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

Matematické algoritmy (K611MA)

Jan Přikryl

2. konzultace K611MA
pátek 26. listopadu 2010

verze: 2010-11-29 12:11

1 Malá Fermatova věta

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

$$a \cdot x \equiv 1 \,(\mathrm{mod}\ \ n). \tag{1}$$

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

$$a^{-1} \equiv a^{p-2} \,(\mathrm{mod}\ \ p). \tag{2}$$

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 $x = 7$: $-3 \cdot 7 \equiv 1 \,(\mathrm{mod}\ \ 11)$.

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 $a = 5$.

2 Čínská věta o zbytcích

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

$$\begin{aligned} x &\equiv 2 \pmod{3}, \\ x &\equiv 3 \pmod{5}, \\ x &\equiv 2 \pmod{7}? \end{aligned}$$

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

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

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

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

2.1 Vlastní tvrzení

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.

2.2 Problém nůše s vejci

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

$$\begin{aligned} v &\equiv 1 \pmod 2 \\ v &\equiv 2 \pmod 3 \\ v &\equiv 4 \pmod 5 \\ v &\equiv 0 \pmod 7 \end{aligned}$$

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

$$\begin{aligned} v &= (1 \cdot 1 \cdot 105 + 2 \cdot 1 \cdot 70 + 4 \cdot 3 \cdot 42 + 0 \cdot 4 \cdot 30) \bmod 210 \\ &= (105 + 140 + 504 + 0) \bmod 210 = 749 \bmod 210 = 119 \end{aligned}$$

3 Šifrování

3.1 Symetrické a asymetrické šifry

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 ...

3.2 Výměna klíčů

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.

Hodnota α se v praxi volí 2 nebo 5.

Výměna klíčů je založena na faktu, že v modulární aritmetice se velmi těžko hledá **diskrétní logaritmus** celého čísla, tedy číslo $x = \log_g(h) \in \mathbb{Z}/n\mathbb{Z}^*$ takové, že $g^x \equiv h \,(\mathrm{mod}\, n)$.

Příklad: Uvažujme kongruenci $3^x \equiv 15 \,(\mathrm{mod}\, 19)$. Jedním z možných řešení je $x = 5$, neboť $3^5 \equiv 15 \,(\mathrm{mod}\, 19)$, není to ale řešení jediné. Z Malé Fermatovy věty totiž plyne $3^{18} \equiv 1 \,(\mathrm{mod}\, 19)$ a proto $3^{5+18k} \equiv 15 \,(\mathrm{mod}\, 19)$ pro libovolné $k \in \mathbb{N}$. Zadanou kongruenci tedy splňují taková x, pro něž platí $x \equiv 5 \,(\mathrm{mod}\, 18)$ a ze zveřejněného $3^x \bmod 19$ jenom velkou náhodou určíme ono konkrétní x, zvolené Alicí.

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)?

Pro ilustraci situace útočníka si povšimněte, že platí $5^7 \equiv 5^{23} \equiv 5^{39} \equiv 5^{7+k\cdot 16} \equiv 10 \,(\mathrm{mod}\, 17)$ pro libovolné $k \in \mathbb{N}$ a že $1271 = 7 + 79 \cdot 16$. Stejně tak je $5^{15} \equiv 5^{31} \equiv 5^{47} \equiv 5^{15+k\cdot 16} \equiv 7 \,(\mathrm{mod}\, 17)$ a $1039 = 15 + 64 \cdot 16$. V našem ilustračním případě může útočník celkem lehce vyzkoušet všechny možné kombinace klíčů (bude jich jenom sedmnáct), ale v případě velkých prvočísel to už nebude praktické: Bude-li $p = 429183283$ a dokážeme-li otestovat 100 klíčů za vteřinu, bude prohledávání celého prostoru možných klíčů trvat přibližně 1192 hodin. Pro p z oblasti 64bitových čísel by prohledávání celého prostoru možných klíčů rychlostí 10^6 klíčů/s mohlo trvat až 585 tisíc let.

