Algoritmická matematika 3

Základní pojmy

Petr Osička

DATA ANALYSIS AND MODELING LAB

Univerzita Palackého v Olomouci

Zimní semestr 2013

O kurzu

Organizace

- přenáška (2h) a cvičení (2h)
- zápočet: písemka + programování
- zkouška: klasická ústní

Vyučující

- Petr Osička (přednáška), Martin Trnečka (cvičení)
- Info o vyučujících (včetně konzultačních hodin) na webu katedry

Obsah

- kurz je věnován základním technikám návrhu algoritmů
- obecný popis technik
- příklady konkrétních algoritmů

Motivace

V životě (práci, škole) často stojíme před problémy (úkoly, otázkami) a potřebujeme k nim najít řešení. Chceme ho umět hledat **automaticky** a dostatečně **efektivně.**

Intuitivnímu pojmu „postup řešení problému" odpovídá v informatice pojem **algoritmus**. Při studiu problémů a algoritmů se lze zkoumat dvě základní otázky

1. **Jak co nejrychleji vyřešit daný problém?**
   Této oblasti se někdy říká **design (návrh) algoritmů** a budeme se jí věnovat v tomto kurzu. Ukážeme si základní šablony návrhu a principy na kterých jsou založeny.

   Příklad: Jak co nejrychleji najít nejkratší cestu mezi dvěma místy? (například ve Starcraftu)
2. **Jde daný problém algoritmicky řešit? Existuje nějaké omezení pro rychlost s jakou jde problém vyřešit?**
   Těmto otázkám se budete věnovat v kurzu **Vyčíslitelnost a složitost**.

   Příklad: Je možné vymyslet algoritmus, který najde nejkratší cestu mezi dvěma místy v lineárním čase? Nebo je problém hledání nejkratších cest přirozeně težší?

Algoritmický problém

Omezíme se na problémy, které jde **přesně specifikovat**. Je nutné uvést množinu **instancí** (vstupů), a relaci mezi instancemi a očekávanými (správnými) řešeními.

Reprezentace instancí

1. *Uvedením objektů, z jakých se vstupní instance skládá. Obvykle to jsou množiny, čísla, funkce, grafy apod.*

   Tuto reprezentaci je výhodné používat při návrhu algoritmů, a budeme ji používat i během kurzu.

2. *Pomocí zakódování do řetězců nad vhodně zvolenou abecedou*

   Tato reprezentace více odpovídá reálným modelům výpočtu (například v paměti počítače jsou instance reprezentovány jako sekvence bitů). Protože je to jednotná reprezentace, umožňuje přesné zavedení pojmů jako je algoritmický problém, algoritmus, velikost instance apod. I kdyz používame reprezentaci z bodu 1, je nutné mít na paměti, že „pod kapotou" je vhodně zvolená reprezentace pomocí řetezců.

V následujícím zavedeme pomocí reprezentace instancí řetezci základní pojmy.

Definice

Abeceda Σ je konečná neprázná množina. Prvkům Σ říkáme symboly. **Slovo (řetězec)** nad abecedou Σ je uspořádaná sekvence znaků z Σ. Délka slova $x \in \Sigma$ je délka této sekvence. Množinu všech slov nad abecedou Σ označujeme jako Σ^*. **Jazyk** L nad Σ je libovolná podmnožina Σ^*.

Příklad
Mějme $\Sigma_{Bool} = \{0, 1\}$. Sekvence 0001, 1110, 00 jsou slovy nad touto abecedou, sekvence 12, $\#43k$ nejsou. Σ^* je pak množina $\{0, 1, 00, 01, 10, 11, 000, \dots\}$. □

Definice

Nechť Σ je abeceda. Pak **algoritmický problém** definujeme jako dvojici $\langle L, R \rangle$, kde $L \subseteq \Sigma^*$ je množina instancí daného problému, a $R \subseteq \Sigma^* \times \Sigma^*$ je relace popisující vztah mezi instancemi a správnými řešeními. Tedy pokud $\langle x, y \rangle \in R$, pak y je správným řešením x.

