Redundantní číselné soustavy pro pokročilé

Jakub „Roman“ Klemsa

Abstrakt. Problém, jak zapisovat čísla, se táhne s lidstvem od nepaměti. Dnes známý poziční zápis, kdy stejný symbol (např. „5“) použitý na různých místech umožňuje zapsat jednou pět, jednou padesát, byl podmíněn vynálezem symbolu „0“ pro „nic“. Každému je jasné, že úloha vynásobení čísel 42 a 27 zapsaných v desítkové soustavě je jednodušší, než násobení XLII a XXVII zapsaných římským způsobem. Přesto se desítkový zápis prosadil v Evropě až ve 13. století.

I dnes má smysl číselné soustavy zkoumat. Jejich renesance přišla s nástupem procesorů – důvodem byla potřeba vyvinout algoritmus pro paralelní sčítání.

Definice. Poziční numerační systém je určen bazí $\beta \in \mathbb{R}$, $\beta > 1$, a konečnou množinou celočíselných cifer nazývanou abeceda $\mathcal{A} \subset \mathbb{Z}$. β-rozvojem čísla x pak rozumíme

$$x = \sum_{k=-\infty}^{n} x_k \beta^k, \quad x_k \in \mathcal{A}.$$

„Naše“ desítková soustava má bázi $\beta = 10$ a abecedu $\mathcal{A} = \{0, 1, \ldots, 9\}$. Ukážeme si dva algoritmy, jak čísla mezi standardními soustavami převádět.

První algoritmus využívá modula. Dá se použít jen na celá čísla a měl by v pseudokódu těžkopádný zápis, ukážeme si ho proto jen na přednášce.

Algoritmus. (Hladový)
`najdi` k`, aby platilo` $\beta^k \leq x < \beta^{k+1}$ `(nespecifikováno jak)`
`repeat`
  $x_k := \lfloor \frac{x}{\beta^k} \rfloor$
  $y := x - x_k \beta^k$
  $k := k - 1$
`until` $y = 0$

Cvičení.

(1) Převeďte $(10010110111)_2$ do osmičkové soustavy. Dokážete to bez myšleného převodu do desítkové? (malá finta)
(2) Zapište prvních n číslic čísla π v sedmičkové soustavě. Který algoritmus použijete?

Základ soustavy nemusí být nutně celočíselný. Dostáváme hned několik otázek:

(1) Jakou volit abecedu, aby byl možný zápis všech reálných čísel?
(2) V kladném případě, bude zápis jednoznačný?

Odpovědi zní:

(1) Pokud zvolíme standardní abecedu nejbližšího vyššího přirozeného čísla, zápis možný bude (hladový algoritmus ho najde). Pokud zvolíme o jedna menší, najdou se čísla, která nepůjdou zapsat (hladový algoritmus se „trefí“ mimo abecedu, de facto ji vynucuje).
(2) Nebude!

Důkaz. Provedeme na přednášce.

Dostali jsme třídu soustav, ve kterých mají některá čísla nejednoznačný zápis! Jako konkrétní příklad si uveďme soustavu o základu $\beta^2 = 9\beta + 1 \Rightarrow 9 < \beta < 10$. Máme tedy abecedu $\mathcal{A} = \{0, 1, \dots, 9\}$ a můžeme psát $\beta^2 = (1\,0\,0\,\bullet)_\beta = (9\,1\,\bullet)_\beta$. Tento „jev“ můžeme vytvořit i pro přirozené báze, stačí uměle zvětšit abecedu.

Cvičení. Zapište číslo 10 v soustavě o základu 4 s abecedou $\mathcal{A} = \{-2, -1, 0, 1, 2\}$. Najdete více možných zápisů?

Paralelní sčítání jako lokální funkce

Definice. Buďte $\mathcal{A}$, $\mathcal{B}$ abecedy a $\mathcal{A}^{\mathbb{Z}}$, $\mathcal{B}^{\mathbb{Z}}$ množiny slov na těchto abecedách. Buďte $r, t \in \mathbb{N}_0$, $p = r + t + 1 \in \mathbb{N}$ a funkce $\Phi\colon \mathcal{B}^p \to \mathcal{A}$. Funkce $\varphi\colon \mathcal{B}^{\mathbb{Z}} \to \mathcal{A}^{\mathbb{Z}}$, $\varphi(u) = v$, kde $v_j = \Phi(u_{j+t} \dots u_j \dots u_{j-r})$, se nazývá p-lokální funkce.

