Malá Fermatova věta, modulární inverze, Čínská věta o zbytcích, RSA. Analýza algoritmů, složitost, NP-úplnost.

Matematické algoritmy (K611MA)

Jan Přikryl

Ústav aplikované matematiky
ČVUT v Praze, Fakulta dopravní

2. konzultace K611MA
pátek 26. listopadu 2010

verze: 2010-11-29 12:09

Obsah konzultace

Definice

Pierre de Fermat, 1640

Pro $a \in \mathbb{Z}$ a prvočíslo $p \in \mathbb{N}$ takové, že $p \nmid a$ platí

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

a v alternativním tvaru

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

Ve skutečnosti je $a^{\phi(p)} \equiv 1 (\mathrm{mod}\, p)$, kde $\phi(p)$ je takzvaná **Eulerova funkce**.

Malá Fermatova věta je základním stavebním kamenem algoritmu generování šifrovacího klíče asymetrické šifry RSA. Je také nutnou podmínkou pro prvočísla.

Multiplikativní inverze

Definice

Pro $a \in \mathbb{Z}$ a $n \in \mathbb{N}$ je celé číslo x **multiplikativní inverzí**, pokud splňuje podmínku

$$a \cdot x \equiv 1 \pmod{n}. \tag{1}$$

Pro **nejmenší multiplikativní inverzi** platí, že x je nejmenší možnou kladnou multiplikativní inverzí k a a označujeme ji a^{-1}.

Z Malé Fermatovy věty přitom plyne, že

$$a^{-1} \equiv a^{p-2} \pmod{p}. \tag{2}$$

pro $a \in \mathbb{Z}$ a prvočíselná $p \in \mathbb{N}$ taková, že $p \nmid a$.

Multiplikativní inverze
Příklad

Výpočet inverze

Chceme spočítat a^{-1} pro $n = 11$ a $a = -3$. Volíme postupně $x = 1, 2, \ldots,$ první kladné číslo x splňující vztah (1) je $x = 7$: $-3 \cdot 7 \equiv 1 \pmod{11}$.

Výpočet inverze pomocí Malé Fermatovy věty

Použitím Malé Fermatovy věty (2) máme $a^{-1} \equiv (-3)^{11-2} \pmod{11}$, tedy $a^{-1} \equiv -19683 \pmod{11}$ což je to samé, jako $a^{-1} \equiv 7 \pmod{11}$ protože jde o stejnou třídu kongruence.

Zkuste si to nyní sami pro $n = 7$ a $a = 5$.

Obsah konzultace

Čínská věta o zbytcích
(Chinese Reminder Theorem, CRT)

Více vzájemně ekvivalentních tvrzení z algebry a teorie čísel.
Nejstarší zmínka z Číny ve 3. století našeho letopočtu.

Motivace: Jak najít x takové, že

$$x \equiv 2 \pmod 3,$$
$$x \equiv 3 \pmod 5,$$
$$x \equiv 2 \pmod 7?$$

Čínská věta o zbytcích

Postup řešení (1/2)

Zbytkové třídy jsou $[2]_3$, $[3]_5$ a $[2]_7$.

V prvním kroku hledáme nulové a jednotkové zbytkové třídy pro kombinace původních modulů. V našem případě platí

$$\begin{aligned}
\kappa_1 &= 70 \equiv 0 \,(\mathrm{mod}\, 5 \cdot 7) \quad \wedge \quad \kappa_1 = 70 \equiv 1 \,(\mathrm{mod}\, 3),\\
\kappa_2 &= 21 \equiv 0 \,(\mathrm{mod}\, 3 \cdot 7) \quad \wedge \quad \kappa_2 = 21 \equiv 1 \,(\mathrm{mod}\, 5),\\
\kappa_3 &= 15 \equiv 0 \,(\mathrm{mod}\, 3 \cdot 5) \quad \wedge \quad \kappa_3 = 15 \equiv 1 \,(\mathrm{mod}\, 7).
\end{aligned}$$

Čínská věta o zbytcích

Postup řešení (2/2)

Řešením dané soustavy kongruencí je v takovém případě číslo

$$\hat{x} = 2 \cdot 70 + 3 \cdot 21 + 2 \cdot 15 = 233.$$

Minimální hodnota x je dána třídou kongruence modulo $3 \cdot 5 \cdot 7 = 105$, tedy

$$x = 233 \bmod 105 = 23.$$

Čínská věta o zbytcích

Vlastní tvrzení

Nechť $n_1, n_2, \ldots, n_k$ jsou navzájem nesoudělná přirozená čísla, $n_i \geq 2$ pro všechna $i = 1, \ldots, k$. Potom řešení soustavy rovnic

$$\begin{aligned} x &\equiv a_1 \pmod{n_1} \\ x &\equiv a_2 \pmod{n_2} \\ &\vdots \\ x &\equiv a_k \pmod{n_k} \end{aligned}$$

existuje a je určeno jednoznačně v modulo $n = n_1 \cdot n_2 \cdot \ldots \cdot n_k$.

Jak si Sun Tzu ušetří práci

Lehký náznak důkazu

Díky nesoudělnosti existuje ve třídě operací modulo n_i ke každému $N_i = n/n_i$ jeho multiplikativní inverze M_i, tedy

$$M_i \cdot N_i \equiv 1 \pmod{n_i}$$

a platí

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

Ve výše uvedeném případě se zbytkovými třídami $[2]_3$, $[3]_5$ a $[2]_7$ je

$$x = 2 \cdot 2 \cdot 35 + 3 \cdot 1 \cdot 21 + 2 \cdot 1 \cdot 15 = 233.$$

Praktický význam věty

Výpočty modulo velké M lze převést na výpočty modulo menší součinitelé čísla M – zrychlení výpočtu.

Lze generalizovat pro soudělná čísla.

