ALGORYTMY I STRUKTURY DANYCH
WYKŁAD VII (materiały pomocnicze)

Algorytmy w teorii liczb, zastosowania

Polsko Japońska Wyższa Szkoła Technik Komputerowych

Warszawa, 11 stycznia 2009

Plan wykładu:

- podstawy:
  - złożoność algorytmów teorioliczbowych,
  - standardowe algorytmy arytmetyki dużych liczb całkowitych,
  - szybki algorytm mnożenia,
  - szybki algorytm potęgowania modularnego,
  - szybki algorytm wyznaczania multiplikatywnej odwrotności,
- problemy teorioliczbowe:
  - badanie pierwszości liczb,
- zastosowania:
  - system kryptograficzny z kluczem publicznym RSA.

Podstawy

(złożoności algorytmów teorioliczbowych)

Podstawy – złożoność algorytmów teorioliczbowych

Definicja. *Długością liczby całkowitej* a nazywamy liczbę cyfr w przyjętym systemie liczbowym o podstawie x niezbędnych do zapisania liczby $a \neq 0$, tj. $\lfloor \log_x |a| \rfloor + 1$ i oznaczamy przez $|a|_x$.

Uwaga! W dalszej części wykładu będziemy domyślnie przyjmowali system dziesiętny reprezentacji liczb i dla uproszczenia $|a|_{10} = |a|$.

Definicja. *Bezwzględną złożonością* czasową/pamięciową algorytmu teorioliczbowego nazywamy złożoność czasową/pamięciową tego algorytmu wyrażoną jako funkcję wartości argumentów (tj. liczb) wejściowych.

Definicja. *Względną złożonością* czasową/pamięciową algorytmu teorioliczbowego nazywamy złożoność czasową/pamięciową tego algorytmu wyrażoną jako funkcję długości argumentów (tj. liczb) wejściowych.

Uwaga! W dalszej części wykładu złożoność algorytmów teorioliczbowych będziemy domyślnie analizowali w wariancie względnym.

Podstawy – złożoność algorytmów teorioliczbowych

Wniosek. Niech Alg będzie algorytmem teorioliczbowym, dla którego argumentem wejściowym jest liczba a długości n. Wtedy:

- jeżeli $T(Alg, a) = O(c^a)$, to $T(Alg, n) = O\left(Alg, c^{10^n}\right)$,
- jeżeli $T(Alg, a) = O(c \cdot a)$, to $T(Alg, n) = O(Alg, c \cdot 10^n)$,
- jeżeli $T(Alg, a) = O(\log_c a)$, to $T(Alg, n) = O(n \cdot \log_c a)$,

gdzie $c > 1$.

Przykład. Szkolny algorytm weryfikacji pierwszości liczby naturalnej a przez dzielenie do skutku aż do wartości $\lfloor\sqrt{a}\rfloor$ ma złożoność czasową bezwzględną rzędu $O\left(\sqrt{a}\right)$ i złożoność czasową względną rzędu $O(10^n)$, gdzie $n = \left|\sqrt{a}\right| = \left\lceil\frac{|a|}{2}\right\rceil$.

Podstawy

(standardowe algorytmy arytmetyki dużych liczb całkowitych)

Podstawy – standardowe algorytmy arytmetyki dużych liczb całkowitych

Założenie. Wprowadzamy dynamiczny typ danych **l_int**, reprezentujący „duże” liczby całkowite zapisane w systemie dziesiętnym, np. 10^6-cyfrowe.

Uwaga! W dalszej części wykładu będziemy pomijali koszt obsługi typu danych l_int.

Fakt. Niech a i b będą liczbami całkowitymi takimi, że $|a| = n$ oraz $|b| = n$, wtedy złożoność czasowa „szkolnej metody” wyznaczenia:

- sumy liczb a i b (tj. $a + b$), mierzona liczbą elementarnych operacji dodawania cyfr dziesiętnych, jest rzędu $O(n)$,
- różnicy liczb a i b (tj. $a - b$), mierzona liczbą elementarnych operacji odejmowania cyfr dziesiętnych, jest rzędu $O(n)$, zakładamy, że $a - b \in \mathbb{N}$,
- iloczynu liczb a i b(tj. $a \cdot b$), mierzona liczbą elementarnych operacji mnożenia cyfr dziesiętnych, jest rzędu $O(n^2)$,
- ilorazu bez reszty liczb a i b (tj. $a \operatorname{div} b$), gdzie $b \neq 0$, mierzona liczbą elementarnych operacji mnożenia cyfr dziesiętnych, jest rzędu $O(n^3)$,
- reszty z dzielenia liczb a i b (tj. $a \bmod b$), gdzie $b > 0$, mierzona liczbą elementarnych operacji mnożenia cyfr dziesiętnych, jest rzędu $O(n^3)$.

