Kryptografia i bezpieczeństwo systemów informatycznych

Andrzej M. Borzyszkowski

Instytut Informatyki
Uniwersytet Gdański

sem. letni 2025/2026

inf.ug.edu.pl/~amb/

Teoria liczb:

źródło funkcji jednokierunkowych

podstawa kryptografii asymetrycznej

Dlaczego teoria liczb?

- Podstawą kryptografii są funkcje jednokierunkowe
  – DES i inne szyfry blokowe są praktyczną implementacją
  – ale nie ma żadnych dowodów jednokierunkowości
- Problemy z teorii liczb
  – znajdywanie dzielników
  – logarytm dyskretny
  – pierwiastek kwadratowy
  – są dobrze znane i istnieje uzasadnione podejrzenie, że są to funkcje jednokierunkowe
- Kryptografia asymetryczna opiera się w całości na algorytmach teorioliczbowych

Arytmetyka

- Podstawy liczenia
  – układ pozycyjny (schemat Hornera):
  $wart_b(Napis\ Cyfra) = wart_b(Napis) \cdot b + wart_b(Cyfra)$
- Zamiana podstaw liczenia
  – $x = b \cdot x/b + x\%b$ (dzielenie całkowite i reszta)
  – czyli $zapis_b(x) = zapis_b(x/b)\, zapis_b(x\%b)$
  (zapis ilorazu i cyfra $[0 \ldots b-1]$ oznaczająca resztę na końcu)
  – części ułamkowe też można zapisywać
- Liczba cyfr: jeśli $k =$ długość$(zapis_b(x))$, to $b^{(k-1)} \leq x < b^k$
  – czyli $k \approx \log_b(x)$, różne podstawy tylko zmieniają stałą
  – większa podstawa ma mniej cyfr ale zapis każdej cyfr wymaga odpowiednio więcej bitów

Działania arytmetyczne

- Dodawanie: złożoność = liczba cyfr większej liczby
- Mnożenie (algorytm szkolny): mnoży się każdą parę cyfr, dodatkowo występuje dodawanie
  – złożoność = iloczyn liczb cyfr
  – istnieją lepsze algorytmy (ostatnie osiągnięcia $\ell \cdot \log \ell$, ℓ–długość)
  – długość($m \cdot n$)=długość(m)+długość(n) lub o 1 mniej
  – przykład: długość($n!$) $\leq n \cdot$ długość(n),
  obliczenie silni: n mnożeń liczb o długościach długość(n) oraz długość($n!$) czyli $(n \cdot \log n)^2$
- Dzielenie: podobna złożoność
- Zamiana podstawy liczenia (np. z dwójkowego na $\approx 2^\ell$) $(\log n)/\ell$ dzieleń, każde ma złożoność $(\log n) \cdot \ell$, razem $(\log n)^2$

Dzielniki

- $a|b$ oznacza, że a dzieli b, tzn. $\exists x. a \cdot x = b$
  – a jest dzielnikiem, b jest wielokrotnością, można czasami żądać, by $b \neq 0$
  – oczywiście $a|0$, $a|a$, $1|a$, jeśli $a|b$ oraz $b|c$ to $a|c$, jeśli $a|b$ oraz $a|c$, to $a|(\beta \cdot b + \gamma \cdot c)$ – kombinacje liniowe
- $p > 1$ jest liczbą pierwszą gdy jedynymi dzielnikami są 1 i p
  – fakt: liczba liczb pierwszych $< m \approx m/(\ln m)$
  np. liczb pierwszych dokładnie 100 cyfrowych jest $\geq 10^{97}$
  – fakt: każda liczba naturalna ma jednoznaczny rozkład na czynniki pierwsze

Algorytm Euklidesa

- $NWD(a, b) = c$ jeśli $c|a$, $c|b$ oraz c jest największe j.w.
  – def: jeśli $NWD(a, b) = 1$ to a i b są względnie pierwsze
  – fakt: jeśli $NWD(a, b) = 1$ to w ich rozkładach na czynniki pierwsze występują zupełnie inne liczby
- Niech $a = \beta \cdot b + r$, wówczas dzielniki (a, b) oraz (b, r) są te same
  – w szczególności $NWD(a, b) = NWD(b, r)$
  – zaczynając od $a \geq b \geq 0$ otrzymujemy algorytm Euklidesa, koniec jest dla $NWD(a, 0) = a$
  – np. w języku python

```
def nwd(a,b):
while b:
  a,b=b,a%b
  return a
```


Algorytm Euklidesa, złożoność, przykład

- Fakt: liczba kroków jest rzędu $\log a$
  – dw.: po dwóch krokach wielkość zadania jest $<$ połowy, $a\%b < a/2$ (najgorszy przypadek to „złoty podział": $a = b + r$ ma te same proporcje co $b = r + s$)
- Złożoność algorytmu: liczba kroków · złożoność dzieleń czyli $(\log a)^3$ (dokładniejsza analiza pokazuje $(\log a)^2$, bo liczby są szybko coraz mniejsze)

nwd(987654321,6543210)

987654321	6543210
6543210	6172821
6172821	370389
370389	246597
246597	123792
123792	122805
122805	987
987	417
417	153
153	111
111	742
42	27
27	15
15	12
12	3