Význam hlavně v šifrovacích systémech.

Problém nůše s vejci

Ilustrace použití CRT

V nůši je v vajec. Pokud z ní odebíráme vejce po dvou, třech a pěti najednou, v nůši nakonec zůstane 1, 2, respektive 4 vejce. Pokud odebíráme vejce po sedmi kusech, v nůši nakonec nezůstane vejce žádné.

Jaká je nejmenší hodnota v pro niž může uvedená situace nastat?

Problém nůše s vejci
Ilustrace použití CRT (2)

Zbytkové třídy jsou $[1]_2$, $[2]_3$, $[4]_5$ a $[0]_7$.

Hledáme řešení soustavy

$$v \equiv 1 \pmod 2$$
$$v \equiv 2 \pmod 3$$
$$v \equiv 4 \pmod 5$$
$$v \equiv 0 \pmod 7$$

Výsledek bude nějaká třída kongruence modulo 210.

Problém nůše s vejci

Ilustrace použití CRT (3)

Pro jednotlivé ekvivalence máme

i	n_i	N_i	M_i	a_i
1	2	105	1	1
2	3	70	1	2
3	5	42	3	4
4	7	30	4	0

$$v = (1 \cdot 1 \cdot 105 + 2 \cdot 1 \cdot 70 + 4 \cdot 3 \cdot 42 + 0 \cdot 4 \cdot 30) \bmod 210$$
$$= (105 + 140 + 504 + 0) \bmod 210 = 749 \bmod 210 = 119$$

Obsah konzultace

Šifrování

Symetrické a asymetrické šifry

Existují dvě základní skupiny šifrovacích algoritmů:

- **Symetrické šifry** u nichž se ten samý klíč používá jak k šifrování, tak i k dešifrování zprávy. Odesílatel i příjemce musí mít k dispozici identické klíče. Příkladem je DES, 3DES, AES.
- **Asymetrické šifry** u nichž se šifruje jiným klíčem, než je klíč určený k dešifrování. Odesílatel po zašifrování již nemá možnost zprávu dešifrovat. Příkladem je RSA (PGP), GnuPG, ElGamal.

Symetrické šifry jsou při stejné délce šifrovacího klíče výrazně bezpečnější, než šifry asymetrické ...

Diffieho-Hellmanova výměna klíčů

Jak se dohodnout na klíči přes nezabezpečený kanál

... ale symetrické šifrování má základní problém: distribuci klíčů.

Diffie a Hellman, 1976

Alice a Bob se na klíči mohou dohodnout přes nezabezpečený komunikační kanál. Je pouze třeba zajistit, aby operace, jež Alice a Bob provádějí, *nebyly výpočetně snadno invertovatelné*.

Diffieho-Hellmanova výměna klíčů

Veřejně známé prvočíslo p a $\alpha \in \{2, \ldots, p-2\}$. Oba jako klíč použijí $\alpha^{xy} \bmod p$ – Alice si vymyslí veliké $x \in \mathbb{N}$ a Bobovi pošle $\alpha^x \bmod p$, Bob pošle Alici $\alpha^y \bmod p$. Alice pak provede $(\alpha^y)^x \bmod p$, Bob obdobně.

Diffieho-Hellmanova výměna klíčů

Vysvětlení

Vzhledem k tomu, že $[a]_p \cdot [b]_p = [ab]_p$ platí pro Alicí přijaté Bobovo $\alpha^y \bmod p$ následující:

$$\alpha^y \bmod p \equiv [\underbrace{\alpha \cdot \alpha \cdots \alpha}_{y-\text{krát}}]_p,$$

a po umocnění na x-tou:

$$\left([\underbrace{\alpha \cdot \alpha \cdots \alpha}_{y-\text{krát}}]_p\right)^x = \underbrace{[\underbrace{\alpha \cdot \alpha \cdots \alpha}_{y-\text{krát}}]_p \cdot [\underbrace{\alpha \cdot \alpha \cdots \alpha}_{y-\text{krát}}]_p \cdots [\underbrace{\alpha \cdot \alpha \cdots \alpha}_{y-\text{krát}}]_p}_{x-\text{krát}} \equiv$$

$$\equiv [\underbrace{\alpha \cdot \alpha \cdots \alpha}_{xy-\text{krát}}]_p.$$

Recipročně to platí i pro Bobem přijaté Aličino $\alpha^x \bmod p$.

Diffieho-Hellmanova výměna klíčů

Příklad

Příklad výměny pro $p = 17$ a $\alpha = 5$

Alice si zvolí $x = 1039$.
Bob si zvolí $y = 1271$.

Po nezašifrovaném spojení pošle Alice Bobovi $5^{1039} \bmod 17 = 7$ a Bob pošle Alici $5^{1271} \bmod 17 = 10$.

Bob si spočte svůj klíč jako $7^{1271} \bmod 17 = 12$, Alice jako $10^{1039} \bmod 17 = 12$.

Ve skutečnosti budou p, α, x, y mnohem větší čísla (proč)?

