# Týden 10

## Přednáška - první část

Začneme zopakováním důležitých pojmů; některé se v předchozím týdnu objevily jen na cvičení.

Připomeňme si, co to znamená rozhodnutelnost problému, v řeči „tabulek“: problém $P$ (typu Ano/Ne) je rozhodnutelný, jestliže existuje algoritmus [Turingův stroj] $M$ takový, že vstupně-výstupní tabulka $T_M$ je konzistentní s tabulkou $T_P$ (stejným vstupům odpovídají stejné výstupy).

Pro zajímavost si všimněme, že o konkrétním problému můžeme zjistit, že je rozhodnutelný, aniž jsme schopni určit konkrétní algoritmus, který jej řeší. Podívejme se např. na následující problém.

Název: 7vPi (*Sedmičky v Ludolfově čísle* $\pi$)

Vstup: Přirozené číslo $k$.

Otázka: Existuje v desetinném rozvoji čísla $\pi$ úsek sedmiček délky $k$ ?

Je jasné, že problému 7vPi odpovídá jedna z následujících tabulek.

| Vstup | Výstup |
|---|---|
| 0 | Ano |
| 1 | Ano |
| 2 | Ano |
| 3 | Ano |
| 4 | Ano |
| 5 | Ano |
| ... | ... |

| Vstup | Výstup |
|---|---|
| 0 | Ano |
| 1 | Ne |
| 2 | Ne |
| 3 | Ne |
| 4 | Ne |
| 5 | Ne |
| ... | ... |

| Vstup | Výstup |
|---|---|
| 0 | Ano |
| 1 | Ano |
| 2 | Ne |
| 3 | Ne |
| 4 | Ne |
| 5 | Ne |
| ... | ... |

| Vstup | Výstup |
|---|---|
| 0 | Ano |
| 1 | Ano |
| 2 | Ano |
| 3 | Ne |
| 4 | Ne |
| 5 | Ne |
| ... | ... |

...

Ke každé tabulce umíme snadno navrhnout příslušný algoritmus, takže problém 7vPi je jistě rozhodnutelný. Nevíme ovšem, která tabulka je ta pravá, takže nejsme schopni předložit konkrétní algoritmus a prokázat, že řeší problém 7vPi.

Na (dřívější) rozšiřující přednášce jsme mj. dokázali nerozhodnutelnost tzv. diagonálního problému zastavení:

Název: DHP (*Diagonal Halting Problem*)

Vstup: Turingův stroj $M$.

Otázka: Zastaví se (výpočet stroje) $M$, když začne pracovat na (vstupním) slově, které je kódem $M$ ?

### Převeditelnost mezi problémy

Důkladně jsme si ujasnili (i využitím tabulek problémů), co to znamená, když se řekne, že

> problém $P_1$ je *algoritmicky převeditelný* (stručněji *převeditelný*) na problém $P_2$; označujeme $P_1 \rightsquigarrow P_2$.

Ilustrovali jsme si na případu DHP$\rightsquigarrow$HP, kde HP je definován níže. Vyvodili jsme, že HP je také nerozhodnutelný (využitím tvrzení 6.11. v části 6.5.).

NÁZEV: HP (*Halting Problem*)

VSTUP: Turingův stroj $M$ a (vstupní) slovo $w$.

OTÁZKA: Zastaví se $M$ na $w$ ? (Je tedy výpočet $M$ pro vstup $w$ konečný ?)

### Částečná rozhodnutelnost, Postova věta

Připomněli jsme *částečnou rozhodnutelnost* problémů. Přitom byla podstatná následující definice, kterou zde formulujeme v řeči „tabulek“ (problému $P$ odpovídá tabulka $T_P$, stroj $M$ přirozeně definuje tabulku $T_M$):

> Turingův stroj $M$ *částečně rozhoduje* problém $P$ (typu ANO/NE), jestliže u každého vstupu, pro nějž je v $T_P$ ANO, je v tabulce $T_M$ výstup ANO a pro každý vstup, pro nějž je v $T_P$ NE, je v $T_M$ výstup NE nebo znak $\perp$ (nedefinováno).