Podstawy

(szybki algorytm mnożenia)

Podstawy – szybki algorytm mnożenia

Idea. Niech a i b będą liczbami całkowitymi takimi, że $|a| = |b| = n$. Załóżmy dla uproszczenia, że $n = m = 2^k$, gdzie $k \in \mathbb{N}$ (to założenie może być łatwo spełnione np. przez właściwe dopisanie odpowiedniej liczby „zer"). Niech $a = a_1 \cdot 10^{\frac{n}{2}} + b_2$, $b = b_1 \cdot 10^{\frac{n}{2}} + b_2$, wtedy:

$$\begin{aligned} a \cdot b &= \left(a_1 \cdot 10^{\frac{n}{2}} + a_2\right) \cdot \left(b_1 \cdot 10^{\frac{n}{2}} + b_2\right) \\ &= a_1 \cdot b_1 \cdot 10^n + (a_1 \cdot b_2 + b_1 \cdot a_2) \cdot 10^{\frac{n}{2}} + a_2 \cdot b_2. \end{aligned}$$

Zauważmy, że jeżeli znamy iloczyny $a_1 \cdot b_1 = x$ i $a_2 \cdot b_2 = y$, to:

$$\begin{aligned} a_1 \cdot b_2 + b_1 \cdot a_2 &= (x + a_1 \cdot b_2 + b_1 \cdot a_2 + y) - x - y \\ &= (a_1 \cdot b_1 + a_1 \cdot b_2 + b_1 \cdot a_2 + a_2 \cdot b_2) - x - y \\ &= (a_1 + a_2) \cdot (b_1 + b_2) - x - y. \end{aligned}$$

Ostatecznie

$$a \cdot b = x \cdot 10^n + ((a_1 + a_2) \cdot (b_1 + b_2) - x - y) \cdot 10^{\frac{n}{2}} + y.$$

Podstawy– szybki algorytm mnożenia

Przykład. Niech $a = 12345678$ i $b = 87654321$ i $n = 8$, stąd :

$$\begin{aligned} a &= 1234 \cdot 10^4 + 5678 \\ b &= 8765 \cdot 10^4 + 4321 \end{aligned}$$

oraz

$$\begin{aligned} x &= 1234 \cdot 8765 = 10816010 \\ y &= 5678 \cdot 4321 = 24534638. \end{aligned}$$

Ponieważ $1234 + 5678 = 6912$ i $8765 + 4321 = 13086$, to

$$\begin{aligned} a \cdot b &= x \cdot 10^8 + (6912 \cdot 13086 - x - y) \cdot 10^4 + y \\ &= 10816010 \cdot 10^8 + 55099784 \cdot 10^4 + 24534638 \\ &= 1082152022374638. \end{aligned}$$

Podstawy – szybki algorytm mnożenia

Wniosek. Problem mnożenia dwóch n-cyfrowych liczb całkowitych, gdzie $n = 2^k$ i $k \in \mathbb{N}$, można sprowadzić do problemu mnożenia dwóch $\frac{n}{2}$-cyfrowych liczb całkowitych.

Algorytm Karatsuby. Niech $a = a_1 \cdot 10^{\frac{n}{2}} + a_2$, $b = b_1 \cdot 10^{\frac{n}{2}} + b_2$, gdzie $n = 2^k$ i $k \in \mathbb{N}$:

- jeżeli $n = 1$, wynikiem jest $a \cdot b$,
- w przeciwnym przypadku oblicz rekurencyjnie $x = a_1 \cdot b_1$ oraz $y = a_2 \cdot b_2$,
- oblicz rekurencyjnie $(a_1 + a_2) \cdot (b_1 + b_2) = z$,
- wynikiem jest $x \cdot 10^n + (z - x - y) \cdot 10^{\frac{n}{2}} + y$.

Złożoność czasowa. Niech $T(n)$ będzie liczbą elementarnych operacji mnożenia na cyfrach dziesiętnych jakie wykonuje algorytm Karatsuby dla danych rozmiaru n, wtedy:

$$T(n) = \begin{cases} 1 & \text{dla } n = 1 \\ 3T\left(\frac{n}{2}\right) + O(1) & \text{dla } n > 1 \end{cases}.$$

Na podstawie twierdzenia o rekurencji uniwersalnej $T(n) = \Theta\left(n^{\lg 3}\right) = O\left(n^{1,585}\right)$.

Podstawy – szybki algorytm mnożenia

Pytanie. Jaka jest złożoność pamięciowa algorytmu Karatsuby?

Pytanie. Jak zmieni się złożoność czasowa i pamięciowa algorytmu Karatsuby, jeżeli zamiast dwupodziału liczb n-cyfrowych zastosujemy trójpodział?

Fakt. Algorytm Karatsuby nie jest optymalnym algorytmem dla problemu mnożenia dużych n-cyfrowych liczb całkowitych a i b. Znane są algorytmy o znacznie lepszej złożoności czasowej, np. algorytm oparty na szybkiej transformacie Fouriera, złożoność czasowa $O(n \lg n)$, stąd $a \operatorname{div} b$ oraz $a \bmod b$ można wyznaczyć w czasie $O\left(n^2 \lg n\right)$.