Příklad: *Problém testování prvočíselnosti*

1. Bez použití řetezců můžeme problém zavést následovně. Množina instancí je dána jako $\{n \mid n \in N, n \geq 1\}$, správné řešení je ANO, pokud je n prvočíslo, jinak NE.
2. Zavedeme-li kódování nad Σ_{Bool} takové, že zakódováním n je slovo skládající se z n znaků 1, kódováním ANO je slovo 1 a kódováním NE je slovo 0, pak je problém popsán pomocí dvojice $\langle L, R\rangle$ takové, že: $L = \{1^n | n \in N, n \geq 1\}$, kde a^n je slovo složené z n znaků a, navíc $\langle 1^n, 1\rangle \in R$, pokud je n prvočíslo, $\langle 1^n, 0\rangle \in R$, pokud n není prvočíslo.

Základní typy problémů

Definition

Problém $\langle L, R\rangle$ je **rozhodovacím problémem**, pokud je množina všech možných výsledků dvouprvková, tj $|\{y \mid (x, y) \in R \text{ pro } x \in L\}| = 2$ a R je zobrazení. Jeden z prvků této množiny pak označujeme jako rozhodnutí ANO, druhý z prvků jako rozhodnutí NE.

Alternativně lze rozhodovací problém reprezentovat jazykem $\{x \mid \langle x, ANO\rangle \in R\}$.

Příklad: *Splnitelnosti formulí výrokové logiky v konjunktivní normální formě (CNF)*

- formule je v CNF, pokud je konjunkcí **klausulí**.
- klausule je disjunkcí **literálů**,
- literál je buď výroková proměnná nebo její negace.
- formule φ je splnitelná, pokud existuje ohodnocení výrokových proměnných takové, že φ je pravdivá
- problém je určen jazykem {zakódování $\varphi \mid \varphi$ je splnitelná formule v CNF}

Definice

Nechť Σ je abeceda. **Optimalizační problém** je čtveřice $\langle L, sol, cost, goal \rangle$, kde

- $L \in \Sigma^*$ je množina instancí daného problému,
- $sol : L \to \mathcal{P}(\Sigma^*)$ je zobrazení přiřazující instanci problému množinu vhodných řešení;
- $cost : L \times \mathcal{P}(\Sigma^*) \to \mathbb{Q}$ je zobrazení přiřazující každé instanci a vhodnému řešení této instance jeho cenu;
- $goal$ je buď minimum (pak mluvíme o minimalizačním problému) nebo maximum (pak mluvíme o maximalizačním problému).

Řekneme, že $y \in \Sigma^*$ je **správným řešením** instance $x \in L$, pokud $y \in sol(x)$.

Řekneme, že $y \in \Sigma^*$ je **optimálním řešením** instance $x \in L$, pokud

$$cost(x, y) = goal\{cost(y, x) \mid x \in sol(x)\}.$$

Příklad:

Úloha batohu (KNAPSACK)	
Instance:	$\{(b, w_1, \ldots, w_n) \mid b, w_1, \ldots, w_n \in \mathbb{N}\}$
Přípustná řešení:	$sol(b, w_1, \ldots, w_n) = \{C \subseteq \{1, \ldots, n\} \mid \sum_{i \in C} w_i \leq b\}$
Cena řešení:	$cost(C, (b, w_1, \ldots, w_n)) = \sum_{i \in C} w_i$
Cíl:	maximum

Čísla w_i odpovídají objemům jednotlivých položek, b je kapacita batohu. Cílem je vybrat takovou podmnožinu položek, která zaplní batoh co nejvíce, ale současně se do batohu vejde.

Pro instanci $I = (b, w_1, \ldots, w_5)$, kde $b = 29, w_1 = 3, w_2 = 6, w_3 = 8, w_4 = 7, w_5 = 12$, existuje jediné optimální řešení $C = \{1, 2, 3, 5\}$ s cenou $cost(C, I) = 29$. Lze snadno ověřit, že všechna ostatní přípustná řešení mají menší cenu.