Modulární mocnění

Výpočet $c \equiv b^r \pmod{n}$

Neefektivně lze opakovaným násobením a redukcí:

$$c = \underbrace{b\,[\,b\,[\ldots[\,b \cdot b}_{r-\text{krát}} \mod n\,]\ldots]\,\mod n\,]\,\mod n$$

Opakovaný kvadrát

Efektivní algoritmus pro $b, r \in \mathbb{N}$ je následující

1. Nechť $r = \sum_{j=0}^{k} a_j \cdot 2^j, a_j \in \{0, 1\}$
2. Inicializujeme $c = 1 + a_0 \cdot (b - 1)$ a $b_0 = b$
3. Opakujeme pro $j = 1 \ldots k$:
   1. Spočteme $b_j = b_{j-1}^2 \mod n$
   2. Pokud je $a_j > 0$, přepíšeme $c \leftarrow c \cdot b_j \mod n$
4. Výsledkem je $c \equiv b^r \pmod{n}$

Modulární mocnění

Příklad

Spočtěte $c = 3^{17} \bmod 7$

Nejprve rozložíme $r = 17 = 10001_{\mathrm{b}}$. Je $a_0 = 1$ a proto prvotní hodnota $c = b = 3$ a $b_0 = 3$. Potom

$$b_1 = 3^2 \bmod 7 = 9 \bmod 7 = 2, a_1 = 0,$$
$$b_2 = 2^2 \bmod 7 = 4 \bmod 7 = 4, a_2 = 0,$$
$$b_3 = 4^2 \bmod 7 = 16 \bmod 7 = 2, a_3 = 0,$$
$$b_4 = 2^2 \bmod 7 = 4 \bmod 7 = 4, a_4 = 1.$$

Nyní přepočteme $c = 3 \cdot 4 \bmod 7 = 12 \bmod 7 = 5$. Další binární cifry už v r nejsou, výsledkem je proto $c = 5$.

Kontrola: $3^{17} = 129140163 \bmod 7 = 5$.

Šifrování veřejným klíčem

Myšlenka RSA

RSA vychází z předpokladu, že faktorizace součinu prvočísel p a q je časově náročná – všichni proto mohou znát šifrovací klíč $n = p \cdot q$, ale nepomůže jim to ke zjištění dešifrovacího klíče, založeného na p a q.

V praxi je klíč $n = p \cdot q \in \{0,1\}^{1024}$ až $\{0,1\}^{4096}$.

Největší faktorizovaný RSA klíč je RSA-768 ($n = \{0,1\}^{768}$, 232 dekadických číslic) v roce 2009 za 2,5 roku na síti několika set pracovních stanic.

Ale pozor: V květnu 2007 padlo $M_{1039} = 2^{1039} - 1$ za 11 měsíců v laboratořích EPFL, Uni Bonn a NTT.

Algoritmy šifrování veřejným klíčem

Prerekvizity

Většina algoritmů (a RSA rozhodně) staví na několika algebraických postupech:

- Modulární mocnění
- Modulární inverze $a^{-1} \cdot a \equiv 1 \pmod{n}$
- Čínská věta o zbytcích
- Eulerova funkce

Algoritmy šifrování veřejným klíčem

Prerekvizity

Většina algoritmů (a RSA rozhodně) staví na několika algebraických postupech:

- Modulární mocnění
- Modulární inverze $a^{-1} \cdot a \equiv 1 \pmod{n}$
- Čínská věta o zbytcích
- Eulerova funkce

Eulerova funkce $\phi(n)$

Rozšíření Malé Fermatovy věty

Leonhard Euler generalizoval Malou Fermatovu větu na

$$a^{\phi(n)} \equiv 1 \pmod{n},$$

kde $\phi(n)$ je již zmíněná **Eulerova funkce**:

- někdy se setkáte s názvem totient
- udává počet přirozených čísel $1 \leq x \leq n$, jež jsou s n nesoudělná
- pro prvočísla $\phi(p) = p - 1$

Pro nesoudělná x a y platí

$$\phi(x \cdot y) = \phi(x) \cdot \phi(y)$$

a pro prvočísla p, q také

$$\phi(p)\phi(q) = (p-1)(q-1).$$

Algoritmus RSA

Generování veřejného a soukromého klíče

Přípravná fáze:

1. Zvolíme nepříliš si blízká prvočísla p a q.
2. Spočteme **modul** šifrovací a dešifrovací transformace, $n = p \cdot q$.
3. Vypočteme Eulerovu funkci pro n, $\phi(n) = (p-1)(q-1)$.
4. Zvolíme **šifrovací exponent** e takový, že $1 < e < \phi(n)$ a $\gcd(e, n) = 1$.
5. Dopočteme **dešifrovací exponent** d tak, aby d bylo multiplikativní inverzí k e modulo $\phi(n)$, $d \cdot e \equiv 1 \pmod{\phi(n)}$.

Veřejný klíč pro zašifrování zprávy je (n, e), **soukromý klíč** pro dešifrování je (n, d).

Algoritmus RSA
Jak to funguje

Princip přenosu zprávy X je primitivní:

Šifrování

Po lince přenášíme šifrovaný text c, jenž vznikne jako

$$c = X^e \bmod n.$$

Dešifrování

Příjemce si z přijatého šifrovaného textu spočítá původní zprávu jako

$$X = c^d \bmod n.$$

Trik celého postupu spočívá v tom, že z pouhé znalosti (n, e) nelze v rozumném čase určit d.

Algoritmus RSA
Důkaz (1/4)

Obdržíme dešifrováním opravdu původní text?

Při dešifrování $c \equiv X^e \pmod{n}$ máme

$$c^d \equiv (X^e)^d \equiv X^{ed} \pmod{n} \equiv X^{ed} \pmod{pq},$$

a $\gcd(p, q) = 1$, což připomíná řešení CRT o dvou kongruencích.

Prozkoumáme vlastnosti $c^d \equiv X^{ed} \pmod{p}$ a $c^d \equiv X^{ed} \pmod{q}$ a zobecníme je na operace modulo n.

