum_{i=0}^{p^\alpha - 1} \binom{p^\alpha - 1}{i} p^i \equiv \binom{p^\alpha - 1}{0} + \binom{p^\alpha - 1}{1} p = p^{\alpha+1} - p + 1 \equiv 1 - p \pmod{p^2},$$

ale $1 \not\equiv 1 - p \pmod{p^2}$ – sprzeczność. Zatem n nie spełnia testu Fermata (a więc również testu Rabina-Millera) dla podstawy $p+1$. Zauważmy, że $\left\{a \in \mathbb{Z}_n^* : a^{n-1} \equiv 1 \pmod{n}\right\}$ jest podgrupą $\mathbb{Z}_n^*$. Wykazaliśmy, że nie jest ona całą grupą, a więc zawiera co najwyżej połowę elementów $\mathbb{Z}_n^*$. Zatem dla co najmniej połowy $a \in \mathbb{Z}_n^*$ już test Fermata wykaże złożoność.

Załóżmy teraz, że $n = uv$ dla $\gcd(u, v) = 1$. Dla s takiego, że $\frac{n-1}{2^s}$ jest nieparzyste mamy $(-1)^{\frac{n-1}{2^s}} = -1$. Zatem istnieje największe takie $m = \frac{n-1}{2^t}$, że dla pewnego $b \in \mathbb{Z}_n^*$ zachodzi $b^m \not\equiv 1 \pmod{n}$. Jeśli $m = n - 1$, to już test Fermata wykaże złożoność n. Możemy więc założyć, że $m < n - 1$ i $a^{2m} \equiv 1 \pmod{n}$ dla każdego $a \in \mathbb{Z}_n^*$.

Pokażemy, że istnieje b' takie, że $b'^m \not\equiv \pm 1 \pmod{n}$. Możemy założyć, że $b^m \equiv -1 \pmod{n}$. Z chińskiego twierdzenia o resztach istnieje takie x, że $x \equiv 1 \pmod{u}$ i $x \equiv b \pmod{v}$. Wtedy $x^m \equiv 1 \pmod{u}$ i $x^m \equiv -1 \pmod{v}$. Zatem x^m nie może przystawać do 1 ani -1 modulo n. Możemy więc przyjąć $b' = x$. Zbiór $\{a \in \mathbb{Z}_n^* : a^m \equiv \pm 1 \pmod{n}\}$ jest podgrupą $\mathbb{Z}_n^*$. Dowiedliśmy właśnie, że nie jest ona całą grupą, zatem podobnie jak w poprzednim przypadku dla co najmniej połowy $a \in \mathbb{Z}_n^*$ test Rabina-Millera wykaże złożoność. □

Uwaga. Można pokazać, że test Rabina-Millera myli się z prawdopodobieństwem nie większym niż $\frac{1}{4}$, a w praktyce jeszcze mniejszym. Jego wielokrotne powtórzenie daje nam algorytm mylący się z bardzo małym prawdopodobieństwem.

W jednej iteracji algorytmu wykonujemy $O(\log n)$ mnożeń liczb długości $\log n$. Zatem całkowita złożoność to $O\left(\log^3 n\right)$ (lub lepsza przy lepszym mnożeniu).

Uwaga. Jeśli $n < 2^{64}$, to wystarczy przyjąć za a wszystkie liczby pierwsze do 37. Jeśli uogólniona hipoteza Riemanna jest prawdziwa, to wystarczy sprawdzić wszystkie $a \leq 3\log^2 n$ – wynika to z faktu, że te liczby generują $\mathbb{Z}_n^*$. Wtedy algorytm Rabina-Millera działa deterministycznie w czasie wielomianowym.

Uwaga. Wiemy, że problem PRIMES (sprawdzenie, czy liczba jest pierwsza) jest w BPP (istnieje randomizowany, wielomianowy algorytm, który myli się z prawdopodobieństwem co najwyżej $\frac{1}{2}$). Jest też w NP – możemy zgadnąć generator g grupy $\mathbb{Z}_n^*$ i rozkład na czynniki pierwsze $n-1 = p_1^{\alpha_1} \dots p_s^{\alpha_s}$. Rekurencyjnie sprawdzamy, czy każde z $p_1, \dots, p_s$ jest pierwsze, a następnie, czy $g^{n-1} \equiv 1 \pmod{n}$ i $g^{\frac{n-1}{p_i}} \not\equiv 1 \pmod{n}$ dla każdego i. Te dwa warunki wymuszają, że g jest generatorem $\mathbb{Z}_n^*$, a więc n musi być pierwsze. Tak naprawdę PRIMES jest w P – istnieje algorytm AKS, który jest deterministyczny i wielomianowy.