Algorytmy Algebry i Teorii Liczb

Zagadnienia na Egzamin

„Zrobią sobie Państwo dużo wrogów, ale i tak zachęcam”

Popełnione przez

Chrząszczowy Pomp

Animowany Ponton

Pompona Vanilla

Kraków

Anno Domini 2025

Spis treści

A Grupa A **4**

A.1 Opisać rozszerzony algorytm Euklidesa znajdowania NWD. 4

A.2 Opisać binarny algorytm Euklidesa znajdowania NWD. 5

A.3 Opisać algorytmy szyfrowania RSA i Elgamal oraz na czym polega trudność ich łamania. 6

A.3.1 Algorytm RSA 6

A.3.2 Algorytm Elgamal 7

A.4 Opisać metodę klucza jednorazowego Vernama, trudność jej łamania i praktyczne zastosowanie. 7

A.5 Zdefiniować problemy Primes oraz Factoring i podać ich umiejscowienie w klasach złożoności. 8

A.5.1 Primes 8

A.5.2 Factoring 9

A.6 Podać efektywną metodę znalezienia liczby pierwszej o zadanej liczbie bitów. . . 9

A.7 Opisać efektywną implementację działań arytmetycznych w ciele $\mathbb{Z}_p/(w)$. . . . 10

A.8 Opisać ideę algorytmu AKS (bez dowodu). 10

A.9 Pokazać, że wielomianowy algorytm na problem pierwiastka dyskretnego da się zamienić na wielomianowy algorytm na faktoryzację. 11

A.10 Zdefiniować problemy Discrete-Log i Diffie-Hellman, ich miejsce w klasach złożoności, opisać protokół Diffiego-Hellmana. 12

A.10.1 Discrete-Log 12

A.10.2 Diffie-Helman 13

A.11 Podać definicję krzywej eliptycznej i grupy z nią związanej. 13

A.11.1 Działania na krzywej eliptycznej 13

A.11.2 Grupa krzywej eliptycznej 14

B Grupa B **15**

B.1 Opisać algorytm Karatsuby mnożenia dużych liczb binarnych. 15

B.2 Opisać algorytm Tooma-Cooka mnożenia dużych liczb binarnych. 15

B.3 Zapisać i udowodnić Chińskie Twierdzenie o Resztach. 16

B.4 Opisać algorytm Millera-Rabina. 17

B.5 Zdefiniować pojęcie ideału, pierścienia ilorazowego oraz pokazać, że $\mathbb{Z}_p[X]/(W)$ jest ciałem wtw gdy W jest nierozkładalny. 19
B.6 Pokazać, że każde ciało skończone ma p^k elementów. 20
B.7 Pokazać, że pierwiastki wielomianu $X^{p^k} - X$ stanowią ciało, podać praktyczną metodę generowania ciała skończonego. 20
B.7.1 Generowanie ciała skończonego 21
B.8 Opisać algorytm Tonellego-Shanksa. 21
B.9 Opisać algorytm faktoryzacji Fermata. 23
B.10 Opisać metodę Baby-Step-Giant-Step. 24
B.11 Opisać kryptosystem plecakowy, dlaczego nie jest stosowany w praktyce. 25

C Grupa C **26**
C.1 Pokazać, że grupa multiplikatywna ciała skończonego jest grupą cykliczną. . . . 26
C.2 Opisać algorytm „ro" Pollarda na faktoryzację. 27
C.2.1 Metoda Floyda 27
C.2.2 Metoda Brenta 28
C.3 Opisać algorytm sita kwadratowego. 28
C.3.1 Faktoryzacja przez liczby B-gładkie 28
C.3.2 Sito kwadratowe 29
C.3.3 Metoda Multi Polynomial Quadratic Sieve (MPQS) 29
C.4 Opisać algorytm „ro" Pollarda na logarytm dyskretny. 30
C.5 Opisać algorytm Pohliga-Hellmana. 30
C.6 Opisać algorytm rachunku indeksów. 32
C.6.1 Atak Logjam 33
C.7 Opisać ideę algorytmu Schönhage-Strassena. 33
C.7.1 Teoretyczna Transformata Numeryczna (NTT) 33
C.7.2 Algorytm Schönhage-Strassena 33
C.8 Opisać algorytm Schreiera-Simsa. 35
C.8.1 Filtr Simsa 37
C.9 Opisać algorytm Grovera. 38
C.10 Opisać algorytm faktoryzacji liczby N za pomocą obliczania rzędu elementu modulo N (klasyczna część algorytmu Shora). 40

Licencja

titlesec chngcntr

Grupa A

A.1 Opisać rozszerzony algorytm Euklidesa znajdowania NWD.

Dla danych $a, b > 0$ algorytm zwraca $d = \gcd(a, b)$ oraz liczby całkowite s i t, które spełniają $d = s \cdot a + t \cdot b$.

Algorytm:

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

Algorytm zwraca poprawne $d = \gcd(a, b)$.

Dowód. Wystarczy pokazać, że $\gcd(a, b) = \gcd(b, r)$, jeśli $a = q \cdot b + r$. Niech $d = \gcd(a, b)$.

$$a = 0 \pmod{d}$$

$$b = 0 \pmod{d}$$

$$r = a - q \cdot b = 0 \pmod{d}$$

□

Algorytm zwraca poprawne (s, t).

Dowód. Z tożsamości Bézout wiemy, że

$$\forall_{a,b \neq 0}\ \exists_{s,t}\ a \cdot s + b \cdot t = \gcd(a, b)$$

Jako krok bazowy indukcji po a, b sprawdzamy przypadek $b = 0$, rzeczywiście $a \cdot 1 + 0 = \gcd(a, 0)$.

Niech $a = q \cdot b + r$. Korzystając z tego, że $\gcd(a, b) = \gcd(b, r) = d$, otrzymujemy:

$$b \cdot s' + r \cdot t' = d,$$

$$b \cdot s' + (a - q \cdot b) \cdot t' = d,$$

czyli $s = t',\ t = s' - q \cdot t'$, co było do pokazania. □

Własności algorytmu:

1. Liczba iteracji: co najwyżej $\log_\phi a$, gdzie $\phi = \frac{1+\sqrt{5}}{2}$
2. Złożoność: $\mathcal{O}(\log^2 a)$
3. $|s| \leq b$, $|t| \leq a$

Ad 1. Żeby obliczenie $\gcd(a, b)$ wymagało k kroków, potrzeba $a \geq F_{k+2}$, $b \geq F_{k+1}$ – dowód przez indukcję po k.
Ad 2. Przy każdym wywołaniu rekurencyjnym przynajmniej jeden z argumentów zmniejsza się o połowę:
Jeśli $b \leq \frac{a}{2}$, to $r < b < \frac{a}{2}$.
Jeśli $b > \frac{a}{2}$, to $r = a - b < \frac{a}{2}$.
Złożoność operacji w kroku rekursji to $\mathcal{O}(\log a)$, więc cały algorytm ma złożoność $\mathcal{O}(\log^2 a)$.

A.2 Opisać binarny algorytm Euklidesa znajdowania NWD.

Dla danych $a, b > 0$ algorytm zwraca $\gcd(a, b)$, wykonując tylko odejmowanie i operacje binarne.

Algorytm:

1. Jeśli $a < b$, zamień a i b.
2. Jeśli $b = 0$, zwróć $d = a$.
3. Jeśli $2 \mid a$ i $2 \nmid b$, zwróć $\gcd\left(\frac{a}{2}, b\right)$,
4. Jeśli $2 \nmid a$ i $2 \mid b$, zwróć $\gcd\left(a, \frac{b}{2}\right)$,
5. Jeśli $2 \mid a$ i $2 \mid b$, zwróć $2 \cdot \gcd\left(\frac{a}{2}, \frac{b}{2}\right)$,
6. Jeśli $2 \nmid a$ i $2 \nmid b$, zwróć $\gcd(b, a - b)$.

Algorytm zwraca poprawne $d = \gcd(a, b)$.

Dowód. Jeśli któryś z argumentów jest nieparzysty (przypadki 3 i 4), to $2 \nmid \gcd(a, b)$, dlatego możemy podzielić parzysty argument przez 2.
Jeśli oba argumenty są parzyste (przypadek 5), to $d = \gcd(a, b)$ też jest parzyste, ponieważ inaczej $2 \cdot d$ dzieliłoby obie liczby, co byłoby sprzeczne z tym, że d jest największym dzielnikiem.

Dlatego możemy wydzielić 2 i obliczyć $2 \cdot \gcd\left(\frac{a}{2}, \frac{b}{2}\right)$.
Jeśli obie liczby są nieparzyste, to obliczamy $d = \gcd(b, a-b) = \gcd(a, b)$, ponieważ:

$$a \equiv 0 \mod d$$

$$b \equiv 0 \mod d$$

$$a - b \equiv 0 \mod d$$

□

W każdym z przypadków od 3 do 5, co najmniej jeden z argumentów zmniejsza się o połowę. Takich wywołań jest więc co najwyżej logarytmicznie wiele. Pozostaje do przeanalizowania przypadek 6. Ponieważ a i b są nieparzyste, to $a - b$ jest parzyste, więc w kolejnym wywołaniu wykona się przypadek 3 lub 4. Czyli któryś z argumentów zmniejszy się o połowę w co najwyżej dwóch krokach. To dowodzi złożoności $\mathcal{O}(\log(a) \cdot M(a))$, gdzie $M(a)$ to koszt odejmowania i operacji bitowych, który jest liniowy od długości zapisu liczb.

A.3 Opisać algorytmy szyfrowania RSA i Elgamal oraz na czym polega trudność ich łamania.

Algorytmy RSA i Elgamal są asymetryczne, czyli wykorzystują jawną funkcję szyfrującą E (klucz publiczny) i tajną funkcję deszyfrującą D (klucz prywatny). Najważniejszym założeniem jest, że nie potrafimy efektywnie odtworzyć klucza prywatnego z publicznego. Dodatkowo zakładamy, że problem Factoring nie jest w klasie BPP.

A.3.1 Algorytm RSA

Algorytm RSA:

1. Oblicz $N = p \cdot q$ dla wybranych liczb pierwszych p, q.
   Wtedy $\varphi(N) = (p-1) \cdot (q-1)$.
2. Wybierz e względnie pierwsze z $\varphi(N)$. Oblicz d takie, że $e \cdot d = 1 \pmod{\varphi(N)}$.
3. Zdefiniuj $E(x) = x^e \pmod N$ oraz $D(x) = x^d \pmod N$. Kluczem publicznym jest (e, n).

Ad 1. Zamiast $\varphi(N)$ można użyć (rozwiązanie często stosowane) funkcji Carmichaela

$$\lambda(N) = \operatorname{lcm}(p-1, q-1)$$

Funkcja λ znajduje najmniejszą liczbę całkowitą x taką, że $x^{\lambda(N)} = 1 \pmod N$ dla każdego x względnie pierwszego z N. Wybór $\lambda(N)$ zamiast $\varphi(N)$ nie zmniejsza bezpieczeństwa, a daje

zazwyczaj mniejszy wykładnik.
Ad 2. Do znalezienia liczby d można użyć rozszerzonego algorytmu Euklidesa:

$$e \cdot s + \varphi(N) \cdot t = 1$$

$$s + \varphi(N) \cdot t \cdot e^{-1} = e^{-1} \pmod{\varphi(N)}$$

$$d = s = e^{-1} \pmod{\varphi(N)}$$

Dlaczego deszyfrowanie działa?
Rząd grupy multiplikatywnej modulo N jest równy $\varphi(N)$, więc dla każdego x względnie pierwszego z N zachodzi

$$(x^e)^d = x^{ed} = x,$$

ponieważ $e \cdot d = 1 \pmod{\varphi(N)}$.

Szyfrowanie RSA można złamać, jeśli wyznaczy się liczby p i q, czyli pozna się rozkład N na czynniki pierwsze. Można wtedy obliczyć $\varphi(N)$ i d algorytmem Euklidesa. Dlatego tak ważne jest założenie o dużej trudności problemu Factoring. W tym dniu (18.05.2025), nie ma ani dowodu na to, że algorytm rozkładu na czynniki pierwsze nie istnieje, ani dowodu NP-zupełności problemu Factoring.

A.3.2 Algorytm Elgamal

Algorytm Elgamal:

1. Wybierz jakąś grupę G (np. grupę multiplikatywną ciała $\mathbb{F}_q$) i element $g \in G$ (najlepiej generator).
2. Wylosuj liczbę $x \in \{1, \ldots, |G| - 1\}$.
3. Klucz publiczny to (g, g^x), a klucz prywatny to (g, x).
4. Szyfrowanie wiadomości M: wylosuj liczbę y, wyślij $(g^y, M \cdot g^{xy})$.

Otrzymawszy $(g^y,\ M \cdot g^{xy})$ i znając klucz prywatny x, można łatwo obliczyć g^{xy}, czyli też odtworzyć $M = (M \cdot g^{xy}) \cdot g^{-xy}$. Gdyby odszyfrowanie M było łatwe, to również problem Diffiego-Hellmana byłby rozwiązywalny, a nie jest.

A.4 Opisać metodę klucza jednorazowego Vernama, trudność jej łamania i praktyczne zastosowanie.

