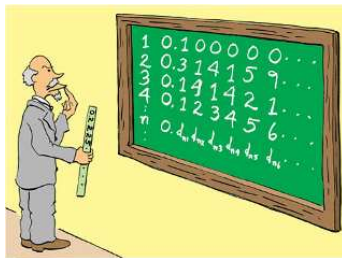


# 11 Nekonečné množiny, Zastavení algoritmu

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.



## Stručný přehled lekce

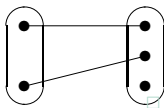
- \* Nekonečné množiny, jejich „velikost“ a Cantorova diagonála.
- \* „Naivní“ množinové paradoxy: Cantorův a Russelův.
- \* Důkaz algoritmické neřešitelnosti problému zastavení programu.

## 11.1 O nekonečných množinách a kardinalitě

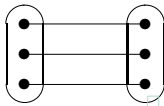
Uvažme problém stanovit, **kolik** má nekonečná množina prvků. Co nám třeba brání zavést pro velikost nekonečné množiny **symbol**  $\infty$ ?  $\square$

– Ponejvíce závažný fakt, že není „nekonečno“ jako „nekonečno“ (Věta 11.2)!  $\square$

**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.  $\square$

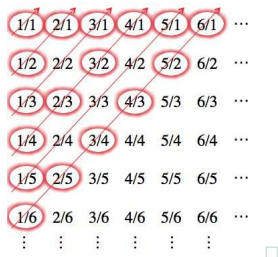


V případech nekonečných množin pak mluvíme formálně o jejich **kardinalitě**.

## Ukázky kardinalit množin

Uvedená definice kardinality množin funguje korektně i pro nekonečné množiny:

- \* Například  $\mathbb{N}$  a  $\mathbb{Z}$  mají stejnou kardinalitu (jsou „stejně velké“, tzv. *spočetně nekonečné*). $\square$
- \* Lze snadno ukázat, že i  $\mathbb{Q}$  je spočetně nekonečná, tj. existuje bijekce  $f : \mathbb{N} \rightarrow \mathbb{Q}$ , stejně jako bijekce  $h : \mathbb{N} \rightarrow \mathbb{N}^2$ .

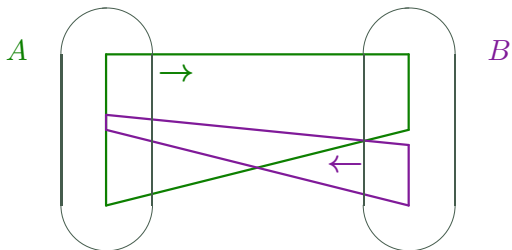


- \* Existují ale i nekonečné množiny, které jsou „striktně větší“ než libovolná spočetná množina (příkladem je  $\mathbb{R}$  ve Větě 11.2). $\square$
- \* Později dokážeme, že existuje nekonečná posloupnost nekonečných množin, z nichž *každá je striktně větší než všechny předchozí*.

## Zjednodušeně místo bijekce

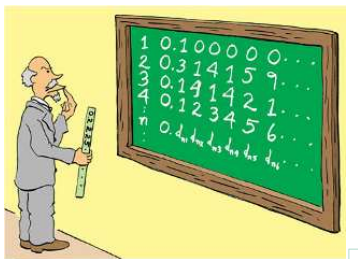
Pro porovnávání velikostí množin někdy s výhodou využijeme následující přirozené, ale nelehké tvrzení (bez důkazu):

**Věta 11.1.** Pro libovolné dvě množiny  $A, B$  platí, že pokud existuje injekce  $A \rightarrow B$  a zároveň i injekce  $B \rightarrow A$ , pak *existuje bijekce* mezi  $A$  a  $B$ .



## Cantorova diagonála, aneb kolik je reálných čísel

**Věta 11.2.** *Neexistuje žádné surjektivní zobrazení  $g : \mathbb{N} \rightarrow \mathbb{R}$ .*



**Důsledek 11.3.** *Neexistuje žádné injektivní (ani bijektivní) zobraz.  $h : \mathbb{R} \rightarrow \mathbb{N}$ . Neformálně řečeno, reálných čísel je striktně více než všech přirozených.*

Věta. Neexistuje žádné surjektivní zobrazení  $g : \mathbb{N} \rightarrow \mathbb{R}$ .

