Kapitola 1

Jak porovnávat algoritmy?

Co je to Algoritmus?
Adam i Božena nezávisle na sobě napsali program,
který hraje šachy. Jak poznat, který program je lepší?

1.1 Algoritmy + Datové struktury = Programy

A co je to ten algoritmus? Neformálně se dá říci, že algoritmus je recept. Dostaneme zadání (co chceme uvařit), přísady (ingredience) a poté provádíme posloupnost kroků podle receptu. Skončíme s požadovaným výsledkem (uvařeným jídlem). Recept, nebo také pracovní postup, by měl mít přesně stanoveno, jak vypadá vstup (suroviny), co má být výstupem (jméno jídla) a dále by měl srozumitelně popisovat, co se má se vstupem dělat, abychom dostali požadovaný výstup.

Odkud pochází název algoritmus? Ještě ve středověku se počítalo s římskými čísly. Umíte sečíst dvě římská čísla, aniž byste je přepsali do desítkové soustavy? Pokud budou čísla dostatečně malá, tak to zvládneme na prstech, ale zkuste sečíst $MCDXLVIII + DCCCXXII$. Moc to nejde a to jsme je ještě nezkoušeli násobit nebo dělit. Desítková soustava vznikla až 6. stol v Indii. Jeden perský astronom a matematik, Al-Khwarizmi, napsal někdy kolem roku 825 spis "O počítání s indickými čísly". Ve spise ukazuje, jak jednoduše sčítat, odčítat, násobit a dělit. Dokonce uměl počítat i druhé odmocniny a relativně dobře vyčíslit π. Jeho spis byl ve 12. století přeložen do latiny jako "Algoritmi de numero Indorum", což znamená něco jako "Algoritmi o číslech od Indů". Algoritmi bylo jméno autora. Lidé názvu špatně porozuměli a od té doby se ujal pojem algoritmus jako metoda výpočtu. Jiní tvrdí, že to bylo na počest autora.

Co jsou to ty datové struktury? Datová struktura je způsob, jak si v počítači organizovat a ukládat data, aby se s nimi dobře a efektivně pracovalo. Příkladem datových struktur je reprezentace čísel, se kterými chceme počítat. Můžeme si je zapsat jako římská čísla nebo v poziční desítkové soustavě. S datovými strukturami jsou přímo spojeny i algoritmy, které na nich pracují.

Samotný recept na jídlo ještě nezaručuje, že bude jídlo výborné. Záleží i na kuchaři. Podobně samotný algoritmus není zárukou skvělého programu, ale záleží na i programátorovi, který algoritmus implementuje pro konkrétní počítač a v konkrétním programovacím jazyce.

Program realizující algoritmus si můžeme představit jako takovou chytrou skříňku, které předhodíme vstup a ona nám po nějaké době "vyplivne" výstup. Čas, za jak dlouho nám skříňka vydá výstup, záleží na algoritmu a vstupu.

Když dostanete nový problém, ne který chcete vymyslet algoritmus, který ho řeší, tak si v první řadě musíte ujasnit, jak vypadají vstupy algoritmu a jak má

vypadat výstup. Teprve pak má smysl přemýšlet o samotném algoritmu.[1] Je dobré dát si nějaký čas na to, abychom si rozmysleli řešení, a pak terpve začít programovat. Řada programátorů se dopouští té chyby, že začnou příliš brzy psát řešení a až v půlce zjistí, že řeší jiný problém. Na druhou stranu nemá smysl příliš dlouho vymýšlet a plánovat. Klidně můžeme nejprve naprogramovat prototyp, získat nové poznatky a zkušenosti, a pak teprve napsat dokonalou verzi.

Při vymýšlení složitějších algoritmů často potřebujeme efektivně vyřešit řadu "základních problémů", jako je třídění, vyhledávání, fronta, zásobník a další. Protože se tyto problémy vyskytují opravdu často, tak se i hodně studují. Známe pro ně celou řada efektivních řešení v podobě algoritmů a datových struktur. Jejich dobrá znalost patří k základnímu programátorskému "know how". Více se o nich můžete dozvědět v literatuře. Pro jednoduchý úvod doporučujeme knihu P. Töpfer: Algoritmy a programovací techniky [**?**]. Pro znalce, hledající opravdové skvosty, doporučujeme D. Knuth: The Art of Computer Programming [**?**, **?**, **?**]. Poměrně dost informací najdete i na internetu. Internetová verze má o proti knize tu výhodu, že může obsahovat plně audiovizuální prezentaci, například animace průběhu algoritmů.

