Modulárna matematika

Sylabus, resp. čo treba vedieť na skúške

- *delenie modulo n, existencia inverzných prvkov*
- *malá Fermatova veta, jej použitie na hľadanie inverzných prvkov*
- *algoritmus rýchleho umocňovania*
- *Bézoutova identita, jej použitie na hľadanie inverzných prvkov*
- *rozšírený Euklidov algoritmus*
- *Čínska zvyšková veta*

Na skúške by ste mali vedieť vysloviť a dokázať spomenuté vety. Pri algoritmoch by ste mali vedieť ako fungujú, akú majú časovú zložitosť a vedieť dokázať ich správnosť.

V programovaní sa často stretneme s problémami, ktoré vyžadujú počítanie modulo nejaké číslo. Najčastejšie sa s takýmito problémami stretneme pri hashovaní alebo kryptológii.

V tejto prednáške si preto predstavíme základné algoritmy, ktoré sa pri riešení takýchto problémov používajú. Ako uvidíme, niektoré z algoritmov budú mať aj širšie uplatnenie, napríklad rýchle umocňovanie alebo hľadanie najväčšieho spoločného deliteľa.

Aritmetické operácie a existencia inverzných prvkov

Majme celé kladné číslo n, ktoré bude predstavovať naše modulo. Ako iste vieme, pre aritmetické operácie $+$, - a $\cdot$ platia nasledovné rovnice:

$$\begin{aligned} a+b &\equiv (a \mod n)+(b \mod n) \pmod n \\ a-b &\equiv (a \mod n)-(b \mod n) \pmod n \\ a\cdot b &\equiv (a \mod n)\cdot(b \mod n) \pmod n \end{aligned}$$

Táto vlastnosť nám uľahčuje počítanie modulo n, pretože nám stačí pamätať si iba zvyškovú triedu, do ktorej naše čísla patria. Problém ale je, že pre / (delenie) takáto vlastnosť neplatí. Nech napríklad $n = 5$. Rovnica $6/2$ dáva zmysel aj modulo 5 a jej výsledok je 3. Ak si však zoberieme $(6 \mod 5)/(2 \mod 5)$, čo je $1/2$ tak to celé prestane dávať zmysel. Necelé čísla totiž nepatria do hodnôt, s ktorými počítame, a je neintuitívne, aby sa $1/2 \equiv 3$.

Obrátiť sa preto musíme na algebru. Majme rovnicu $\frac{1}{b} = x$ a našou úlohou je nájsť x spĺňajúce túto rovnicu. Inak povedané, hľadáme také číslo x, že $b \cdot x = 1$. A to bola predsa definícia **inverzných prvkov** v algebre. Je teda jasné, že delenie číslom b sa dá nahradiť násobením inverzným prvkom b^{-1}. Otázka však je, ako túto hodnotu vypočítať a či vôbec taká hodnota existuje.

Lemma 1.1. *Nech n je ľubovoľné kladné číslo. Potom pre každé $x \in \mathbb{Z}_n$ nesúdeliteľné s n existuje práve jedno $y \in \mathbb{Z}_n$ také, že*

$$x \cdot y \equiv 1 \pmod n$$

Toto y označujeme x^{-1}.

Proof. Zoberme si ľubovoľné n a $x \in \mathbb{Z}_n$. Chceme nájsť také y, že $x \cdot y \equiv 1 \pmod n$. Zoberme si teraz n čísel $0 \cdot x$, $1 \cdot x \ldots (n-1) \cdot x$, všetky samozrejme modulo n. Chceme dokázať, že práve jedno z nich je rovné 1.

Ukážme preto, že všetky tieto čísla sú navzájom rôzne. Pre spor, nech $i \cdot x \equiv j \cdot x \pmod n$, pre čísla $i \neq j$, pričom $i, j < n$. Túto rovnicu potom vieme upraviť na

$$x \cdot (i-j) \equiv 0 \pmod n$$

Teda $x \cdot (i-j)$ je deliteľné číslom n. Číslo x je ale **nesúdeliteľné** s n. To znamená, že hodnota $i-j$ musí byť deliteľná n. Keďže však $i, j < n$, tak hodnota $i-j$ môže byť iba medzi $-(n-1)$ a $n-1$. A jediné číslo deliteľné n v tomto rozsahu je 0. Je teda jasné, že ak $i-j=0$, tak $i=j$.