(Pro vstupy, kterým problém $P$ přiřazuje odpověď NE, nemusí stroj $M$ svůj výpočet skončit.)

Jak se dá očekávat, o problému řekneme. že je *částečně rozhodnutelný*, jestliže existuje algoritmus (Turingův stroj), který jej částečně rozhoduje.

Speciálně jsme si uvědomili Postovu větu (Věta 7.2.)

Uvědomili jsme si také, že problém HP je částečně rozhodnutelný (viz Univerzální TS), a že tedy $\overline{\text{HP}}$ (doplňkový problém k problému HP) není (ani) částečně rozhodnutelný.

### Univerzální Turingův stroj

Algoritmus, kterým jsme prokazovali částečnou rozhodnutelnost problému zastavení (HP) byl tento: na zadaný stroj (program) $M$ a vstup $w$ spusť „interpret“, který provádí činnost (výpočet) $M$ na vstupu $w$. Takový interpret je ovšem také program, tedy algoritmus, a šlo by jej proto realizovat (naprogramovat) ve formě konkrétního Turingova stroje $U$ (viz Věta 7.3.).

### Riceova věta

Uvědomili jsme si, že každá vlastnost $V$ Turingových strojů (např. „stroj $M$ má více než 100 stavů“, „výpočet stroje $M$ je pro každý vstup konečný“, apod.) rozdělí množinu všech Turingových strojů na dvě disjunktní podmnožiny: jedna je množina strojů, které vlastnost $V$ mají, a druhá je množina strojů, které vlastnost $V$ nemají. Vlastnost $V$ je *triviální*, jestliže je jedna z oněch dvou příslušných množin prázdná (tedy buď všechny stroje vlastnost $V$ mají nebo ji nemá ani jeden). Vlastnost $V$ je *netriviální*, jestliže ji alespoň jeden stroj má a alespoň jeden stroj nemá.

Důkladně jsme si promysleli, co to je *vstupně/výstupní vlastnost*, zkráceně též *I/O vlastnost*, Turingových strojů (či obecně „programů“).

Speciálně jsme si uvědomili, že vlastnost $V$ není I/O vlastností právě tehdy, když existují dva stroje $M_1$, $M_2$, které mají stejnou I/O tabulku (tedy stejné vstupně/výstupní chování), ale jeden z nich vlastnost $V$ má a druhý ji nemá.

Probrali jsme si pak (níže uvedené) vlastnosti v řešeném příkladu 7.1. a uvědomili si, které jsou I/O vlastnostmi. Speciálně jsme si všimli, že podle definice je každá triviální vlastnost I/O vlastností.

a/ Zastaví se $M$ na řetězec 001 ?

b/ Má $M$ více než sto stavů ?

c/ Má v nějakém případě výpočet stroje $M$ více kroků než tisícinásobek délky vstupu ?

d/ Platí, že pro libovolné $n$ se $M$ na vstupech délky nejvýše $n$ vícekrát zastaví než nezastaví ?

e/ Zastaví se $M$ na každém vstupu $w$ za méně než $|w|^2$ kroků?

f/ Je pravda, že pro lib. vstupní slovo $M$ realizuje jeho zdvojení ?

g/ Je pravda, že $M$ má nejvýše sto stavů nebo více než sto stavů ?

Pak jsme se zamysleli nad Riceovou větou:

*Každá netriviální vstupně/výstupní vlastnost programů je nerozhodnutelná.*

(V rozšiřující přednášce ukážeme důkaz využitím tzv. věty o rekurzi.)

### Složitost algoritmů

Připomněli jsme si, že složitostí algoritmů většinou nemyslíme složitost jejich struktury či obtížnost jejich návrhu, ale časovou (či paměťovou) náročnost jimi definovaných výpočtů.

