Složitost problémů

Složitost problémů

- Ukazuje se, že různé (algoritmické) problémy jsou různě těžké.
- Obtížnější jsou ty problémy, k jejichž řešení potřebujeme více času a paměti.
- Obtížnost problémů chceme nějak posuzovat, a to jak
  - absolutně – kolik času a kolik paměti potřebujeme k jejich řešení, tak
  - relativně – o kolik je jejich řešení obtížnější nebo naopak jednodušší oproti jiným problémům.
- Proč se u některých problémů nedaří nalézt efektivní algoritmy? Může vůbec nějaký efektivní algoritmus pro daný problém existovat?
- Kde přesně jsou limity toho, co můžeme prakticky zvládnout?

Složitost problémů

Je potřeba odlišovat **složitost algoritmu** a **složitost problému**.

Pokud například zkoumáme časovou složitost v nejhorším případě, mohli bychom neformálně říct:

- **složitost algoritmu** – funkce, která vyjadřuje, kolik kroků maximálně udělá daný algoritmus pro vstup velikosti n
- **složitost problému** – jaká je časová složitost „nejefektivnějšího“ algoritmu, který řeší daný problém

Zavedení pojmu „složitost problému“ ve výše uvedeném smyslu naráží na značné technické obtíže. Pojem „složitost problému“ se tedy jako takový nedefinuje, ale obchází se zavedením tzv. **tříd složitosti**.

Třídy složitosti

Třídy složitosti jsou podmnožiny množiny všech (algoritmických) **problémů**.

Daná konkrétní třída složitosti je vždy charakterizována nějakou vlastností, kterou mají problémy do ní patřící.

Typickým příkladem takové vlastnosti je vlastnost, že pro daný problém existuje nějaký algoritmus s určitým omezením (např. časové nebo prostorové složitosti):

- Do dané třídy pak patří všechny problémy, pro které takovýto algoritmus existuje.
- Naopak do ní nepatří problémy, pro které žádný takový algoritmus neexistuje.

Třídy složitosti

Definice

Pro libovolnou funkci $f : \mathbb{N} \to \mathbb{N}$ definujeme třídu $\mathcal{T}(f(n))$ jako třídu obsahující právě ty problémy, pro něž existuje algoritmus s časovou složitostí $O(f(n))$.

Příklad:

- $\mathcal{T}(n)$ – třída všech problémů pro něž existuje algoritmus s časovou složitostí $O(n)$
- $\mathcal{T}(n^2)$ – třída všech problémů pro než existuje algoritmus s časovou složitostí $O(n^2)$
- $\mathcal{T}(n \log n)$ – třída všech problémů pro než existuje algoritmus s časovou složitostí $O(n \log n)$

Třídy složitosti

Definice

Pro libovolnou funkci $f : \mathbb{N} \to \mathbb{N}$ definujeme třídu $\mathcal{S}(f(n))$ jako třídu obsahující právě ty problémy, pro něž existuje algoritmus s prostorovou složitostí $O(f(n))$.

Příklad:

- $\mathcal{S}(n)$ – třída všech problémů pro něž existuje algoritmus s prostorovou složitostí $O(n)$
- $\mathcal{S}(n^2)$ – třída všech problémů pro než existuje algoritmus s prostorovou složitostí $O(n^2)$
- $\mathcal{S}(n \log n)$ – třída všech problémů pro než existuje algoritmus s prostorovou složitostí $O(n \log n)$

Třídy složitosti

Poznámka:

Všimněte si, že u tříd $\mathcal{T}(f)$ a $\mathcal{S}(f)$ může to, které problémy do dané třídy patří, záviset na použitém výpočetním modelu (zda je to stroj RAM, jednopáskový Turingův stroj, vícepáskový Turingův stroj, ...).

Třídy složitosti

Pomocí tříd $\mathcal{T}(f(n))$ a $\mathcal{S}(f(n))$ můžeme definovat třídy PTIME a PSPACE jako

$$\text{PTIME} = \bigcup_{k \geq 0} \mathcal{T}(n^k) \qquad \text{PSPACE} = \bigcup_{k \geq 0} \mathcal{S}(n^k)$$

- PTIME je třída všech problémů, pro které existuje algoritmus s polynomiální časovou složitostí, tj. s časovou složitostí $O(n^k)$, kde k je nějaká konstanta.
- PSPACE je třída všech problémů, pro které existuje algoritmus s polynomiální prostorovou složitostí, tj. s prostorovou složitostí $O(n^k)$, kde k je nějaká konstanta.

Třídy složitosti

Poznámka: Vzhledem k tomu, že všechny (rozumné) výpočetní modely jsou schopné se navzájem simulovat tak, že při dané simulaci nevzroste počet kroků ani množství použité paměti víc než polynomiálně, není definice tříd PTIME a PSPACE závislá na použitém výpočetním modelu. Pro jejich zadefinování můžeme použít kterýkoliv výpočetní model.