Algoritmus RSA
Důkaz (2/4)

Z definice součinu ed v algoritmu RSA plyne

$$ed \equiv 1 \,(\mathrm{mod}\, \phi(n)) \Rightarrow \exists f \in \mathbb{Z} : ed = 1 + f(p-1)(q-1),$$

což můžeme dále upravit na

$$ed = 1 + f(p-1)(q-1) = 1 + g(q-1) = 1 + h(p-1)$$

a tedy

$$ed \equiv 1 \,(\mathrm{mod}\, \phi(n)) \equiv 1 \,(\mathrm{mod}\, \phi(q)) \equiv 1 \,(\mathrm{mod}\, \phi(p)).$$

Algoritmus RSA
Důkaz (3/4)

Dokazujeme stále pro p a q odděleně:

Pro $p \nmid X$ je podle Malé Fermatovy věty $X^{p-1} \equiv 1 \pmod{p}$ a tedy

$$X^{ed} = X^{1+h(p-1)} = X^{h(p-1)}X = \left(X^{(p-1)}\right)^h X \equiv 1^h X \equiv X \pmod{p}.$$

Pro $p|X$ je

$$X^{ed} \equiv 0^{ed} \pmod{p} \equiv X \pmod{p}$$

To samé platí pro q a tedy

$$X^{ed} \equiv X \pmod{p}$$
$$X^{ed} \equiv X \pmod{q}$$

Algoritmus RSA
Důkaz (4/4)

Jedním z důsledků CRT je pro nesoudělná x a y ekvivalence

$$\begin{aligned} a &\equiv b \pmod{x} \\ a &\equiv b \pmod{y} \end{aligned} \quad \Leftrightarrow \quad a \equiv b \pmod{xy}.$$

Proto také z

$$X^{ed} \equiv X \pmod{p}$$
$$X^{ed} \equiv X \pmod{q}$$

plyne

$$X^{ed} \equiv X \pmod{pq}.$$

RSA pomocí CRT

Urychlení dešifrování (1/3)

Jak modul n, tak i dešifrovací exponent d jsou hodně velká čísla, a proces dešifrování

$$X \equiv c^d \pmod{n}$$

trvá dlouho.

K urychlení dešifrovací transformace lze použít rozklad na výpočet s menšími moduly pomocí Čínské věty o zbytcích.

RSA pomocí CRT
Urychlení dešifrování (2/3)

Pro $n = pq$ použijeme již jednou provedený trik

$$X \equiv X_p \pmod{p} \equiv c^{d_p} \pmod{p}$$
$$X \equiv X_q \pmod{q} \equiv c^{d_q} \pmod{q}$$

přičemž pro p

$$d_p \equiv d \pmod{\phi(p)} \Leftrightarrow d = d_p + j(p-1)$$

a tedy

$$c^d \equiv c^{d_p + j(p-1)} \pmod{p} \equiv c^{d_p} 1^j \pmod{p} \equiv c^{d_p} \pmod{p}$$

Pro q je to obdobně.

RSA pomocí CRT
Urychlení dešifrování (3/3)

Zpráva X je tedy řešením soustavy dvou kongruencí sestavených pro c:

$$X \equiv c^{d_p} \pmod{p},$$
$$X \equiv c^{d_q} \pmod{q}.$$

Řešením je

$$X = \left[c^{d_p} M_p q + c^{d_q} M_q p \right] \bmod pq,$$

kde $M_p = q^{-1} \bmod p$ a $M_q = p^{-1} \bmod q$.

Algoritmus RSA

Prolomení při nevhodné volbě p a q

Pokud zvolím p a q nevhodně (blízko sebe, příliš malá, atd.), útočník využije znalosti (n, e):

1. Faktorizuje n na p a q.
2. Vypočte Eulerovu funkci pro n, $\phi(n) = (p-1)(q-1)$.
3. Dopočte **dešifrovací exponent** d tak, aby $d \cdot e \equiv 1 \pmod{\phi(n)}$.

Můj **soukromý klíč** pro dešifrování (n, d) v ten okamžik zná i útočník a může moje zprávy dešifrovat.

Obsah konzultace

Algoritmy a programy

Co je co

Algoritmus:

- myšlenka řešení nějakého problému
- konečný počet kroků řešení
- vyjadřujeme nejčastěji slovně nebo pseudokódem

Program:

- implementace algoritmu ve zvoleném programovacím jazyce

Bez algoritmu nelze napsat program.

Analýza složitosti algoritmů

Proč nás to vlastně zajímá

Ideální kombinace: efektivní algoritmus + efektivní implementace

Analýza algoritmu:

- poskytne **předpověď výkonnosti** algoritmu
- je **vhodnější než experimenty**
- umožní výběr **vhodné varianty** řešení

Asymptotická složitost **paměťová** × složitost **časová**

Analýza složitosti algoritmů (2)

Předpověď chování

Analýza efektivity: Identifikace efektivních a neefektivních částí algoritmu umožní soustředit se na ty části, jejichž úprava přinese nejvyšší nárůst výkonu.

Předpověď výkonnosti programu

Velké projekty potřebují apriorní odhad výkonnosti pro daný hardware – je třeba učinit odhad bez znalosti detailů programového kódu.

Identifikace úzkých míst a jejich vhodné ošetření ještě před vlastním naprogramováním.

Analýza složitosti algoritmů (2)

Předpověď chování

Analýza efektivity: Identifikace efektivních a neefektivních částí algoritmu umožní soustředit se na ty části, jejichž úprava přinese nejvyšší nárůst výkonu.

Předpověď výkonnosti programu

Velké projekty potřebují apriorní odhad výkonnosti pro daný hardware – je třeba učinit odhad bez znalosti detailů programového kódu.
Identifikace úzkých míst a jejich vhodné ošetření ještě před vlastním naprogramováním.

Analýza složitosti algoritmů (3)

Ověření vlastností

Vhodnější než experimenty

Experimenty ověří chování pouze ve vybraných krizových případech – místo „dokázali jsme, že daný algoritmus funguje správně“ lze pouze tvrdit: „nenalezli jsme způsob, jak prokázat, že algoritmus je špatně“.

Záruky funkčnosti poskytuje jedině **formální analýza**.