**Důkaz** (Věty 11.2 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:

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

□  
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  $\alpha = \mathbf{0.21211\dots}$  □

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

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

## 11.2 „Naivní“ množinové paradoxy

Analogickým způsobem k Větě 11.2 lze dokázat následovné zobecnění.

**Věta 11.4.** *Nechť  $M$  je libovolná množina. Pak existuje injektivní zobrazení  $f : M \rightarrow 2^M$ , ale neexistuje žádné bijektivní zobrazení  $g : M \rightarrow 2^M$ . □*

**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. □

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  $y \in M$  takové, že  $g(y) = K$ . □  
Nyní rozlišíme dvě možnosti:

- $y \in g(y)$ . Tj.  $y \in K$ . Pak ale  $y \notin g(y)$  z definice  $K$ , což je spor.
- $y \notin g(y)$ . To podle definice  $K$  znamená, že  $y \in K$ , tj.  $y \in g(y)$ , spor. □

Dvě navazující poznámky.

- Technika použitá v důkazech **Vět 11.2 a 11.4** se nazývá *Cantorova diagonální metoda*, nebo také zkráceně *diagonalizace*. □

Konstrukci množiny  $K$  lze znázornit pomocí následující tabulky:

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

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  $a \notin g(d)$ . □

- 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), a proto nutně existují problémy, které **nejsou algoritmicky řešitelné**.



## Cantorův paradox

**Příklad 11.5.** *Uvážíme-li nyní nekonečnou posloupnost množin*

$$A_1, A_2, A_3, A_4, \dots$$

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 kardinalit bude „množina všech množin“?  $\square$  Na tuto otázku, jak sami asi cítíte, nelze podat odpověď. Co to však znamená?  $\square$

- \* Takto se koncem 19. století objevil první **Cantorův paradox** teorie množin.  $\square$
- \* Dnešní moderní vysvětlení paradoxu je jednoduché, prostě „množinu všech množin“ **nelze definovat**, taková 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.

- Necht'  $X = \{M \mid M \text{ je množina taková, že } M \notin M\}$ . □ 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. Jak je ale něco takového vůbec možné?

Vidíte u Russelova paradoxu podobnost přístupu s Cantorovou diagonalizací? Russelův paradox se objevil začátkem 20. století a jeho „jednoduchost“ zasahující úplně 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 inženýrských disciplín vystačí s „**intuitivně bezpečnými**“ množinami.

## 11.3 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'  $\Sigma$  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  $j \in \mathbb{N}$  označme symbolem  $P_j$  program  $f(j)$ . Pro **každý program**  $P$  tedy existuje  $i \in \mathbb{N}$  takové, že  $P = P_i$ .  $\square$
- Každý možný vstup každého možného programu lze zapsat jako **konečnou posloupnost** symbolů z konečné množiny  $\Gamma$ . Množina všech možných vstupů je tedy spočetně nekonečná a existuje bijekce  $g$  mezi množinou  $\mathbb{N}$  a množinou všech vstupů. Pro každé  $j \in \mathbb{N}$  označme symbolem  $V_j$  vstup  $g(j)$ .  $\square$
- 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.

**Věta 11.6.** Neexistuje program *Halt*, který by pro vstup  $(P_i, V_j)$  správně rozhodl, zda se program  $P_i$  zastaví na vstupu  $V_j$ .

**Důkaz:** Sporem uvažme program *Diag* s následujícím kódem:

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

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

	$P_0$	$P_1$	$P_2$	$P_3$	...
$V_0$	✓	—	—	✓	...
$V_1$	✓	—	—	✓	...
$V_2$	—	✓	—	✓	...
$V_3$	—	—	✓	✓	...
⋮	⋮	⋮	⋮	⋮	⋮

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

```

if  Halt(k,k) = ano  then
    while true do ; done
fi

```

	$P_0$	$P_1$	$P_2$	$P_3$	...
$V_0$	✓	–	–	✓	...
$V_1$	✓	✓	–	✓	...
$V_2$	–	✓	✓	✓	...
$V_3$	–	–	✓	✓	...
⋮	⋮	⋮	⋮	⋮	⋮

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.