Říkáme, že tyto třídy jsou **robustní** – jejich definice nezávisí na použitém výpočetním modelu.

Třídy složitosti

Analogicky můžeme zavést další třídy:

EXPTIME – množina všech problémů, pro které existuje algoritmus s časovou složitostí $2^{O(n^k)}$, kde k je nějaká konstanta

EXPSPACE – množina všech problémů, pro které existuje algoritmus s prostorovou složitostí $2^{O(n^k)}$, kde k je nějaká konstanta

LOGSPACE – množina všech problémů, pro které existuje algoritmus s prostorovou složitostí $O(\log n)$

Poznámka: Místo $2^{O(n^k)}$ bychom mohli psát také $O(c^{n^k})$, kde c a k jsou nějaké konstanty.

Třídy složitosti

Při definici třídy LOGSPACE musíme přesněji specifikovat, co považujeme za prostorovou složitost algoritmu.

Uvažujeme například Turingův stroj, který pracuje se třemi páskami:

- **Vstupní páskou**, na které je na začátku výpočtu zapsán vstup. Z této pásky je možno pouze číst.
- **Pracovní páskou**, která je na začátku výpočtu prázdná. Z této pásky je možno číst i na ni zapisovat.
- **Výstupní páskou**, která je také na začátku výpočtu prázdná a na kterou je možno pouze zapisovat.

Množství použité paměti je pak definováno, jako počet použitých políček na pracovní pásce.

Třídy složitosti

Další příklady tříd složitosti:

2-EXPTIME – množina všech problémů, pro které existuje algoritmus s časovou složitostí $2^{2^{O(n^k)}}$, kde k je nějaká konstanta

2-EXPSPACE – množina všech problémů, pro které existuje algoritmus s prostorovou složitostí $2^{2^{O(n^k)}}$, kde k je nějaká konstanta

ELEMENTARY – množina všech problémů, pro které existuje algoritmus s časovou (či prostorovou) složitostí

$$2^{2^{2^{\cdot^{\cdot^{\cdot^{2^{2^{O(n^k)}}}}}}}}$$

kde k je konstanta a počet exponentů je omezen konstantou.

Vztahy mezi třídami složitosti

Pokud Turingův stroj provede m kroků, tak použije maximálně m políček na pásce.

Pokud tedy existuje pro nějaký problém algoritmus s časovou složitostí $O(f(n))$, má tento algoritmus paměťovou složitost (nejvýše) $O(f(n))$.

Je tedy zřejmé, že platí následující vztah.

Pozorování

Pro libovolnou funkci $f : \mathbb{N} \to \mathbb{N}$ platí $\mathcal{T}(f(n)) \subseteq \mathcal{S}(f(n))$.

Poznámka: Analogicky bychom mohli argumentovat například pro stroj RAM.

Vztahy mezi třídami složitosti

Z předchozího okamžitě plyne:

$$\begin{aligned} \text{PTIME} &\subseteq \text{PSPACE} \\ \text{EXPTIME} &\subseteq \text{EXPSPACE} \\ \text{2-EXPTIME} &\subseteq \text{2-EXPSPACE} \\ &\vdots \end{aligned}$$

Vzhledem k tomu, že polynomiální funkce rostou pomaleji než exponenciální, zjevně platí:

$$\text{PTIME} \subseteq \text{EXPTIME} \subseteq \text{2-EXPTIME} \subseteq \cdots$$

$$\text{LOGSPACE} \subseteq \text{PSPACE} \subseteq \text{EXPSPACE} \subseteq \text{2-EXPSPACE} \subseteq \cdots$$

- Pro libovolná dvě reálná čísla $0 \leq \epsilon_1 < \epsilon_2$ platí

$$\mathcal{S}(n^{\epsilon_1}) \subsetneq \mathcal{S}(n^{\epsilon_2})$$

- LOGSPACE $\subsetneq$ PSPACE
- PSPACE $\subsetneq$ EXPSPACE
- EXPSPACE $\subsetneq$ 2-EXPSPACE

- Pro libovolná dvě reálná čísla $0 \leq \epsilon_1 < \epsilon_2$ platí

$$\mathcal{T}(n^{\epsilon_1}) \subsetneq \mathcal{T}(n^{\epsilon_2})$$

- PTIME $\subsetneq$ EXPTIME
- EXPTIME $\subsetneq$ 2-EXPTIME

Vztahy mezi třídami složitosti

Při zkoumání vztahů mezi třídami složitosti se ukazuje jako užitečný pojem **konfigurace**.

Konfigurací budeme rozumět celkový stav, ve kterém se během jednoho kroku nachází stroj, provádějící nějaký daný algoritmus.