Příklad:
Pro sekvenci prvků $a = a_1, \ldots, a_n$ a jejich totální uspořádání $\leq$ je počet inverzí definován jako

$$Inv(a) = |\{(a_i, a_j) \mid i < j, a_i \not\leq a_j\}|,$$

tedy jako počet dvojic prvků, které jsou nejsou uspořádány ve správném pořadí.

Problém třídění	
Instance:	$a = a_1, \ldots, a_n$, uspořádání $\leq$
Přípustná řešení:	$sol(a) =$ množina všech permutací prvků $a_1, \ldots, a_n$
Cena řešení:	pro $x \in sol(a)$ je $cost(a, x) = Inv(x)$
Cíl:	minimum

Poznámky:

- Rozhodovací a optimalizační problémy rozhodně nepokrývají množinu všech existujících problémů. Jejich význam je v tom, že mnoho přirozených problémů se dá formulovat jako rozhodovací nebo optimalizační problém.
- Příkladem dalšího typu problémů jsou **counting (počítací) problémy**. Úkolem je pro dané omezení spočítat, kolik struktur, které odpovídají danému omezení, existuje.

  Příklad: *Kolik různých koster existuje v grafu? Kolik existuje takových ohodnocení proměnných, které splní formuli v CNF?*
- Problémy, které neumíme nikam zařadit, většinou označujeme jako **searching (vyhledávací) problémy**.

  Příklad: *Spočítejte součin dvou matic. Spočítejte součin dvou polynomů.*

Algoritmus

Formální definici si neuvedeme (dozvíte se ji v kurzu o vyčíslitelnosti).

Dva základní pohledy

1. Algoritmus je procedura zapsaná ve vhodném symbolickém jazyce ve tvaru (konečně mnoha) jednoznačných instrukcí. Instrukce jsou prováděny za sebou v jednoznačném pořadí a k jejich provedení není potřeba žádná inteligence, intuice, lze je provádět strojově.
2. Algoritmus je konečná množina pravidel, pomocí kterých se dynamicky mění stav. Stav může být popsán například pomocí proměnných. Každá z instrukcí pak mění hodnotu jedné proměnné v závislosti na tom, v jakém stavu se systém právě nachází. Existuje také pravidlo, které určí, že vývoj systému je u konce.

Správnost a optimalita algoritmu

Algoritmus danému vstupu (instanci nějakého problému) přiřazuje unikátní výstup. Z vnějšku jej lze tedy chápat jako zobrazení. Formálně zapíšeme fakt, že algorimus A pro vstup x vrátí y jako $A(x) = y$.

Správnost
Řekneme, že A řeší problém $\langle L, R\rangle$ pokud $\langle x, A(x)\rangle \in R$ pro každé $x \in L$, tedy pokud pro každou instanci problému vrací A správné řešení.

Optimalita
Řekneme, že A řeší problém $\langle L, sol, cost, goal\rangle$ optimálně, pokud pro každé $x \in L$ je $A(x)$ optimálním řešením.

Pokud algoritmus A nevrací vždy optimální řešení, ale $A(x) \in sol(x)$ a $cost(A(x))$ není příliš daleko od ceny optimálního řešení, nazýváme A **aproximačním algoritmem**. Takový algoritmus je užitečný v případech, kdy by výpočet optimálního řešení trval příliš dlouho.

Složitost

Velikost instance

1. Pokud instanci reprezentujeme řetězcem x, je velikost instance $|x|$rovna délce tohoto řetezce.
2. Pokud instanci reprezentuje jinými objekty, je nutné velikost instance vhodně zvolit. Například velikostí grafu může být počet uzlů, případně počet hran, velikostí pole čísel může být jejich počet, apod.

Definice

Časová složitost (dále jen složitost) algoritmu A je funkce

$$Time_A : \mathbb{N} \to \mathbb{N}$$

přiřazující velikosti vstupní instance počet instrukcí, které algoritmus A během výpočtu provede.