3.3 Modulární mocnění

Modulární mocnění Výpočet $c \equiv b^r \,(\mathrm{mod}\, n)$

Neefektivně lze opakovaným násobením a redukcí:

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

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 $j = 1 \ldots k$:
   (a) Spočteme $b_j = b_{j-1}^2 \bmod n$
   (b) Pokud je $a_j > 0$, přepíšeme $c \leftarrow c \cdot b_j \bmod n$
4. Výsledkem je $c \equiv b^r \,(\mathrm{mod}\, n)$

Je možná jednodušší si postup přiblížit následujícím příkladem: Mám-li počítat $a \equiv 21^{41} \,(\mathrm{mod}\, 43)$, nebudu postupně vyčíslovat $a_1 = 21 \cdot 21 \bmod 43$, $a_2 = a_1 \cdot 21 \bmod 43$ až $a = a_3 \cdot 21 \bmod 43$. Jednodušší je si uvědomit, že $41 = 32 + 8 + 1$ a že tedy $a \equiv 21^4 1 \,(\mathrm{mod}\, 43) \equiv 21 \cdot 21^8 \cdot 21^{32} \,(\mathrm{mod}\, 43)$, kde pro výpočet dvojkových mocnin čísla 21 potřebujeme pouze opakovaně umocňovat na druhou:

$$\begin{aligned}
21^2 &\equiv 11 \,(\mathrm{mod}\, 43),\\
21^4 &\equiv 11^2 \,(\mathrm{mod}\, 43) \equiv 35 \,(\mathrm{mod}\, 43),\\
21^8 &\equiv 35^2 \,(\mathrm{mod}\, 43) \equiv 21 \,(\mathrm{mod}\, 43),\\
21^{16} &\equiv 21^2 \,(\mathrm{mod}\, 43) \equiv 11 \,(\mathrm{mod}\, 43),\\
21^{32} &\equiv 11^2 \,(\mathrm{mod}\, 43) \equiv 35 \,(\mathrm{mod}\, 43).
\end{aligned}$$

Výsledek obdržíme jako $a \equiv 21 \cdot 21^8 \cdot 21^{32} \pmod{43} \equiv 21 \cdot 21 \cdot 35 \pmod{43}$ a po úpravách $a \equiv 41 \pmod{43}$.

Počet kroků nutných pro umocnění b^r opakovaným kvadrátem je $\log_2 r$ oproti r krokům potřebným pro opakované násobení.

Modulární mocnění Příklad

Spočtěte $c = 3^{17} \bmod 7$

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

$$\begin{aligned} b_1 &= 3^2 \bmod 7 = 9 \bmod 7 = 2, a_1 = 0, \\ b_2 &= 2^2 \bmod 7 = 4 \bmod 7 = 4, a_2 = 0, \\ b_3 &= 4^2 \bmod 7 = 16 \bmod 7 = 2, a_3 = 0, \\ b_4 &= 2^2 \bmod 7 = 4 \bmod 7 = 4, a_4 = 1. \end{aligned}$$

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

3.4 RSA (Rivest, Shamir a Adelman 1977)

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.

Faktorizovat 1024-bitový RSA klíč by podle Kleinjunga a kolegů [2] bylo asi 1000× náročnější, než jejich faktorizace 768-bitového klíče. Vzhledem k tomu, že 768-bitový klíč je několikanásobně složitější na faktorizaci, než klíč o délce 512 bitů, a že mezi faktorizací těchto dvou klíčů uplynulo přibližně deset let, lze očekávat, že kilobitový klíč (tyto klíče se standardně používají v každodenním šifrovaném provozu) bude považován za faktorizovatelný (a tedy prolomitelný) už někdy během příštích deseti let.

Poslední faktorizovaný RSA klíč je přitom RSA-180, 596 bitů [1].

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

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

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

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

Pro libovolné přirozené n platí také

$$\phi(n) = n \prod_{p|n} \left(1 - \frac{1}{p}\right).$$

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 $\phi(n)$, $d \cdot e \equiv 1 \,(\mathrm{mod}\, \phi(n))$.

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.