Osoby A i B mają bezpieczny kanał komunikacji do przesyłania kluczy i niechroniony kanał do zwykłej komunikacji. Osoba A chce przesłać osobie B ciąg bitów M. Szyfruje go kluczem k o takiej samej długości za pomocą bitowej operacji XOR: $E(M) = M \oplus k$. Osoba A przesyła

bezpiecznym kanałem wartość k i normalnym kanałem wartość $E(M)$. Żeby osoba B mogła odczytać M wystarczy, że obliczy $E(E(M))$, ponieważ XOR jest łączny:

$$(M \oplus k) \oplus k = M \oplus (k \oplus k) = M \oplus 0 = M$$

Przy „naprawdę losowym" k każdy wynik szyfrowania jest równie prawdopodobny, więc zaszyfrowana wiadomość ma takie same właściwości, co gdyby wylosować osobno każdy bit.

Pozostaje jedno uzasadnione zastrzeżenie – jeśli musimy mieć sposób na bezpieczne przesłanie klucza o długości $|M|$, równie dobrze można po prostu przesłać M. Jednak metoda może być przydatna, jeśli potrzebne jest awaryjne, jednorazowe przesłanie zaszyfrowanej wiadomości i korespondenci wcześniej mieli okazję wymienić się kluczem o długości z odpowienim zapasem.

A.5 Zdefiniować problemy Primes oraz Factoring i podać ich umiejscowienie w klasach złożoności.

A.5.1 Primes

Wejście: Liczba p
Problem: Czy p jest liczbą pierwszą?
Klasa złożoności: P

Primes $\in$ coNP:
Jeśli otrzymamy od wyroczni d potencjalny dzielnik p, możemy w wielomianowym czasie sprawdzić, czy d rzeczywiście jest dzielnikiem i odpowiedzieć NIE na pytanie o pierwszość.

Primes $\in$ NP:
Korzystamy z tego, że $\mathbb{Z}_p^*$ ma rząd równy $p-1$ wtedy i tylko wtedy, gdy p jest pierwsze.
Jeśli otrzymamy od wyroczni g, generator $\mathbb{Z}_p^*$ i rozkład na czynniki pierwsze $p-1 = p_1^{\alpha_1} \cdots p_s^{\alpha_s}$, możemy:

- rekurencyjnie sprawdzić pierwszość p_i i poprawność rozkładu,
- sprawdzić, czy $g^{p-1} = 1 \pmod{p}$,
- sprawdzić, czy $g^{\frac{p-1}{p_i}} \neq 1 \pmod{p}$.

Jeżeli wszystkie z powyższych warunków są spełnione, to odpowiedzią jest TAK.

Primes $\in$ BPP:
Dowodem jest algorytm Millera-Rabina, który myli się przy odpowiedzi TAK z prawdopodobieństwem nie większym niż $\frac{1}{2}$.

Primes $\in$ P:
Dowodem jest algorytm Agrawala-Kayala-Saxena (AKS).

A.5.2 Factoring

Wejście: Liczby n, k
Problem: Czy istnieje nietrywialny dzielnik n mniejszy lub równy k?
Klasa złożoności: NP $\cap$ coNP?

Factoring $\in$ NP:
Jeśli otrzymamy od wyroczni $2 \leq d \leq k$ potencjalny dzielnik n, możemy w wielomianowym czasie sprawdzić, czy d rzeczywiście jest dzielnikiem i odpowiedzieć TAK.

Factoring $\in$ coNP:
Otrzymujemy od wyroczni rozkład liczby n na czynniki pierwsze $n = p_1^{\alpha_1} \cdots p_s^{\alpha_s}$. Po sprawdzeniu poprawności rozkładu odpowiadamy $(p_1 \leq k)$? TAK : NIE, gdzie p_1 to najmniejszy z dzielników.

Na ten moment nie wiadomo nic więcej.

A.6 Podać efektywną metodę znalezienia liczby pierwszej o zadanej liczbie bitów.

Dla zadanej liczby k chcemy znaleźć liczbę pierwszą $p \in \left[2^k, 2^{k+1} - 1\right]$.

Liczby pierwsze są dość gęsto upakowane – w przedziale od 1 do n jest około $\frac{n}{\log n}$ liczb pierwszych. Zatem w przedziale $\left[2^k, 2^{k+1} - 1\right]$ jest ich około

$$\frac{2^{k+1} - 1}{k+1} - \frac{2^k - 1}{k} = \frac{2^k\left(1 - \frac{1}{k}\right) - 1}{k+1} = \Theta\left(\frac{2^k}{k}\right)$$

Możemy więc po prostu losować liczbę całkowitą p z przedziału $\left[2^k, 2^{k+1} - 1\right]$, dopóki nie trafi się liczba pierwsza. Skoro w przedziale jest $\Theta\left(2^k\right)$ liczb całkowitych, z których $\Theta\left(\frac{2^k}{k}\right)$ jest liczbami pierwszymi, to w oczekiwaniu po $\mathcal{O}(k)$ losowaniach trafimy na liczbę pierwszą. Sprawdzanie, czy liczba jest pierwsza, wykonuje się w czasie $\mathcal{O}(k^3)$, więc oczekiwana złożoność algorytmu losowania liczby pierwszej to $\mathcal{O}(k^4)$.

A.7 Opisać efektywną implementację działań arytmetycznych w ciele $\mathbb{Z}_p/(w)$.

Jeśli w jest wielomianem nierozkładalnym stopnia n, to elementy elementy $\mathbb{Z}_p/(w)$ to wielomiany stopnia co najwyżej n nad $\mathbb{Z}_p$.

- Dodawanie – $\mathcal{O}(n)$
  Dodajemy po współrzędnych modulo p.
- Mnożenie – $\mathcal{O}(n^2)$
  Mnożymy „w słupku" z wynikiem modulo w. Można szybciej algorytmem Karatsuby, Tooma-Cooka lub Schönhage-Strassena w $\mathcal{O}(n \log n)$.
- Dzielenie – $\mathcal{O}(n^2)$
  Mnożymy przez odwrotność obliczoną rozszerzonym algorytmem Euklidesa. Korzystamy z tego, że wszystkie wielomiany w $\mathbb{Z}_p/(w)$ są względnie pierwsze z w.

A.8 Opisać ideę algorytmu AKS (bez dowodu).

Algorytm opiera się na poniższym twierdzeniu:

Twierdzenie A.8.1. Niech $n, a \in \mathbb{Z}$ będą liczbami względnie pierwszymi. Równość

$$(X + a)^n = X^n + a \pmod{n}$$

zachodzi wtedy i tylko wtedy, gdy n jest liczbą pierwszą.

Szkic dowodu. Jeśli n jest liczbą pierwszą, to wszystkie współczynniki $\binom{n}{k}$ dla $0 < k < n$ są podzielne przez n, więc $(X + a)^n = X^n + a^n \pmod{n}$ oraz z Małego Twierdzenia Fermata wiemy, że $a^n = a \pmod{n}$.

Jeśli n nie jest liczbą pierwszą, to któryś współczynnik $\binom{n}{k}$ nie jest podzielny przez n, więc wielomian $(X + a)^n$ zawiera wyraz $\binom{n}{k} X^k a^{n-k}$, gdzie k jest najmniejszym dzielnikiem n. □

Ponieważ obliczenie $(X + a)^n$ jest zbyt czasochłonne, przenosimy się do pierścienia ilorazowego $\mathbb{Z}_n[X]/(X^r - 1)$ dla pewnego r takiego, że rząd n modulo r jest wysoki, co najmniej $\log^2 n$. Przeprowadzenie testu dla jednego a nie wystarczy, ale potrzeba pierwiastkowo-logarytmicznie mało powtórzeń.

Algorytm AKS:

1. Jeśli $n = m^k$ dla $k \geq 2$, zwróć ZŁOŻONA.
2. Znajdź najmniejsze r takie, że rząd n modulo r jest większy od $\log^2 n$.
3. Jeśli $1 < \gcd(a, n) < n$ dla jakiegoś $a \leq r$, zwróć ZŁOŻONA.
4. Jeśli $n \leq r$, zwróć PIERWSZA.
5. Dla $1 \leq a \leq \sqrt{r} \log n$ sprawdź równość:

$$(X + a)^n = X^n + a \mod (p, X^r - 1)$$

   Jeśli równość nie zachodzi, zwróć ZŁOŻONA.
6. Zwróć PIERWSZA.

(W kolorze wyróżniony jest właściwy algorytm. Pozostałe kroki obsługują przypadki brzegowe.)

Złożoność
Sprawdzamy $\mathcal{O}(\sqrt{r} \log n)$ równań na wielomianach stopnia r każde w $\widetilde{\mathcal{O}}(r \log^2 n)$ krokach, co daje złożoność $\widetilde{\mathcal{O}}(r^{3/2} \log^3 n) = \widetilde{\mathcal{O}}(\log^{10.5} n)$.

Panowie Lenstry i Pomerance przyspieszyli algorytm AKS do $\widetilde{\mathcal{O}}(\log^6 n)$.

A.9 Pokazać, że wielomianowy algorytm na problem pierwiastka dyskretnego da się zamienić na wielomianowy algorytm na faktoryzację.

Pomysł opiera się na tym, że możemy sprowadzić faktoryzację n do szukania nietrywialnych rozwiązań równania $x^2 = y^2 \mod n$.

Lemat A.9.1. Niech n będzie nieparzystą liczbą złożoną, nie potęgą liczby pierwszej. Dla dowolnego s równanie:

$$x^2 = s \pmod n,$$

jeśli ma jakiekolwiek rozwiązanie, to ma co najmniej 4 różne.

Dowód. Przedstawmy liczbę n jako $n = p \cdot q$, gdzie p, q są względnie pierwsze. Jeśli x jest rozwiazaniem równania postaci $x^2 = s \pmod p$, to $-x$ również. Zapiszmy kongruencje:

$$x^2 = s \pmod p$$

$$x^2 = s \pmod q$$

Z Chińskiego Twierdzenia o Resztach wynika, że rozwiązań oryginalnego równania jest co najmniej tyle, ile wynosi iloczyn liczby rozwiązań poszczególnych równań, czyli $2 \cdot 2 = 4$. □

To wystarcza, żeby skonstruować algorytm.

Algorytm na faktoryzację:

1. Wylosuj $x < n$ i oblicz $s = x^2 \pmod{n}$.
2. Oblicz $y = \text{sqrt}(s, n)$.
3. Jeśli $x \neq \pm y$, to $\gcd(x + y, n)$ lub $\gcd(x - y, n)$ jest nietrywialnym dzielnikiem n.
4. Jeśli $x = y$ lub $x = -y$, powtórz losowanie.

(Funkcja $\text{sqrt}(x, n)$ to czarna skrzynka, która oblicza lub zgaduje w czasie wielomianowym pierwiastek z liczby x modulo n.)

Algorytm działa poprawnie, bo skoro $x^2 = y^2 \pmod{n}$, to $n \mid (x+y)(x-y)$. W każdej iteracji szansa na znalezienie nietrywialnego dzielnika jest równa $\frac{1}{2}$, ponieważ tyle wynosi prawdopodobieństwo, że trafiliśmy na $x \neq \pm y$, przy losowym wyborze x, czyli każde z 4 rozwiązań równania $y^2 = s \pmod{n}$ jest równie prawdopodobne.

Gdyby algorytm Tonellego-Shanksa działał na grupie $\mathbb{Z}_n$ dla dowolnej liczby całkowitej n, to dałoby się w czasie wielomianowym rozłożyć n na czynniki pierwsze.

A.10 Zdefiniować problemy Discrete-Log i Diffie-Hellman, ich miejsce w klasach złożoności, opisać protokół Diffiego-Hellmana.

Niech $G = \langle g \rangle$ będzie pewną grupą cykliczną.

A.10.1 Discrete-Log

Wejście: $a \in G$
Wyjście: x takie, że $g^x = a$

Discrete-Log $\in$ NP:
Jeśli otrzymamy od wyroczni x, możemy sprawdzić, czy $g^x = a$.

W kryptografi zakładamy, że Discrete-Log $\notin$ P i Discrete-Log $\notin$ BPP, ale nie jest to udowodnione. Nie wiadomo też, czy ten problem jest NP-trudny.

A.10.2 Diffie-Helman

Wejście: $g^x,\ g^y$
Wyjście: g^{xy}

Diffie-Helman $\in$ NP:
Jeśli otrzymamy od wyroczni x, po sprawdzeniu z g^x, możemy obliczyć $(g^y)^x = g^{xy}$.

Tak jak w poprzednim przypadku, nie ma dowodu, że Diffie-Helman $\notin$ P. Za pomocą Discrete-Log możemy rozwiązać problem Diffiego-Helmana.

Protokół Diffiego-Helmana:
Korespondenci uzgadniają klucz symetryczny – jeden z nich ustala swój klucz prywatny x, drugi y, po czym publicznym kanałem razem z wiadomością przesyłają g^x i g^y. Kluczem deszyfrującym wiadomość jest g^{xy}.

A.11 Podać definicję krzywej eliptycznej i grupy z nią związanej.

