

11 Nekonečné množiny a zastavení algoritmu

Bystrého čtenáře může snadno napadnout myšlenka, proč se vlastně zabýváme dokazováním zprávnosti algoritmů a programů, když by to přece (snad?) mohl za nás dělat automaticky počítač samotný.

Bohužel to však nejde a je hlavním cílem této lekce ukázat důvody proč. Konkrétně si dokážeme, že nelze algoritmicky rozhodnout, zda se daný algoritmus na svém vstupu zastaví nebo ne.

	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	...
<i>g(a)</i>	✓	–	–	✓	...
<i>g(b)</i>	✓	–	–	✓	...
<i>g(c)</i>	–	✓	–	✓	... □
<i>g(d)</i>	–	–	✓	✓	...
⋮	⋮	⋮	⋮	⋮	

Stručný přehled lekce

- * Nekonečné množiny, jejich kardinalita a Cantorova diagonála.
- * „Naivní“ množinové paradoxy: Cantorův a Russelův.
- * Důkaz algoritmické neřešitelnosti problému zastavení programu.

11.1 O kardinalitě a nekonečných množinách

Definice: Množina A je „nejvýše tak velká“ jako množina B , právě když existuje injektivní funkce $f : A \rightarrow B$. \square

Množiny A a B jsou „stejně velké“ právě když mezi nimi existuje bijekce. \square

V případech nekonečných množin místo „velikosti“ mluvíme formálně o jejich *kardinalitě*.

Tyto definice kardinality množin „fungují“ dobře i pro nekonečné množiny.

- * Například \mathbb{N} a \mathbb{Z} mají stejnou kardinalitu („stejně velké“), tzv. *spočetně nekonečné*. \square
- * Lze snadno ukázat, že i \mathbb{Q} je spočetně nekonečná, tj. existuje bijekce $f : \mathbb{N} \rightarrow \mathbb{Q}$. \square
- * Existují ale i nekonečné množiny, které jsou „striktně větší“ než libovolná spočetná množina (příkladem je \mathbb{R}). \square
- * Později dokážeme, že *existuje nekonečná posloupnost nekonečných množin, z nichž každá je striktně větší než všechny předchozí*.

Věta 11.1. Neexistuje žádné surjektivní (ani bijektivní) zobrazení $g : \mathbb{N} \rightarrow \mathbb{R}$. \square

Důkaz sporem. Necht' takové g existuje a pro zjednodušení se omezme jen na funkční hodnoty v intervalu $(0, 1)$. Podle hodnot zobrazení g si takto můžeme „uspořádat“ dekadické zápisy **všech reálných** čísel v intervalu $(0, 1)$ po řádcích do tabulky:

$$\begin{array}{rcccccccccccc} g(0) = 0. & 1 & 5 & 4 & 2 & 7 & 5 & 7 & 8 & 3 & 2 & 5 & \dots \\ g(1) = 0. & & 2 & & & & & & & & & & \dots \\ g(2) = 0. & & & 1 & & & & & & & & & \dots \\ g(3) = 0. & & & & 3 & & & & & & & & \dots \quad \square \\ g(4) = 0. & & & & & 9 & & & & & & & \dots \\ \vdots & \vdots & & & & & \ddots & & & & & & \dots \end{array}$$

Nyní sestrojíme číslo $\alpha \in (0, 1)$ následovně; jeho i -tá číslice za desetinnou čárkou bude 1, pokud v i -tém řádku tabulky na diagonále není 1, jinak to bude 2. V našem příkladě $\alpha = 0.21211\dots$ \square

Kde se naše číslo α v tabulce nachází? (Nezapomeňme, g byla surjektivní, takže tam α musí být!) Konstrukce však ukazuje, že α se od každého čísla v tabulce liší na aspoň jednom desetinném místě, to je spor.

(Až na drobný technický detail s rozvojem $\dots\bar{9}$.) \square

V obecnosti lze dokonce analogickým způsobem dokázat následovně.

Věta 11.2. *Bud' M libovolná množina. Pak existuje injektivní zobrazení $f : M \rightarrow 2^M$, ale neexistuje žádné bijektivní zobrazení $g : M \rightarrow 2^M$.* \square

Důkaz: Dokážeme nejprve existenci f . Stačí ale položit $f(x) = \{x\}$ pro každé $x \in M$. Pak $f : M \rightarrow 2^M$ je zjevně injektivní. \square

Neexistenci g dokážeme sporem. Předpokládejme tedy naopak, že existuje bijekce $g : M \rightarrow 2^M$. Uvažme množinu $K \subseteq M$ definovanou takto:

$$K = \{x \in M \mid x \notin g(x)\}.$$

Jelikož g je bijektivní a $K \in 2^M$, musí existovat $x \in M$ takové, že $g(x) = K$.

\square Nyní rozlišíme dvě možnosti:

- $x \in g(x)$. Tj. $x \in K$. Pak ale $x \notin g(x)$ z definice K , spor.
- $x \notin g(x)$. To podle definice K znamená, že $x \in K$, tj. $x \in g(x)$, spor. \square

Dvě navazující poznámky.

- Z toho, že nekonečna mohou být „různě velká“, lze lehce odvodit řadu dalších faktů. V jistém smyslu je např. množina všech „problémů“ větší než množina všech algoritmů (obě množiny jsou nekonečné), proto nutně existují problémy, které **nejsou algoritmicky řešitelné**. □
- Technika použitá v důkazech **Vět 11.1 a 11.2** se nazývá **Cantorova diagonální metoda**, nebo také zkráceně **diagonalizace**.