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.
Z definice součinu ed v algoritmu RSA plyne

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

což můžeme dále upravit na

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

a tedy

$$ed \equiv 1 \pmod{\phi(n)} \equiv 1 \pmod{\phi(q)} \equiv 1 \pmod{\phi(p)}.$$

Důkaz: $ed \equiv 1 \pmod{\phi(n)} \equiv 1 \pmod{(p-1)(q-1)} \Leftrightarrow ed = 1 + g(p-1)(q-1)$ a tedy $ed = 1 + g_p(q-1)$ a $ed = 1 + g_q(p-1)$.
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

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

Jedním z důsledků CRT je pro nesoudělná x a y ekvivalence

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

Proto také z

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

plyne

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

Ekvivalence

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

platí neboť z první rovnice

$$\begin{aligned} a &= k_1 x + b \\ a - b &= k_1 x \end{aligned}$$

a z druhé obdobně

$$\begin{aligned} a &= k_2 y + b \\ a - b &= k_2 y. \end{aligned}$$

Vzhledem k tomu, že x a y jsou nesoudělná, obě dělí výraz $a - b$,

$$a - b = k_1 x = k_2 y,$$

musí být

$$k_1 x = k_2 y = \kappa\, xy$$

a tedy také

$$a - b = \kappa\, xy$$

a proto

$$a \equiv b \,(\mathrm{mod}\, xy).$$

3.5 CRT-RSA

V úvodu přednášky zmíněná Čínská věta o zbytcích má velmi zajímavé využití právě při výpočtech dešifrovací části šifry RSA.

RSA pomocí CRT Urychlení dešifrování (1/3)

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

$$X \equiv c^d \,(\mathrm{mod}\, n)$$

trvá dlouho.

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

Pro $n = pq$ použijeme již jednou provedený trik

$$\begin{aligned} X &\equiv X_p \,(\mathrm{mod}\, p) \equiv c^{d_p} \,(\mathrm{mod}\, p) \\ X &\equiv X_q \,(\mathrm{mod}\, q) \equiv c^{d_q} \,(\mathrm{mod}\, q) \end{aligned}$$

přičemž pro p

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

a tedy

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

Pro q je to obdobně.

Plné znění je

$$\begin{aligned} d_p &\equiv d \,(\mathrm{mod}\, p-1) \Leftrightarrow d = d_p + j(p-1) \\ d_q &\equiv d \,(\mathrm{mod}\, q-1) \Leftrightarrow d = d_q + k(q-1) \end{aligned}$$

a tedy

$$c^d \equiv c^{d_p+j(p-1)} \,(\mathrm{mod}\, p) \equiv c^{d_p} 1^j \,(\mathrm{mod}\, p) \equiv c^{d_p} \,(\mathrm{mod}\, p)$$
$$c^d \equiv c^{d_q+k(q-1)} \,(\mathrm{mod}\, q) \equiv c^{d_q} 1^k \,(\mathrm{mod}\, q) \equiv c^{d_q} \,(\mathrm{mod}\, q)$$

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

$$X \equiv c^{d_p} \,(\mathrm{mod}\, p),$$
$$X \equiv c^{d_q} \,(\mathrm{mod}\, q).$$

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

Prolomení RSA při nevhodné volbě p a q 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 $d \cdot e \equiv 1 \,(\mathrm{mod}\, \phi(n))$.

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

Příklad RSA: musíme doplnit, externí viz například `http://en.wikipedia.org/wiki/RSA#A_working_example`

4 Analýza algoritmů

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.

Každý problém lze v matematice řešit mnoha různými postupy. Existuje tedy mnoho různých algoritmů, které vedou ke stejnému cíli. Pokud programátor píše program, který bude použit jenom jednou, zvolí patrně řešení, které lze snadno a rychle implementovat. Jaký algoritmus bychom ale měli preferovat v případě, kdy bude zvolený kus kódu používán opakovaně? V takovém případě je třeba posuzovat různá subjektivní hlediska (strozumitelnost kódu, rozšiřitelnost algoritmu na různé typy vstupních dat, přenositelnost na jiné architektury počítačů a efektivitu výpočtu).

Efektivita algoritmu přímo závisí na tom, kolik zvolený algoritmus spotřebovává výpočetních zdrojů, tedy například jak dlouho bude trvat jeho vykonávání pro určitou množinu vstupních dat, kolik bude potřebovat operační paměti, do jaké míry bude využívat diskový subsystém či síťové rozhraní.