Zadanie (***). Zaimplementować algorytm Karatsuby.

Zadanie (*****). Zaimplementować algorytm Schonhage-Strassena mnożenia liczb.

Podstawy

(szybki algorytm potęgowania modularnego)

Podstawy – szybki algorytm potęgowania modularnego

Zadanie. Niech a, b i c będą liczbami całkowitymi takimi, że $|a| = |b| = |c| = n$ oraz $a, b \leq c$ i $c > 0$. Podaj algorytm wyznaczający $a^b \mod c$.

Rozwiązanie „naiwne”.

```
int ModExp(l_int a,l_int b,l_int c) { // wp:  a,b ≤ c ∧ c > 0
   l_int i=0,s=1;

   while (i<b) { // nz:  s = a^i mod c ∧ s ≤ c
      s=s*a; // s/a ≡ a^i mod c
      i=i+1; // s/a ≡ a^(i−1) mod c ⇒ s ≡ a^i mod c
      s=s mod c; // (x·c + s) = a^i mod c ∧ s ≤ c ⇒
                 // (x·c mod c + s mod c) = a^i mod c ∧ s ≤ c ⇒
                 // s = a^i mod c ∧ s ≤ c
   }

   return s; // wk:  s = a^b mod c ∧ s ≤ c
}
```

Podstawy – szybki algorytm potęgowania modularnego

Złożoność czasowa. Załóżmy, że operacją dominującą są elementarne działania mnożenia, wtedy

$$
\begin{aligned}
T(n) &= O\left(\sum_{i=0}^{10^n} (\text{złożoność mnożenia rezultatu częściowego przez } a)\right) + \\
&\quad O\left(\sum_{i=0}^{10^n} (\text{złożoność dzielenia rezultatu częściowego modulo } c)\right) \\
&= O\left(10^n n \lg n\right) + O\left(10^n n^2 \lg n\right) \\
&= O\left(10^n n^2 \lg n\right).
\end{aligned}
$$

Pytanie. Jaka jest złożoność pamięciowa algorytmu ModExp?

Podstawy – szybki algorytm potęgowania modularnego

Rozwiązanie szybkie.

```
int FastModExp(l_int a,l_int b,l_int c) { // wp:  a,b ≤ c ∧ c > 0
   l_int s=1,p=a,w=b;

   while (w > 0) {
      if (w mod 2==0) {
         p=p*p;
         w=w/2;
         p=p mod c;
      } else {
         s=s*p;
         w=w-1;
         s=s mod c;
      }
   }

   return s; // wk:  s = a^b mod c
}
```

Podstawy – szybki algorytm potęgowania modularnego

Poprawność. Rozważmy formułę $s \cdot p^w \equiv a^b \mod c \ \wedge \ s, p \leq c$, wtedy dla $s = 1$, $p = a$ i $w = b$ zachodzi $1 \cdot a^b \equiv a^b \mod c \ \wedge \ s, p \leq c$, następnie

- dla $w \mod 2 = 0$

  `p=p*p;` // $s \cdot \sqrt{p}^w \equiv a^b \mod c \ \wedge \ s \leq c$
  `w=w/2;` // $s \cdot \sqrt{p}^{2w} \equiv a^b \mod c \ \wedge \ s \leq c \Rightarrow s \cdot p^w \equiv a^b \mod c \ \wedge \ s \leq c$
  `p=p mod c;` // $s \cdot (x \cdot c + p)^w \equiv a^b \mod c \ \wedge \ s, p \leq c$

  ponieważ

$$
\begin{aligned}
s \cdot (x \cdot c + p)^w \mod c \quad & \equiv \quad s \cdot \prod_{i=0}^{w} (x \cdot c + p) \mod c \\
& \equiv \quad s \cdot \prod_{i=0}^{w} ((x \cdot c + p) \mod c) \mod c \\
& \equiv \quad s \cdot \prod_{i=0}^{w} (x \cdot c \mod c + p \mod c) \mod c,
\end{aligned}
$$

  to ostatecznie

$$s \cdot (x \cdot c + p)^w \mod c \equiv s \cdot p^w \mod c \ \wedge \ s, p \leq c \Rightarrow s \cdot p^w \equiv a^b \mod c \ \wedge \ s, p \leq c,$$

Podstawy – szybki algorytm potęgowania modularnego

Poprawność c.d.