n čísel $0 \cdot x$, $1 \cdot x$ ... $(n-1) \cdot x$ je teda navzájom rôznych a keďže všetky sú medzi 0 až $n-1$, tak **práve jedno** je rovné 1. Týmto číslom je potom určená hodnota y, teda hľadaný inverzný prvok. □

Lemma 1.1 nám síce nehovorí nič o tom, ako takýto inverzný prvok nájsť, hovorí nám však, kedy existuje. Už aj toto je však užitočná informácia. Napríklad by teraz mohlo byť jasné, prečo je $(\mathbb{Z}_n, +, *)$ pole iba ak je n prvočíslo. Iba v takom prípade má každý prvok $\mathbb{Z}_n$ inverzný prvok ležiaci v množine $\mathbb{Z}_n$, čo je podmienka poľa.

Skôr ako sa pustíme do hľadania inverzných prvkov, povedzme si ešte niečo málo o počítaní modulo v programovacích jazykoch. Napríklad v `C++` máme znak '%' predstavujúci modulo. A je jasné, že aj pre záporné čísla funguje modulo. Napríklad $-1 \equiv n-1 \pmod{n}$. Dôležité je však vedieť, že ak budeme v `C++` (alebo aj inom programovacom jazyku s touto vlastnosťou) modulovať záporné číslo, dostaneme **záporné** číslo z rozsahu $-(n-1)$ až 0. To môže byť občas pre náš program problém, väčšinou by sme boli radi, aby to boli iba kladné čísla. Práve kvôli tomu sa pri odčítavaní používa nasledovný príkaz

```
((a-b)%n + n)%n
```

Sami si rozmyslite, že tento výraz vráti vždy kladné číslo z rozsahu 0 až $n-1$.

Hľadanie inverzného prvku – malá Fermatova veta

Prvý spôsob hľadania inverzného prvku čísla x modulo n bude fungovať iba ak bude n prvočíslo. Napriek tomu je tento spôsob pomerne užitočný, pretože je rýchly, elegantný a prvočísla sa pri modulovaní používajú naozaj často (napríklad pri spomínaných hashovaniach alebo kryptografii). Najskôr budeme potrebovať vysloviť nasledovnú vetu:

Veta 1.2 (Malá Fermatova veta)**.** *Nech p je ľubovoľné prvočíslo a nech a je ľubovoľné celé číslo nedeliteľné p. Potom platí:*

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

Proof. Dôkaz tejto vety bude veľmi podobný dôkazu lemmy 1.1. Zoberme si $p-1$ čísel $1 \cdot a$, $2 \cdot a$ ... $(p-1) \cdot a$. Z nedeliteľnosti čísla a číslom p je jasné, že žiadne z týchto čísel nie je rovné 0. Naviac, každé dve tieto čísla sú rôzne.

Pre spor predpokladajme, že pre dve hodnoty $i \neq j$ $(0 \leq i, j < p)$ platí $i \cdot a \equiv j \cdot a \pmod{p}$. Potom platí $a \cdot (i-j) \equiv 0 \pmod{p}$. Číslo p je však prvočíslo a preto je s číslom a nesúdeliteľné. Preto musí hodnota p deliť číslo $i-j$. Jediné možná hodnota $i-j$, ktorá je deliteľná p je však 0, z čoho vyplýva, že $i=j$, čo je spor s predpokladom. Z toho vyplýva, že všetky uvedené čísla sú navzájom rôzne.

Tieto čísla sú navzájom rôzne, nie je medzi nimi 0, preto sú len permutáciou hodnôt 1 až $p-1$. Označme si súčin $1 \cdot 2 \cdot \cdots \cdot (p-1)$ hodnotou S. Potom platí:

$$(1 \cdot a) \cdot (2 \cdot a) \cdot \cdots \cdot ((p-1) \cdot a) \equiv S \pmod{p}$$

Ľavú časť rovnice však vieme upraviť na $a^{p-1} \cdot (1 \cdot 2 \cdot \cdots \cdot (p-1))$ čo je vlastne $a^{p-1} \cdot S$.