Primárním hladiskem posuzování efektivity algoritmu je čas. Uvědomme si ale, že v mnoha případech musí při implementaci docházet k určitým kompromisům – algoritmus může být možné zrychlit pouze za cenu neúměrného zvýšení spotřeby operační paměti. Mnohdy také platí, že implementace velmi efektivních algoritmů jsou mnohem hůře pochopitelné a udržovatelné.

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**

Rychlost vykonávání programu (implentace zvoleného algoritmu) závisí na zvoleném programovacím jazyce, výpočetním výkonu počítače, jeho momentální zátěži či na zvoleném kompilátoru. Přesto bychom rádi znali předem orientační odhad toho, jak efektivní je algoritmus, jenž jsme zvolili k implementaci.

Existují dva základní způsoby, jak si udělat představu o tom, jak se algoritmus chová:

- experimentální ověření chování implementace
- analýza na základě pseudokódu

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.*

5 Vývojová stádia algoritmu

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?

5.1 Algoritmus pomocí explicitního řešení

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 Přímá varianta výpočtu $F(n)$.

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?

5.2 Rekurzivní algoritmus

Rekurzivní algoritmus Výpočet podle definice

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

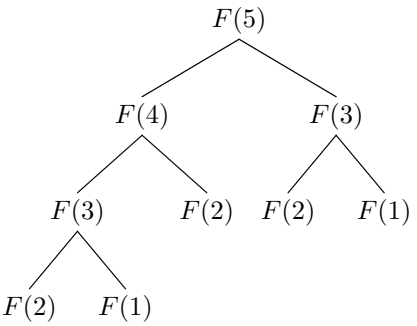
Algoritmus 2 Varianta výpočtu $F(n)$ rekurzivním algoritmem.

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 $\tau(n) = 3 \cdot F(n) - 2$.

5.3 Dynamické programování

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 Varianta výpočtu $F(n)$ pomocí dynamického programování přístupem shora dolů.

Require: $n \geq 0$
Ensure: $y = F(n)$
1: Alokuj $f[1 \dots n]$
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 Varianta výpočtu $F(n)$ pomocí dynamického programování přístupem zdola nahoru.

Require: $n \geq 0$
Ensure: $y = F(n)$
1: **if** $n = 0$ **then**
2: $\quad y \Leftarrow 0$
3: **else**
4: $\quad a \Leftarrow 1, b \Leftarrow 1$
5: $\quad$ **for** $i = 3$ to n **do**
6: $\quad\quad c \Leftarrow a + b$
7: $\quad\quad a \Leftarrow b, b \Leftarrow c$
8: $\quad$ **end for**
9: $\quad y \Leftarrow b$
10: **end if**

5.4 Maticová varianta pomocí opakovaného mocnění

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 $O(\log n)$.

Důkaz. Indukcí pro $n \to n+1$. Začneme pro $n = 1$, kde platí

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^1 = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} F_2 & F_1 \\ F_1 & F_0 \end{bmatrix}$$

a vztah tedy pro $n = 1$ dává korektní výsledek. Za předpokládu, že vztah platí pro n, dostaneme pro $n + 1$

$$\begin{aligned}\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n+1} &= \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} = \\ &= \begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} = \\ &= \begin{bmatrix} F_{n+1} + F_n & F_{n+1} \\ F_n + F_{n-1} & F_n \end{bmatrix} = \begin{bmatrix} F_{n+2} & F_{n+1} \\ F_{n+1} & F_n \end{bmatrix}\end{aligned}$$

□

Odvození celého maticového předpisu vychází z teorie dvouprvkových rekurencí a na žádost některých z vás zde uvádím stručný výtah. K dispozici máte volně stažitelnou plnou verzi článku [3], z níž jsem čerpal.

Definice 1 (Množina $\mathcal{R}(a, b)$)**.** Mějme dvojici reálných čísel a a $b \neq 0$ a posloupnost A_n definovanou rekurentním vztahem

$$A_{n+2} = a \cdot A_{n+1} + b \cdot A_n.$$

Potom pro pevné hodnoty parametrů a a b označíme $\mathcal{R}(a, b)$ množinu všech takto parametrizovaných posloupností.

Jedním z význačných prvků $\mathcal{R}(1, 1)$ je Fibonacciho posloupnost F s počátečními členy $F_0 = 0$ a $F_1 = 1$.