- U Turingova stroje je konfigurace dána stavem jeho řídící jednotky, obsahem pásky (resp. pásek) a pozicí hlavy (resp. hlav).
- U stroje RAM je konfigurace dána obsahem paměti, obsahem všech registrů (včetně IP), obsahem vstupní a výstupní pásky a pozicemi čtecí a zapisovací hlavy.

Vztahy mezi třídami složitosti

Mělo by být jasné, že konfigurace (resp. jejich popisy) můžeme zapisovat jako slova v nějaké abecedě.

Navíc můžeme konfigurace zapisovat tak, že délka těchto slov bude zhruba stejná jako množství paměti použité algoritmem (tj. počet políček na pásce použitých Turingovým stojem, počet bitů paměti použitých strojem RAM apod.).

Poznámka: Pokud máme abecedu Σ, kde $|\Sigma| = c$, tak:

- Počet slov délky n je c^n, tj. $2^{\Theta(n)}$.
- Počet slov délky nejvýše n je

$$\sum_{i=0}^{n} c^n = \frac{c^{n+1} - 1}{c - 1}$$

tj. také $2^{\Theta(n)}$.

Vztahy mezi třídami složitosti

Je jasné, že během výpočtu korektního algoritmu se žádná konfigurace nemůže zopakovat, protože jinak by se algoritmus zacyklil a běžel by donekonečna.

Pokud tedy víme, že paměťová složitost nějakého algoritmu je $O(f(n))$, znamená to, že počet různých konfigurací dosažitelných během výpočtu je $2^{O(f(n))}$.

Protože se konfigurace během žádného výpočtu neopakují, je i časová složitost daného algoritmu maximálně $2^{O(f(n))}$.

Pozorování

Pro libovolnou funkci $f : \mathbb{N} \to \mathbb{N}$ platí, že $\mathcal{S}(f(n)) \subseteq \mathcal{T}(2^{f(n)})$.

Vztahy mezi třídami složitosti

Z předchozího plynou následující důsledky:

$$\begin{aligned} \text{LOGSPACE} &\subseteq \text{PTIME} \\ \text{PSPACE} &\subseteq \text{EXPTIME} \\ \text{EXPSPACE} &\subseteq \text{2-EXPTIME} \\ &\vdots \end{aligned}$$

Vztahy mezi třídami složitosti

Shrnutí:

$$\text{LOGSPACE} \subseteq \text{PTIME} \subseteq \text{PSPACE} \subseteq \text{EXPTIME} \subseteq \text{EXPSPACE} \subseteq$$
$$\subseteq \text{2-EXPTIME} \subseteq \text{2-EXPSPACE} \subseteq \cdots \subseteq \text{ELEMENTARY}$$

Navíc je známo, že:

- $\text{PTIME} \subsetneq \text{EXPTIME} \subsetneq \text{2-EXPTIME} \subsetneq \cdots$
- $\text{LOGSPACE} \subsetneq \text{PSPACE} \subsetneq \text{EXPSPACE} \subsetneq \text{2-EXPSPACE} \subsetneq \cdots$

Horní a dolní odhady složitosti problémů

Horním odhadem složitosti problému rozumíme to, že složitost problému není vyšší než nějaká uvedená.

Většinou je to formulováno tak, že daný problém patří do nějaké určité třídy složitosti.

Příklady tvrzení, které se týkají horních odhadů složitosti:

- Problém dosažitenosti v grafu je v PTIME.
- Problém ekvivalence dvou regulárních výrazů je v EXPSPACE.

Pokud chceme zjistit nějaký horní odhad složitosti problému, stačí ukázat, že existuje algoritmus s danou složitostí.

Horní a dolní odhady složitosti problémů

Dolním odhadem složitosti problému rozumíme to, že složitost problému je alespoň taková jako nějaká uvedená.

Obecně je zjišťování (netriviálních) dolních odhadů složitosti problémů mnohem obtížnější než zjišťování horních odhadů.

Pro odvození dolního odhadu musíme totiž ukázat, že **každý** algoritmus řešící daný problém má danou složitost.

Horní a dolní odhady složitosti problémů

Problém „Třídění“

Vstup: Posloupnost prvků $a_1, a_2, \ldots, a_n$.

Výstup: Prvky $a_1, a_2, \ldots, a_n$ setříděné od nejmenšího po největší.

Dá se dokázat, že každý algoritmus, který řeší problém “Třídění” a na prvcích tříděné posloupnosti používá pouze operaci porovnávání (tj. nezkoumá obsah těchto prvků), má časovou složist v nejhorším případě v $\Omega(n \log n)$ (tj. pro každý takový algoritmus existují konstanty $c > 0$ a $n_0 \geq 0$ takové, že pro každé $n \geq n_0$ existuje vstup velikosti n, pro který provede algoritmus nejmémě $cn \log n$ operací).