Výběr vhodné varianty

Ne vždy je nejvhodnější varianta ta, jenž je v počtu instrukcí nejefektivnější a tedy nejrychlejší – *např. implementace pro jednočipový počítač × pracovní stanice.*

Analýza složitosti algoritmů (3)

Ověření vlastností

Vhodnější než experimenty

Experimenty ověří chování pouze ve vybraných krizových případech – místo „dokázali jsme, že daný algoritmus funguje správně“ lze pouze tvrdit: „nenalezli jsme způsob, jak prokázat, že algoritmus je špatně“.
Záruky funkčnosti poskytuje jedině **formální analýza**.

Výběr vhodné varianty

Ne vždy je nejvhodnější varianta ta, jenž je v počtu instrukcí nejefektivnější a tedy nejrychlejší – *např. implementace pro jednočipový počítač* × *pracovní stanice*.

Obsah konzultace

Ilustrační příklad
Fibonacciho posloupnost

Poprvé popsána italským matematikem Leonardem z Pisy, známým také jako Fibonacci (1202).

Růstu populace králíků za poněkud idealizovaných podmínek.

Číslo $F(n)$ popisuje velikost populace po n měsících, předpokládáme-li, že

- první měsíc se narodí jediný pár,
- nově narozené páry jsou produktivní od druhého měsíce svého života,
- každý měsíc zplodí každý produktivní pár jeden další pár,
- králíci nikdy neumírají, nemají predátory.

Ilustrační příklad

Stavy populace králíků

Fibonacciho číslo	Stav
$F(1) = 1$	začínáme s jedním párem
$F(2) = 1$	ještě jsou příliš mladí
$F(3) = 2$	tento měsíc již zplodí první potomky
$F(4) = 3$	druhý pár potomků
$F(5) = 5$	první potomci třetí generace

Obecně

$$F(n) = F(n-1) + F(n-2)$$

Jak efektivně zjistit $F(n)$ pro zvolené n?

Explicitní řešení

Přímé vyjádření $F(n)$

Explicitní nerekurzivní vztah pro n-tý člen Fibonacciho posloupnosti je

$$F(n) = \frac{\phi^n - (1-\phi)^n}{\sqrt{5}},$$

kde ϕ je hodnota zlatého řezu,

$$\phi = \frac{1+\sqrt{5}}{2} = 1{,}61803398874989\cdots \approx 1{,}618.$$

Algoritmus 1

Spočti

$$F(n) = \frac{\phi^n - (1-\phi)^n}{\sqrt{5}}.$$

Explicitní řešení

Přímé vyjádření $F(n)$

Explicitní nerekurzivní vztah pro n-tý člen Fibonacciho posloupnosti je

$$F(n) = \frac{\phi^n - (1-\phi)^n}{\sqrt{5}},$$

kde ϕ je hodnota zlatého řezu,

$$\phi = \frac{1+\sqrt{5}}{2} = 1{,}61803398874989\cdots \approx 1{,}618.$$

Algoritmus 1

Spočti

$$F(n) = \frac{\phi^n - (1-\phi)^n}{\sqrt{5}}.$$

Analýza Algoritmu 1

Aneb co všechno může dopadnout špatně

Reprezentace čísel v plovoucí řádové čárce má svá omezení. V počítači budeme vztah reprezentovat jako

$$F(n) = \frac{1{,}61803^n - 0{,}61803^n}{2{,}23606}.$$

Jaké budou výsledky?

$F(2) = 1{,}00000$ je v pořádku

$F(3) = 1{,}78884$ zaokrouhlíme na 2

$F(20) = 6764{,}69$ ještě zaokrouhlíme na 6765

$F(21) = 10945{,}4$ už zaokrouhlíme na 10945 místo 10946

$F(25) = 75020{,}6$ by mělo být 75025

Existuje přesnější varianta výpočtu?

Analýza Algoritmu 1

Aneb co všechno může dopadnout špatně

Reprezentace čísel v plovoucí řádové čárce má svá omezení. V počítači budeme vztah reprezentovat jako

$$F(n) = \frac{1{,}61803^n - 0{,}61803^n}{2{,}23606}.$$

Jaké budou výsledky?

$F(2) = 1{,}00000$ je v pořádku
$F(3) = 1{,}78884$ zaokrouhlíme na 2
$F(20) = 6764{,}69$ ještě zaokrouhlíme na 6765
$F(21) = 10945{,}4$ už zaokrouhlíme na 10945 místo 10946
$F(25) = 75020{,}6$ by mělo být 75025

Existuje přesnější varianta výpočtu?

Analýza Algoritmu 1

Aneb co všechno může dopadnout špatně

Reprezentace čísel v plovoucí řádové čárce má svá omezení. V počítači budeme vztah reprezentovat jako

$$F(n) = \frac{1{,}61803^n - 0{,}61803^n}{2{,}23606}.$$

Jaké budou výsledky?

$F(2) = 1{,}00000$ je v pořádku
$F(3) = 1{,}78884$ zaokrouhlíme na 2
$F(20) = 6764{,}69$ ještě zaokrouhlíme na 6765
$F(21) = 10945{,}4$ už zaokrouhlíme na 10945 místo 10946
$F(25) = 75020{,}6$ by mělo být 75025

Existuje přesnější varianta výpočtu?

Analýza Algoritmu 1

Aneb co všechno může dopadnout špatně

Reprezentace čísel v plovoucí řádové čárce má svá omezení. V počítači budeme vztah reprezentovat jako

$$F(n) = \frac{1{,}61803^n - 0{,}61803^n}{2{,}23606}.$$

Jaké budou výsledky?

$F(2) = 1{,}00000$ je v pořádku
$F(3) = 1{,}78884$ zaokrouhlíme na 2
$F(20) = 6764{,}69$ ještě zaokrouhlíme na 6765
$F(21) = 10945{,}4$ už zaokrouhlíme na 10945 místo 10946
$F(25) = 75020{,}6$ by mělo být 75025

Existuje přesnější varianta výpočtu?