Definice 2 (Posloupnost A)**.** Všechny prvky posloupnosti $A_0, A_1, A_2, \ldots$ tvořící vektor v $\mathbb{R}^\infty$ označíme jako *posloupnost* A.

Definice 3. Operátor *posunu doleva*, označovaný $\lhd$, odstraní z posloupnosti A její nejlevější prvek.

Z původní posloupnosti $A = A_0, A_1, A_2, A_3, \ldots$ vznikne posunem doleva o jednu pozici posloupnost $\lhd A = A_1, A_2, A_3, A_4, \ldots$.

Studiem vlastností dvouprvkových rekurencí snadno dospějeme k závěru, že vlastnosti rekurencí $A \in \mathcal{R}(a, b)$ jsou zcela určeny volbou A_0 a A_1. Prostor $\mathcal{R}(a, b)$ je proto dvourozměrný a každé $A \in \mathcal{R}(a, b)$ musíme být schopni vyjádřit jako lineární kombinaci dvou bázových posloupností X a Y s prvky $X_0 = 1, X_1 = 0$ a $Y_0 = 0, Y_1 = 1$,

$$\begin{aligned}X &= 1, 0, b, ab, a^2b + b^2, \ldots, \\ Y &= 0, 1, a, a^2 + b, a^3 + 2ab, \ldots.\end{aligned}$$

Pro všechna $A \in \mathcal{R}(a, b)$ je tedy

$$A = A_0X + A_1Y.$$

Všimněte si, že pro $a = 1$ a $b = 1$ je $Y = F$ (půjde o Fibonacciho posloupnost) a také, že

$$\lhd X = b \cdot Y,$$

takže můžeme jakoukoliv posloupnost $A \in \mathcal{R}(a, b)$ psát jako posloupnost členů

$$A_n = A_0bY_{n-1} + A_1Y_n = A_0bF_{n-1} + A_1F_n.$$

Operátor posunu doleva lze v $\mathcal{R}(a, b)$ s bázovými prvky X a Y vyjádřit maticí $\mathbf{M}$ jako

$$\lhd \begin{bmatrix} X \\ Y \end{bmatrix} = \mathbf{M}^\mathsf{T} \begin{bmatrix} X \\ Y \end{bmatrix}.$$

Prvky matice $\mathbf{M}$ jsou dány vztahy

$$\begin{aligned}\lhd X &= b \cdot Y, \\ \lhd Y &= X + a \cdot Y,\end{aligned}$$

a proto

$$\mathbf{M} = \begin{bmatrix} 0 & 1 \\ b & a \end{bmatrix}.$$

Posloupnost $A \in \mathcal{R}(a, b)$ je v bázi $[X, Y]$ možno zapsat jako

$$A = \begin{bmatrix} A_0 \\ A_1 \end{bmatrix}$$

přičemž pro n-krát posunutou posloupnost bude platit

$$\overset{n}{\lhd} A = \begin{bmatrix} A_n \\ A_{n+1} \end{bmatrix}.$$

Vzhledem k definici transformační matice $\mathbf{M}$ je ale také $\lhd^n A = \mathbf{M}^n A$, a proto

$$\begin{bmatrix} 0 & 1 \\ b & a \end{bmatrix}^n \begin{bmatrix} A_0 \\ A_1 \end{bmatrix} = \begin{bmatrix} A_n \\ A_{n+1} \end{bmatrix}.$$

Pro n-násobné posuny bázové posloupnosti X a Y můžeme psát

$$\begin{bmatrix} 0 & 1 \\ b & a \end{bmatrix}^n \begin{bmatrix} X_0 \\ X_1 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} m_{11} \\ m_{21} \end{bmatrix} = \begin{bmatrix} X_n \\ X_{n+1} \end{bmatrix},$$

respektive

$$\begin{bmatrix} 0 & 1 \\ b & a \end{bmatrix}^n \begin{bmatrix} Y_0 \\ Y_1 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} m_{12} \\ m_{22} \end{bmatrix} = \begin{bmatrix} Y_n \\ Y_{n+1} \end{bmatrix},$$

z čehož vyplývá

$$\begin{bmatrix} 0 & 1 \\ b & a \end{bmatrix}^n = \begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix} = \begin{bmatrix} X_n & Y_n \\ X_{n+1} & Y_{n+1} \end{bmatrix}.$$