1.2 Jak poznat, který algoritmus je lepší?

Když Vy i Váš kamarád dostanete stejné zadání problému, tak nejspíš každý navrhnete jiný algoritmus. Někdy i Vás samotné napadne více řešení. Jak poznat, který algoritmus je lepší? Mohli byste navrhnout, že ten, který proběhne rychleji. A nebo to bude ten, který bude potřebovat méně paměti? A nebo oba algoritmy proběhnou tak rychle a spotřebují tak málo paměti, že vybereme ten, který má kratší a jednoduší zdrojový kód? Všechny odpovědi mohou být správné. Záleží, co chceme optimalizovat:

- **Rychlost výpočtu,** tj. za jak dlouho program proběhne. Jak dlouho budeme muset čekat, než se dozvíme výsledek?
- **Paměťovou náročnost.** Kolik paměti program zabere? Bude stačit paměť, kterou v počítači máme? Pokud program potřebuje více paměti, než je k dispozici, tak se chybějící paměť nahradí pamětí na pevném disku. Ta je výrazně pomalejší a proto se tím zpomalí výpočet.
- **Rychlost, za jak dlouho program napíšeme.** Některé věci potřebujeme spočítat jen jednou. Potom nezáleží tolik na rychlosti výpočtu, jako na celkovém čase, než program napíšeme a než program proběhne. Je jedno, jestli program poběží 1s nebo 5min, protože jeho naprogramování nám zabere podstatně více času.

  Do času, za jak dlouho program napíšeme, spadá i odladění chyb. A to někdy trvá hodně dlouho. V kratších a jednodušších algoritmech je menší šance, že uděláme chybu.

Dobrý programátor si nejprve promyslí, na které z těchto kritérií se chce zaměřit a podle toho vybere vhodný algoritmus.

První dvě kritéria nás přivádí ke dvěma důležitým pojmům – časové a prostorové složitosti. Zhruba řečeno, *časová složitost* říká, jak dlouho algoritmus poběží v závislosti na velikosti vstupních dat. *Prostorová (paměťová) složitost* říká, kolik paměti je potřeba k vykonání algoritmu v závislosti na velikosti vstupních dat.

[1] Odpovídá to i řízení projektů a komunikaci mezi lidmi vůbec. Nejprve se musíme shodnout na pojmech a jejich významu, v těchto pojmech si ujasníme, co po nás kdo chce a co očekává. Teprve když už tohle všechno víme, tak můžeme začít projekt řešit.

Tyto dvě věci spolu úzce souvisí. Zrychlení výpočtu můžeme dosáhnout tím, že si něco předpočítáme. Příkladem může být program, který na vstupu dostane $n \in \{1, \ldots, 50\}$, a má odpovědět jestli lze šachovnici velikosti $n \times n$ proskákat šachovým koněm tak, abychom každé políčko navštívili právě jednou. Všechny odpovědi si můžeme předpočítat a uložit do tabulky. Nejrychlejší program se jen podívá do tabulky a vrátí odpověď. Použitelnější příklad může být předpočítání si prvočísel.

Předpočítané věci si ale musíme někam uložit a to nám může zvýšit prostorovou složitost. Proto se často stává, že rychlejší algoritmy mají větší prostorovou složitost a naopak.[2]

1.2.1 Praktické porovnávání algoritmů

Porovnání algoritmů lze provádět teoreticky i prakticky. Teoreticky můžeme nalézt odhad na počet kroků algoritmu (tj. jeho rychlost) a na spotřebu paměti. Často už v teoretických odhadech dostaneme takové rozdíly, že nemá smysl algoritmy dále porovnávat. Ale pokud je algoritmus složitý, tak může být nalezení správných odhadů těžké. Nejlepší odhady, které umíme ukázat mohou být mnohokrát horší než reálné hodnoty. Také záleží na tom, na kterých datech/vstupech budeme algoritmy používat. Z lepší znalosti vstupních dat můžeme ukázat mnohem lepší odhady než pro obecná data. Proto je lepší brát odhady časových složitostí jen jako první a hrubé porovnání. Pro lepší porovnání algoritmů nám nezbude nic jiného, než oba algoritmy naprogramovat. Potom oba programy spustíme na používaných datech, změříme časy výpočtu a velikosti zabrané paměti, a naměřené údaje porovnáme.