Predelením hodnotou S dostaneme požadovanú kongruenciu

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

□

Malá Fermatova veta nám však veľmi nepomohla. Akurát sme sa dozvedeli, čomu sa rovná hodnota a^{p-1}. Majme však prvočíslo p, ktorým modulujeme a hodnotu a, o ktorej by sme chceli vedieť inverzný prvok. Prenásobmme preto rovnicu z tejto vety inverzným prvkom a^{-1}, ktorý hľadáme:

$$a^{p-1} \cdot a^{-1} = a^{p-2} \equiv 1 \cdot a^{-1} = a^{-1} \pmod{p}$$

Zisťujeme, že inverzný prvok a^{-1} je vlastne rovný hodnote a^{p-2}. A keďže $p-2$ je nezáporné, hodnotu a^{p-2} vieme vypočítať umocnením. Na počítanie inverzného prvku modulo prvočíslo p preto existuje nasledovný jednoduchý algoritmus:

Listing programu (C++)

```
int inverz(int a, int p) {
    return umocni(a,p-2,p); //umocni cislo a na p-2 modulo p
}
```

Rýchle umocňovanie

Otázka je, akú zložitosť má takýto algoritmus. Je jasné, že to závisí od zložitosti funkcie `umocni()`. A jej triviálna implementácia pomocou `for` cyklu má zložitosť $O(p)$. Takéto riešenie je však nepriateľné, ak je p veľké, alebo chceme počítať inverzný prvok pre veľa čísel. Chceli by sme preto rýchlejší algoritmus, ktorý počíta mocninu a^b.

Na to použijeme nasledovnú rekurzívnu myšlienku. Nech b je párne. Potom hodnotu a^b môžeme vypočítať ako

$$(a^{b/2})^2$$

Hodnotu $a^{b/2}$ pritom vypočítame rekurzívne a umocníme na druhú v konštantnom čase. No a ak je b nepárne, použijeme

$$a \cdot (a^{b/2})^2$$

V každom kroku teda zmenšíme hodnotu b na polovicu. To znamená, že náš algoritmus spraví iba $O(\log(b))$ krokov na vypočítanie hodnotu a^b a teda algoritmus počítajúci inverzné prvky bude mať zložitosť $O(\log(p))$.

Prvá implementácia rýchleho umocňovania používa vyššie uvedenú rekurzívnu myšlienku. Druhá implementácia je trochu trikovejšia. Využíva fakt, že číslo b sa dá poskladať z niekoľkých mocnín 2 (binárne číslo) a počítať a^{2^k} vieme veľmi jednoducho umocňovaním na druhú. Výhoda tohto postupu je, že sa programuje trošku jednoduchšie a keďže nepoužíva rekurziu, tak má konštantnú pamäťovú zložitosť.

Listing programu (C++)

```
int umocni(int a, int b, int mod) { //umocni a na b a vysledok vrat modulo mod
    if(b==0) return 1;
    int x = umocni(a,b/2,mod); //problem polovicnej velkosti
    if(b%2 == 0) return (x*x)%mod;
    return (a*x*x)%mod;
}
```

Listing programu (C++)

```
int umocni(int a, int b, int mod) { //umocni a na b a vysledok vrat modulo mod
    int res = 1;
    while(b>0) {
        if(b%2 == 1) res=(res*a)%mod;
        a = (a*a)%mod;
        b/=2;
    }
    return res;
}
```

Hľadanie inverzného prvku – Bézoutova identita

Už sme zistili, že vieme hľadať inverzné prvky modulo prvočíslo. Ako nám však ukázala veta 1.1, inverzný prvok existuje pre ľubovoľnú dvojicu nesúdeliteľných čísel. Aby sme však vedeli hľadať inverzné prvky aj pre neprvočíselné modulo, budeme potrebovať nasledovnú vetu.

Veta 1.3 (Bézoutova identita)**.** *Nech $a, b \in \mathbb{Z}$, aspoň jedno z nich je nenulové. Nech d je najväčší spoločný deliteľ čísel a, b. Potom existuje dvojica čísel $u, v \in \mathbb{Z}$ taká, že*

$$d = au + bv$$

Proof. Ak je jedno z čísel a, b nulové tak veta očividne platí. Preto predpokladajme, že obe čísla sú nenulové.

Zoberme si množinu $M = \{ax + by; x, y \in \mathbb{Z}\} \cap \mathbb{N}$. Nech $m = \min M$. Zrejme $m = au + bv$ pre nejaké $u, v \in \mathbb{Z}$. Chceme však ukázať, že $m = d$.