Definice

Označme si $t_A(x)$ počet instrukcí, které algoritmus A provede pro instanci $x \in L$.
Časová složitost v **nejhorším případě** je definovaná

$$Time_A(n) = \max\{t_A(x) \mid x \in L, |x| = n\}.$$

Složitost v **průměrném případě** je pak

$$Time_A(n) = \frac{\sum_{x \in L, |x|=n} t_A(x)}{|\{x \mid |x| = n\}|}.$$

Poznámky

- Složitost v nejhorším případě je snažší nalézt. Stačí najít nejhorší případ.
- Složitost v průměrném případě vyžaduje použití sofistikovanějších metod. Vztah z předchozí definice je pro větší velikosti instancí (velkých instancí je obrovské množství) v praxi nepoužitelný.

Definice

Nechť U je algoritmický problém.

- $O(g(n))$ je **horním omezením složitosti** problému U, pokud existuje algoritmus A řešící U takový, že $Time_A(n) \in O(g(n))$.
- $\omega(f(n))$ je **dolním omezením složitosti** problému U, pokud pro každý algoritmus B řešící U platí, že $Time_B(n) \in \omega(f(n))$

- Za horní omezení složitosti problému bereme složitost nejlepšího známého algoritmu.
- Nalézt dolní omezení složitosti je oproti tomu mnohem komplikovanější. Musíme totiž vyloučit existenci algoritmu s lepší složitostí než dolní hranicí.
- Příkladem problému, pro který je tato hranice známá (a je stejná jako horní) je třídění porovnáváním, které má dolní i horní omezení složitosti $O(n \log n)$.

Praktická řešitelnost problému

U tohoto rozdělení se řídíme horním odhadem složitosti problému.

Řešitelné problémy

- jeho horní omezení složitosti je $O(p(n))$, kde $p(n)$ je polynom.
- je rozumné požadovat, aby $p(n)$ měl rozumně nízký stupeň
- polynom je výhodná funkce, protože

Proč polynom?

- součin dvou polynomů je také polynom. Odtud plyne, že složením dvou algoritmů s polynomickou složitostí dostaneme opět algoritmus s polynomickou složitostí.
- polynom je největší funkce, která „neroste rychle".
- všechny rozumné definice pojmu algorimus jsou schopné se vzájemně simulovat s polynomickým zpožděním.

Asymptotická notace

Definice

$O(f(n))$ je **množina** všech **funkcí** $g(n)$ takových, že existují kladné konstanty c a n_0 takové, že

$$0 \leq g(n) \leq cf(n) \text{ pro všechna } n \geq n_0.$$

$\Omega(f(n))$ je **množina** všech **funkcí** $g(n)$ takových, že existují kladné konstanty c a n_0 takové, že

$$g(n) \geq cf(n) \geq 0 \text{ pro všechna } n \geq n_0.$$

$\Theta(f(n))$ je **množina** všech **funkcí** $g(n)$ takových, že existují kladní konstanty c_1, c_2 a n_0 takové, že

$$0 \leq c_1 f(n) \leq g(n) \leq c_2 f(n) \text{ pro všechna } n \geq n_0.$$

Práce s asymptotickou notací

Jednostranná rovnost

- $g(n) = O(f(n))$, rovnítko interpretujeme $=$ jako $\in$
- pokud je asymptotická notace na obou stranách, interpretujeme $=$ jako $\subseteq$

Další manipulace

- Nechť S a T jsou množiny funkcí. Pak

$$\begin{aligned} S + T &= \{g + h \mid g \in S, h \in T\}, \\ S - T &= \{g - h \mid g \in S, h \in T\}, \\ S \cdot T &= \{g \cdot h \mid g \in S, h \in T\}, \\ S \div T &= \{g \div h \mid g \in S, h \in T\}. \end{aligned}$$

- pro funkci f máme $f(O(g(n))) = \{f(h(n)) \mid h \in O(g(n))\}$
- $O(g(n) + O(f(n))) = \bigcup\{O(h) \mid h \in g(n) + O(f(n))\}$