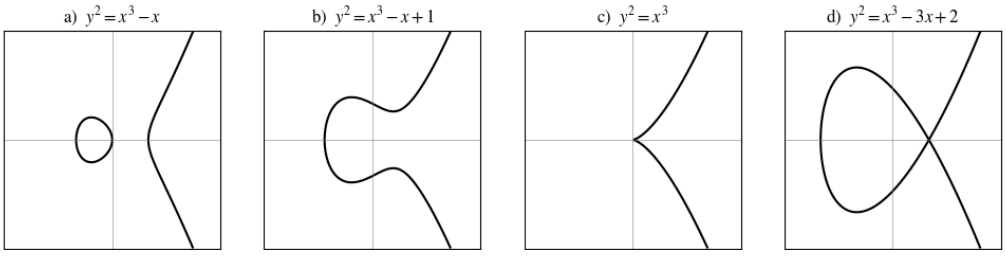
Definicja A.11.1. Krzywa eliptyczna to zbiór rozwiązań (x, y) w pewnym ciele $\mathbb{F}$ równania:

$$y^2 + a_1xy + a_2y = x^3 + b_1x^2 + b_2x + b_3,$$

które można sprowadzić do postaci Weierstrassa:

$$y^2 = x^3 + ax + b$$

Chcemy, żeby krzywa była „gładka" (nie jak Rys. A.1d) i nie miała „ostrza" (nie jak Rys. A.1c), czyli formalnie, żeby wyznacznik krzywej $\Delta = 4a^3 + 27b^2$ był różny od 0.

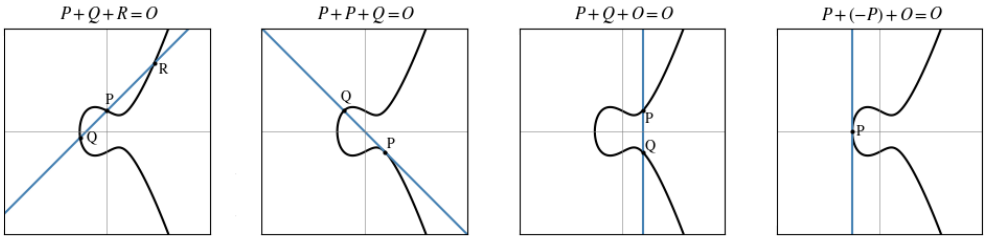


Rysunek A.1: Możliwe kształty krzywych eliptycznych w $\mathbb{R}^2$

A.11.1 Działania na krzywej eliptycznej

Jeśli punkt $P = (x, y)$ leży na krzywej eliptycznej, to $-P = (x, -y)$ również. Jeśli P i Q są punktami na krzywej eliptycznej, to prosta PQ przecina krzywą jeszcze w jednym punkcie R.

Ustalamy, że $P + Q + R = O$, gdzie O jest sztucznym punktem w nieskończoności.
(Niestety nie wiadomo, gdzie konkretnie jest O – szczęśliwy znalazca proszony o kontakt.)



Rysunek A.2: Możliwe przecięcia prostej z krzywą eliptyczną

Jeżeli $Q = P$, traktujemy punkt P trochę jak podwójny pierwiastek wielomianu, który odbija się od osi, nie przecinając jej. Za prostą PQ przyjmujemy wtedy styczną do krzywej w P (Rys. A.2b).

Jeżeli $Q = -P$, przyjmujemy, że $P + Q = P + (-P) = O$ oraz $P + O = P$ (Rys. A.2c).

Podsumowując:

Definicja A.11.2 (Dodawanie)**.** Jeśli P, Q są punktami na krzywej eliptycznej, to punkt R przecięcia prostej PQ z krzywą daje wynik dodawania $P + Q = -R$. Jeśli prosta PQ nie przecina krzywej w dodatkowym punkcie, to $P + Q = O$.

Ponieważ mnożenie wymaga stałej liczby operacji, to dla dowolnego punktu P oraz liczby k można obliczyć $k \cdot P$ w $\mathcal{O}(\log k)$ operacjach arytmetycznych w ciele, stosując coś w rodzaju szybkiego potęgowania.

A.11.2 Grupa krzywej eliptycznej

Dodawanie punktów na krzywej jest przemienne i (co nieoczywiste) łączne, więc otrzymujemy grupę przemienną.

Najczęściej używa się krzywych:

- nad $\mathbb{Z}_p$, gdzie p jest dużą liczbą pierwszą,
- nad ciałem $\mathrm{GF}(2^s)$.

Twierdzenie A.11.1. Niech $\mathbb{F}$ będzie ciałem, $|\mathbb{F}| = q$ oraz niech E będzie grupą dla pewnej krzywej eliptycznej nad $\mathbb{F}$. Grupa E jest albo grupą cykliczną, albo produktem dwóch grup cyklicznych.

Niech pozostanie bez dowodu.

Rozmiar grupy $|E|$ może być rozumiany jako liczba rozwiązań równania opisującego krzywą. Dla połowy $x \in \mathbb{F}$ wartość $x^3 + ax + b$ jest resztą kwadratową, a wtedy odpowiadają jej dwa możliwe y, więc punktów powinno być blisko q.

Grupa B

B.1 Opisać algorytm Karatsuby mnożenia dużych liczb binarnych.

Algorytm mnożenia Karatsuby opiera się o lubianą technikę „dziel i zwyciężaj". Chcemy pomnożyć n-cyfrowe liczby A i B zapisane w systemie binarnym. Zakładamy, że n jest potęgą dwójki (zawsze można dodać zera wiodące).
Zapisujemy liczby w postaci:

$$A = A_1 \cdot K + A_0,$$

$$B = B_1 \cdot K + B_0,$$

gdzie $K = 2^{\frac{n}{2}}$. Wtedy wynikiem jest:

$$A \cdot B = A_1 B_1 \cdot K^2 + (A_0 B_1 + A_1 B_0) \cdot K + A_0 B_0$$

Mnożenie przez K to po prostu przesunięcie bitowe, więc możemy wykonać je w czasie $\mathcal{O}(n)$, dodawanie też. Korzystając z obserwacji, że:

$$A_0 B_1 + A_1 B_0 = (A_0 + A_1) \cdot (B_0 + B_1) - A_0 B_0 - A_1 B_1$$

pozostają nam do wykonania 3 rekurencyjne mnożenia: $A_0 B_0$, $A_1 B_1$ i $(A_0 + A_1) \cdot (B_0 + B_1)$. Czyli mamy złożoność:

$$T(n) = 3T\left(\frac{n}{2}\right) + \mathcal{O}(n),$$

więc z Uniwersalnego Twierdzenia o Rekurencji algorytm działa w $\mathcal{O}(n^{\log_2 3}) \approx \mathcal{O}(n^{1.59})$.

B.2 Opisać algorytm Tooma-Cooka mnożenia dużych liczb binarnych.

Omówimy algorytm w wersji Toom-m z przykładem dla $m = 3$. Chcemy pomnożyć n-cyfrowe liczby A i B zapisane w systemie binarnym. Rozkładamy je podobnie jak w algorytmie Karatsuby, tym razem na $K = 2^{n/m}$ składników:

$$A = A_{m-1} \cdot K^{m-1} + \cdots + A_1 \cdot K + A_0$$

$$B = B_{m-1} \cdot K^{m-1} + \cdots + B_1 \cdot K + B_0$$

Traktujemy tak zapisane liczby jak wielomiany:

$$\mathcal{A}(X) = A_{m-1} \cdot X^{m-1} + \cdots + A_1 \cdot X + A_0$$

$$\mathcal{B}(X) = B_{m-1} \cdot X^{m-1} + \cdots + B_1 \cdot X + B_0$$

Wtedy wynikiem mnożenia jest $\mathcal{C}(K) = \mathcal{A}(K) \cdot \mathcal{B}(K)$. Ponieważ wielomian $\mathcal{C}$ jest stopnia $2m-2$, obliczamy wartości wielomianów $\mathcal{A}(X)$ i $\mathcal{B}(X)$ w $2m-1$ punktach. Następnie mnożymy punktowo i interpolujemy $\mathcal{C}$, korzystając z twierdzenia, że dla dowolnych wielomianów f, g zachodzi:

$$f(x) \cdot g(x) = (f \cdot g)(x)$$

Wartość wielomianu w punkcie obliczamy w czasie $\mathcal{O}(n)$.
Przyjrzyjmy się złożoności całego algorytmu dla $m = 3$. Dzielimy A i B na 3 części i traktujemy je jak wielomiany $\mathcal{A}$ i $\mathcal{B}$ stopnia 2. Wyznaczamy wartości tych wielomianów w 5 wybranych punktach rekurencyjnie na liczbach długości $\frac{n}{3}$. To daje złożoność $T(n) = 5T(\frac{n}{3}) + \mathcal{O}(n)$, czyli $T(n) \in \mathcal{O}(n^{\log_3 5}) \approx \mathcal{O}(n^{1.46})$.

B.3 Zapisać i udowodnić Chińskie Twierdzenie o Resztach.

Twierdzenie B.3.1 (Chińskie Twierdzenie o Resztach)**.** Dla parami względnie pierwszych liczb całkowitych $m_1, \ldots m_k \geq 1$, $M = m_1 \cdot \ldots \cdot m_k$ oraz liczb $a_1, \ldots, a_k$, gdzie $0 \leq a_i \leq m_i$, układ kongruencji:

$$\begin{aligned} x &= a_1 \pmod{m_1} \\ &\cdots \\ x &= a_k \pmod{m_k} \end{aligned}$$

ma dokładnie jedno rozwiązanie $x < M$.

Dowód. Trzeba udowodnić, że rozwiązanie istnieje i że wszystkie rozwiązania przystają do siebie modulo M.
Istnienie[1]
Korzystamy z twierdzenia Bézout, z którego wynika, że dla względnie pierwszych liczb x, y istnieją takie liczby całkowite a i b, że $ax + by = 1$.
Definiujemy $M_i = \frac{M}{m_i}$, wtedy M_i oraz m_i są względnie pierwsze, więc istnieją takie liczby B_i, b_i, że $B_i M_i + b_i m_i = 1$. Możemy zatem skonstruować rozwiązanie:

$$x = \sum_{i=1}^{k} a_i B_i M_i$$

[1]Dowód został zaczerpnięty z „Chinese remainder theorem", Wikipedia.

Rzeczywiście spełnia ono układ kongruencji, ponieważ:

$$a_i B_i M_i = 0 \pmod{m_j} \quad \text{dla } i \neq j$$

$$a_i B_i M_i = a_i(1 - b_i m_i) = a_i \pmod{m_i}$$

Jedyność
Załóżmy, że istnieją dwa rozwiązania x oraz y, które są różne modulo M. Wiemy, że zachodzi:

$$x = a_i \pmod{m_i},$$
$$y = a_i \pmod{m_i},$$

czyli $x - y = 0 \pmod{m_i}$. Skoro m_i są parami względnie pierwsze, to $x - y = 0 \pmod{M}$. Ponieważ $x, y < M$, to x i y muszą być równe. □

B.4 Opisać algorytm Millera-Rabina.

Każda liczba pierwsza p spełnia Małe Twierdzenie Fermata:

$$a^{p-1} = 1 \pmod{p}$$

Byłby to dobry test pierwszości, gdyby nie to, że liczba złożona też może go zdać dla szczególnych a. Algorytm Millera-Rabina rozwiązuje, a przynajmniej minimalizuje ten problem.

W grupie $\mathbb{Z}_p^*$ równanie $x^2 = 1 \pmod{p}$ ma dokładnie dwa rozwiązania: 1 i $p - 1 = -1$. Mając a takie, że $a^{n-1} = 1 \pmod{n}$:

- jeśli $a^{\frac{n-1}{2}} \neq \pm 1$, to mamy dowód, że n jest liczbą złożoną.
- jeśli $a^{\frac{n-1}{2}} = -1 \pmod{n}$, to a nie nada się na świadka niepierwszości n.
- jeśli $a^{\frac{n-1}{2}} = 1 \pmod{n}$, to możemy powtórzyć sprawdzenie dla $a^{\frac{n-1}{4}}$.

Składając powyższe przypadki w całość, otrzymujemy algorytm.

Algorytm Millera-Rabina:

1. Znajdź s i nieparzyste k, dla których $n - 1 = k \cdot 2^s$.
2. Wylosuj $a \in \{2, \ldots, p-1\}$.
3. Oblicz $(r_1, \ldots, r_s) = (a^k, a^{2k}, \ldots, a^{2^{s-1}k}, a^{n-1})$.
4. Jeśli $(r_1, \ldots, r_s) = (1, \ldots, 1)$, zwróć PIERWSZA.
5. Jeśli $(r_1, \ldots, r_s) = (\ldots, -1, 1, \ldots, 1)$, zwróć PIERWSZA.
6. Zwróć ZŁOŻONA.

Jeśli algorytm orzeka złożoność, na pewno tak jest. W przeciwnym przypadku prawdopodobieństwo pomyłki jest nie większe niż $\frac{1}{2}$, w praktyce mniejsze od $\frac{1}{4}$.

Lemat B.4.1. Niech n będzie liczbą złożoną. Dla losowego a liczba n nie przechodzi testu Millera-Rabina z prawdopodobieństwem co najmniej $\frac{1}{2}$.