Dokonce se může stát, že praktickým porovnáním algoritmů dostane opačný závěr než teoretickým porovnáním. Tedy že se algoritmus s teoreticky vysokou časovou složitostí může prakticky chovat lépe než algoritmus s teoreticky nízkou časovou složitostí.[3]

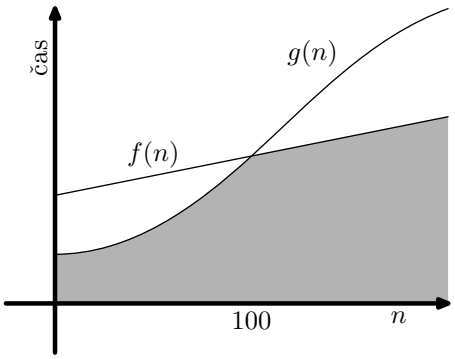
Praktickému porovnávání algoritmů se v tomto textu nebudeme příliš věnovat, ikdyž je to v praxi velmi důležité. Jenom poznamenejme, že je potřeba měřit rychlost na stejném počítači a také na stejných vstupních datech. Také dost záleží na tom, ve kterém programovacím jazyce a jak dobře je který algoritmus implementován. I elegantní algoritmus lze implementovat úplně neelegantně. Čas běhu programu je také značně závislý na hardwaru, na kterém program běží (například drobné zvětšení vstupních dat může výrazně zpomalit běh celého programu, protože se nám najednou data nevejdou do paměti a budou se muset ukládat na disk). Podobná je i závislost na operačním systému.

Pokud už máme dobře proměřené chování algoritmů (jejich časové složitosti) na různě velkých datech, tak je můžeme porovnat. Následující obrázek zachycuje grafy časové složitosti dvou programů v závisloti na velikosti vstupu n. Může se stát, že jeden algoritmus bude lepší pro menší data (například pro $n \leq 100$) a druhý pro

[2] V dnešní době se klade větší důraz na časovou složitost, protože paměti je relativně dost a není tak drahé ji dokoupit.

[3] Příkladem může být Simplexový algoritmus pro úlohy lineárního programování. V nejhorším případě má exponenciální časovou složitost, ale na praktických datech se chová daleko lépe než všechny známé polynomiální algoritmy pro lineární programování. Ty totiž mají časovou složitost něco jako $\mathcal{O}(n^{50})$ a to ještě nemluvíme o konstantách schovaných ve velkém O.

větší data (pro $n > 100$). Který algoritmus vybrat?



Oba. Nejlepšího výsledku dosáhneme kombinací obou algoritmů. Pro malá data použijeme první algoritmus a pro velká data ten druhý.

Poznámka: Výborným tréninkem praktického programování je programovací soutěž ACM. Na adrese `http://uva.onlinejudge.org/` si ji můžete vyzkoušet online mimo soutěž. (řada dalších serverů s podobnou službou vygooglíte při hledání"online judge"). Server obsahuje archiv úloh, které se už někdy objevily programovacích soutěžích.

Vyberete si úlohu a dostanete zadání, které obsahuje popis problému, formát vstupů a formát výstupů. Až napíšete program, který úlohu řeší, tak ho odešlete na server. Soudce na serveru posoudí, jestli vaše řešení vrací správné výsledky a také změří, jestli je dostatečně rychlé. Do pár vteřin vám oznámí výsledek. Pokud váš program běží déle, než je stanovený limit, tak se řešení odmítne a vy musíte přemýšlet, jak to naprogramovat lépe. Na tom se přesně naučíte prakticky srovnávat algoritmy na konkrétních vstupech (velikost vstupu stejně jako čísel na vstupu je u každé úlohy omezena konstantou ze zadání).

K tomu, abyste byli v soutěži dobří, potřebujete rychle analyzovat problém, vymyslet dostatečně dobré řešení a co nejrychleji ho naprogramovat a odladit.

1.2.2 Teoretické porovnávání algoritmů

K teoretickému porovnání dvou algoritmů (tj. aniž bychom oba algoritmy naprogramovali) stačí odhadnout počet kroků, které každý algoritmus udělá v závisloti na velikosti vstupu.

Jak to udělat si vysvětlíme v následující kapitole **??** o časové složitosti.