Analýza Algoritmu 1

Aneb co všechno může dopadnout špatně

Reprezentace čísel v plovoucí řádové čárce má svá omezení. V počítači budeme vztah reprezentovat jako

$$F(n) = \frac{1{,}61803^n - 0{,}61803^n}{2{,}23606}.$$

Jaké budou výsledky?

$F(2) = 1{,}00000$ je v pořádku
$F(3) = 1{,}78884$ zaokrouhlíme na 2
$F(20) = 6764{,}69$ ještě zaokrouhlíme na 6765
$F(21) = 10945{,}4$ už zaokrouhlíme na 10945 místo 10946
$F(25) = 75020{,}6$ by mělo být 75025

Existuje přesnější varianta výpočtu?

Rekurzivní algoritmus

Výpočet podle definice

Hodnoty $F(0)$ až $F(2)$ předpočítáme, zbytek lze s jejich pomocí vyjádřit.

Algoritmus 2

Require: $n \geq 0$
Ensure: $y = F(n)$
1: **if** $n = 0$ **then**
2: $\quad y \Leftarrow 0$
3: **else if** $n \leq 2$ **then**
4: $\quad y \Leftarrow 1$
5: **else**
6: $\quad y \Leftarrow F(n-1) + F(n-2)$
7: **end if**

Jak efektivní je daný algoritmus?

Rekurzivní algoritmus

Výpočet podle definice

Hodnoty $F(0)$ až $F(2)$ předpočítáme, zbytek lze s jejich pomocí vyjádřit.

Algoritmus 2

Require: $n \geq 0$
Ensure: $y = F(n)$

1: **if** $n = 0$ **then**
2: $\quad y \Leftarrow 0$
3: **else if** $n \leq 2$ **then**
4: $\quad y \Leftarrow 1$
5: **else**
6: $\quad y \Leftarrow F(n-1) + F(n-2)$
7: **end if**

Jak efektivní je daný algoritmus?

Rekurzivní algoritmus

Výpočet podle definice

Hodnoty $F(0)$ až $F(2)$ předpočítáme, zbytek lze s jejich pomocí vyjádřit.

Algoritmus 2

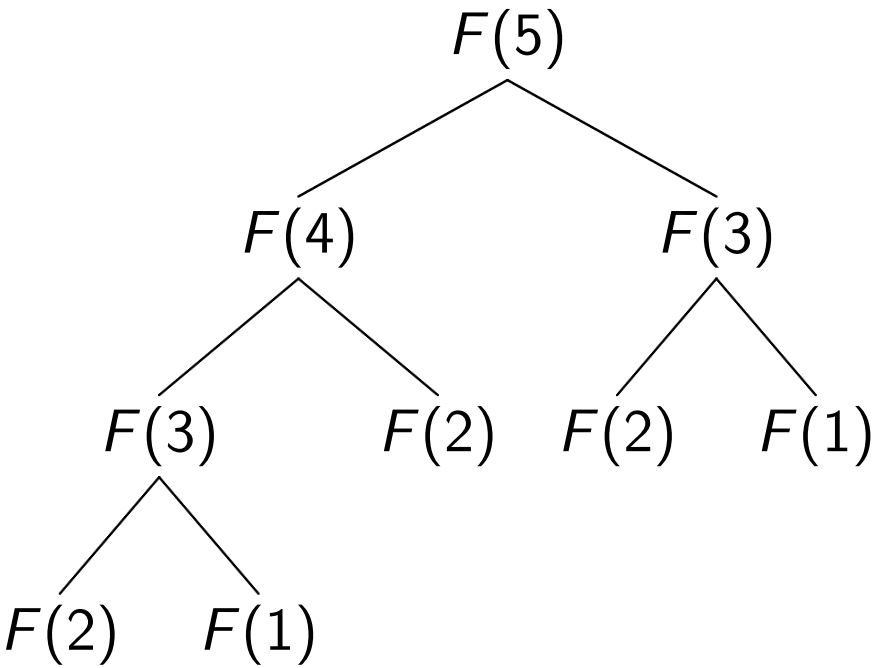
Require: $n \geq 0$
Ensure: $y = F(n)$

1: **if** $n = 0$ **then**
2: $\quad y \Leftarrow 0$
3: **else if** $n \leq 2$ **then**
4: $\quad y \Leftarrow 1$
5: **else**
6: $\quad y \Leftarrow F(n-1) + F(n-2)$
7: **end if**

Jak efektivní je daný algoritmus?

Rekurzivní algoritmus
Efektivita

Posloupnost výpočtu $F(5)$:



Počet kroků pro $F(n)$ je $\tau(n) = 3 \cdot F(n) - 2$.

Dynamické programování
Varianta „shora dolů“

Technika matematické optimalizace.

Dekompozice problému na identické podproblémy.

Dva základní přístupy:

- **shora dolů** – řešíme podproblémy postupně a pamatujeme si řešení
- **zdola nahoru** – vyřešíme všechny potřebné podproblémy a skládáme je

Algoritmus 3

Require: $n \geq 0$
Ensure: $y = F(n)$
1: Alokuj $f[1 \ldots n]$
2: $f[0] \Leftarrow 0$
3: $f[2] \Leftarrow f[1] \Leftarrow 1$
4: **for** $i = 3$ to n **do**
5: $\quad f[i] \Leftarrow f[i-1] + f[i-2]$
6: **end for**
7: $y \Leftarrow f[n]$

Dynamické programování

Varianta „shora dolů“

Technika matematické optimalizace.

Dekompozice problému na identické podproblémy.