Již dříve jsme si všimli, že $Y = F$ (a tedy $Y_n = F_n$) a že $\Lambda X = b \cdot F$ (a tedy $X_n = b \cdot F_{n-1}$). V $\mathcal{R}(a, b)$ bude proto platit

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

a v $\mathcal{R}(1, 1)$ ekvivalentně

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

Výše uvedený zápis lze pomocí opakovaného mocnění zapsat Algoritmem 5. Hlavní cyklus využívá funkce matpow, kterou definujeme Algoritmus 6.

Algoritmus 5 Algoritmus maticové reprezentace Fibonacciho posloupnosti.

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: $\mathbf{M} \Leftarrow \text{matpow}(\mathbf{M}, n-1)$
6: $y \Leftarrow M[0, 0]$
7: **end if**

Algoritmus 6 Algoritmus opakovaného mocnění matic.

Require: $n \geq 0, \mathbf{A}[2 \times 2]$
Ensure: $\mathbf{B} = \text{matpow}(\mathbf{A}, n)$
1: **if** $n > 1$ **then**
2: $\quad \mathbf{B} \Leftarrow \text{matpow}(\mathbf{A}, n/2)$
3: $\quad \mathbf{B} \Leftarrow \mathbf{BA}$
4: **end if**
5: **if** n je liché **then**
6: $\quad \mathbf{B} \Leftarrow \mathbf{A} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$
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	$O(\log N)$	$O(\log N)$

6 Asymptotická složitost

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.

Složitost $\mathcal{O}(N)$ znamená lineárně rostoucí nároky, $\mathcal{O}(N) \sim k \cdot N + q$ pro nějaká $k, n \in \mathbb{N}$, pro $\mathcal{O}(1)$ jsou nároky konstantní, $O(1) \sim q$.

Asymptotická složitost Velká $\mathcal{O}$ notace

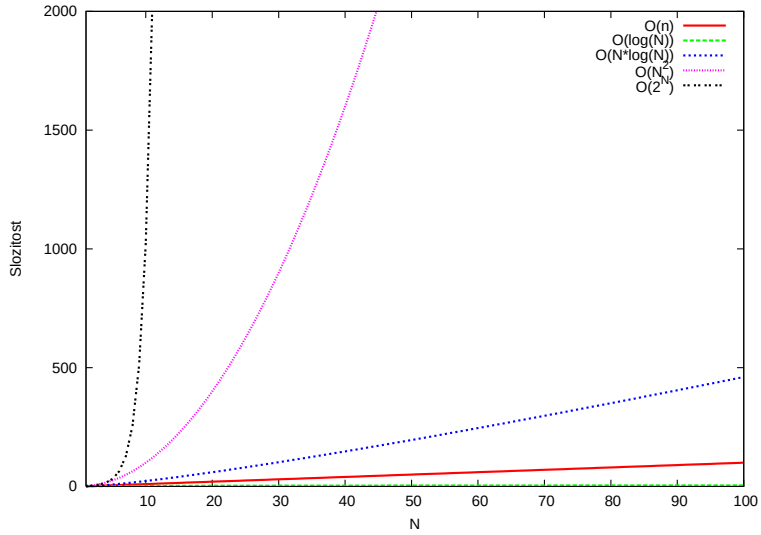
Vliv asymptotické časové složitosti

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



7 NP-úplné problémy

Jistě jste už někde slyšeli či četli, že nějaký matematický problém je **NP-úplný** (angl. *NP-complete*, v češtině se někdy také používá označení NP-složitý). Podívejme se na závěr této přednášky krátce na to, co tento termín znamená a proč je z hlediska algoritmů tak důležitý. Detailnější pohled na celý problém lze nalézt například v poznámkách prof. Demlové na FEL [4].

Rozhodovací úlohy, třída $\mathcal{P}$

Jako **rozhodovací úlohu** označujeme úlohu, jejímž řešením jsou výroky „ANO“ respektive „NE“. Jako výstup v případě počítače můžeme uvažovat hodnoty `true` a `false` nebo 1 a 0.