Algorytm Euklidesa, rozszerzony

- Tw.: jeśli $NWD(a,b) = c$, to $\exists \alpha, \beta . \alpha \cdot a + \beta \cdot b = c$
- Dw.: każda reszta w algorytmie jest kombinacją liniową argumentów, $NWD(a,b)$ jest ostatnią niezerową resztą.
- Wniosek z dowodu: współczynniki można obliczyć efektywnie podczas obliczania NWD (rozszerzony algorytm Euklidesa)

```
#!/usr/bin/env python
#Tadeusz Andrzej Kadlubowski, marzec 2008
a,b=map(int,file("dane.txt").readlines()[:2])
p,q,r,s = 1,0,0,1
while b: a,b,p,q,r,s = b,a%b,r,s,p-a/b*r,q-a/b*s
file("nwd.txt","w").write("%d\n%d\n%d\n"%(a,p,q))
```

- dw.: jeśli A i B są pierwotnymi wartościami a i b to w każdym przebiegu pętli zachodzi $p \cdot A + q \cdot B = a$ oraz $r \cdot A + s \cdot B = b$.

Kongruencje (arytmetyka modularna)

- $a = b \mod n$ jeśli $n | a - b$ ($n \neq 0$, w praktyce $n > 0$)
  – czyli $\exists \lambda . a = b + \lambda \cdot n$
- Relacja równoważności (zwrotność, symetria, przechodniość)
  – $a \equiv a$, $a \equiv b$ implikuje $b \equiv a$, $a \equiv b$, $b \equiv c$ implikuje $a \equiv c$
  – reprezentantami są $0, 1, \ldots, n-1$, oznaczenie $\mathbb{Z}_n = \{0, \ldots, n-1\}$
  – dla każdej liczby istnieje reszta, $a = r \mod n$, $r \in \mathbb{Z}_n$
- Można wykonywać działania dodawania i mnożenia
  – tzn. $a = b \mod n$ oraz $c = d \mod n$ implikuje $a + c = b + d \mod n$, $a \cdot c = b \cdot d \mod n$
- Tw.: jeśli $ab = ac \mod n$ oraz $NWD(a,n) = 1$, to $b = c \mod n$ (tzn. dzielenie ma sens)
- dw. $\exists \alpha, \beta . \alpha \cdot a + \beta \cdot n = 1$ czyli $\alpha \cdot a \cdot (b-c) + \beta \cdot n \cdot (b-c) = b - c$, ponieważ $n | \beta \cdot n \cdot (b-c)$ i $n | \alpha \cdot a \cdot (b-c)$ (założenie) więc $n | b - c$
  – np. $5 \cdot x = 7 = 18 = 29 = 40 = 5 \cdot 8 \mod 11$ i $NWD(5,11) = 1$ więc $x = 8 \mod 11$
  – ale $4 \cdot 4 = 4 \cdot 1 \mod 12$, a jednak $4 \neq 1 \mod 12$

Złożoność działań w arytmetyce modularnej

- Zasada: najpierw redukcja modulo, potem działanie
  – $a \cdot b \mod N = ((a \mod N) \cdot (b \mod N)) \mod N$
  – złożoność: $\log a + \log b + \log N$ (obliczenie reszt mod N) $+ (\log N)^2$ (obliczenia dla reszt)
- Przykład
  – $10986576198654210 43 \cdot 87397655445287387347 \mod 100$ $= 43 \cdot 47 = 2021 = 21 \mod 100$

Szybkie potęgowanie

Obliczyć $2^{100} \mod 123$

b	n	r
2	100	1
4	50	1
16	25	1
16	24	16
16^2=10	12	16
100	6	16
100^2=37	3	16
37	2	16·37
37^2=16	1	100
16	0	100·16
		1

```
r=1;
while(n) /* inv:r*b^n mod m */{
 if(n%2){n--; r=(r*b)%m;}
 else{n=n/2; b=(b*b)%m;}
}
```

```
def potm(b,n,m):
if n==0: return 1
if n%2==1: return (b*potm(b,n-1,m))%m
else: return potm((b*b)%m,n/2,m)%m
```

– liczba kroków ≈ długość wykładnika
– przy każdej okazji dokonuje się redukcji modulo
– nigdy nie operuje się na liczbach większych niż kwadrat modułu

„Odwrotności" liczb w arytmetyce modularnej

- $5 \cdot x = 7 \mod 11$ można rozwiązać znajdując odwrotność 5:
  – $9 \cdot 5 = 1 \mod 11$ więc $x = 9 \cdot 7 = 8 \mod 11$
- Jeśli $NWD(a, n) = 1$, to $\exists \alpha, \beta. \alpha \cdot a + \beta \cdot n = 1$, czyli $\alpha \cdot a = 1 \mod n$
  – $11111 \cdot x = 4 \mod 12345$, rozwiązanie $x = 2471 \cdot 4 = 9884$
  – ponieważ $2471 \cdot 11111 = 1 \mod 12345$ zostało obliczone
  w rozszerzonym algorytmie Euklidesa
- Jeśli $\exists \alpha. \alpha \cdot a = 1 \mod n$, to $\exists \beta. \alpha \cdot a + \beta \cdot n = 1$ czyli
  $NWD(a, n) = 1$
- Problem: rozwiązać równanie $a \cdot x = b \mod n$
  – jeśli istnieje rozwiązanie, to $NWD(a, n)|b$
  – jeśli $NWD(a, b) = c$, $c|b$, to można rozwiązać równoważny problem
  $(a/c) \cdot x = b/c \mod n/c$
  – np. $12 \cdot x = 21 \mod 39$ daje $4 \cdot x = 7 \mod 13$, $x = 5$
  w oryginalnym problemie są jeszcze rozwiązania 18 i 31