- dla $w \bmod 2 = 1$

  `s=s*p;` // $\frac{s}{p} \cdot p^w \equiv a^b \bmod c \ \wedge \ p \leq c$

  `w=w-1;` // $\frac{s}{p} \cdot p^{w+1} \equiv a^b \bmod c \ \wedge \ p \leq c \Rightarrow s \cdot p^w \equiv a^b \bmod c \ \wedge \ p \leq c$

  `s=s mod c;` // $(x \cdot c + s) \cdot p^w \equiv a^b \bmod c \ \wedge \ s, p \leq c$

  ponieważ

  $$(x \cdot c + s) \cdot p^w \bmod c \quad \equiv \quad x \cdot c \cdot p^w \bmod c + s \cdot p^w \bmod c$$

  to ostatecznie

  $$(x \cdot c + s) \cdot p^w \bmod c \equiv s \cdot p^w \bmod c \ \wedge \ s, p \leq c \Rightarrow s \cdot p^w \equiv a^b \bmod c \ \wedge \ s, p \leq c.$$

Stąd po wykonaniu pętli głównej prawdą jest, że $w = 0$ oraz $s \cdot p^0 \equiv a^b \bmod c \ \wedge \ s, p \leq c$, czyli

$$s = a^b \bmod c.$$

Podstawy – szybki algorytm potęgowania modularnego

Pytanie. Jaka jest pesymistyczna liczba iteracji pętli głównej w algorytmie FastModExp wyrażona względem długości zapisu dziesiętnego liczby b?

Złożoność czasowa. Załóżmy, że operacją dominującą są elementarne działania mnożenia, wtedy

$$\begin{aligned} W(n) &= O\left(n \cdot (\text{złożoność mnożenia rezultatu częściowego przez } p)\right) + \\ &\quad O\left(n \cdot (\text{złożoność dzielenia rezultatu częściowego modulo } c)\right) \\ &= O\left(n^2 \lg n\right) + O\left(n^3 \lg n\right) \\ &= O\left(n^3 \lg n\right). \end{aligned}$$

Pytanie. Jaka jest złożoność średnia algorytmu FastModExp wyrażona względem przyjętej wcześniej miary?

Pytanie. Jaka jest złożoność pamięciowa algorytmu FastModExp?

Podstawy

(szybki algorytm wyznaczania multyplikatywnej odwrotności)

Podstawy – szybki algorytm wyznaczania multyplikatywnej odwrotności

Definicja. Liczby całkowite a i b, nazywamy *względnie pierwszymi*, jeżeli $gcd\,(a, b) = 1$.

Przykład. Liczby 5 i 17 są względnie pierwsze, liczby 5 i 24 są względnie pierwsze, liczby 10 i 45 nie są względnie pierwsze, bo $gcd\,(10, 45) = 5$.

Definicja. Niech a i b, będą liczbami całkowitymi względnie pierwszymi, gdzie $b > 1$. Liczbę c nazywamy *multyplikatywną odwrotnością liczby a modulo b,* i oznaczamy przez $\left(a^{-1} \bmod b\right)$, jeżeli zachodzi

$$a \cdot c \equiv 1 \bmod b.$$

Przykład. $7 = \left(5^{-1} \bmod 17\right)$, $5 = \left(5^{-1} \bmod 24\right)$.

Lemat. Dla dowolnej pary liczb względnie pierwszych a i b, gdzie $b > 1$, istnieje dokładnie jedna liczba c będąca multyplikatywną odwrotnością liczby a modulo b.

Algorytm Reciprocal. Niech a i b, będą liczbami całkowitymi względnie pierwszymi (tj. $gcd\,(a, b) = 1$), gdzie $b > 1$. Wyznaczamy, bazując na *algorytmie Euklidesa*, rozwiązanie równania

$$a \cdot c + b \cdot x = 1,$$

dla niewiadomej c oraz x, stąd $c = \left(a^{-1} \bmod b\right)$.

Podstawy – szybki algorytm wyznaczania multyplikatywnej odwrotności

Algorytm Reciprocal – implementacja.

```
(l_int,l_int) Reciprocal(l_int a,l_int b) {
// wp:  a,b ∈ ℤ, b > 1, gcd(a,b) = 1

   l_int c=1, x=0;

   if (b==0) // zachodzi a = 1
      return (c,x); // wk:  a·c + b·x = 1
   else {
      (c,x)=Reciprocal(b,a mod b);
      (c,x)=(x,c-(a div b)*x);

      return (c,x); // wk:  a·c + b·x = 1
   }
}
```

Zadanie (***). Udowodnij przez indukcję poprawność częściową algorytmu Reciprocal.

Podstawy – szybki algorytm wyznaczania multyplikatywnej odwrotności

Algorytm Reciprocal – przykład dla $a = 73$ **i** $b = 37$**.**

a	b	c	x	$a \cdot c + b \cdot x = 1$
73	37	–	–	–
37	36	–	–	–
36	1	–	–	–
1	0	–	–	–
1	0	1	0	$1 \cdot 1 + 0 \cdot 0 = 1$
36	1	0	1	$36 \cdot 0 + 1 \cdot 1 = 1$
37	36	1	-1	$37 \cdot 1 + 36 \cdot (-1) = 1$
73	37	-1	2	$73 \cdot (-1) + 37 \cdot 2 = 1$

Podstawy – szybki algorytm wyznaczania multyplikatywnej odwrotności

Fakt. Liczba wywołań rekurencyjnych algorytmu Reciprocal jest rzędu $O(n)$, gdzie n jest długością liczby b.