Běžné úlohy v matematice lze snadno převést na rozhodovací úlohy. Například hledání nejkratší cesty v ohodnoceném grafu lze převést na rozhodovací úlohu „Lze v grafu najít cestu, jejíž délka je menší, než nějaké c?“

Definice 4. 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 $\mathcal{NP}$

Definice 5. 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 ⇒strom). Jeho chování si lze představit tak, že v každém přechodu může dojít k naklonování několika

kopií stroje se stejnou historií, ale jiným cílovým stavem daného přechodu. Výsledkem provádění důkazu rozhodovací úlohy je potom neustále se rozšiřující strom možností, z nichž v určitém čase jedna povede k cíli.

Platí $\mathcal{P} \subseteq \mathcal{NP}$. Pokud použijeme nedeterministický Turingův stroj v takové konfiguraci, že v každém přechodu nedojde k naklonování žádné kopie, stroj bude vlastně deterministický a jsme v třídě $\mathcal{P}$.

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

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

Definice 6. 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*!

Základním úskalím NP-úplných úloh je to, že je sice možné rychle (tedy v polynomiálním čase) ověřit, zda zadané řešení úlohy platí, není ale známý žádný efekltivní způsob, jak toto řešení pro dané vstupy najít. Tento paradox je jednou z nejvýznamnějších vlastností této třídy úloh: Pro NP-úplnou úlohu není znám algoritmus, jenž by dokázal dostatečně rychle najít její řešení. Čas, potřebný k vyřešení takové úlohy jakýmkoliv v současné době známým algoritmem roste velmi rychle spolu s tím, jak roste „velikost“ řešeného problému (například počet cifer faktorizovaného čísla). Doba řešení takových úloh v současné době známými algortimy dosahuje i pro rozumně velké rozsahy vstupních data klidně miliónů let bez ohledu na to, jak roste výpočetní výkon současných počítačů.

Příklady NP-úplných problémů

Problém batohu – lze zabalit batoh tak, aby jeho hmotnost nepřesáhla m a cena věcí byla alespoň c? Vlastní problém lze možná lépe popsat takto: pokud mám množinu $\mathscr{A}$ obsahující N předmětů o hmotnosti μ a ceně γ, existuje $\mathscr{B} \subseteq \mathscr{A}$ taková, že

$$\sum_{j \in \mathscr{B}} \mu_j \leq m \text{ a zároveň } \sum_{j \in \mathscr{B}} \gamma_j \geq c?$$

Problém obchodního cestujícího – existuje v grafu hamiltonovská kružnice o délce nejvýše c? *Hamiltonovská kružnice* v grafu prochází právě jednou všemi jeho vrcholy, stejně tak jako si přejeme, aby náš obchodní cestující navštívil každé město právě jednou.

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? *Eulerovská kružnice* v grafu prochází právě jednou všechny hrany, stejně tak, jako pošťák musí projet všechny ulice, z nichž některé jsou jednosměrné.

Reference

[1] Danilov, S. A. – Popovyan, I. A.: Factorization of RSA-180. [online] Cryptology ePrint Archive. Dostupné na WWW: `http://eprint.iacr.org/2010/270.pdf` (staženo 13.10.2010).

[2] Kleinjung, T. – Aoki, K. – Franke, J. – Lenstra, A. – Thome, E. – Bos, J. – Gaudry, P. – Kruppa, A. – Montgomery, P. – Osvik, D. A. – te Riele, H. – Timofeev, A. – Zimmermann, P. Factorization of a 768-bit RSA modulus. [online] Cryptology ePrint Archive. Dostupné na WWW: `http://eprint.iacr.org/2010/006.pdf` (staženo 13.10.2010).

[3] Kalman, D. – Mena, R.: The Fibonacci Numbers—Exposed. Mathematics Magazine, vol. 76, no. 3, 2003, pp. 167–181. Preprint dostupný na WWW: `http://www.american.edu/cas/mathstat/People/kalman/pdffiles/fibpaper.pdf` (staženo 18.11.2008).

[4] Demlová, M: Teroie algoritmů – stručný obsah přednášek. ČVUT FEL, Katedra matematiky, 2010. Dostupné na WWW: `http://math.feld.cvut.cz/demlova/teaching/tal/predn_tal.html` (staženo 30.10.2010).