Dowód. Rozważamy dwa przypadki:

- n nie jest potęgą liczby pierwszej.
  Zapisujemy $n = p \cdot q$, gdzie liczby $p,\ q > 1$ są względnie pierwsze. Dla grupy

  $$\mathbb{Z}_n^* = \{a : 1 \leq a < n, \gcd(a, n) = 1\}$$

  dobieramy takie $m = \frac{n-1}{2^t}$ dla pewnego t, żeby jakieś $a \in \mathbb{Z}_n^*$ nie spełniało $a^m = 1$. Sprawdzając $m = n-1, \frac{n-1}{2}, \ldots, \frac{n-1}{2^s}$, w końcu znajdziemy odpowiednie m, ponieważ $\frac{n-1}{2^s}$ jest nieparzyste, więc dla $a = -1$ na pewno $a^{\frac{n-1}{2^s}} = -1 \neq 1$.

  Wybieramy największe takie m, czyli albo $m = n-1$, albo $a^{2m} = 1$ dla każdego $a \in \mathbb{Z}_n^*$. Oznaczmy któreś z a takich, że $a^m \neq 1$ przez α.

  Wykażemy, że istnieje β takie, że $\beta^m \neq \pm 1$. Jeśli $\alpha^m \neq -1$, to mamy $\beta = \alpha$. W przeciwnym przypadku rozwiązujemy układ równań:

  $$\beta = 1 \pmod{p}$$

  $$\beta = \alpha \pmod{q}$$

  Chińskie Twierdzenie o Resztach zapewnia istnienie rozwiązania modulo $p \cdot q = n$. Wtedy $\beta^m = 1 \pmod{p}$ oraz $\beta^m = \alpha^m = -1 \pmod{q}$. Zatem niemożliwe, żeby β^m przystawało do ± 1 modulo n.

  Ustalamy zbiór $M = \{a : a^m = \pm 1\}$, który stanowi podgrupę $\mathbb{Z}_n^*$, ponieważ jest podzbiorem elementów odwracalnych zamkniętym na mnożenie. Wiadomo też, że $M \subsetneq \mathbb{Z}_n^*$, ponieważ $\beta \in \mathbb{Z}_n^* \setminus M$. Rząd M musi więc dzielić rząd $\mathbb{Z}_n^*$, w szczególności $|M| < \frac{n-1}{2}$. Zatem losując $a \in \{1, \ldots, n-1\}$ trafimy na $a \notin M$ z prawdopodobieństwem co najmniej $\frac{1}{2}$. Taka podstawa a świadczy o niepierwszości n, ponieważ dla $m = n-1$ albo $a^{n-1} \neq 1$, albo $a^{2m} = 1$ i $a^m \neq \pm 1$, co jest wychwytywane przez algorytm.

- $n = p^k$ jest potęgą liczby pierwszej.
  Liczby $M = \{a : a^{n-1} = 1 \mod n\}$ stanowią podgrupę $\mathbb{Z}_n^*$. Chcemy pokazać, że jest to podgrupa właściwa, czyli istnieje $\beta \in \mathbb{Z}_n^* \setminus M$ Dla $\beta = p + 1$ zachodzi:

  $$\beta^p = (p+1)^p = p^p + \ldots + \binom{p}{1} p + 1 = 1 \pmod{p^2},$$

  ponieważ p^2 dzieli wszystkie składniki oprócz 1.

Skoro $n = p^k$ dla $k > 1$, to $b^n = 1 \pmod{p^2}$. Gdyby było tak, że $b^{n-1} = 1 \pmod{n}$, to również $b^{n-1} = 1 \pmod{p^2}$, czyli $b^n = b = p + 1 \pmod{p^2}$, co daje sprzeczność. Zatem $b^{n-1} \neq 1 \pmod{n}$.

To wystarcza, żeby zamknąć dowód analogicznie jak w pierwszym przypadku.

□

Oczywiście można uzyskać jeszcze lepszą wiarygodność, powtarzając test dla kilku a, na przykład 12 kolejnych liczb pierwszych.
Złożoność jednego powtórzenia algorytmu to $\mathcal{O}(\log^3(n))$, ponieważ trzeba wykonać $\mathcal{O}(\log(n))$ mnożeń na liczbach długości $\log(n)$. Można zrobić to szybciej, używając FFT.

B.5 Zdefiniować pojęcie ideału, pierścienia ilorazowego oraz pokazać, że $\mathbb{Z}_p[X]/(W)$ jest ciałem wtw gdy W jest nierozkładalny.

Definicja B.5.1 (Ideał)**.** Jeżeli R jest pierścieniem, to $I \subseteq R$ jest ideałem wtedy i tylko wtedy, gdy:

- $x, y \in I \implies x + y \in I$ (zamknięcie na sumę)
- $x \in I, y \in R \implies y{\cdot}x \in I$ (własność wciągania – silniejsza niż zamkniętość na mnożenie)

Definicję ideału można uogólnić na więcej elementów – ideał $I \subseteq R$ generowany przez elementy $(a_1, \ldots, a_s)$ to:

$$\{x_1a_1 + \ldots + x_sa_s : a_1, \ldots, a_s \in I, x_1, \ldots, x_s \in R\}$$

zbiór sum wszystkich wielokrotności elementów $a_1, \ldots, a_s$.

Ideałem głównym nazywamy ideał generowany przez jeden element.
Przykład: zbiór liczb całkowitych jest pierścieniem ideałów głównych, czyli każdy jego ideał jest ideałem głównym.

Definicja B.5.2 (Pierścień ilorazowy)**.** Definiujemy relację równoważności R następująco:

$$x \sim y \iff x - y \in I$$

Pierścień ilorazowy to zbiór klas abstrakcji $R/I = \{[x]_\sim : x \in R\} = \{x + I : x \in R\}$.

Mniej formalnie mówimy po prostu o operacjach modulo ideał I.

Przykład: pierścień $\mathbb{Z}$ z ideałem głównym $n\mathbb{Z} = \{n \cdot x : x \in \mathbb{Z}\}$, dla pewnego n. Wtedy $\mathbb{Z}/n\mathbb{Z}$ to pierścień liczb całkowitych z działaniami modulo n.

Twierdzenie B.5.1. Pierścień $\mathbb{Z}_p[X]/(W)$ jest ciałem wtw, gdy W jest nierozkładalny, czyli nie daje się przedstawić jako iloczyn wielomianów stopnia co najmniej 1.

Dowód. Pierścień $\mathbb{Z}_p[X]/(W)$ jest ciałem $\implies$ W jest nierozkładalny.
Załóżmy nie wprost, że $\mathbb{Z}_p[X]/W$ jest ciałem i W jest rozkładalny, czyli można zapisać $W = g_1 \cdot g_2$, gdzie $1 < \deg(g_1), \deg(g_2) < \deg(W)$. Wynika z tego, że $g_1 \cdot g_2 = 0$, czyli $g_1,\ g_2$ są dzielnikami zera w tym ciele. Nie mają więc odwrotności względem mnożenia, co daje sprzeczność.

Pierścień $\mathbb{Z}_p[X]/(W)$ jest ciałem $\impliedby$ W jest nierozkładalny.
Wiemy, że $\mathbb{Z}_p[X]/W$ jest pierścieniem. Wystarczy pokazać, że każdy wielomian g stopnia mniejszego niż W ma odwrotność. Skoro W jest nierozkładalny, to $\gcd(g, W) = 1$. Możemy więc zastosować rozszerzony algorytm Euklidesa do znajdowania odwrotności g, która istnieje dla każdego $g \neq 0$. Zatem $\mathbb{Z}_p[X]/W$ jest ciałem. □

B.6 Pokazać, że każde ciało skończone ma p^k elementów.

Twierdzenie B.6.1. Ciało skończone $\mathbb{F}$ o charakterystyce p ma p^k elementów dla pewnego $k \in \mathbb{Z}$.

Dowód. Wiedząc, że $\mathbb{Z}_p$ jest ciałem, możemy wprowadzić mnożenie przez element z $\mathbb{Z}_p$ elementów z ciała $\mathbb{F}$:

$$a \cdot x = \underbrace{x + x + ... + x}_{a},\ a \in \mathbb{Z}_p,\ x \in \mathbb{F}$$

Definiujemy w ten sposób przestrzeń liniową $\mathbb{F}$ nad $\mathbb{Z}_p$, jako że mamy dodawanie i mnożenie przez skalar. Nasza przestrzeń liniowa ma skończony wymiar k i bazę $x_1, \ldots, x_k$.
Każdy element $x \in \mathbb{F}$ możemy zapisać jako:

$$a_1 x_1 + a_2 x_2 + \cdots + a_k x_k$$

Różne ciągi $(a_1, \ldots, a_k)$ dają różne elementy $\mathbb{F}$ z definicji bazy. Stąd liczba elementów $\mathbb{F}$ jest równa liczbie ciągów $(a_1, \ldots, a_k)$, czyli p^k. □

B.7 Pokazać, że pierwiastki wielomianu $X^{p^k} - X$ stanowią ciało, podać praktyczną metodę generowania ciała skończonego.

Twierdzenie B.7.1. Dane jest ciało $\mathbb{F}$ charakterystyki p i liczba $q = p^k$ dla pewnego k. Jeśli wielomian $f(X) = X^q - X$ ma q pierwiastków, to stanowią one q-elementowe podciało $\mathbb{F}$.

Dowód. Niech $A = \{a \in \mathbb{F} : f(a) = 0\} = \{a \in \mathbb{F} : a^q = a\}$ będzie zbiorem pierwiastków. Wystarczy pokazać, że zbiór A jest zamknięty na działania, bo inne aksjomaty ciała (łączność, przemienność, rozdzielność, etc.) wynikają z faktu, że pracujemy w $\mathbb{F}$, które od początku je spełniało.

- Elementy neutralne: $0, 1 \in A$
- Element $-1 \in A$:
  $(-1)^q = -1$ (również dla $p = 2$)
- Elementy odwrotne:
  $(a^{-1})^q \cdot a = (a^{-1})^q \cdot a^q = 1$, czyli $(a^{-1})^q = a^{-1} \in A$
- Zamkniętość na mnożenie:
  $(a \cdot b)^q = a^q \cdot b^q = a \cdot b$
  W szczególności, jeśli $a \in A$, to $-a = -1 \cdot a \in A$, stąd mamy elementy odwrotne w dodawaniu.
- Zakmniętość na dodawanie:
  Najpierw udowodnimy, że w $\mathbb{F}$ zachodzi $(a+b)^p = a^p + b^p$ dla dowolnych $a,\ b$.
  Wiemy, że:
  $$(a+b)^p = a^p + \ldots + \binom{p}{i} a^{p-i} b^i + \ldots + b^p$$
  Wszystkie współczynniki $\binom{p}{i}$ dla $1 \leq i < p$ są podzielne przez p, bo licznik jest równy $p!$ a mianownik jest niepodzielny przez p. Zatem w ciele charakterystyki p wszystkie składniki oprócz $a^p,\ b^p$ są równe 0. Korzystając z tego, dostajemy:

$$(a+b)^q = ((a+b)^p)^{p^{k-1}} = (a^p+b^p)^{p^{k-1}} = (a^{p^2}+b^{p^2})^{pk-2} = \ldots = a^{p^k}+b^{p^k} = a^q+b^q = a+b$$

□

B.7.1 Generowanie ciała skończonego

Losujemy wielomian w stopnia k i sprawdzamy, czy jest nierozkładalny i mamy dużą szansę trafić. Wtedy $\mathbb{Z}_p^*[X]/(w)$ jest ciałem.

B.8 Opisać algorytm Tonellego-Shanksa.

Algorytm Tonellego-Shanksa służy do obliczania pierwiastka dyskretnego w czasie wielomianowym randomizowanym.

Lemat B.8.1. Grupa cykliczna G o liczbie elementów $n = 2m$, zawiera dokładnie m kwadratów, a każdy kwadrat ma dokładnie dwa pierwiastki.

Dowód. Zauważmy, że każdy element grupy G należy do zbioru $\{g^0, g^1, \ldots, g^{2m-1}\}$, czyli traktujemy potęgi generatora modulo $2m$. Do rozważenia są dwa przypadki:

1. $a = g^{2k}$ dla pewnego całkowitego k:
   Wtedy piewiastkami a są g^k i g^{k+m}, ponieważ

$$(g^k)^2 = g^{2k} = a$$

$$(g^{k+m})^2 = g^{2k+2m} = g^{2k} = a$$

   Element a nie ma też żadnego innego pierwiastka.

2. $a = g^{2k+1}$ dla pewnego całkowitego k:
   Dla dowodu nie wprost, załóżmy że istnieje $b = g^j$, które jest pierwiastkiem a. Wtedy

$$b^2 = g^{2j} = g^{2k+1},$$

   czyli $2j = 2k + 1 \pmod{2m}$ – sprzeczność.

Wniosek: tylko i dokładnie parzyste potęgi generatora są kwadratami. □

Lemat B.8.2. Niech G będzie grupą cykliczną o $n = 2m$ elementach. Element a jest kwadratem w G wtedy i tylko wtedy, gdy $a^m = 1$.