Złożoność czasowa. Załóżmy, że operacją dominującą są elementarne działania mnożenia, wtedy

$$\begin{aligned} W(n) &= O\left(n \cdot (\text{złożoność dzielenia } a \bmod b)\right) + \\ &\quad O\left(n \cdot \left(\text{złożoność operacji } c - \left\lfloor \frac{a}{b} \right\rfloor \cdot x\right)\right) \\ &= O\left(n^3 \lg n\right) + O\left(n^3 \lg n\right) \\ &= O\left(n^3 \lg n\right). \end{aligned}$$

Pytanie. Jaka jest złożoność pamięciowa algorytmu Reciprocal?

Zadanie (**). Udowodnij przytoczony powyżej fakt szacujący liczbę wywołań rekurencyjnych algorytmu Recprocal dla ustalonego n.

Problemy teorioliczbowe

(badanie pierwszości liczb)

Problemy teorioliczbowe – badanie pierwszości liczb – gęstość rozmieszczenia l. p.

Twierdzenie Gaussa. *Niech* $\pi(n)$ *oznacza liczbę liczb pierwszych mniejszych bądź równych* n*, wtedy*

$$\lim_{n\to\infty} \frac{\pi(n)}{Li(n)} = 1,$$

gdzie $Li(n) = \int\limits_2^n \frac{dx}{\ln|x|} = \frac{n}{\ln n} + \frac{n}{\ln^2 n} + \frac{2n}{\ln^3 n} + \ldots = \sum\limits_{i=0}^{\infty} \frac{(i-1)!n}{\ln^i n}.$

Wniosek. Dla dostatecznie dużego n zachodzi $\pi(n) \approx \mathrm{Li}(n) \approx \frac{n}{\ln n}$, np. dla $n = 2^{1024}$ dostajemy $\pi(n) \approx 2^{1024}/710 \approx 2^{1024}/2^{10} \approx 2^{1014}$.

Wniosek. Prawdopodobieństwo, że losowo wybrana liczba naturalna jest liczbą pierwszą jest w przybliżeniu równe $\frac{1}{\ln n}$, np. dla $n = 2^{1024}$ wynosi około $\frac{1}{710} \approx 0,0014$.

Hipoteza Riemanna. *Niech* $\pi(n)$ *oznacza liczbę liczb pierwszych mniejszych bądź równych* n*, wtedy*

$$\pi(n) = Li(n) + O\left(\sqrt{n}\ln n\right).$$

Problemy teorioliczbowe – badanie pierwszości liczb – test Fermata

Twierdzenie Fermata. *Jeżeli p jest liczbą pierwszą, to*

$$a^{p-1} \equiv 1 \,(\, mod\ p) \,,$$

dla każdego $a \in \mathbb{Z}_p^+$, gdzie $\mathbb{Z}_p^+ = \{1, 2, \ldots, p-1\}$.

Definicja. Jeżeli p jest liczbą złożoną i $a^{p-1} \equiv 1 \,(\bmod p)$, dla pewnego $a \in \mathbb{Z}_p^+$, to liczba p nazywamy *liczbą pseudopierwszą* przy podstawie a.

Wniosek. Jeżeli istnieje liczba $a \in \mathbb{Z}_p^+$ taka, że $a^{p-1}1 \equiv 1 \,(\bmod p)$, to liczba p jest albo liczbą pierwszą, albo liczbą pseudopierwszą przy podstawie a.

Wniosek. Jeżeli istnieje liczba $a \in \mathbb{Z}_p^+$ taka, że $a^{p-1}1 \not\equiv 1 \,(\bmod p)$, to liczba p jest liczbą złożoną.

Przykład. Niech $a = 2$ oraz:

- $p = 17$, wtedy $2^{17-1} = 65536$ oraz $65536 \equiv 1 \,(\bmod 17)$, zatem p jest liczbą pierwszą albo liczbą pseudopierwszą przy podstawie 2,
- $p = 18$, wtedy $2^{18-1} = 131072$ oraz $131072 \equiv 14 \,(\bmod 18)$, zatem p jest złożoną.

Problemy teorioliczbowe – badanie pierwszości liczb – test Fermata

Test Fermata.

```
bool Fermat(l_int p) { // wp:  p ∈ ℕ ∧ p > 1
   if (FastModExp(2,p-1,p)==1) // sprawdzamy, czy 2^(p-1) ≡ 1 (mod p)
      return TRUE; // wk:  p jest liczbą pierwszą (mamy nadzieję)
   else
      return FALSE; // wk:  p jest liczbą złożoną
}
```

Pytanie. Jaka jest złożoność czasowa algorytmu Fermat?

Fakt. Prawdopodobieństwo tego, że $Fermat\,(p) = TRUE$ i p jest liczbą pseudopierwszą przy podstawie 2, dla $p \approx 2^{1024}$ wynosi co najwyżej 10^{-41}.