Přirozeně jsme navrhli chápat

*časovou složitost Turingova stroje* $M$ jako funkci $T_M : \mathbb{N} \to \mathbb{N}$

tak, že $T_M(n)$ udává počet kroků výpočtu stroje $M$ nad vstupem velikosti (tj. délky) $n$ v *nejhorším případě* (worst-case complexity).

*Poznámka.* O časové složitosti má tedy rozumný smysl hovořit jen u strojů, jejichž (všechny) výpočty jsou konečné.

Pak jsme navrhli (rámcově) Turingův stroj $M_1$ přijímající (přechodem do stavu $q_{accept}$) právě ta slova v abecedě $\{0, 1\}$, která jsou z jazyka $\{0^m 1^m \mid m \geq 1\}$.

Jednalo se o standardní jednopáskový Turingův stroj (s jednostranně nekonečnou páskou).

Při analýze jeho časové složitosti jsme si připomněli

> asymptotickou notaci $f(n) \in O(g(n))$, $f(n) \in o(g(n))$, $f(n) \in \Theta(g(n))$ (včetně běžných funkcí, které se vyskytují při analýze složitosti algoritmů)

a přišli na to, že u našeho konkrétního stroje $M_1$ platí $T_{M_1}(n) \in \Theta(n^2)$.

Pak jsme se zamysleli a přišli na nápad, jak navrhnout (standardní) stroj $M_2$ řešící tentýž problém, pro nějž jsme odvodili $T_{M_2}(n) \in \Theta(n \log n)$.

Přitom jsme si uvědomili, proč při takové analýze nezáleží na základu logaritmu (který zde bereme jako 2, pokud není řečeno jinak).

Pak jsme si ještě všimli, že umíme navrhnout *dvoupáskový* (či jen dvouhlavý) Turingův stroj $M_3$, který řeší výše uvedený problém a pro nějž je $T_{M_3}(n) \in \Theta(n)$.

Jen letmo jsme zaznamenali jako fakt, že

> mezi rozumnými výpočetními modely existují (vzájemné) polynomiální simulace,

takže třída PTIME, tedy *třída problémů řešitelných polynomiálními algoritmy* (tedy algoritmy s časovou složitostí omezenou polynomy), je nezávislá na tom, ve kterém (rozumném) výpočetním modelu (tedy ve kterém „programovacím jazyku“) algoritmy realizujeme.

Pozn. Další připravený text zde nechávám, i když jsme se k němu nedostali.

### Dynamické programování

Uvažovali jsme o problému nejdelší společné podposloupnosti ze sekce 10.2.4. Nejprve jsme celkem přímočaře sestrojili rekurzivní proceduru $LCS(u, v)$. Analýzou jsme zjistili, že počet volání $LCS$ při řešení instance $u, v$, kde $|u| = |v| = n$, může být větší než $2^n$. Navržený algoritmus tedy jistě není polynomiální.

Kladli jsme si otázku, zda se nedá navrhnout polynomiální algoritmus řešící uvedený problém. Zlepšení nás napadlo, když jsme si uvědomili, že při rekurzivním řešení se mnohé podpřípady mohou zbytečně řešit vícekrát (nezávisle na sobě). Přitom každý podpřípad stačí vyřešit jednou a řešení poznamenat do vhodné „tabulky“ (v našem případě dvourozměrného pole). Přitom postupujeme od menších podpřípadů k větším. To je princip tzv. metody *dynamického programování*. Výsledný algoritmus (také ilustrovaný jednou z animací) už polynomiální očividně je.

### Metoda „Rozděl a panuj“

Připomněli jsme si i tuto obecnou metodu návrhu (efektivních) algoritmů. Naznačili jsme si odvození řešení rekurentních rovnic užitečných při analýze složitosti rekurzivních algoritmů, jak je to probráno v části 10.2.2.