Dowód. Jeśli a jest kwadratem, to $a^m = 1$.
Z lematu B.8.1 wynika, że jeśli a jest kwadratem, to musi być postaci $a = g^{2k}$. Zatem

$$a^m = \left(g^{2k}\right)^m = g^{2m \cdot k} = g^0 = 1$$

Jeśli a nie jest kwadratem, to $a^m \neq 1$.
Z lematu B.8.1 wynika, że a jest postaci $a = g^{2k+1}$ dla pewnego nieparzystego k. Zatem

$$g^{(2k+1)m} = g^{2m \cdot k} \cdot g^m = g^m = \xi = -1,$$

ponieważ rząd g jest równy $2m$. □

Tak przygotowani możemy przejść do algorytmu:
Mając grupę G o liczności $|G| = n = 2m$ oraz dane na wejściu $a \in G$, celem jest znaleźć x takie, że $x^2 = a$ w grupie G.

Algorytm Tonellego-Shanksa:

1. Wylosuj z takie, że $z^m \neq 1$.
2. Przypisz $q = m,\ t = 2m$.
3. Dopóki $2 \mid q$:
   (a) Zaktualizuj $q = q/2,\ t = t/2$.
   (b) Jeśli $a^q \cdot z^t \neq 1$, to zaktualizuj $t = t + m$.
4. Zwróć $a^{\frac{q+1}{2}} \cdot z^{\frac{t}{2}}$.

Algorytm zachowuje niezmienniki:

- $a^q \cdot z^t = 1$
- Jeśli $2^k \mid q$, to $2^{k+1} \mid t$.

Początkowo $a^q = z^t = 1$ oraz $t = 2q$. Z pierwszego niezmiennika wiadomo, że $a^{\frac{q}{2}} \cdot z^{\frac{t}{2}} = \pm 1$, ponieważ kwadrat tej liczby jest jedynką. Jeśli $a^{\frac{q}{2}} \cdot z^{\frac{t}{2}} = 1$, to możemy podzielić $q,\ t$ przez 2 i niezmiennik pozostaje zachowany. Jeśli $a^{\frac{q}{2}} \cdot z^{\frac{t}{2}} = \xi$, to

$$a^{\frac{q}{2}} \cdot z^{\frac{t}{2}+m} = (a^{\frac{q}{2}} \cdot z^{\frac{t}{2}}) \cdot z^m = \xi \cdot \xi = 1$$

Korzystamy z tego, że m niezmiennie jest wielokrotnością $2q'$.

Ostateczny wynik to:

$$x^2 = a^{q+1} \cdot z^t = a \cdot (a^q \cdot z^t) = a$$

Algorytm wykonuje $\mathcal{O}(\log n)$ iteracji, bo po tylu dzieleniach q staje się nieparzyste, więc jest to algorytm wielomianowy. Z lematu B.8.1 wynika, że przy losowaniu z znajdziemy niekwadrat z prawdopodobieństwem $\frac{1}{2}$, czyli w oczekiwaniu po stałej liczbie kroków będzie można przejść do kroku 2. Nie potrafimy zdeterminizować tego kroku, ponieważ gdyby dało się znaleźć niekwadrat bez losowania, czyli policzyć pierwiastek dyskretny deterministycznie, to również można by deterministycznie rozłożyć liczbę na czynniki pierwsze (patrz: pytanie A.9).

B.9 Opisać algorytm faktoryzacji Fermata.

Celem jest rozłożyć liczbę n na czynniki pierwsze. Zakładamy, że n jest nieparzyste (można wydzielić największą potęgę 2) i nie jest liczbą pierwszą, czyli, $n = p \cdot q$, gdzie $p > q$ są

nieparzyste. Definiujemy $a = \frac{p+q}{2}$ oraz $b = \frac{p-q}{2}$. Wtedy:

$$a^2 - b^2 = (a+b)\cdot(a-b) = p \cdot q = n$$

Jeśli znajdziemy a takie, że $a^2 - n = b^2$, gdzie b^2 jest dowolnym kwadratem liczby naturalnej, to możemy łatwo wyliczyć p i q.

Algorytm faktoryzacji Fermata:

1. Ustal $a = \lceil \sqrt{n} \rceil$.
2. Jeśli istnieje $b \in \mathbb{N}$ takie, że $b^2 = a^2 - n$, to zwróć dzielniki $p = a + b,\ q = a - b$.
3. Wpp powtórz procedurę dla $a = a + 1$.

Algorytm znajduje rozkład n na dwa czynniki po około $a - \sqrt{n}$ krokach, bo trzeba sprawdzić po kolei $\sqrt{n}, \ldots, a$, każde w $\Theta(1)$. Całkowita liczba operacji jest równa:

$$a - \sqrt{n} = \frac{a^2 - n}{a + \sqrt{n}} = \frac{b^2}{a + \sqrt{n}} \leq \frac{b^2}{\sqrt{n}} = \frac{(p-q)^2}{4\sqrt{n}}$$

Pesymistyczna złożoność algorytmu to $\mathcal{O}(n)$. Jednak przy małej różnicy $|p - q|$ algorytm działa szybko, a dla $|p - q| \leq \sqrt[4]{n}$ nawet w czasie stałym.

Wniosek jest taki, żeby do RSA nie używać liczb pierwszych położonych blisko siebie.

B.10 Opisać metodę Baby-Step-Giant-Step.

Algorytm Baby-Step-Giant-Step służy do rozwiązywania problemu logarytmu dyskretnego w czasie wykładniczym około $\sqrt{G}$, czyli prawie najlepszym osiągalnym aktualnie dla ludzkości.

W pewnej grupie G dla zadanych $a,\ b \in G$, chcemy obliczyć x, spełniające $a^x = b$.

Algorytm Baby-Step-Giant-Step:

1. Ustal $s = \left\lceil \sqrt{|G|} \right\rceil$.
2. Oblicz $A = \{a^0, \ldots, a^{s-1}\}$.
3. Oblicz $B = \left\{b \cdot a^{-s}, \ldots, b \cdot a^{-(s-1)s}\right\}$.
4. Jeśli $A \cap B = \varnothing$, nie ma rozwiązania.
5. Wybierz $u,\ v$ takie, że $b \cdot a^{-us} = a^v \in A \cap B$.
6. Zwróć $x = u \cdot s + v$.

Wadą tej metody jest złożoność pamięciowa równa czasowej, czyli $\Theta\left(\sqrt{|G|}\right)$.

B.11 Opisać kryptosystem plecakowy, dlaczego nie jest stosowany w praktyce.

Kryptosystem plecakowy jest systemem kryptograficznym opartym na założeniu o trudności problemu Subset-Sum.

Problem Subset-Sum

Wejście: Zbiór $V = \{v_1, v_2, \ldots, v_n\}$ oraz liczba s
Wyjście: Czy istnieje taki zbiór $A \subseteq V$, że $\sum_{a \in A} a = s$?
Klasa złożoności: NP (problem NP-zupełny)

Problem Subset-Sum jest jednak łatwy do rozwiązania, jeśli $v_1, \ldots, v_n$ tworzą ciąg nadrosnący, czyli $v_i > v_{i-1} + \ldots + v_1$. W takim przypadku działa liniowy algorytm zachłanny, ponieważ jeśli $v_i \leq s$ jest największym elementem nie przekraczającym s, to musi zostać wybrany, bo $v_1 + \ldots + v_{i-1} < s$.

W kryptosystemie plecakowym ustalamy nadrosnący ciąg $v_1, \ldots, v_n$, liczbę $m > v_1 + \ldots + v_n$ oraz a względnie pierwsze z m. Konstruujemy ciąg $w_1, \ldots, w_n$, obliczając $w_i = a \cdot v_i \mod m$, który posłuży za klucz publiczny.

Szyfrowanie:
Chcąc zaszyfrować bity $b_1, \ldots, b_n$, obliczamy i wysyłamy $s = \sum_i w_i \cdot b_i$. Odtworzenie wartości b_i jest równoznaczne z wyborem podzbioru $\{w_1, \ldots, w_n\}$, który sumuje się do s.

Deszyfrowanie:
Otrzymawszy s, obliczamy odwrotność a modulo m i otrzymujemy równanie:

$$s \cdot a^{-1} = \sum_i b_i \cdot v_i \pmod{m}$$

Ponieważ $m > \sum_i v_i$, rozwiązanie można znaleźć wspomnianym algorytmem zachłannym.

Kryptosystem plecakowy można złamać, ponieważ wielokrotność ciągu nadrosnącego jest szczególnym przypadkiem problemu Subset-Sum, rozwiązywalnym w czasie wielomianowym, co udowodnił Adi Shamir w 1982.

Grupa C

C.1 Pokazać, że grupa multiplikatywna ciała skończonego jest grupą cykliczną.

Twierdzenie C.1.1. Dla ciała skończonego $\mathbb{F}_q$ istnieje taki element g, że $\mathbb{F}_q^* = \{1, g, g^2, \dots\}$.

Dowód. Oznaczmy $n = \left|\mathbb{F}_q^*\right| = q - 1$. Definiujemy zbiory:

$$A_d = \left\{a \in \mathbb{F}_q : a^d = 1\right\}$$

$$B_d = \left\{a \in \mathbb{F}_q : a^d = 1, a^{d'} \neq 1 \text{ dla } d' < d\right\}$$

czyli B_d zawiera elementy, których rząd jest równy d. Widać, że $B_d \subseteq A_d$.

Z twierdzenia Lagrange'a wynika, że B_d może być niepusty tylko dla d będącego dzielnikiem n, a także $\sum_{d|n} |B_d| = n$, bo każdy element należy do któregoś B_d. Chcemy pokazać, że $B_n \neq \varnothing$, czyli istnieje element rzędu n.

Jeśli dla pewnego d zachodzi $B_d \neq \varnothing$, to $|A_d| \geq d$, bo jeśli $a^d = 1$, to również $(a^j)^d = 1$ dla $0 \leq j < d$. Z drugiej strony, $|A_d| \leq d$, bo wielomian $X^d - 1$ może mieć co najwyżej d pierwiastków. Zatem jeśli istnieje element rzędu d, to $|A_d| = d$. Wtedy $|B_d| = \varphi(d)$ – na ćwiczeniach udowodniliśmy, że jeśli istnieje jakiś generator, to można znaleźć $\varphi(d)$ generatorów. Podsumowując, dla każdego d zachodzi jedna z dwóch możliwości:

- $|B_d| = 0$
- $|B_d| = \varphi(d)$

Wynika stąd, że

$$n = \sum_{d|n} |B_d| \leq \sum_{d|n} \varphi(d),$$

przy czym jeśli chociaż jedno d ma $B_d = \varnothing$, to nierówność jest ostra.

Wiemy jednak, że $\sum_{d|n} \varphi(d) = n$, ponieważ możemy rozważyć wszystkie n ułamków właściwych postaci $\frac{k}{n}$ dla $1 \leq k \leq n$. Po skróceniu mają postać $\frac{k'}{d}$, gdzie $d \mid n$ oraz k' jest względnie pierwsze z d, więc dokładnie $\varphi(d)$ ułamków ma taki mianownik. To dowodzi równości.

Zatem nierówność nie może być ostra i zachodzi $B_d \neq \varnothing$ dla każdego $d \mid n$, w szczególności istnieje element rzędu n. □

C.2 Opisać algorytm „ro" Pollarda na faktoryzację.

Celem jest rozłożyć na czynniki liczbę złożoną n. Załóżmy, że $p \leq \sqrt{n}$ jest jakimś jej dzielnikiem pierwszym. Wybieramy łatwo obliczalną funkcję, niech to będzie $f(x) = x^2+1$. Chcemy zbadać ciąg $x_0 = 2$, $x_k = f(x_{k-1})$. Zakładamy, że wartości będą się rozkładały równomiernie modulo n. Ciąg powinien się zapętlić po około $\sqrt{n}$ krokach. Przypuszczamy jednak, że uda się to zaobserwować już wcześniej, po $\sqrt{p}$ krokach.

Jeśli znajdzie się $x_i = x_j \pmod{p}$, to $\gcd(x_i - x_j, n)$ powinno być nietrywialnym dzielnikiem n, ponieważ wykonaliśmy zbyt mało kroków, żeby $n \mid (x_i - x_j)$. Jeśli powstanie cykl modulo p o długości s, to taka sytuacja będzie się powtarzać dla dowolnych $x_i,\ x_{i+s}$. Zamiast co krok sprawdzać dla wszystkich $x_{i<j}$ przystawania do x_j, wykorzystujemy metodę Floyda lub Brenta, żeby wykryć długość cyklu.

Algorytm Pollarda:

1. Jeśli $2 \mid n$, zwróć 2.
2. Zdefiniuj $x = x_0, y = x_0, d = 1$.
3. Dopóki d jest trywialny:
   (a) Oblicz $x = f(x) \pmod{n}$, $y = f(f(y)) \pmod{n}$.
   (b) Oblicz $d = \gcd(x - y, n)$.
4. Jeśli $d \neq n$, zwróć d.
5. Powtórz procedurę od kroku 2, losując nowe x_0 oraz funkcję f.

Zapętlenie oraz wyłapanie tego wymaga w oczekiwaniu około $\sqrt{p}+s \leq 2\sqrt{p} = \mathcal{O}(\sqrt[4]{n})$ kroków. Jeśli algorytm kończy się powodzeniem, po tylu krokach $\gcd(x_{2k} - x_k, n)$ jest dzielnikiem n. Są jednak dwa scenariusze niepowodzenia:

- Wyrazy ciągu mogą nie powtórzyć się wystarczająco szybko
  – przerywamy procedurę po $5\sqrt[4]{n}$ krokach.
- Ciąg pętli się modulo p i modulo n, czyli $\gcd(x_i - x_j, n) = n$
  – przerywamy od razu.