Fakt. Istnieje nieskończenie wiele liczba złożonych p, nazywanych *liczbami Carmichaela*, które są pseudopierwsze dla dowolnego $a \in \mathbb{Z}_p^+$, zatem w szczególności dla $a = 2$, np. 561, 1105, 1729, 2465. Liczb tych jest „stosunkowo niewiele". Jedynie 8238 liczb mniejszych od 10^{12} jest liczbami Carmichaela.

Wniosek. Prawdopodobieństwo błędu algorytmu Fermat zależy jedynie od postaci argumentu wejściowego p. W przypadku pesymistycznym wynosi dokładnie 1.

Problemy teorioliczbowe – badanie pierwszości liczb – test Millera-Rabina

Definicja. Liczbę a nazywamy *nietrywialnym pierwiastkiem kwadratowym z 1 modulo* p, jeżeli zachodzi

$$a^2 \equiv 1 \pmod{p}$$

oraz $a \not\equiv 1 \pmod{p}$ i $a \not\equiv (p-1) \pmod{p}$.

Przykład. Ponieważ:

- $5^2 \equiv 1 \pmod{24}$, $5 \not\equiv 1 \pmod{24}$ i $5 \not\equiv 23 \pmod{24}$, to 5 jest nietrywialnym pierwiastkiem kwadratowym modulo 24,
- $7^2 \equiv 1 \pmod{12}$, $7 \not\equiv 1 \pmod{12}$ i $7 \not\equiv 11 \pmod{12}$, to 7 jest nietrywialnym pierwiastkiem kwadratowym modulo 12,
- $7^2 \equiv 1 \pmod{8}$, $7 \not\equiv 1 \pmod{8}$ i $7 \equiv 7 \pmod{8}$, to 7 jest trywialnym pierwiastkiem modulo 12.

Lemat. *Niech $a, b \in \mathbb{Z}$, jeżeli a jest nietrywialnym pierwiastkiem kwadratowym z 1 modulo p, to p jest liczbą złożoną.*

Problemy teorioliczbowe – badanie pierwszości liczb – test Millera-Rabina

Procedura Witness – idea. Rozważmy przystawanie będące podstawą twierdzenia Fermata, tj. $a^{p-1} \equiv 1 \pmod{p}$, gdzie $p > 1$ jest liczbą nieparzystą. Ponieważ $p - 1 = 2^x y$, dla ustalonych $x, y \geq 1$, to

$$a^{p-1} = a^{2^x y} = (a^y)^{2^x}$$

Zatem, jeżeli $a^y \not\equiv 1 \pmod{p}$ oraz dla pewnego $1 \leq i \leq x$ zachodzi $(a^y)^{2^i} \equiv 1 \pmod{p}$, to $(a^y)^{2^{i-1}}$ jest nietrywialnym pierwiastkiem kwadratowym z 1 modulo p. Stąd zgodnie z przedstawionym lematem, liczba p jest liczbą złożoną. W przeciwnym przypadku zachodzą wnioski z twierdzenia Fermata.

Przykład. Niech $p = 1537$, $a = 30$, wtedy $p - 1 = 1536 = 2^9 \cdot 3$, stąd

$$\begin{aligned}(30^3)^{2^9} \bmod 1537 &= \left(30^3 \bmod 1537\right)^{2^9} \bmod 1537 = \left(871^2 \bmod 1537\right)^{2^8} \bmod 1537 \\ &= \left(900^2 \bmod 1537\right)^{2^7} \bmod 1537 = (1 \bmod 1537)^{2^7} \bmod 1537 = 1,\end{aligned}$$

czyli 900 jest nietrywialnym pierwiastkiem kwadratowym z 1 modulo 1537. Zatem 1537 jest liczbą złożoną. Faktycznie, $1537 = 29 \cdot 53$.

Problemy teorioliczbowe – badanie pierwszości liczb – test Millera-Rabina

Procedura Witness.

```
bool Witness(l_int a, l_int p) { // wp: a,p ∈ ℕ, p > 1, p mod 2 = 1
   l_int x=FindX(p), y=FindY(p), // p − 1 = 2^x y
      i=0, s=FastModExp(a,y,p), old; // s = a^y mod p

   while (i<=x) {
      old=s;
      s=(s*s) mod p;
      if ((s==1)&&(old!=1)&&(old!=p-1))
         return TRUE; // pierwiastek nietrywialny,
                      // wk: p jest liczbą złożoną
      i=i+1;
   }

   if (s==1) return FALSE; // wk: p jest liczbą pierwszą (mamy nadzieję)
   else return TRUE; // wk: p jest liczbą złożoną
}
```

Problemy teorioliczbowe – badanie pierwszości liczb – test Millera-Rabina

Złożoność czasowa. Załóżmy, że operacją dominującą są elementarne działania mnożenia, wtedy

$$\begin{aligned} W(n) &= O\left(n^3 \lg n\right) + O\left(n \cdot (\text{złożoność operacji } (s \cdot s) \bmod p)\right) \\ &= O\left(n^3 \lg n\right) + O\left(n \cdot \left(O(n \lg n) + O\left(n^2 \lg n\right)\right)\right) \\ &= O\left(n^3 \lg n\right) + O\left(n^3 \lg n\right) = O\left(n^3 \lg n\right). \end{aligned}$$

Pytanie. Jaka jest złożoność średnia algorytmu Witness wyrażona względem przyjętej wcześniej miary?

