Týden 10

Přednáška

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).
Např. u problému obchodního cestujícího hltavý přístup nefunguje. Rozhodovací (neboli Ano/Ne) verze tohoto optimalizačního problému je tato:

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. $d(i_1, i_2) + d(i_2, i_3) + \cdots + d(i_{n-1}, i_n) + d(i_n, i_1) \leq \ell$?

Nedeterministické algoritmy a jejich složitost; třída NPTIME

Ujasnili jsme si pojem nedeterministického algoritmu (Turingova stroje) a definici toho, co to znamená, že nějaký nedeterministický Turingův stroj M rozhoduje daný problém P.
Také jsme přímočaře rozšířili pojem (časové) složitosti na nedeterministické stroje a definovali jsme třídu NPTIME.
Ukázali jsme si (nedeterministické) algoritmy prokazující, že problém SAT (splnitelnost booleovských formulí) a IS (Independent Set, rozhodovací verze problému nezávislé množiny v grafu) jsou v NPTIME.

Polynomiální převeditelnost; NP-úplné problémy

Již známý pojem převeditelnosti mezi problémy jsme využili v definici polynomiální převeditelnosti mezi problémy. (Příslušný převádějící algoritmus musí mít polynomiální časovou složitost, tedy časovou složitost omezenou polynomem.)
Všimli jsme si, jak může prokázaná polynomiální převeditelnost mezi problémy pomoci v určování (ne)příslušnosti k PTIME či NPTIME.
Definovali jsme pojem NP-úplného problému; problémy SAT, 3-SAT, HC, HK, 3-CG a IS jsou příklady NP-úplných problémů.
Připomenutí: animace ukazují důkaz Cookovy věty, tedy důkaz toho, že SAT je NP-úplný, a dále demonstrují SAT $\lhd$ 3-SAT, 3-SAT $\lhd$ IS, IS $\lhd$ HC, HC $\lhd$ HK (a také nyní požadovaný převod HK $\lhd$ TSP.)

Partie textu k prostudování

Složitost algoritmů (8.1., 8.2.). Dynamické programování (10.2.4.), část 10.2.2. (metoda „rozděl a panuj“). Části 8.3., 8.4., 8.5. (třída PTIME, třída NPTIME, NP-úplné problémy).

Cvičení

Příklad 10.1

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 10.2

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 10.3

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

Příklad 10.4

Definujte pojem *polynomiální převeditelnosti* jako speciální případ dříve uvedené (algoritmické) převeditelnosti mezi problémy. (Příslušný převádějící algoritmus musí mít polynomiální časovou složitost, tedy časovou složitost omezenou polynomem.)
Vysvětlete nejdříve přesně, co máme udělat, chceme-li prokázat polynomiální převeditelnost problému HC (hamiltonovský cyklus v orientovaném grafu) na problém HK (hamiltonovská kružnice v neorientovaném grafu).
Pak to udělejte.

Příklad 10.5

Vysvětlete pojem „NP-úplný problém“.
Definujte problémy SAT, 3-SAT, HC, HK, 3-CG a IS (s příklady pozitivních a negativních instancí). Tyto problémy jsou NP-úplné. U každého si alespoň uvědomte algoritmus, prokazující příslušnost k NP.

Příklad 10.6

Uvažujme následující problém (jeden z často uváděných NP-úplných problémů).

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. $d(i_1, i_2) + d(i_2, i_3) + \cdots + d(i_{n-1}, i_n) + d(i_n, i_1) \leq \ell$?

Je to rozhodovací (neboli ANO/NE) verze optimalizačního problému. Odvoďte nejdříve, jak vypadá onen optimalizační problém (tedy co je jeho vstupem a co odpovídajícím výstupem).
Dále ukažte nějakou malou (ale ne úplně triviální) instanci (tedy vstup) uvedeného problému TSP, pro niž je odpověď ANO, a instanci, pro niž je odpověď NE.
Pak prokažte (návrhem konkrétního nedeterministického algoritmu), že TSP je v NP.
Nakonec zkuste vymyslet důkaz NP-obtížnosti problému TSP.
(Nápověda. Můžete využít faktu, že problém hamiltonovské kružnice (HK) je NP-úplný.)

Příklad 10.7

Uvažujme problém

NÁZEV: ILP (*problém celočíselného lineárního programování*)

VSTUP: Matice A typu $m \times n$ a sloupcový vektor b velikosti m, jejichž prvky jsou celá čísla.

OTÁZKA: Existuje celočíselný sloupcový vektor x (velikosti n) tž. $Ax \geq b$?

Ukažte nejprve nějakou malou (ale ne úplně triviální) instanci (tedy vstup) uvedeného problému ILP, pro niž je odpověď ANO, a instanci, pro niž je odpověď NE.

Vysvětlete přesně, co bychom museli udělat, kdybychom chtěli ukázat, že 3-SAT $\triangleleft$ ILP.

(Zbytek příkladu je nepovinný.)

Zbude-li čas, zkuste tuto převeditelnost dokázat. Přinejmenším ale uveďte, co bychom mohli říci o složitosti problému ILP poté, co bychom prokázali 3-SAT $\triangleleft$ ILP.

Dále pouvažujte o tom, zda ILP patří do NP.

Je to tak, ale je to příklad problému, jehož příslušnost k NP není ihned zřejmá – na rozdíl od dřívějších příkladů problémů v NP.

(Spokojíme se zde jen s odkazem na fakt, že se dá ukázat, že existuje-li řešení nerovnosti $Ax \geq b$, pak existuje i řešení „dostatečně malé“ – jeho zápis je polynomiální vzhledem k zápisu A a b; řešení se tedy dá v polynomiálním čase „uhodnout“ a ověřit.)

Příklad 10.8

(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.

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