Dva základní přístupy:

- **shora dolů** – řešíme podproblémy postupně a pamatujeme si řešení
- **zdola nahoru** – vyřešíme všechny potřebné podproblémy a skládáme je

Algoritmus 3

Require: $n \geq 0$

Ensure: $y = F(n)$

1: Alokuj $f[1 \ldots n]$
2: $f[0] \Leftarrow 0$
3: $f[2] \Leftarrow f[1] \Leftarrow 1$
4: **for** $i = 3$ to n **do**
5: $\quad f[i] \Leftarrow f[i-1] + f[i-2]$
6: **end for**
7: $y \Leftarrow f[n]$

Dynamické programování
Varianta „zdola nahoru“

Algoritmus 3 potřebuje pole n prvků pro uchování minulých členů posloupnosti.
Jde to ale i bez něj.

Algoritmus 4

Require: $n \geq 0$
Ensure: $y = F(n)$

```
1:  if n = 0 then
2:      y ⇐ 0
3:  else
4:      a ⇐ 1, b ⇐ 1
5:      for i = 3 to n do
6:          c ⇐ a + b
7:          a ⇐ b, b ⇐ c
8:      end for
9:      y ⇐ b
10: end if
```

Dynamické programování
Varianta „zdola nahoru“

Algoritmus 3 potřebuje pole n prvků pro uchování minulých členů posloupnosti.
Jde to ale i bez něj.

Algoritmus 4

Require: $n \geq 0$
Ensure: $y = F(n)$

```
1:  if n = 0 then
2:      y ⇐ 0
3:  else
4:      a ⇐ 1, b ⇐ 1
5:      for i = 3 to n do
6:          c ⇐ a + b
7:          a ⇐ b, b ⇐ c
8:      end for
9:      y ⇐ b
10: end if
```

Maticová varianta

Jiná explicitní forma

Pro členy Fibonacciho posloupnosti platí také

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n = \begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix}$$

Použitím opakovaného mocnění snížíme počet kroků na $O(\log n)$.

Maticová varianta

Pomocí opakovaného mocnění

Algoritmus 5

Require: $n \geq 0$
Ensure: $y = F(n)$

1: **if** $n = 0$ **then**
2: $y \Leftarrow 0$
3: **else**
4: $\mathbf{M} \Leftarrow \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$
5: $\mathbf{M} \Leftarrow$ matpow$(\mathbf{M}, n-1)$
6: $y \Leftarrow M[0,0]$
7: **end if**

Algoritmus 5a

Require: $n \geq 0, \mathbf{A}[2 \times 2]$
Ensure: $\mathbf{B}$ =matpow$(\mathbf{A}, n)$

1: **if** $n > 1$ **then**
2: $\mathbf{B} \Leftarrow$ matpow$(\mathbf{A}, n/2)$
3: $\mathbf{B} \Leftarrow \mathbf{BA}$
4: **end if**
5: **if** n je liché **then**
6: $\mathbf{B} \Leftarrow \mathbf{A} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$
7: **end if**

Porovnání variant

Vlastnosti jednotlivých algoritmů

Algoritmus	Paměťové nároky	Časová složitost
1	$O(1)$	$O(\log N)$
2	$O(N)$	$O(F(N))$
3	$O(N)$	$O(N)$
4	$O(1)$	$O(N)$
5	$O(\log N)$	$O(\log N)$

Obsah konzultace

Asymptotická složitost
Velká $\mathcal{O}$ notace

Jakým způsobem se bude chování algoritmu měnit v závislosti na velikosti (počtu, objemu) vstupních dat?

Dva základní typy:

- **časová složitost** – vliv na dobu výpočtu
- **paměťová složitost** – nároky na operační paměť

Značíme:

- $\mathcal{O}(N)$ – lineární složitost,
- $\mathcal{O}(N^2)$ – kvadratická složitost,
- $\mathcal{O}(\log N)$ – logaritmická složitost.