## Přednáška - druhá část

Pokračovali jsme v diskusi dalších problémů, hlavně šlo o vysvětlení znění a pak aplikace věty o rekurzi ...
(Podrobnější poznámky lze nalézt v průběhu výuky v loňském roce ... )

### Partie textu k prostudování

Univerzální Turingův stroj, Riceova věta (v části 7.). Složitost algoritmů (8.1., 8.2.). Dynamické programování (10.2.4.), část 10.2.2. (metoda „rozděl a panuj“).

## Cvičení

### Prezentace referátů 17 a 18

Ještě jedno upozornění na zápočtovou písemku (viz např. podklady z minulého týdne), případná stručná diskuse.

### Příklad 10.1

Vysvětlete, co dělá univerzální Turingův stroj $U$, když dostane jako vstup slovo tvaru $uv$, kde slovo $u$ je kódem stroje $U$.

### Příklad 10.2

Uveďte alespoň tři vlastnosti Turingových strojů, pro něž plyne nerozhodnutelnost z Riceovy věty, a alespoň tři vlastnosti, pro něž nerozhodnutelnost z Riceovy věty neplyne.

### Příklad 10.3

Připomeňte si, jak se používá a co vyjadřuje značení $O$, $o$, $\Theta$.
Pak seřaďte následující tři funkce podle rychlosti jejich růstu.
a/ $n/2005$, $\sqrt{n} \cdot 3n$, $n + n \cdot \log n$
b/ $(\log n)^n$, $n^n$, $2^{\sqrt{n}}$

### Příklad 10.4

Nechť $a, b > 1$. Ukažte:

- $\exists c : \forall x : \log_a x = c \cdot \log_b x$ (tedy $\log_a n \in \Theta(\log_b n)$)
- $a^{\log_b n} = n^{\log_b a}$ (návod: aplikujte na obě strany funkci $\log_b$)

**Příklad 10.5**

(Nepovinně.)
Věta o řešení rekurentních rovnic se dá úvest i takto (mírně obecněji než je v učebním textu):
Nechť $a \geq 1$, $b > 1$ jsou konstanty, $f$ je funkce (typu $\mathbb{N} \to \mathbb{N}$, či alespoň asymptoticky kladná) a pro funkci $T : \mathbb{N} \to \mathbb{N}$ platí rekurentní vztah

$$T(n) = aT(n/b) + f(n).$$

Pak platí:

1. Je-li $f(n) = O(n^c)$ a $c < \log_b a$, pak $T(n) = \Theta(n^{\log_b a})$.
2. Je-li $f(n) = \Theta(n^{\log_b a})$, pak $T(n) = \Theta(n^{\log_b a} \cdot \log n)$.
   Obecněji: je-li $f(n) = \Theta(n^{\log_b a} \log^k n)$, $k \geq 0$, pak $T(n) = \Theta(n^{\log_b a} \cdot \log^{k+1} n)$.
3. Je-li $f(n) = \Omega(n^c)$ a $c > \log_b a$
   a $a \cdot f(n/b) \leq d \cdot f(n)$ pro nějaké $d < 1$ a skoro všechna $n$ (tedy až na konečně mnoho výjimek),
   pak $T(n) = \Theta(f(n))$.

Ve 3. případu lze vyvodit, že když $f(n) = \Theta(n^c)$, tak $T(n) = \Theta(n^c)$. (Můžete ověřit.)
Pak zjistěte u následujících příkladů, zda se na ně věta vztahuje, a v kladném případě odvoďte řešení.

- $T(n) = 3T(n/2) + n^2$
- $T(n) = 4T(n/2) + n^2$
- $T(n) = 8T(n/2) + n^2$
- $T(n) = 7T(n/2) + n^2$
- $T(n) = 2T(n/2) + n \log n$
- $T(n) = T(n/2) + 2^n$
- $T(n) = 2^n T(n/2) + n$