Pytanie. Jaka jest złożoność pamięciowa algorytmu Witness?

Definicja. Jeżeli $Witness(a,p) = TRUE$, to liczbę a nazywamy *świadkiem złożoności liczby* p.

Twierdzenie. *Niech p będzie nieparzystą liczbą złożoną. Wtedy liczba świadków złożoności liczby p, tj. liczb $a \in \mathbb{Z}_p^+$ takich, że $Witness(a,p) = TRUE$, wynosi co najmniej $\left\lfloor \frac{n}{2} \right\rfloor$.*

Wniosek. Prawdopodobieństwo tego, że $Witness(a,p) = FALSE$ i p jest liczbą złożoną jest równe co najwyżej $\frac{1}{2}$ i nie zależy od postaci argumentu wejściowego p.

Problemy teorioliczbowe – badanie pierwszości liczb – test Millera-Rabina

Test Millera-Rabina.

```
bool Miller_Rabin(l_int p,l_int t) { // wp:  p,t ∈ ℕ
   l_int i=0;

   while (i<t) {
      if (Witness(Random(2,p-1),p)==TRUE) // losowanie bez powtórzeń
         return FALSE // wk:  p jest liczbą złożoną
      i=i+1;
   }

   return TRUE // wk:  p jest liczbą pierwszą (z dużym prawdopodobieństwem)
}
```

Pytanie. Jaka jest złożoność czasowa algorytmu Miller_Rabin?

Wniosek. Prawdopodobieństwo błędu algorytmu Millera-Rabina zależy jedynie od rezultatu losowania liczb będących argumentami procedury Witness.

Wniosek. Prawdopodobieństwo tego, że $Miller\_Rabin(p,t) = TRUE$ i p jest liczbą złożoną wynosi co najwyżej $\left(\frac{1}{2}\right)^t$.

Problemy teorioliczbowe – badanie pierwszości liczb – algorytm deterministyczny

Fakt. Do 2002 roku problem efektywnej (tj. wielomianowej względem długości liczby wejściowej) weryfikacji pierwszości dowolnej liczby naturalnej znajdował jedynie rozwiązanie w algorytmach niedeterministycznych, których przykładem jest przedstawiony wcześniej test Millera-Rabina.

Fakt. W 2002 roku M. Agrawal, N. Kayal i N. Saxena z Indyjskiego Instytutu Technologii w Kanpur przedstawili pierwszy deterministyczny algorytm AKS, weryfikujący pierwszeństwo dowolnej n-cyfrowej liczby naturalnej ze złożonością działania rzędu wielomianowego, dokładnie $O\left(n^{12+\epsilon}\right)$, gdzie ϵ jest pewną stałą.

Fakt. W 2005 roku C. Pomerance H. Lenstra Jr. przedstawili zmodyfikowaną wersję algorytmu AKS, której złożoność ograniczona jest przez $O\left(n^{6+\epsilon}\right)$, gdzie ϵ jest pewną stałą.

Info. Więcej informacji o algorytmie AKS i sposobach jego implementacji można znaleźć na stronie `http://fatphil.org/maths/AKS/`.

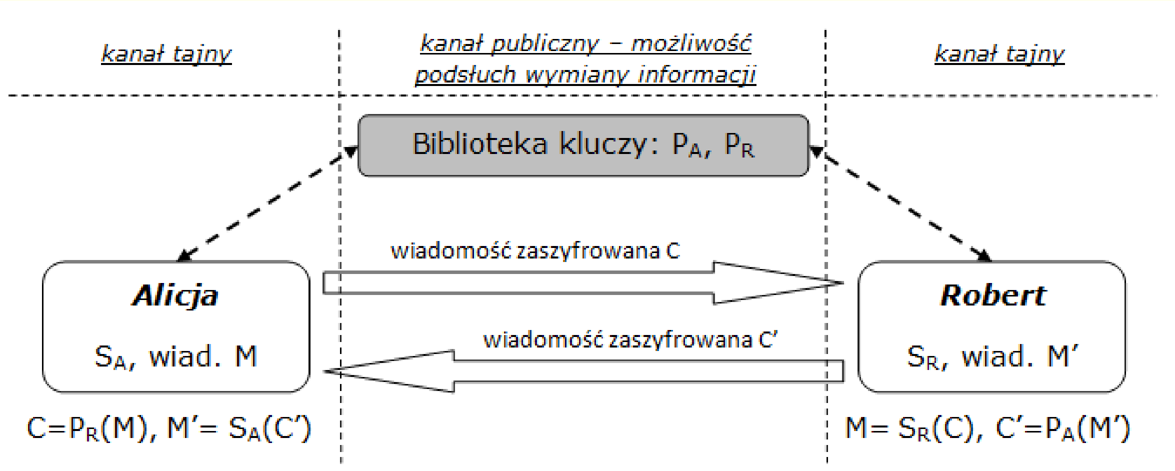
Zastosowania

(system kryptograficzny z kluczem publicznym RSA)