Asymptotická složitost

Velká $\mathcal{O}$ notace

Vliv asymptotické časové složitosti

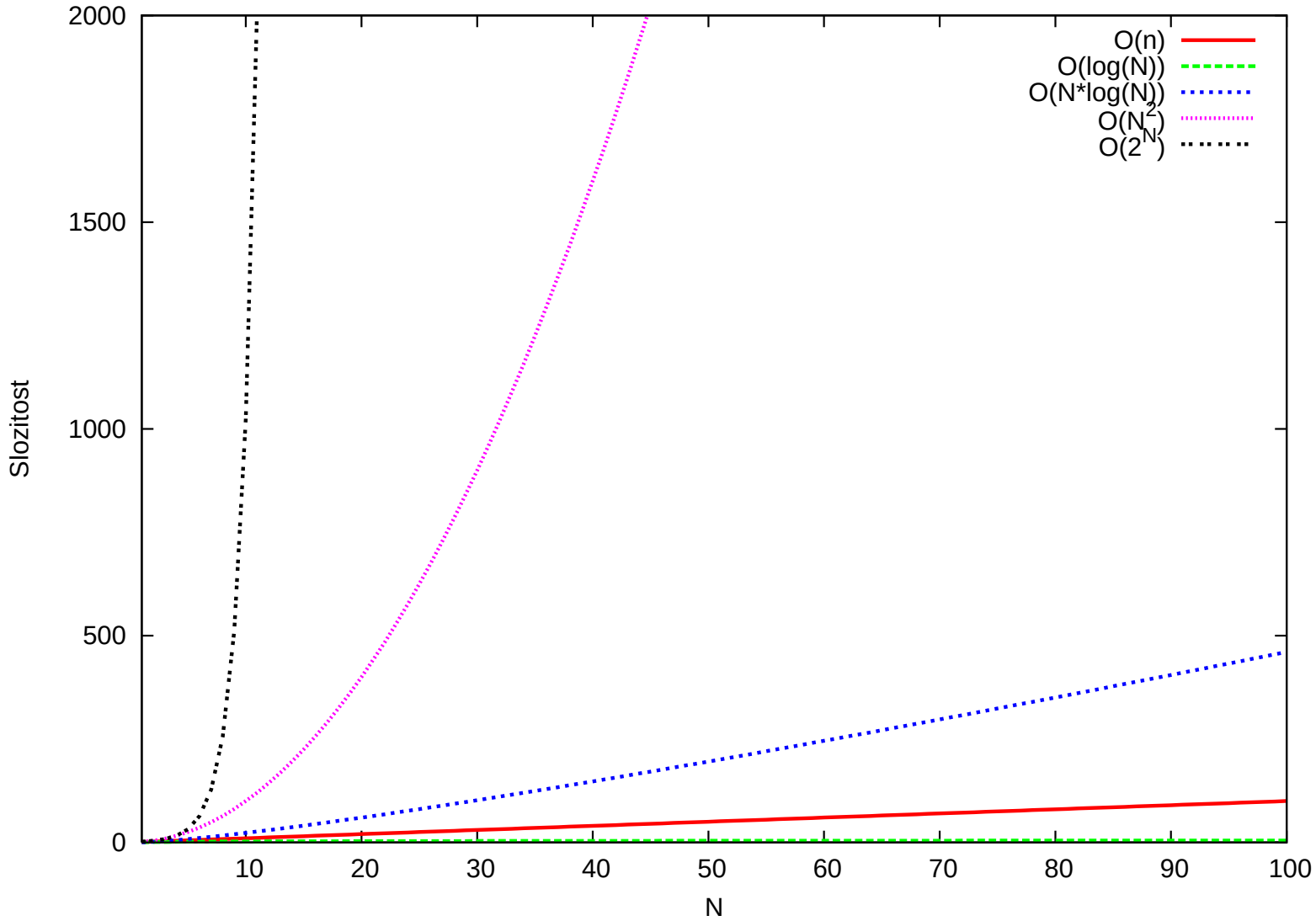
Pro $O(N^2)$ má zdvojnásobení objemu vstupních dat za následek čtyřnásobnou dobu vykonávání algoritmu.
Pro $O(\log N)$ může mít čtyřnásobný počet dat na vstupu za následek dvojnásobnou dobu vykonávání algoritmu.
Pro $O(1)$ je doba vykonávání algoritmu nezávislá na velikosti vstupu.

Vliv asymptotické paměťové složitosti

Pro $O(N)$ má zdvojnásobení velikosti vstupu za následek dvojnásob vysoké nároky na operační paměť.
Pro $O(2^N)$ čtyřnásobná velikost vstupu zosmínásobí paměťové nároky.

Asymptotická složitost

Obrázek



Obsah konzultace

Rozhodovací úlohy, třída $\mathcal{P}$

Velmi krátce z teorie výpočetní složitosti

Jako **rozhodovací úlohu** označujeme úlohu, jejímž řešením jsou výroky „ANO“ respektive „NE“.

Běžné úlohy v matematice lze snadno převést na rozhodovací úlohy.

Definice

Rozhodovací úloha L náleží do třídy $\mathcal{P}$, pokud existuje *deterministický* Turingův stroj, který tuto úlohu rozhodne v polynomiálním čase.

Příklady rozhodovacích úloh v třídě $\mathcal{P}$

Minimální kostra grafu – existuje kostra s ohodnocením menším, než c?

Nejkratší cesta v grafu – existuje cesta mezi dvěma uzly s ohodnocením menším, než c?

Lineární programování – existuje $\arg\max_{\boldsymbol{x}} \boldsymbol{w}^\top \boldsymbol{x} > c$ za daných omezujících podmínek?

Komprese dat (LZW) – přidá komprese řetězce s do slovníku slovo t?

Třída $\mathcal{NP}$

Nedeterministicky polynomiální úlohy

Definice

Rozhodovací úloha L náleží do třídy $\mathcal{NP}$, pokud existuje *nedeterministický* Turingův stroj, který tuto úlohu rozhodne v polynomiálním čase.

Nederministický Turingův stroj: vstupům může odpovídat více, než jedna jediná akce (sekvence $\Rightarrow$strom).

Platí $\mathcal{P} \subseteq \mathcal{NP}$.

Příklady rozhodovacích úloh v třídě $\mathcal{NP}$

Všechny úlohy třídy $\mathcal{P}$.

Izomorfismus grafu – lze dané dva grafy nakreslit stejně?

Faktorizace čísel – pro dané n a k, existuje $f : 1 < f < k, f|n$?

Všechny NP-úplné úlohy.

Třída NP-úplných úloh

Nejtěžší z třídy $\mathcal{NP}$

Třída NP-úplných problémů je třídou rozhodovacích úloh, pro něž platí následující definice:

Definice

Rozhodovací úloha L je NP-úplná, pokud náleží do třídy $\mathcal{NP}$ a zároveň jde o úlohu NP-těžkou

Co to znamená:

- Jakékoliv řešení L lze ověřit v polynomiálním čase
- Jakýkoliv problém z třídy NP lze převést na L transformací vstupů opět v polynomiálním čase

Tyto typy úloh umíme řešit pouze přibližně!

Příklady NP-úplných problémů

Problém batohu – lze zabalit batoh tak, aby jeho hmotnost nepřesáhla m a cena věcí byla alespoň c?

Problém obchodního cestujícího – existuje v grafu hamiltonovská kružnice o délce nejvýše c?

Obarvení grafu – lze uzly daného grafu obarvit nejvýše c barvami tak, aby sousedící uzly neměly stejnou barvu?

Problém čínského listonoše (pouze na smíšeném grafu) – existuje v grafu eulerovská kružnice o délce nejvýše c?