Týden 9

Přednáška

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.*

Důkaz věty. Uvažujme libovolnou netriviální vstupně/výstupní vlastnost V Turingových strojů. Nechť stroj M_1, který se nezastaví na žádný vstup, vlastnost V nemá, a nechť jistý stroj M_2 vlastnost V má, nebo naopak; takové dva stroje M_1, M_2 nutně musí existovat.

Uvažme nyní instanci M, w problému zastavení. Sestrojme k ní stroj M', který se chová takto: vstup u nejprve ignoruje a simuluje stroj M na w; teprve pokud se tato simulace zastaví (tedy M se zastaví na w), tak M' pokračuje simulací stroje M_2 na vstup u. Vidíme: pokud M se zastaví na w, tak M' má stejné I/O chování (má stejnou I/O tabulku) jako M_2; pokud se M nezastaví na w, tak M' má stejné I/O chování (má stejnou I/O tabulku) jako M_1. Kdyby tedy vlastnost V byla rozhodnutelná, byl by rozhodnutelný i problém zastavení.

Nerozhodnutelnost ekvivalence bezkontextových gramatik

Uvedli jsme myšlenku redukce problému zastavení na ekvivalenci dvou (nedeterministických) zásobníkových automatů.

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.

Třída PTIME

Připomněli jsme si definici třídy PTIME. Přitom jsem zdůraznil, že všechny problémy ve studijním textu (nejen ty, které jsou v PTIME) je třeba důkladně promyslet – pak nemůže být pro nikoho problémem u zkoušky nějaký požadovaný problém přesně definovat a uvést příklady instancí s pozitivní odpovědí a instancí s negativní odpovědí.

Dále zmíníme několik obecných metod návrhu polynomiálních algoritmů.

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.

Jako pěkný příklad dynamického programování jsme zmínili také algoritmus Cocke-Younger-Kasami pro rozhodování příslušnosti k jazyku generovanému bezkontextovou gramatikou.

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. Ilustrováno na mergesortu $O(n \log n)$ a násobení matic: klasicky v $O(n^3)$ (kde n je rozměr matic), zmíněna složitost Strassenova algoritmu v $O(n^{\log_2 7})$.

Hltavý (greedy) přístup u optimalizačních problémů

Ilustrováno na problému minimální kostry v grafu (kde tento přístup funguje).

Partie textu k prostudování

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

Příklad 9.1

Uveďte alespoň tři vlastnosti Turingových strojů, pro něž plyne nerozhodnutelnost z Riceovy věty, a alespoň tři vlastnosti, pro něž nerozhodnutelnost z Riceovy věty neplyne. (Jiné příklady než byly na přednášce.)

Příklad 9.2

Připomeňte si, jak se používá a co vyjadřuje značení O, o, Θ a přesně jej zadefinujte.
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 9.3

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 9.4

Připomeňte si definici třídy PTIME. Uveďte precizně příklady alespoň dvou problémů z PTIME a prokažte, že jsou v PTIME.

Příklad 9.5

Demonstrujte hltavý přístup při řešení problému minimální kostry v grafu (na rozumném příkladu s několika vrcholy) a ukažte, že se jedná o polynomiální algoritmus. (Ideální by bylo, kdybyste také podali argumenty, proč tento přístup vede k optimálnímu řešení.)

Příklad 9.6

Uvažte následující rozhodovací (neboli Ano/Ne) verzi jistého optimalizačního problému:

Název: TSP (*problém obchodního cestujícího (*Ano/Ne *verze)*)

Vstup: množina „měst“ $\{1, 2, \ldots, n\}$, přir. čísla („vzdálenosti“) d_{ij} ($i = 1, 2, \ldots, n$, $j = 1, 2, \ldots, n$); dále číslo ℓ („limit“).

Otázka: existuje „okružní jízda“ dlouhá nejvýše ℓ, tj. existuje permutace $\{i_1, i_2, \ldots, i_n\}$ množiny $\{1, 2, \ldots, n\}$ tž. $d(i_1, i_2) + d(i_2, i_3) + \cdots + d(i_{n-1}, i_n) + d(i_n, i_1) \leq \ell$?

Zformulujte přirozenou metodu hltavého přístupu k tomuto problému a zkuste na malém příkladu prokázat, že nemusí vést k optimu.

Příklad 9.7

(Nepovinně; doplnění k metodě „rozděl a panuj“)
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.

- $T(n) = 3T(n/2) + n^2$
- $T(n) = 4T(n/2) + n^2$

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