11 Nekonečné množiny a zastavení algoritmu

Bystrého čtenáře může snadno napadnout myšlenka, proč se vlastně zabýváme dokazováním správnosti 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.

	a	b	c	d	$\cdots$
$g(a)$	$\surd$	$-$	$-$	$\surd$	$\cdots$
$g(b)$	$\surd$	$-$	$-$	$\surd$	$\cdots$
$g(c)$	$-$	$\surd$	$-$	$\surd$	$\cdots$
$g(d)$	$-$	$-$	$\surd$	$\surd$	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	

□

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 $f : A \to B$. □

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

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*).□
- Lze snadno ukázat, že i $\mathbb{Q}$ je spočetně nekonečná, tj. existuje bijekce $f : \mathbb{N} \to \mathbb{Q}$.□
- Existují ale i nekonečné množiny, které jsou „striktně větší“ než libovolná spočetná množina (příkladem je $\mathbb{R}$).□
- 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* $g : \mathbb{N} \to \mathbb{R}$. □

Důkaz sporem. Nechť 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:

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

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 $\alpha = 0.21211\ldots$ □

Kde se naše číslo α v tabulce nachází? (Nezapomeňme, g byla surjektivní, takže tam α musí být!) Kostrukce 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 $\ldots \overline{9}.$) □

V obecnosti lze dokonce analogickým způsobem dokázat následovné.

Věta 11.2. *Buď M libovolná množina. Pak existuje injektivní zobrazení $f : M \to 2^M$, ale neexistuje žádné bijektivní zobrazení $g : M \to 2^M$.*□

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

Neexistenci g dokážeme sporem. Předpokládejme tedy naopak, že existuje bijekce $g : M \to 2^M$. 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 $g(x) = K$. □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.

□

Dvě navazující poznámky.

- 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	$\cdots$
$g(a)$	$\surd$	$-$	$-$	$\surd$	$\cdots$
$g(b)$	$\surd$	$-$	$-$	$\surd$	$\cdots$
$g(c)$	$-$	$\surd$	$-$	$\surd$	$\cdots$
$g(d)$	$-$	$-$	$\surd$	$\surd$	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	

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

- 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.

„Naivní“ množinové paradoxy

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

$$A_1, A_2, A_3, \cdots$$

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.* □

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

Takto se koncem 19. století objevil první Cantorův paradox nově vznikající teorie množin. □

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

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

Russelův paradox

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

- $X = \{M \mid M$ 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 zapamatujeme si, že většina matematických a informatických 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.) Nechť 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 $P = P_j$. □
- 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 $g(i)$. □
- 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 pomocí následující tabulky:

	P_0	P_1	P_2	P_3	$\cdots$
V_0	$\surd$	$-$	$-$	$\surd$	$\cdots$
V_1	$\surd$	$-$	$-$	$\surd$	$\cdots$
V_2	$-$	$\surd$	$-$	$\surd$	$\cdots$
V_3	$-$	$-$	$\surd$	$\surd$	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Symbol $\surd$ 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 $Diag = P_j$. □

Zastaví $Diag$ pro vstup V_j? □

- Ano. Podle kódu $Diag$ pak ale tento program vstoupí do nekonečné smyčky, tedy nezastaví. □
- Ne. Podle kódu $Diag$ pak ale `if` test neuspěje, a tento program tedy zastaví. □

Předpoklad existence programu $Halt$ tedy vede ke sporu. □

Otázkami algoritmické (ne)řešitelnosti problémů se zabývá teorie vyčíslitelnosti. □

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