Uvažujme konečné β-reprezentace čísel v abecedě $\mathcal{A}$:

$$\mathrm{Fin}_{\mathcal{A}}(\beta) = \left\{ x = \sum\nolimits_{k \in I} x_k \beta^k \;\middle|\; I \subset \mathbb{Z},\ |I| < \infty,\ x_k \in \mathcal{A} \right\}.$$

Hledáme algoritmus, který umožní sečíst dvě taková čísla v konstantním čase:

$$\sum_{k \in I_1} (\underbrace{x_k + y_k}_{\in \mathcal{A}+\mathcal{A}=\mathcal{B}}) \beta^k = \sum_{k \in I_1} \underbrace{u_k}_{\in \mathcal{B}} \beta^k = \sum_{k \in I_2} \underbrace{v_k}_{\in \mathcal{A}} \beta^k. \qquad (\clubsuit)$$

K tomu potřebujeme znát p-lokální funkci φ takovou, že funkce Φ (dle definice) přiřadí p-tici dvojic cifer $(u_{j+t} \dots u_j \dots u_{j-r})$ cifru v_j tak, aby platila rovnost ($\clubsuit$) a $v_j \in \mathcal{A}$. Na příkladu si ukážeme, že standardní číselné soustavy nám nestačí.

$$\begin{array}{r} 1999\ldots99999 \\ \underline{0000\ldots00001} \\ 20000\ldots00000 \end{array}$$

Na vznik dvojky na začátku součtu má vliv každá cifra horního čísla. Takové situaci se chceme vyhnout. Algoritmus, jak to udělat, byl objeven až při použití redundantní

číselné soustavy. První takový algoritmus navrhl Avizienis r. 1961. Pracoval s desítkovou soustavou a abecedou $\{-6,-5,\dots,5,6\}$. Své aplikace se podobný algoritmus dočkal v aritmetických jednotkách procesorů, například Pentium používá soustavu se základem $\beta = 4$ a pěti ciframi $\mathcal{A} = \{-2,-1,0,1,2\}$. Nyní ale zpět k algoritmu.

Mějme $x, y \in \mathrm{Fin}_{\mathcal{A}}(4)$. Cílem je spočítat $z \in \mathrm{Fin}_{\mathcal{A}}(4)$ tak, aby platilo

$$\sum x_i 4^i + \sum y_i 4^i = \sum z_i 4^i.$$

Algoritmus. (♠)

```
for i in I_1 ∪ I_2 do
   w_i := x_i + y_i
   if (w_i ≥ 3 or (w_i = 2 and w_{i-1} ≥ 2))
      q_i := 1
   elseif (w_i ≤ −3 or (w_i = −2 and w_{i-1} ≤ −2))
      q_i := −1
   else
      q_i := 0
   end
   z_i := w_i − 4q_i + q_{i-1}
end
```

Cvičení.

(1) Ověřte, že $z_i \in \mathcal{A}$.
(2) Určete p-lokální funkci z algoritmu (♠). Jaké je p?

Dané číslo však musíme nejdříve do nové soustavy převést. To lze provést paralelně. Číslo rozložíme na součet dvou čísel tak, že 0, 1 a 2 zůstanou, 3 rozepíšeme na 1 a 2. Tím se oběma čísly dostaneme do naší abecedy, v níž paralelně sčítat umíme. Převod zpět už paralelně provést nelze.

Dále potřebujeme umět čísla porovnávat, což na první pohled paralelně nelze. Běžné lexikografické porovnávání v tomto případě selže – například $(1\,\overline{2}\,\overline{2}\,\bullet)_{4,\mathcal{A}} < (2\,2\,\bullet)_{4,\mathcal{A}}$, protože $6 < 10$. Čísla musíme nejdříve paralelně odečíst a pak už první cifra rozhodne.

Literatura a zdroje

[1] E. Pelantová, Š. Starosta, *Nestandardní zápisy čísel*, Pokroky matematiky, fyziky a astronomie, JČMF, Praha, 2011,
[2] E. Pelantová, *prezentace Redundantní číselné soustavy*, FJFI ČVUT, Praha, 2011.