System kryptograficzny z kluczem publiczny RSA – podstawy

Idea kryptosystemów z kluczem publicznym. Strony wiany informacji – Alicja i Robert:

- klucz publiczny/tajny Alicji P_A/S_A, klucz publiczny/tajny Roberta P_R/S_R,
- wiadomość oryginalna (tajna) od Alicji do Roberta i odwrotnie – M i M',
- wiadomość zaszyfrowana (publiczna) od Alicji do Roberta i odwrotnie – C i C'.



System kryptograficzny z kluczem publiczny RSA – podstawy

Idea kryptosystemów z kluczem publicznym w ujęciu matematycznym. Niech $\mathcal{U}$ będzie zbiorem wszystkich możliwych wiadomości, wtedy:

- klucz publiczny P – funkcja $P : \mathcal{U} \to \mathcal{U}$ będąca bijekcją w zbiorze wiadomości $\mathcal{U}$,
- klucz tajny S – funkcja $S : \mathcal{U} \to \mathcal{U}$ będąca funkcją odwrotną do P, tj. $S = P^{-1}$, w zbiorze wiadomości $\mathcal{U}$,
- szyfrowanie wiadomości tajnej $M \in \mathcal{U}$ – obliczenie $C = P(M)$,
- deszyfrowanie wiadomości publicznej $C \in \mathcal{U}$ – obliczenie $M = S(C)$.

Wniosek. Ponieważ klucz publiczny P jest funkcją będącą bijekcją, to istnieje jednoznacznie określona funkcją S będąca kluczem tajnym, stąd dla każdej wiadomości $M \in \mathcal{U}$ zachodzi $S(P(M)) = M = P(S(M))$, tj. kodowanie i odkodowanie odbywa się w sposób jednoznaczny.

Fakt. Bezpieczeństwo kryptosystemów z kluczem publicznym opiera się na ”trudności obliczeniowej” znalezienia funkcji odwrotnej $S = P^{-1}$ do zadanej funkcji P.

System kryptograficzny z kluczem publiczny RSA – działanie

Schemat. Niech $\mathcal{U}$ będzie zbiorem wszystkich możliwych wiadomości, których długość zapisu bitowego nie przekracza l bitów, wtedy:

- wybierz dwie różne liczby pierwsze p i q co najmniej $\left\lfloor \frac{l}{2} \right\rfloor + 1$ bitowe, oblicz $n = p \cdot q$,
- wybierz stosunkowo małą liczbę pierwszą e taką, że $gcd\,(e, (p-1) \cdot (q-1)) = 1$,
- oblicz $d = e^{-1} \mod ((p-1) \cdot (q-1))$,
- klucz publiczny $P = (e, n)$,
- klucz tajny $S = (d, n)$,
- szyfrowanie wiadomości tajnej $M \in \mathcal{U}$ – obliczenie $C = P\,(M) = M^e \mod n$,
- deszyfrowanie wiadomości publicznej $C \in \mathcal{U}$ – obliczenie $M = S\,(C) = C^d \mod n$.

System kryptograficzny z kluczem publiczny RSA – działanie

Przykład. Niech $\mathcal{U}$ będzie zbiorem wszystkich możliwych słów długości co najwyżej 7 bitów, dla alfabetu $\Sigma = \{\mathrm{A,B,C}\}$ i kodowania binarnego znaków A – 0, B – 10, C – 11, zatem:

- wybieramy $p = 13 = 1101_2$, $q = 17 = 10001_2$, wtedy $n = 13 \cdot 17 = 221 = 11011101_2$,
- wybieramy $e = 7 = 111_2$, obliczamy $d = 7^{-1} \mod (12 \cdot 16) = 55 = 110111_2$,
- klucz publiczny $P = (7, 221)$, klucz tajny $S = (55, 221)$,
- szyfrowanie wiadomości tajnej $\mathrm{BACA} \in \mathcal{U}$ – wyznaczamy $\mathrm{BACA} = 100110_2 = 38$, obliczamy
$$C = P\,(\mathrm{BACA}) = 38^7 \mod 221 = 64,$$
- deszyfrowanie wiadomości publicznej $\mathrm{BAAAA} \in \mathcal{U}$ – wyznaczamy $\mathrm{BAAAA} = 100000_2 = 64$, obliczamy
$$M = P\,(\mathrm{BAAAA}) = 64^{55} \mod 221 = 38 = 100110_2 = \mathrm{BACA}.$$

System kryptograficzny z kluczem publiczny RSA – działanie

Schemat z użyciem prezentowanych algorytmów.