Konstrukci množiny K lze znázornit pomocí následující tabulky:

	a	b	c	d	\dots
$g(a)$	✓	–	–	✓	\dots
$g(b)$	✓	–	–	✓	\dots
$g(c)$	–	✓	–	✓	\dots
$g(d)$	–	–	✓	✓	\dots
\vdots	\vdots	\vdots	\vdots	\vdots	

Symbol ✓ resp. – říká, že prvek uvedený v záhlaví sloupce patří resp. nepatří do množiny uvedené v záhlaví řádku. Tedy např. $d \in g(b)$ a $a \notin g(d)$.

„Naivní“ množinové paradoxy

Příklad 11.3. *Uvážíme-li nyní nekonečnou posloupnost množin*

$$A_1, A_2, A_3, \dots$$

kde $A_1 = \mathbb{N}$ a $A_{i+1} = 2^{A_i}$ pro každé $i \in \mathbb{N}$, je vidět, že všechny množiny jsou nekonečné a každá je „striktně větší“ než libovolná předchozí. \square

Kde však v tomto řazení mohutností bude „množina všech množin“? \square

Takto se koncem 19. století objevil první **Cantorův paradox** nově vznikající teorie množin. \square

(Dnešní vysvětlení je jednoduché, prostě „množinu všech množin“ **nelze definovat**, prostě v matematice neexistuje.) \square

Brzy se však ukázalo, že je ještě **mnohem hůř**...

Russelův paradox

Fakt: Není pravda, že **každý soubor prvků** lze považovat za množinu.

- $X = \{M \mid M \text{ je množina taková, že } M \notin M\}$. □ Platí $X \in X$? □
 - * Ano. Tj. $X \in X$. Pak ale X splňuje $X \notin X$. □
 - * Ne. Pak X splňuje vlastnost $X \notin X$, tedy X je prvkem X , tj., $X \in X$. □
- Obě možné odpovědi vedou ke sporu. X tedy **nelze** prohlásit za množinu.

Vidíte zde podobnost přístupu s Cantorovou diagonalizací? Russelův paradox se objevil začátkem 20. století a jeho „jednoduchost“ zasahující úplné základy matematiky (teorie množin) si vynutila hledání seriózního rozřešení a rozvoj formální matematické logiky.

□

Pro ilustraci tohoto paradoxu, slyšeli jste už, že „**v malém městečku žije holič, který holí právě ty muže městečka, kteří se sami neholí**“? □

Zmíněné **paradoxy naivní teorie množin** zatím v tomto kurzu nerozřešíme, ale zapa-
matujeme si, že většina matematických a inforatických disciplín vystačí s „**intuitivně bezpečnými**“ množinami.

11.2 Algoritmická neřešitelnost problému zastavení

Fakt: Uvědomme si (velmi neformálně) několik základních myšlenek.

- Program v každém programovacím jazyce je konečná posloupnost složená z **konečně mnoha** symbolů (písmena, číslice, mezery, speciální znaky, apod.) Necht' Σ je množina všech těchto symbolů. Množina všech programů je tedy jistě podmnožinou množiny $\bigcup_{i \in \mathbb{N}} \Sigma^i$, která je spočetně nekonečná. Existuje tedy bijekce f mezi množinou \mathbb{N} a množinou všech programů. Pro každé $i \in \mathbb{N}$ označme symbolem P_i program $f(i)$. Pro **každý program** P tedy existuje $j \in \mathbb{N}$ takové, že $P = P_j$. \square
- Každý možný vstup každého možného programu lze zapsat jako **konečnou posloupnost** symbolů z konečné množiny Γ . Množina všech možných vstupů je tedy spočetně nekonečná a existuje bijekce g mezi množinou \mathbb{N} a množinou všech vstupů. Pro každé $i \in \mathbb{N}$ označme symbolem V_i vstup $g(i)$. \square
- Předpokládejme, že **existuje program** *Halt*, který pro dané $i, j \in \mathbb{N}$ zastaví s výstupem *ano/ne* podle toho, zda P_i pro vstup V_j zastaví, nebo ne.
- Tento předpoklad dále dovedeme ke **sporu** dokazujícímu, že problém zastavení nemá algoritmické řešení.

Tvrzení 11.4. Předpoklad existence programu *Halt* vede ke sporu.

Důkaz: Uvažme program *Diag* s následujícím kódem:

```
input k;  
if Halt(k,k) = ano then while true do ; done□
```

(Program $Diag(k)$ má na rozdíl od *Halt* jen jeden vstup k , což bude důležité.)

Fungování programu *Diag* lze znázornit za pomoci následující tabulky:

	P_1	P_2	P_3	P_4	\dots
V_1	✓	—	—	✓	\dots
V_2	✓	—	—	✓	\dots
V_3	—	✓	—	✓	\dots
V_4	—	—	✓	✓	\dots
\vdots	\vdots	\vdots	\vdots	\vdots	\ddots

Symbol ✓ resp. — říká, že program uvedený v záhlaví sloupce zastaví resp. nezastaví pro vstup uvedený v záhlaví řádku. Program *Diag* „zneguje“ diagonálu této tabulky.

Podle našeho předpokladu (*Diag* je program a posloupnost P_i zahrnuje **všechny programy**) existuje $j \in \mathbb{N}$ takové, že $Diag = P_j$. \square

Zastaví *Diag* pro vstup V_j ? \square

- **Ano**. Podle kódu *Diag* pak ale tento program vstoupí do nekonečné smyčky, tedy **nezastaví**. \square
- **Ne**. Podle kódu *Diag* pak ale **if** test neuspěje, a tento program tedy **zastaví**. \square

Předpoklad existence programu *Halt* tedy vede ke sporu. \square

Otázkami algoritmické (ne)řešitelnosti problémů se zabývá **teorie vyčíslitelnosti**. \square

Metoda diagonalizace se také často využívá v **teorii složitosti** k důkazu toho, že dané dvě složitostní třídy jsou různé.