C.2.1 Metoda Floyda

Pomysł polega na tym, żeby w każdym kroku sprawdzać tylko wartość $\gcd(x_{2k} - x_k, n)$, ponieważ te dwa wyrazy przystają do siebie modulo p, kiedy k jest wielokrotnością s. Ciąg x_{2k} definiujemy jako $x_{2k} = y_k = f(f(y_{k-1}))$, $y_0 = x_0$, unikając w ten sposób przechowywania wielu poprzednich wartości.

C.2.2 Metoda Brenta

Sprawdzamy, czy ciąg się zapętlił, wykonując kolejno $K = 1, 2, 4, 8, \ldots$ kroków. Jeśli w K krokach wyraz ciągu powtórzy się modulo p, to udało się znaleźć wspólny dzielnik. Jeśli nie, to powtarzamy próbę dla kolejnego K, zaczynając od miejsca, w którym skończyliśmy.

Ten wariant ma nieco lepszą stałą niż metoda Floyda.

C.3 Opisać algorytm sita kwadratowego.

C.3.1 Faktoryzacja przez liczby B-gładkie

Definicja C.3.1. Dla ustalonego B liczba B-gładka to taka, której wszystkie pierwsze dzielniki są mniejsze od B.

Celem jest rozłożyć N na czynniki pierwsze. Ustalamy pewne $a_1, \ldots, a_k$, po czym obliczamy $b_i = a_i^2 \pmod N$. Chcemy wybrać taki podzbiór b_i, żeby iloczyn jego elementów był pełnym kwadratem.

Lemat C.3.1. Niech $\{p_1, \ldots, p_s\}$ będzie zbiorem liczb pierwszych mniejszych od B. W każdym zbiorze liczb B-gładkich o liczności większej od s istnieje podzbiór, którego iloczyn jest kwadratem.

Dowód. Niech $\{x_1, \ldots, x_k\}$ dla $k > s$ będą liczbami B-gładkimi. Rozkładamy każdą z nich na czynniki pierwsze:

$$x_i = p_1^{\alpha_{1,i}} \cdots p_s^{\alpha_{s,i}}$$

a następnie przypisujemy jej wektor:

$$v_i = (\alpha_{1,i}, \ldots, \alpha_{s,i}) \mod 2$$

Jeśli dla $\{x_1, \ldots, x_k\}$ każdy czynnik p_i występuję parzyście wiele razy, czyli $v_1 \oplus \ldots \oplus v_k = 0$, jest to poszukiwany podzbiór. Skoro $k > s$, to wektory $v_1, \ldots, v_k$ są liniowo zależne w $\mathbb{Z}_2^s$, więc taki podzbiór musi istnieć i można go znaleźć za pomocą eliminacji Gaussa w $\mathcal{O}(s^3)$. □

Powyższą teorię można przekształcić na praktyczną metodę faktoryzacji liczby N.

Algorytm faktoryzacji:

1. Znajdź takie $a_1, \ldots, a_k$, że $b_i = a_i^2 \pmod N$ jest B-gładkie.
2. Dla każdego b_i wyznacz wektor v_i.
3. Znajdź taki podzbiór $I \subseteq \{1, \ldots, k\}$, że wektory $v_{i \in I}$ są liniowo zależne.
4. Przypisz $u = \prod_{i \in I} a_i \pmod N$ oraz $v = \sqrt{\prod_{i \in I} b_i} \pmod N$.

5. Jeśli $u \neq \pm v$, to $\gcd(u+v, N)$ jest nietrywialnym dzielnikiem N, ponieważ $u^2 = v^2 \pmod N$.
6. Jeśli $u = \pm v$, to powtórz procedurę.

Krok 1. jest na razie czarną skrzynką, ale można za nią podstawić algorytm sita kwadratowego.

C.3.2 Sito kwadratowe

Poszukiwane liczby a_i muszą być większe od $\sqrt{N}$, więc sprawdzamy po kolei dla potencjalnych kandydatów, czy reszty są B-gładkie. Dla ustalonego K oraz $i \in \{0, \ldots, K-1\}$ definiujemy:

$$X[i] = \left\lceil \sqrt{N} \right\rceil + i$$

$$Y[i] = X[i]^2 \pmod N = \left(\left\lceil \sqrt{N} \right\rceil + i \right)^2 - N$$

Trzeba rozłożyć liczby $Y[i]$ na czynniki – najlepiej wszystkie na raz, poniższym algorytmem.

Algorytm sita kwadratowego:

1. Dla każdego $p < B$:
   (a) Usuń ze zbioru $\{p_1, \ldots, p_s\}$ te liczby, dla których $X[j]^2 \neq N \pmod p$, czyli $Y[i] \neq 0 \pmod p$.
   (b) Algorytmem Tonellego-Shanksa znajdź rozwiązanie $X[i] = \pm k$ równania $X[i]^2 = N \pmod p$.
   (c) Podziel przez p wszystkie $Y[i]$, dla których $X[j] = \pm k \pmod p$.
2. Jeśli $Y[i]$ zmaleje do 1, to początkowo musiało być liczbą B-gładką.

W ramach dodatkowego usprawnienie algorytmu, możemy uniknąć dzielenia dużych liczb. Zamiast wartości $Y[i]$ przechowujemy $Z[i] = \log Y[i]$. Wtedy operację $\frac{Y[i]}{p}$ zastępujemy działaniem $Z[i] - \log p$. Warunkiem B-gładkości jest (prawie) wyzerowanie się $Z[i]$. Problemem może być występowanie czynników $Y[i]$ z potęgą $\alpha > 1$. Należy wykryć od razu liczby $Y[i]$ podzielne przez p^α dla każdego możliwego α, stosując tę samą metodę.

Przy optymalnym wyborze parametrów, czyli B, K rzędu $e^{\sqrt{\ln N \ln\ln N}}$, złożoność algorytmu wynosi $\mathcal{O}(e^{(1+o(1))\sqrt{\ln N \ln\ln N}})$, czyli jest podwykładnicza.

C.3.3 Metoda Multi Polynomial Quadratic Sieve (MPQS)

Algorytm sita kwadratowego można zrównoleglić, używając wielu różnych wielomianów postaci $(A \cdot X + B)^2 - N$ zamiast $X^2 - N$.

C.4 Opisać algorytm „ro" Pollarda na logarytm dyskretny.

Trudno na ten moment poprawić pierwiastkową złożoność czasową algorytmu Baby-Step-Giant-Step, można jednak zmniejszyć użycie pamięci – w algorytmie „ro" Pollarda wystarcza $\mathcal{O}(1)$.

Główny pomysł polega na tym, żeby generować pary postaci $a^{\alpha_i} \cdot b^{\beta_i}$, dla różnych α_i oraz β_i, aż wystąpi kolizja.

Jeśli $a^{\alpha_i} \cdot b^{\beta_i} = a^{\alpha_j} \cdot b^{\beta_j}$, to $a^{\alpha_i - \alpha_j} = b^{\beta_j - \beta_i}$. Wystarczy więc znaleźć $\beta^* = (\beta_j - \beta_i)^{-1}$ modulo $|G|$ i otrzymamy $a^{\beta^*(\alpha_i - \alpha_j)} = b$.

Ciąg (α_i, β_i) definiujemy, losując wartości początkowe (α_0, β_0), a kolejne obliczając za pomocą jakiejś deterministycznej funkcji f, która zapewni pseudolosowość wyników. Jeśli w pewnym momencie $(\alpha_i, \beta_i) = (\alpha_j, \beta_j)$, to tak już zostaje, czyli ciąg wpada w pętlę literki „ro". Drugi ciąg można zdefiniować za pomocą metody Floyda (patrz: pytanie C.2):

$$(\alpha'_{i+1}, \beta'_{i+1}) = f(f(\alpha'_i, \beta'_i))$$

W każdym kroku sprawdzamy, czy $(\alpha_i, \beta_i) = (\alpha'_i, \beta'_i)$. Jeśli ciągi są losowe, to kolizja wystąpi w oczekiwaniu po $\sqrt{G}$ krokach.

Pozostaje uszczegółowić wybór funkcji f. Powinna ona rozrzucać wartości ciągu możliwie przypadkowo wśród elementów G. Technika stosowana w praktyce jest następująca:

- Wybieramy małą liczbę naturalną n.
- Wybieramy funkcję haszującą h, która parze liczb całkowitych (α, β) przyporządkowuje liczbę ze zbioru $\{1, 2, \ldots, n\}$.
- Losujemy liczby naturalne $x_1, \ldots, x_n$ oraz $y_1, \ldots, y_n$.
- Definiujemy $f(\alpha, \beta) = (\alpha + x_s, \beta + y_s)$, gdzie $s = h(\alpha, \beta)$.

Algorytm „ro" działa w oczekiwanym czasie $\mathcal{O}\left(\sqrt{|G|}\right)$ i stałej pamięci.

C.5 Opisać algorytm Pohliga-Hellmana.

Algorytm jest heurystyką, korzystającą z faktu, że rząd grupy często jest liczbą złożoną. Mając daną grupę $G = \langle g \rangle$ o n elementach oraz $b \in G$, chcemy znaleźć x, rozwiązanie równania $g^x = b$.

Algorytm Pohliga-Hellmana:

1. Jeśli n jest liczbą pierwszą, to wyznacz x algorytmem Baby-Step-Giant-Step $\rightarrow \mathcal{O}(\sqrt{n})$.
2. Jeśli $n = p^k$, gdzie p jest pierwsze oraz $k > 1$, to:

(a) Zapisz (nieznany jeszcze) x w systemie o podstawie p jako

$$x = x_0 + x_1 \cdot p + \ldots + x_{k-1} \cdot p^{k-1}$$

(b) Na podstawie równania:

$$(g^x)^{p^{k-1}} = (g^{x_0})^{p^{k-1}} = b^{p^{k-1}},$$

zdefiniuj $\gamma = g^{p^{k-1}}$ o rzędzie p.

(c) Rozwiąż równanie postaci $\gamma^{x_0} = b^{p^{k-1}}$ dla x_0, czyli oblicz logarytm dyskretny w grupie $\{1, \gamma, \gamma^2, \ldots, \gamma^{p-1}\} \to \mathcal{O}(\sqrt{p})$ (przypadek 1).

(d) Sprowadź równanie do postaci:

$$g^{x_1 \cdot p + \ldots + x_{k-1} \cdot p^{k-1}} = b \cdot g^{-x_0}$$

Po podniesieniu równania do potęgi p^{k-2}, oblicz x_1, rozwiązując $\gamma^{x_1} = (b \cdot g^{-x_0})^{p^{k-2}}$.

Powtórz procedurę dla $x_0, x_1, \ldots, x_{k-1} \to \mathcal{O}(k\sqrt{p})$.

3. Jeśli $n = q_1 \cdot q_2$, gdzie $\gcd(q_1, q_2) = 1$, to:

(a) Zapisz x jako $x = u_1 \cdot q_1 + r_1$.

(b) Podnieś równanie do potęgi q_2:

$$g^{u_1 n + r_1 q_2} = b^{q_2}$$

Przy $g^n = 1$ daje to $g^{r_1 q_2} = b^{q_2}$, czyli $h = g^{q_2}$ ma rząd q_1.

(c) Oblicz r_1 takie, że $x = r_1 \pmod{q_1}$, rekurencyjnie na grupie $H = \langle h \rangle$.

(d) Analogicznie, oblicz r_2 takie, że $x = r_2 \pmod{q_2}$.

(e) Z CRT wyznacz x, rozwiązanie układu kongruencji $x = u_i \cdot q_i + r_i$.

Jeżeli $n = p_1^{\alpha_1} \cdot \ldots \cdot p_s^{\alpha^s}$, to rekurencyjnie wywołaj przypadek 2 dla każdego $p_i^{\alpha_i}$ $\to \mathcal{O}(\sum_i \alpha_i \sqrt{p_i}) \leq \mathcal{O}(\log n \cdot \sqrt{\max p_i})$, bo $\alpha_i \leq \log n$.

Algorytm działa szybko, jeśli rząd grupy ma tylko małe dzielniki. Zazwyczaj radzi sobie też z grupą $GF(p^k)$ ciała skończonego, ponieważ rząd grupy $p^k - 1$ najczęściej jest liczbą złożoną. Rozsądnym wyborem jest $\mathbb{Z}_p^*$ dla $p = 2q + 1$, gdzie p, q są pierwsze (liczby pierwsze Sophie Germain).

C.6 Opisać algorytm rachunku indeksów.

Rachunek indeksów to obecnie najlepszy znany algorytm na logarytm dyskretny w $\mathbb{Z}_p$.

Działając w grupie $\mathbb{Z}_p$ o znanym generatorze g, dla $g^x = b$ chcemy znaleźć $x = \log b$. Taki logarytm zachowuje własności zwykłego logarytmu.