Keďže $d|a$ (d delí a) a $d|b$ (d delí b), tak $d|m$. A keďže d aj m sú kladné celé čísla, tak platí $d \leq m$.

Aby sme dokázali, že $d = m$, tak potrebujeme ešte dokázať opačnú nerovnosť, že $d \geq m$. Najskôr však ukážeme, že $m|a$ a $m|b$.

Vieme, že číslo a vieme zapísať ako $a = q \cdot m + r$, pričom $0 \leq r < m$, teda r je zvyšok a po delení m. Pre spor, nech m nedelí a. Potom je $r > 0$. To znamená, že $r = a - mq = a - (au + bv)q = a(1 - uq) - bvq$. Číslo $a(1 - uq) + b(-vq)$ však leží v M a zároveň je táto hodnota (r) menšia ako m, čo je spor s výberom hodnoty m. To znamená, že $r = 0$ a číslo m delí číslo a.

Rovnakým spôsobom dokážeme, že m delí číslo b. Číslo m je preto spoločný deliteľ čísel a a b. A keďže d je najväčší spoločný deliteľ, platí $d \geq m$. Spojením týchto dvoch nerovností dostaneme hľadanú rovnosť $d = m$. □

Opäť raz to vyzerá, že sme si príliš nepomohli. Veta 1.3 nám dokonca nehovorí ani to, ako takéto u a v nájsť. A ako nám to vôbec pomôže nájsť inverzné prvky?

Majme číslo n a k nemu nesúdeliteľné číslo a. Našou úlohou je nájsť inverzný prvok čísla a modulo číslo n. Z nesúdeliteľnosti čísel a a n vieme, že najväčší spoločný deliteľ týchto čísel je 1. Z Bézoutovej identity potom vieme, že existujú také dve čísla u a v, že $1 = au+nv$. Ak sa však pozrieme na túto rovnicu modulo n, dostaneme

$$1 \equiv au + nv \equiv au \pmod{n}$$

Číslo u je teda inverzný prvok čísla a modulo n. Ak by sme teda vedeli vypočítať hodnoty u a v, dostali by sme algoritmus počítajúci inverzný prvok.

Rozšírený Euklidov algoritmus

Euklidov algoritmus (alebo tiež algoritmus gcd) je algoritmus na hľadanie najväčšieho spoločného deliteľa (greatest common divisor). Ako sa ukáže, rozšírenie tohto algoritmu nám pomôže hľadať hodnoty u, v také, že $d = au + bv$. Začnime preto s tým, že zistíme, ako sa hľadá najväčší spoločný deliteľ dvoch čísel.

Lemma 1.4. *Nech $a, b \in \mathbb{Z}$, pričom $a \leq b$. Nech d je najväčší spoločný deliteľ čísel a, b. Potom d je aj najväčší spoločný deliteľ čísel a a $b - a$.*

Proof. Keďže d je (najväčší) spoločný deliteľ čísel a a b, tak $a = dx$ a $b = dy$ pre nejaké x a y. Číslo $b - a$ potom môžeme zapísať ako $b - a = d(y - x)$, čiže je tiež deliteľné číslom d. Každý spoločný deliteľ čísel a a b je preto spoločný deliteľ aj čísel a a $b - a$.

Nech d' je spoločný deliteľ čísel a a $b - a$. Potom $a = d'x$ a $b = d'y$ pre vhodné x a y. Číslo b potom vieme zapísať ako $b = d'(x + y)$. Každý spoločný deliteľ čísel a a $b - a$ je preto aj spoločný deliteľ čísel a a b.

Keďže dvojice (a, b) a $(a, b - a)$ majú rovnakých spoločných deliteľov, majú aj rovnakého najväčšieho spoločného deliteľa. □

Je jasné, že najväčší spoločný deliteľ čísel 0 a a je $|a|$ (v absolútnej hodnote). Vieme preto navrhnúť rekurzívny program, ktorý vie riešiť tento jednoduchý prípad a v prípade, že a a b sú nenulové, tak sa zavolá na dvojicu $(a, b - a)$. Je však jasné, že ak je b oveľa väčšie ako a, bude sa hodnota a odčítavať až kým b nebude menšie. A v tom prípade bude rovné hodnote $b\%a$. Náš rekurzívny program sa preto zavolá na dvojicu $(a, b\%a)$.

