1. CVIČENÍ Z ADS 1

Viktor Němeček 18. 2. 2019

https://kam.mff.cuni.cz/~viki/vyuka/ads11819/

V informatice bychom často rádi řekli, jak dlouho nějaký algoritmus trvá vzhledem k délce vstupu. Protože ale nechceme řešit, jak dlouho vůči sobě trvají různé instrukce a také nechceme, aby se změnila rychlost algoritmu, pokud ho pustíme na dvakrát výkonnějším stroji, je potřeba porovnávat čas běhu způsobem, který nebere v potaz multiplikativní a aditivní konstanty, je ale přesto dostatečně jemný.

Definice. Mějme funkce $f, g : \mathbb{N} \to \mathbb{R}$. Pak řekneme, že:

- $f(n)$ je třídy $\mathcal{O}(g(n))$, pokud existuje kladné reálné číslo c takové, že $f(n) \leq c \cdot g(n)$ PSVn[1].
- $f(n)$ je třídy $\Omega(g(n))$, pokud existuje kladné reálné číslo c takové, že $f(n) \geq c \cdot g(n)$ PSVn.
- $f(n)$ je třídy $\Theta(g(n))$, pokud je $f(n)$ třídy $\mathcal{O}(g(n))$ i $\Omega(g(n))$.
- $f(n)$ je třídy $o(g(n))$, pokud $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$.
- $f(n)$ je třídy $\omega(g(n))$, pokud $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$.

Tvrzení „$f(n)$ je třídy $\mathcal{O}(g(n))$“ často zapisujeme jako $f(n) = \mathcal{O}(g(n))$, nejedná se však o rovnost v jejím běžném pojetí. Chceme-li zdůraznit, že $\mathcal{O}(g(n))$ je v jistém smyslu množina, můžeme psát, že $f(n) \in \mathcal{O}(g(n))$. V této notaci poté dávají smysl tvrzení jako například $o(g(n)) \subseteq \mathcal{O}(g(n))$.

Nadále se budeme zabývat pouze situací, kdy f i g nabývá pouze nezáporných hodnot. Povolíme-li totiž i záporné hodnoty, začnou se různé běžně zaměnitelné definice použitých tříd chovat poněkud ošklivě.

$f(n) = \Theta(g(n))$ pak v nějakém smyslu znamená, že f a g rostou (klesají) stejně rychle, $\mathcal{O}$ že f roste nejvýše tak rychle (klesá alespoň tak rychle) jako g a o že f roste pomaleji (klesá rychleji). Uvědomme si ale, že ne každé dvě funkce musí být porovnatelné, a že Θ nezaručuje tak silnou vlastnost, jak bychom si mysleli.

Příklad 1. Najděte funkce $f, g : \mathbb{N} \to \mathbb{R}_0^+$ takové, že $f(n) \notin \mathcal{O}(g(n))$, ale $g(n) \notin \mathcal{O}(f(n))$.

Příklad 2. Najděte funkce $f, g : \mathbb{N} \to \mathbb{R}_0^+$ takové, že $f(n) \in \Theta(g(n))$, ale $\lim_{n\to\infty} \frac{f(n)}{g(n)}$ neexistuje.

Pozorování. Máme-li funkce $f, g, h : \mathbb{N} \to \mathbb{R}_0^+$ a platí $f(n) = \Theta(g(n) + h(n))$, pak $f(n) = \Theta(\max(g(n), f(n)))$.

Na druhou stranu, můžeme si všimnout, že pokud budeme Θ brát jako relaci na funkcích z $\mathbb{N}$ do $\mathbb{R}_0^+$, jedná se o ekvivalenci, a $\mathcal{O}$ bude neostré částečné uspořádání na jejích ekvivalenčních třídách.

Příklad 3. Zkuste najít co nejvíce vztahů mezi následujícími funkcemi: n, $\log n$, $\log(n^2)$, $\log\log n$, $\sqrt{n}$, $n \log n$, 4^n, 2^{2n}, $6n + 8$, 2^n, $n^{3/2}$, $n!$, $(n + 1)!$, n^n.

Nyní ale již k samotným algoritmům a odhadům jejich složitosti. Prozatím předpokládejme, že všechny běžné operace (sčítání, násobení, dělení, bitové operace, porovnání) s čísly, která jsou na vstupu či která mají být na výstupu, dovedeme provést v jednotkovém čase, naopak počítání s absurdně velkými čísly (jako například celým vstupem zakódovaným do jednoho dlouhého čísla) trvá déle, k tomu však více na přednášce a možná příště.

[1] Pro skoro všechna n. Přesněji, existuje N takové, že tvrzení platí pro všechna $n > N$.

Příklad 4. Jak dlouho bude v závislosti na n běžet následující pseudokód?

```
1       Dokud n>0
2           pokud n je liché
3               n := n-1
4           jinak
5               n := n/2
```

Pozor na tvrzení „Každý algoritmus se vstupem délky n musí udělat alespoň $\Omega(n)$ kroků protože musí alespoň načíst vstup“. To ale není pravda, v praxi často algoritmy nepoužíváme samostatně, ale jako součást delšího programu, a vstup tedy často již máme v paměti. Typickým příkladem algoritmu, který pak nemusí načíst ani celý vstup, je binární vyhledávání. Podobně bychom mohli najít argument proti tvrzení, že algoritmus musí běžet alespoň tak dlouho, jak dlouhý má výstup (ten lze totiž někdy odevzdat pouze jako odkaz do paměti, a pokud výstup je například nějaký podřetězec vstupu, nemusíme ho muset ani celý zapsat. Představme si například situaci, kdy by naším vstupem byl seznam slov seřazených podle abecedy, a my měli vrátit všechna slova začínající na písmeno „k“.)

Příklad 5. Najděte co nejrychlejší algoritmy na následující problémy:

a) Na vstupu máte seřazenou posloupnost a_i přirozených čísel délky n a číslo k. Najděte indexy k a ℓ, takové, že $a_k + a_\ell = k$ (nebo odpovězte, že neexistují).

b) Na vstupu máte číslo a a číslo k. Spočtěte a^k.

c) V paměti máte matici $n \times n$ celých čísel takovou, že každé číslo v ní je ostře větší než jeho soused směrem dolů a také než jeho soused směrem doprava (pokud příslušní sousedé existují). Zjistěte, zda tato matice obsahuje nulu.

Příklad 6. Mějme n-patrový mrakodrap a k vajíček. Vejce je objekt, pro který existuje hodnota m taková, že když vejce vyhodíme z okna v p-tém patře, tak pokud $p \leq m$, vejce najdeme pod mrakodrapem vždy nepoškozené (a můžeme ho tedy bez obav sebrat a jít ho vyhodit i z jiného okna), ale pokud $p > m$, vejce se vždy rozbije (a je pro naše pokusy nadále nepoužitelné). Hodnota m je navíc pro všechna vejce stejná. Vymyslete algoritmus, jak určit hodnotu m, kdy počet kroků měříme jako počet vyhození vajíčka z okna (případné počítání nás nic nestojí), pro případ, kdy

a) $k = 1$,

b) $k = \infty$,

c) $k = 2$.