Faza I:
Ustalamy zbiór małych liczby pierwszych $Q = \{q_1, \ldots, q_r\}$ – bazę czynników. Wybieramy lub losujemy k, tak żeby uzyskać r różnych liczb, dla których $g^k \pmod p$ rozkłada się na czynniki z bazy Q. W ten sposób otrzymujemy r równań postaci:

$$g^k = q_1^{\beta_1} \cdot \ldots \cdot q_r^{\beta_r} \pmod p,$$

$$\text{czyli } k = \beta_1 \cdot \log q_1 + \ldots + \beta_r \cdot \log q_r \pmod{p-1}$$

Faza II:
Znając wszystkie β_i oraz k, chcemy rozwiązać układ równań:

$$k^{(1)} = \beta_1^{(1)} \cdot \log q_1 + \ldots + \beta_r^{(1)} \cdot \log q_r$$

$$\ldots$$

$$k^{(r)} = \beta_1^{(r)} \cdot \log q_1 + \ldots + \beta_r^{(r)} \cdot \log q_r$$

z niewiadomymi $\log q_i$. Możemy wykorzystać do tego metody algebry liniowej z dokładnością do jednej trudności – operacje są modulo $p-1$, które nie jest pierwsze, więc niekoniecznie da się policzyć odwrotność względem mnożenia. Jako wynik otrzymujemy logarytmy dyskretny liczb $q_1, \ldots, q_r$.

Faza III:
Żeby znaleźć $\log b$, próbujemy wygenerować jakąkolwiek liczbę postaci $b^u \cdot g^v$, która jest gładka modulo p. Wystarczy przyjąć $u = 1$ i losować v do skutku. Z równości

$$b^u \cdot g^v = q_1^{\gamma_1} \cdot \ldots \cdot q_r^{\gamma_r}$$

ostatecznie wynika:

$$x = \log b = u^{-1} \cdot (\gamma_1 \cdot \log q_1 + \ldots + \gamma_r \cdot \log q_r - v)$$

Oczekiwana złożoność algorytmu to $e^{(\sqrt{2}+\mathcal{O}(1))\sqrt{\ln n \ln\ln n}}$, czyli podwykładnicza. Metoda rachunku indeksów daje się uogólnić na ciała skończone $\mathbb{Z}_p/h(X)$, gdzie h jest pewnym wielomianem. Wtedy baza czynników, zamiast liczb pierwszych, zawiera nierozkładalne wielomiany małych stopni.

C.6.1 Atak Logjam

Rachunek indeksów, jeśli stosowany w sposób leniwy, ma słaby punkt – fazy I i II są niezależne od wejściowego elementu b, co gorsza można je łatwo zrównoleglić. Żeby rozszyfrować ciąg informacji, dla których zawsze używana była ta sama liczba pierwsza p, wystarczy wykonać fazy I i II tylko raz, a faza III jest już bardzo szybka.

C.7 Opisać ideę algorytmu Schönhage-Strassena.

C.7.1 Teoretyczna Transformata Numeryczna (NTT)

Transformata NTT (*Number Theoretic Transform*) jest wersją FFT, która nie używa liczb rzeczywistych – zamiast nad $\mathbb{R}$, obliczenia są wykonywane nad $\mathbb{Z}_p$. Wzory rekurencyjne i własność wzajemnej odwrotności DFT i IDFT przenoszą się na pierścień $\mathbb{Z}_p$. Jedynym warunkiem jest, żeby $\omega \in \mathbb{Z}_p$ spełniało $\omega^n = 1$ oraz $\omega^{\frac{n}{2}} = -1$, czyli żeby ω było rzędu n. Z twierdzenia Lagrange'a oraz faktu, że $\mathbb{Z}_p$ jest cykliczna wiemy, że taki element istnieje, jeśli $n \mid (p-1)$. Algorytm FFT modulo p (czyli właśnie NTT) działa dla p postaci $q \cdot n + 1$.

Splatanie ciągów $A = (a_0, a_1, \ldots, a_n)$ i $B = (b_0, b_1, \ldots, b_n)$:

1. Znajdź p postaci $q \cdot n + 1$, gdzie q jest niedużą liczbą pierwszą – można wylosować i sprawdzić pierwszość lub szukać po kolei.
2. Znajdź generator g grupy $\mathbb{Z}_p^*$ – generatorów jest $\varphi(q \cdot n) = (q-1) \cdot \frac{n}{2}$, więc losowanie zadziała. Generator musi spełniać $g^{\frac{p-1}{2}} \neq 1 \pmod p$ oraz $g^{\frac{p-1}{q}} \neq 1 \pmod p$.
3. Ustal $\omega = g^q$ rzędu n i oblicz splot za pomocą NTT.

Można wybrać na tyle duże p, żeby modulo nie miało wpływu na wynik. Przy mnożeniu liczb binarnych wystarczy $p > n$.

C.7.2 Algorytm Schönhage-Strassena

Algorytm służy do mnożenia dużych liczb w zapisie binarnym. Ponieważ daje on zysk praktyczny dopiero w przypadku *naprawdę* dużych liczb, musi radzić sobie z problemami wynikającymi z długości ich zapisu.

Pierwszym z wyzwań jest wybór pierścienia do obliczeń – jeśli liczba p jest duże, to operacje w $\mathbb{Z}_p$ nie są jednostkowe.

Liczba p nie musi być pierwsza, wystarczy nam względna pierwszość z n, (potrzebne jest dzielenie przez n) i musi istnieć element ω rzędu n modulo p. Te warunki spełniają $p = 2^m + 1$, $\omega = 2^\alpha$ takie, żeby $\omega^n = 1 \pmod p$. Sprytny dobór liczb upraszcza obliczenia:

- branie reszty modulo p – zauważamy, że $2^m = p - 1 = -1 \pmod p$, więc możemy podzielić

liczbę na m-cyfrowe fragmenty i dodać je z naprzemiennymi znakami:

$$101001110 \pmod{2^3+1} = 101 - 001 + 110 \pmod{2^3+1},$$

- mnożenie przez ω – to mnożenie przez potęgę 2, czyli dopisanie zer na końcu liczby binarnej i wyciągnięcie reszty modulo p,
- mnożenie dwóch liczb w $\mathbb{Z}_p$ – nadal trudne, ale wykonywane niewiele razy i można sobie z nim poradzić rekurencją na liczbach o długości m,
- dzielenie przez $\omega = 2^\alpha$ – to mnożenie przez $\omega^{-1} = -2^{m-\alpha}$, czyli też potęgę 2.

Algorytm Schönhage-Strassena służy do obliczania iloczynu N-bitowych liczb A, B modulo $p = 2^N + 1$. Jeśli chce się otrzymać wynik pełny (nie modulo), wystarczy przyjąć N większe od sumy długości liczb na wejściu. Zakładamy, że N jest potęgą 2. Algorym działa rekurencyjnie, sprowadzając iloczyn A i B do splotu wielomianów w mniejszym pierścieniu.

Algorytm fastmul(A, B, N):

1. Rozbijamy liczby A i B na fragmenty o długości około $\sqrt{N}$, czyli na t binarnych liczb b-cyfrowych. Liczby $t,\ b$ powinny być potęgami dwójki wielkości mniej więcej $\sqrt{N}$.

   $$A = u_0 + u_1 \cdot 2^b + u_2 \cdot 2^{2b} + \ldots + u_{t-1} \cdot 2^{(t-1)b}$$

   $$B = v_0 + v_1 \cdot 2^b + v_2 \cdot 2^{2b} + \ldots + v_{t-1} \cdot 2^{(t-1)b}$$

   Zadanie sprowadza się do pomnożenia dwóch liczb t-cyfrowych w systemie o podstawie 2^b.

2. Wykonujemy algorytm NTT w pierścieniu $\mathbb{Z}_p$ dla $p = 2^{b'} + 1$. Wymaga to obliczenia $2t$ iloczynów modulo $2^{b'} + 1$ rekurencyjnie.

W punkcie 2 splatamy liczby w pierścieniu $\mathbb{Z}_p$ dla $p = 2^{b'} + 1$, gdzie $b' \approx 2b + \log t$ oraz $t \mid b'$. Bierze się to stąd, że wynik splotu nie przekracza p, czyli $p > t \cdot \left(2^b\right)^2$, więc $b' \geq 2b + \log t$. Podzielność $t \mid b'$ jest ważna, żeby rząd $\omega = 2^{\frac{2b}{t}}$ był równy t, czyli $\omega^{\frac{t}{2}} = 2^{b'} = -1 \pmod{p}$.

Algorytm można opisać ogólnym równaniem

$$T(n) = 2\sqrt{n} \cdot T(\alpha\sqrt{n}) + \Theta(nlogn)$$

dla pewnego całkowitego α.

Sprowadzamy mnożenie dwóch liczb N-bitowych do około $2\sqrt{N}$ mnożeń liczb o długości $4\sqrt{N}$, co daje złożoność $\mathcal{O}(N \log^2 N)$. Poprawienie jej do $\mathcal{O}(N \log N \log\log N)$ wymaga zmniejszenia stałych $(2, 4) \to (1, 2)$ technicznymi sztuczkami.

W praktyce rekursja jest płytka, ponieważ dla $N < 10^3$ lepiej użyć prostszych algorytmów, na

przykład Tooma-Cooka albo nawet kwadratowego mnożenia.

C.8 Opisać algorytm Schreiera-Simsa.

Definicja C.8.1. Grupa permutacji to zbiór S_n z działaniem składania permutacji.

Definicja C.8.2. Stabilizatorem i w grupie permutacji A jest zbiór $A_i = \{\pi \in A : \pi(i) = i\}$.

Definicja C.8.3. Orbitą i w grupie permutacji A jest zbiór $O_i = \{\pi(i) : \pi \in A\}$.

Dla danego zbioru permutacji $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_k\} \subset S_n$ chcemy określić, ile elementów ma grupa $A \subseteq S_n$ generowana przez Σ.

Korzystamy z następującego twierdzenia:

Twierdzenie C.8.1. $|A| = |A_1| \cdot |O_1|$

Dowód. Dzielimy grupę A na podzbiory $B_1, \ldots, B_n$ zdefiniowane jako:

$$B_x = \{\pi \in A : \pi(1) = x\}$$

Zbiór B_x jest niepusty, jeśli $x \in O_1$, czyli istnieje permutacja $\pi \in A$ taka, że $\pi_x(1) = x$. Zatem odwzorowanie z A_1 do B_x:

$$\zeta \mapsto \pi_x \circ \zeta$$

jest bijekcją i każdy niepusty zbiór B_x jest równoliczny z A_1. Stąd otrzymujemy:

$$|A| = |B_1| + \ldots + |B_n| = |A_1| \cdot |O_1|,$$

co było do pokazania. □

Z twierdzenia wynika schemat algorytmu:

1. Znajdź zbiór O_1.
2. Utwórz zbiór $\Sigma' \subseteq \Sigma$, który generuje A_1.
3. Rekurencyjnie wyznacz $|A_1|$.
4. Oblicz $|A| = |O_1| \cdot |A_1|$.

Jak wyznaczyć orbitę O_1?

1. Utwórz graf $G = (\{1, \ldots, n\}, E)$, gdzie $E = \{(x, \sigma_i(x)) : 1 \leq i \leq k,\ x \in V\}$.
2. Wyznacz O_1, czyli elementy osiągalne z 1, na przykład BFSem.
3. Dla każdego $x \in O_1$ zapamiętaj $\pi_x \in A$ taką, że $\pi_x(1) = x$.
   Permutacja π_x jest złożeniem generatorów na ścieżce od 1 do x.

Jak wyznaczyć zbiór Σ' generujący A_1?

Dla $\sigma_i \in \Sigma$ oraz $x \in \{1, 2, \ldots, n\}$ konstruujemy:

$$\tau_{i,x} = \pi^{-1}_{\sigma_i(x)} \circ \sigma_i \circ \pi_x$$

Permutacja $\tau_{i,x}$ działa następująco:

$$1 \xrightarrow{\pi_x} x \xrightarrow{\sigma_i} \sigma_i(x) \xrightarrow{\pi^{-1}_{\sigma_i(x)}} 1,$$

czyli $\tau_{i,x} \in A_1$.

Lemat C.8.1 (Schreiera)**.** Zbiór $\Sigma' = \{\tau_{i,x} : 1 \leq i \leq k,\ 1 \leq x \leq n\}$ generuje A_1.

Dowód. Dowolna permutację $\zeta \in A_1$ musi być iloczynem generatorów:

$$\zeta = \alpha_s \circ \alpha_{s-1} \circ \ldots \circ \alpha_2 \circ \alpha_1$$

Ustalamy $y_j = \alpha_j \circ \ldots \circ \alpha_1(1)$, dla $j = 1, 2, \ldots, s$ i dodajemy $\pi_{y_j} \circ \pi^{-1}_{y_j}$ między każde α_j i α_{j+1}:

$$\zeta = \alpha_s \circ \pi_{y_{s-1}} \circ \pi^{-1}_{y_{s-1}} \circ \alpha_{s-1} \circ \pi_{y_{s-2}} \circ \pi^{-1}_{y_{s-2}} \circ \ldots \circ \alpha_2 \circ \pi_{y_1} \circ \pi^{-1}_{y_1} \circ \alpha_1$$

Ponieważ $\pi_1 = \mathrm{id}$, otrzymujemy:

$$\zeta = (\pi^{-1}_1 \circ \alpha_s \circ \pi_{y_{s-1}}) \circ (\pi^{-1}_{y_{s-1}} \circ \alpha_{s-1} \circ \pi_{y_{s-2}}) \circ \ldots \circ (\pi^{-1}_{y_1} \circ \alpha_1 \circ \pi_1)$$

Udało się wygenerować permutację ζ jako złożenie generatorów z Σ'. □

Konstrukcja zbioru Σ' jest poprawna, ale w rekurencji jego rozmiar może dojść do $n!$, ponieważ $|\Sigma'| \approx n \cdot |\Sigma|$. Do kontroli nad tym służy filtr Simsa.