Listing programu (C++)

```
int gcd(int a, int b) { //najdi najvacsieho spolocneho delitela cisel a,b
    if(a>b) return gcd(b,a); //nech je cisla a mensie ako b
    if(a==0) return b;
    return gcd(b%a,a);
}
```

Aká je časová zložitosť takéhoto algoritmu? Uvedomme si nasledovnú vlastnosť – za dve rekurzívne volania sa zmenší veľkosť väčšieho z dvoch prvkov na aspoň polovicu. Ukážme si prečo. Začínajme dvojicou (a, b), pričom prvý prvok dvojice bude vždy menší ako druhý. V ďalšom rekurzívnom volaní budeme mať dvojicu $(b\%a, a)$ a v druhom volaní dvojicu $(a\%(b\%a), b\%a)$. Zaujíma nás, aká veľká je hodnota $b\%a$ oproti hodnote b.

Nech je $a \leq b/2$. Potom aj hodnota $b\%a < b/2$, keďže po vymodulovaní číslom a nemôžeme dostať číslo väčšie ako a. No a ak je $a > b/2$, tak platí $b\%a = b - a$, pretože nemôžeme odčítať hodnotu a viac ako raz. Aj v tomto prípade je výsledok $b - a$ najviac $b/2$.

Počas dvoch rekurzívnych volaní sa hodnota väčšieho čísla zmenší na aspoň polovicu. To znamená, že nemôžeme spraviť viac ako $2 \log b$ rekurzívnych volaní. Časová zložitosť gcd algoritmu je preto $O(\log b)$.

Ostáva nám rozšíriť tento algoritmus, aby počítal hodnoty u, v také, že $\gcd(a, b) = au + bv$. Uvedomme si, že ak $a = 0$, tak najväčší spoločný deliteľ je číslo b. Preto je vhodná dvojica (u, v) dvojica $(0, 1)$. Ako sa budeme vynárať z rekurzie algoritmu gcd, budeme preto postupne prepočítavať hodnoty (u, v).

Predstavme si, že počítame najväčší spoločný deliteľ pre dvojicu (a, b), pričom vieme, že a je menšie ako b. Rekurzívne sme sa zavolali na dvojicu $(b\%a, a)$ a vypočítali sme hodnoty u' a v' také, že $\gcd(a, b) = (b\%a)u' + av'$. Uvedomme si, že

$$b\%a = b - a(b \div a)$$

kde $\div$ je celočíselné delenie zaokrúhľujúce nadol. Nahraďme pravou stranou hodnotu $b\%a$ v predchádzajúcej rovnici. Dostaneme

$$\gcd(a,b) = (b - a(b \div a))u' + av' = bu' + a(v' - u'(b \div a))$$

Z toho vyplýva, že dvojica $(v' - u'(b \div a), u')$ je nami hľadaná dvojica (u, v), že $\gcd(a,b) = ua + bv$. Túto myšlienku potom môžeme veľmi jednoducho použiť v algoritme gcd, čím dostaneme rozšírený algoritmus gcd, ktorý môžeme použiť na počítanie inverzných prvkov, ako si môžete pozrieť v nasledujúcom programe.

Listing programu (C++)

```
pair<int,pair<int,int> > egcd(int a, int b) { //prve cislo je gcd, druha dvojica koeficienty u,v
    pair<int,pair<int,int> > res;
    if(a>b) {
        res = egcd(b,a);
        return {res.first,{res.second.second,res.second.first}};
    }
    if(a==0) return {b,{0,1}};
    res = egcd(b%a,a);
    return {res.first,{res.second.second-res.second.first*(b/a),res.second.first}};
}

int inverz(int a, int n) { //inverzny prvok k a modulo n
    pair<int,pair<int,int> > res = egcd(a,n);
    if(res.first != 1) return -1; //prvok a nema inverzny prvok
    return res.second.first%n;
}
```

Čínska zvyšková veta

Posledná veta, ktorú si ukážeme v tejto prednáške je Čínska zvyšková veta (Chinese remainder theorem), ktorá svoje meno získala preto, lebo sa prvýkrát vyskytla v knihe čínskeho matematika Sun Tzua. Je to pomerne užitočná veta, ktorá hovorí o riešení niektorých sústav kongruencií.

