5. Základy teorie složitosti

- zaměřuje se na klasifikaci výpočetních problémů dle jejich vlastní složitosti
- k studiu/určení složitosti problému se definují jisté výpočetní modely, Turingův stroj
- rozlišujeme dva základní typy složitosti, časová a prostorová
- výpočetní problém lze chápat jako nekonečnou sbírku příkladů a jejich řešení, přičemž zadání problému (instanci problému) nelze zaměňovat s problémem samotným; problém je třeba řešit bez ohledu na jeho zadání (test na prvočíslo, vstupem je jakékoliv číslo, výstupem je ano, či ne)
  - jeden ze základních typů problémů je tzv. rozhodovací problém, výstup je *ano*, nebo *ne*; příkladem může být problém, zda je daný graf souvislý (viz teorie grafů)
  - problém funkce je další typ výpočetního problému, jeden výstup funkce může být dosažen několika vstupy, ale totéž platí o rozhodovacím problému, ale od problému funkce se neočekává jednoduchá odpověď *ano*, či *ne*
    * příklad, násobíme dvě čísla, vstupem je čtveřice (a, b, c, d), ale vrátí se kladná odpověď, když platí $\frac{a \cdot b}{c} = d$

1. Zjištění složitosti problému

- problém P je řešitelný algoritmem, jehož časová složitost je $s(n)$
- předmětem je důkaz, že neexistuje algoritmus řešící problém P s lepší časovou složitostí, až poté můžeme tvrdit, že složitost problému je $s(n)$
- často se musíme spokojit s horní složitostí problému, analýza vhodného algoritmu, a dolní složitostí, odvozenou typicky matematickým argumentem
- složitost problému a složitost algoritmu
  - složitost algoritmu je složitost konkrétního algoritmu, který řeší daný problém, zajímá nás přitom horní odhad, “největší složitost”
  - zatímco u složitosti problému nás zajímá nejlepšího algoritmu, který daný problém řeší, takže nás zajímá dolní odhad, zároveň však každý takový algoritmus má svůj horní odhad složitosti problému
- čas, jenž je potřebný k zpracování algoritmem, závisí na velikosti vstupu, teorie složitosti zkoumá jak se daný algoritmus s různou velikostí vstupu vypořádá
  - pakliže je vstupem n a pro výpočet je zapotřebí času $\tau(n)$, pokud je $\tau(n)$ polynomiální, pak hovoříme o polynomiálním algoritmu, poté lze algoritmus vyřešit v polynomiálním čase
- měření velikosti vstupu záleží na konkrétní implementaci algoritmu

Časová složitost

- závislost velikosti vstupu na čase, který algoritmus potřebuje k běhu - jedná se tedy o funkci
- pro různě veliké vstupy bude časová složitost různá

Prostorová složitost

- vyjadřuje paměťové nároky algoritmu
- je nutné, kolik elementárních buněk algoritmus spotřebuje, tou může být proměnná (integer, float, double, ...)
- opět se jedná o funlci závislosti velikosti vstupu na množství využitých elementárních buněk

2. Asymptotický růst funkcí

- Landalova notace $\mathcal{O}$
- funkce $f(x)$ má složitost $\mathcal{O}(g(x))$, pakliže existuje kladné raálné číslo k takové, že $f(x) \leq k \cdot g(x)$ platí pro skoro všechna x (múže selhat pouze konečný počet vyjímek), poté pravíme, že funkce $g(x)$ je *asymptotický horní odhad* funkce $f(x)$
- poté můžeme pravit, že $\mathcal{O}(g)$ je množina všech funkcí f, které splňují předešlou definici
- takže $\mathcal{O}(n^2) \subseteq \mathcal{O}(n^3)$
- asymptotická složitost je způsob klasifikace algoritmů

3. Složitost základních algoritmů

- $\mathcal{O}(log(n))$: binární vyhledávání (vyhledávací strom)
- $\mathcal{O}(1)$: skok v poli dle indexu
- $\mathcal{O}(n)$: vyhledávání v neseřazeném poli
- $\mathcal{O}(n \cdot log(n))$: merge sort, typ řadícího algoritmu

4. Třídy složitosti N a NP

- již jsme hovořili o rozhodovacích problémech, příklad: zda-li je daný prvek obsažen v grafu
- pakliže nám někdo předloží už takový odjekt, je snadné ověřit, jestli daný objekt splňuje požadovanou vlastnost
- problém je když takový objekt nemáme, potřebujeme jej najít, proto zavádíme dvě základní třídy složitosti
- **Třída složitosti P**: Třída rozhodovacích problémů, které jsou řešitelné v polynomiálním čase. Jinými slovy problém náleží množině P právě tehdy, pokud pro každý vstup algoritmus řešící daný problém doběhne v čase $f(x)$, f je polynom k-tého stupně. Třída tedy zachycuje ty problémy, které se dají efektivně řešit.
- **Třída složitosti NP**: Opět se jedná o třídu rozhodovacích algoritmů. Aby problém L náležel této třídě, musí existovat problém K, který náleží třídě P, tak, že pro každý vstup x je $L(x) = 1$, když pro nějaký polynomiální řetězec y platí $K(x, y) = 1$. Takže algoritmus K řeší problém L, ale kromě vstupu x má i polynomiálně dlouhou nápovědu y. Tedy je-li L(x) = 1, musí existovat alespoň jedna nápověda y, kterou algoritmus K schválí. Můžeme si to ještě představit tak, že y je certifikát, který je algoritmem K ověřitelný v polynomiálním čase. Pro kladnou odpověď musí existovat alespoň jeden schválený certifikát, pro zápornou musí být všechna y odmítnuta.
- platí, že $P \subseteq NP$