C.8.1 Filtr Simsa

Twierdzenie C.8.2. Dla dowolnego zbioru $P \subseteq S_n$ istnieje zbiór $Q \subseteq S_n$ taki, że podgrupy generowane przez P i Q są równe oraz $|Q| \leq \frac{n(n-1)}{2}$.

Dowód twierdzenia jest algorytmizowalny:
Dzielimy permutacje na klasy – permutacja $\zeta \neq \mathrm{id}$ jest klasy (i, j), jeśli:

$$\zeta(1) = 1, \zeta(2) = 2, \ldots, \zeta(i-1) = i-1,$$

$$\zeta(i) = j > i.$$

Chcemy, żeby w Q był co najwyżej jeden element z każdej klasy, ponieważ to gwarantuje, że $|Q| \leq \frac{n(n-1)}{2}$.

Zaczynając od $Q = \varnothing$, przekładamy permutacje z P do Q, utrzymując niezmienniki, że grupa generowana przez $P \cup Q$ nie zmienia się i w Q jest co najwyżej jeden element każdej klasy.

Dla każdego elementu $\zeta \in P$ klasy (i, j):

1. Jeśli w Q nie ma nic klasy (i, j), przekładamy ζ do Q.
2. Jeśli w Q jest element φ klasy (i, j), zastępujemy ζ przez $\zeta' = \varphi^{-1} \circ \zeta$ i powtarzamy.

Procedura zachowuje niezmienniki, ponieważ $\zeta = \varphi \circ \zeta'$, więc zbiór generowany przez $P \cup Q$ pozostaje taki sam. Zmienia się natomiast klasa elementu, ponieważ $\zeta'(i) = i$, więc ζ' jest klasy (i', j'), gdzie $i' > i$.

Algorytm się nie zapętli, ponieważ P ma co najwyżej k elementów, które można przerzucić do Q i można co najwyżej n razy zmienić klasę ζ, bo każda zmiana zwiększa i. Filtr Simsa działa więc w czasie $\mathcal{O}(k \cdot n^2)$, bo operacje na permutacjach są liniowe.

Poszczególne etapy algorytm Schreiera-Simsa mają złożoność:

1. Obliczenie orbity $O_1 \longrightarrow |\Sigma| \cdot n$
2. Wyznaczenie $\Sigma' \longrightarrow |\Sigma'| = \mathcal{O}(n \cdot |\Sigma|)$
3. Filtrowanie $\Sigma' \longrightarrow n^2 \cdot |\Sigma'| = \mathcal{O}(n^3 \cdot |\Sigma|)$

Algorytm wykonuje n wywołań rekurencyjnych, a $|\Sigma|$ w każdym z nich mieści się w $\mathcal{O}(n^2)$, więc łączna złożoność to $\mathcal{O}(n^6)$.

C.9 Opisać algorytm Grovera.

Definicja C.9.1 (Bramka H)**.** Bramka zdefiniowana jako

$$H\left(\begin{bmatrix}\alpha\\ \beta\end{bmatrix}\right) = \begin{bmatrix}\frac{1}{\sqrt{2}}(\alpha+\beta)\\ \frac{1}{\sqrt{2}}(\alpha-\beta)\end{bmatrix}$$

o własności $H(H(x)) = x$.

Definicja C.9.2 (Bramka Toffolego)**.** Trójargumentowa bramka

$$\mathrm{CCNOT}(x, y, z) = (x, y, (x \wedge y) \oplus z)$$

jest odwracalna i uniwersalna.

Algorytm Grovera to kwantowy algorytm wyszukiwania. Dla zadanej funkcji:

$$f : \{0, 1, \ldots, N-1\} \to \{0, 1\}$$

znajduje z prawdopodobieństwem $\frac{2}{3}$, ciąg x^* wartościowany przez f inaczej niż pozostałe. Na potrzeby algorytmu zakładamy, że $N = 2^n$ oraz f jest określona na ciągach z $\{0,1\}^n$, z których tylko na x^* przyjmuje wartość -1, a na pozostałych 1.

Tworzymy n-wejściową bramkę O_f, określoną regułą $O_f(|x\rangle) = f(x) \cdot |x\rangle$.

Wykorzystując bramkę O_f, konstruujemy obwód kwantowy, który:

- składa się z bramek H, T i O_f,
- ma n wejść i n wyjść,
- otrzymuje na wejściu $|00\ldots0\rangle$,
- zwraca na wyjściu ciąg x^*, dla którego $f(x^*) = -1$ z prawdopodobieństwem $\frac{1}{10}$,
- zawiera $\mathcal{O}(\sqrt{N}\log N)$ bramek, w tym $\mathcal{O}(\sqrt{N}) = \mathcal{O}(\sqrt{2^n})$ bramek O_f.

Dysponując opisanym wyżej obwodem, możemy przejść do właściwego algorytmu.

Algorytm Grovera:

1. Przepuść każdy z n kubitów przez bramkę H:

$$|0\rangle \to \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$$

   Układ n kubitów ma stan $\frac{1}{\sqrt{2^n}}\sum_{x\in\{0,1\}^n}|x\rangle$, czyli każdy n-bitowy ciąg jest równie prawdopodobny.

2. Kilkukrotnie przepuść kubity przez warstwę bramek, przekształcającą:

$$\sum_{x\in\{0,1\}^n} \delta_x|x\rangle \to \sum_{x\in\{0,1\}^n} \delta'_x|x\rangle$$

Każda warstwa zwiększa współczynnik przy x^*, a zmniejsza przy $x \neq x^*$, zachowując niezmienniki:

- $\delta_x \in \mathbb{R}^+$,
- $\delta'_{x^*} > \delta_{x^*} + \frac{1}{\alpha\sqrt{N}}$ dla stałej α.

Po $\mathcal{O}(\sqrt{N})$ krokach $|\delta_{x^*}|^2 > \frac{1}{10}$, czyli z prawdopodobieństwem co najmniej $\frac{1}{10}$ ciągiem wyjściowym jest x^*. Do osiągnięcia tego służą operacje liniowe:

- bramka O_f:
$$\delta_{x^*} \to -\delta_{x^*},\ \delta_{x^*} \to -\delta_{x^*} \text{ dla } x \neq x^*$$
- bramka dyfuzyjna Grovera (D):

$$(\delta_0, \delta_1, \ldots, \delta_{N-1}) \to (2s - \delta_0, 2s - \delta_1, \ldots, 2s - \delta_{N-1}),$$

gdzie $s = \frac{1}{N}(\delta_0 + \delta_1 + \ldots + \delta_{N-1})$, czyli jest średnią współczynników. Operacja D odbija symetrycznie każde δ_x względem s.

Lemat C.9.1. Operacje D i O_f wystarczają, żeby zwiększyć współczynnik przy x^*, a zmniejszyć przy pozostałych $x \neq x^*$.

Dowód. Stan układu kubitów to $\sum_x \delta'_x|x\rangle$. Początkowo $\delta_x = \frac{1}{\sqrt{N}}$ dla każdego x, a w każdym kroku jeden ze współczynników (δ_{x^*}) jest zamieniany na przeciwny. Następnie odejmujemy wszystkie współczynniki od $2s$. Wynika z tego, że po k krokach wszystkie (δ_x) dla $x \neq x^*$ są sobie równe z wyjątkiem $(\delta_{x^*}) = b_k$. Bramka O_f zamienia b_k na $-b_k$, po czym bramka D oblicza $b_{k+1} = 2s_k + b_k$, gdzie s_k jest średnią współczynników w k-tym kroku. Można udowodnić indukcyjnie, że $b_{k+1} - b_k > \frac{1}{\sqrt{N}}$ tak długo, jak $b_k < \frac{1}{2}$. Stąd po co najwyżej $\frac{1}{10}\sqrt{N}$ krokach dostaniemy $b_k > \frac{1}{10}$, co było do pokazania. □

Lemat C.9.2. Operację D można wyrazić jako kombinację $\mathcal{O}(\log N)$ bramek H i T.

Dowód. Bramka D jest opisana macierzą:

$$D = \frac{2}{N}\begin{bmatrix} 1 & 1 & \ldots & 1 \\ 1 & 1 & \ldots & 1 \\ \ldots & & & \\ 1 & 1 & \ldots & 1 \end{bmatrix} - I,$$

czyli $D = 2 \cdot v \cdot v^T - I$, gdzie $v = \frac{1}{\sqrt{N}} \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}$. Macierz D jest postaci Householdera, więc jest unitarna. Faktem jest, że $D = H_n \cdot Z_0 \cdot H_n$, gdzie Z_0 jest macierzą jednostkową z -1 zamiast 1 w komórce $(0,0)$, a H_n jest macierzą opisującą bramkę H na każdym z kubitów. Składa się ona z n bramek H. Do konstrukcji bramki Z_0 również wystarcza $\mathcal{O}(n)$ dwuwejściowych bramek logicznych. Zatem całościowo potrzebujemy $\mathcal{O}(n) = \mathcal{O}(\log N)$ bramek, co było do pokazania. □

C.10 Opisać algorytm faktoryzacji liczby N za pomocą obliczania rzędu elementu modulo N (klasyczna część algorytmu Shora).

Dla zadanej liczby A chcemy znaleźć jej rozkład na czynniki pierwsze, umiejąc wyznaczyć rząd dowolnego elementu z $\mathbb{Z}_A^*$. Pomysł jest następujący:

1. Wylosuj $a \in \mathbb{Z}_A^*$ o parzystym rzędzie r.
2. Jeśli $d = \gcd(A,\ a^{\frac{r}{2}} - 1)$ jest nietrywialnym dzielnikiem, zwróć d.
3. Jeśli $d' = \gcd(A,\ a^{\frac{r}{2}} + 1)$ jest nietrywialnym dzielnikiem, zwróć d'.
4. Wpp powtórz losowanie.

Ponieważ $A = (a^{\frac{r}{2}} + 1)(a^{\frac{r}{2}} - 1)$, to z dużym prawdopodobieństem d jest nietrywialnym dzielnikiem A.

Jak trudno jest wylosować odpowiednie a?

Twierdzenie C.10.1. Niech N będzie nieparzystą liczbą złożoną, nie potęgą liczby pierwszej. Jeśli r jest rzędem $a \in \{0, \ldots, N-1\}$, to prawdopodobieństwo, że r jest nieparzyste lub $a^{\frac{r}{2}} = -1$ nie przekracza $\frac{1}{2}$.

Dowód. Niech $N = p_1^{\alpha_1} \cdot \ldots \cdot p_s^{\alpha_s}$, gdzie $s \geq 2$ i $2 \nmid p_i$. Z Chińskiego Twierdzenia o Resztach wynika, że losowanie a jest równoważne niezależnemu losowaniu s reszt a_i modulo $p_i^{\alpha_i}$. Oznaczmy przez r_i rząd a_i modulo $p_i^{\alpha_i}$ oraz niech $r = \mathrm{lcm}(r_1, \ldots, r_s)$.

Żeby r było nieparzyste, wszystkie $r_1, \ldots, r_s$ muszą być nieparzyste. Inaczej byłoby takie i, dla którego $2r_i \mid r$, więc też $a^{r_i} \mid a^{r/2}$. Zatem $a^{r/2} = 1 \pmod{p_i^{\alpha_i}}$, co jest sprzeczne z tym, że $a^{r/2} = -1 \mod N$. Żeby $a^{r/2} = -1 \pmod{N}$, musi być spełnione $v(r_1) = \ldots = v(r_s)$, gdzie $v(r_i)$ oznacza liczbę dwójek w rozkładzie r_i. Jest to też warunek konieczny, żeby r było nieparzyste.

Jakie jest prawdopodobieństwo, że tak będzie?

Korzystamy z tego, że $G_i = \mathbb{Z}^*_{p_i^{\alpha_i}}$ jest grupą cykliczną. Niech G_i ma generator g i rząd $t = \varphi(p_i^{\alpha_i} - 1)$. Zbiory $G_i' = \{g^1, g^3, g^5, \ldots\}$ i $G_i'' = \{1, g^2, g^4, \ldots\}$ są równoliczne, niezależne i $G_i = G_i' \cup G_i''$. Rząd każdego elementu z G_i' ma tyle samo dwójek w rozkładzie co t, a rząd elementu z G_i'' ma ich mniej. Wynika z tego, że losując element z G_i, mamy szansę co najmniej $\frac{1}{2}$ nie trafić w element rzędu r taki, że $v(r) = q$ dla ustalonego q.

Prawdopodobieństwo, że dla wylosowanych $a_1, \ldots, a_s$ zajdzie $v(r_1) = \ldots = v(r_s)$, wynosi co najwyżej $\frac{1}{2^{s-1}} \leq \frac{1}{2}$. □