Veta 1.5 (Čínska zvyšková veta)**.** *Nech $m_1, \dots m_n$ sú po dvoch nesúdeliteľné čísla. Nech $b_1, \dots b_n \in \mathbb{Z}$. Potom systém kongruencií*

$$\begin{aligned} x &\equiv b_1 \pmod{m_1} \\ x &\equiv b_2 \pmod{m_2} \\ &\vdots \\ x &\equiv b_n \pmod{m_n} \end{aligned}$$

má práve jedno riešenie modulo $m_1 \cdot m_2 \dots m_n$.

Proof. Označme si $m = m_1 \cdot m_2 \dots m_n$ a $M_i = \frac{m}{m_i}$ pre všetky i. Je jasné, že pre $i \neq j$ platí $m_j | M_i$, pretože súčin M_i v sebe obsahuje aj m_j. Avšak, M_i neobsahuje m_i (lebo sme ho ním vydelili) a všetky zvyšné m_j sú s m_i nesúdeliteľné. Z toho vyplýva, že aj m_i a M_i sú navzájom nesúdeliteľné.

Podľa lemmy 1.1 preto existuje inverzný prvok k číslu M_i modulo m_i. Označme si tento inverzný prvok c_i. Platí teda, že $c_i M_i \equiv 1 \pmod{m_i}$, a teda $c_i M_i b_i \equiv b_i \pmod{m_i}$.

Dostávame

$$\begin{aligned} c_i M_i b_i &\equiv b_i \pmod{m_i} \\ c_i M_i b_i &\equiv 0 \pmod{m_j} \end{aligned}$$

pre $j \neq i$, pretože M_i je deliteľné m_j. Teraz stačí tieto riešenia pospájať.

Nech $x_0 = \sum_{i=1}^{n} c_i M_i b_i$. Z vyššie uvedených kongruencií ale vyplýva, že $x_0 \equiv b_i \pmod{m_i}$ pre všetky $i = 1, \dots n$. Takto zvolené x_0 je skutočne riešením našej sústavy.

Ešte potrebujeme dokázať jednoznačnosť tohto riešenia. Predpokladajme, že máme dve rôzne riešenia x_0 a x_1, pre ktoré platí $x_0 \equiv x_1 \equiv b_i \pmod{m_i}$ pre všetky $i = 1, \dots n$. Potom platí, že $m_i | x_0 - x_1$. Keďže sú však všetky m_i navzájom nesúdeliteľné, tak platí aj $m_1 \cdot m_2 \dots m_n | x_0 - x_1$ a preto sú tieto čísla rovnaké modulo m, čo je spor. □

Veta 1.5 sa dá dokázať aj iným spôsobom. Tento je však pekný preto, lebo je to dôkaz konštrukčný. Nielenže nám hovorí, že dané riešenie existuje, ale ukazuje nám, ako také riešenie aj vypočítať.

Vidíme, že pre danú sústavu kongruencií stačí sčítať hodnoty $c_i M_i b_i$. Hodnoty b_i sú pritom zadané, M_i vieme vypočítať ako súčin vhodných m_i a hodnota c_i je inverzný prvok M_i modulo m_i. A počítať inverzné prvky sme sa naučili v predchádzajúcich častiach. Je preto jednoduché napísať program riešiaci zadanú sústavu kongruencií.

Listing programu (C++)

```
int crt(vector<int> B, vector<int> M) { //pole hodnot b_i a m_i
    int n=B.size();
    int m=1;
    for(int i=0; i<n; i++) m*=M[i];
    int res=0;
    for(int i=0; i<n; i++) res+=B[i]*(m/M[i])*inverz(m/M[i],M[i]);
    return res;
}
```

Čínska zvyšková veta sa používa v rôznych situáciách. Veľmi často ju nájdete pri šifrovaní. Ďalším použitím je optimalizácia časovej zložitosti. Urobiť veľa aritmetických operácií s veľkými číslami je totiž značne pomalé. Preto sa občas oplatí nepamätať si celé číslo, ale niekoľko zvyškov po delení vhodne malými číslami, s týmito zvyškami spraviť všetky aritmetické operácie a výsledok na konci len poskladať z týchto zvyškov pomocou Čínskej zvyškovej vety.

Takáto optimalizácia sa naozaj používa napríklad v RSA šifrovaní, kde sa výsledok počíta modulo n, ktoré vznikne ako násobok dvoch prvočísel p a q. Preto sa na počítanie používajú zvyšky po týchto dvoch prvočíslach a výsledok sa skladá až na konci.