

Asymptotická složitost algoritmů

Daněk Antonín

ČVUT FEL, STM, 1. ročník, paralelka 109, danek@antonindanek.cz

Abstrakt: Je vysvětleno, co je to algoritmus a dále se plynule přejde na samotnou asymptotickou složitost. Čtenář by po přečtení článku měl být schopen určit asymptotickou složitost algoritmu a měl by chápat zásadní rozdíly mezi různě složitými algoritmy. Vše je vysvětleno na příkladech a v závěru je ukázka složitosti některých řadících algoritmů.

Klíčová slova: algoritmus, složitost, asymptotická složitost algoritmu, řadící algoritmy

I. Definice algoritmu

Hlavním tématem tohoto článku je v nadpisu zmíněná asymptotická složitost algoritmů, nicméně nejdříve bychom si měli ujasnit, co to je algoritmus samotný.

Algoritmus je přesný postup práce, jakým lze vyřešit daný problém. Jak je vidět, tato definice je velmi volná, a proto si jistě každý dokáže hned představit několik algoritmů, ze všedního života. Oblíbeným příkladem je kuchařka, která obsahuje algoritmy pro přípravu jídel. Ještě jednodušším příkladem pak může být algoritmus zvaný „ranní rutina“, který má každý z nás zakódovaný ve své hlavě a před odchodem do práce či školy jej automaticky provádí.

Dále je algoritmus blíže určen svými vlastnostmi, což jsou: konečnost, obecnost, determinovanost a resultativnost. Vlastnosti už však nebudeme v tomto článku blíže rozebírat. Pro náš účel bude postačovat vědět, že **asymptotická složitost algoritmů úzce souvisí s konečností a resultativností.**

II. Výpočet složitosti algoritmu

Představme si dva programy, které mají stejný výsledek (výstup), ale každý trvá různě

dlouho. Předpokládáme-li stejné podmínky, v jakých oba programy běží, jediný důvod časového rozdílu může být různý počet provedených operací. Jinak řečeno, **pokud mají programy stejný výstup, ale provádí různý počet operací, pak mají různou složitost.** Z tohoto už je krásně vidět, jak se asi bude složitost algoritmů počítat. Budeme počítat základní operace.

Základními operacemi tady ovšem ve většině případů nebudeme uvažovat mikro-operace procesoru, ale základní operace vyššího programovacího jazyka. Tzn. aritmetické operace, podmínky (testy) a přesuny v paměti.

```
static int count=0;
for(int i=0; i<pole.length; i++) {
    if(pole[i]<=i) count++;
}
```

Příklad 1: výpočet složitosti algoritmu

Podívejme se na [příklad výše](#). Na prvním řádku máme deklaraci a inicializaci proměnné, což budeme uvažovat jako jednu operaci. Následuje průchod celým polem (pole.length), které nám zde (a ve většině případů) reprezentuje data. Cyklus tedy proběhne $N \times$, kde N je velikost dat. V těle cyklu je

podmínka, která proběhne také $N \times$ a navýšení proměnné count, které nemusí proběhnout ani jednou, ale také $N \times$ (v případě, že pole bude obsahovat samé nuly).

Tím se dostáváme k pojům nejlepší, průměrný a nejhorší případ složitosti algoritmu. Při psaní programů však obvykle nemůžeme spoléhat na průměrné nebo dokonce ideální případy, a tak **složitost počítáme především pro nejhorší možný případ dat**. Složitost [našeho příkladu](#) je tedy $3N + 1$.

III. Výpočet asymptotické složitosti algoritmu

Všimněme si rozdílu mezi tímto titulkem a na první pohled stejně vypadajícím titulkem nadpisu II.

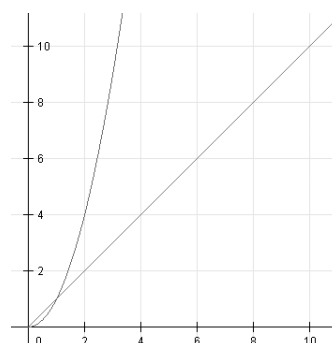
Zatím jsme počítali „obyčejnou“ složitost. Otázka však zní, jestli je něco takového vhodné a hlavně jestli má takový přesný výpočet smysl. Odpověď zní jistě ne.

Dejme si předpoklad, že mikroprocesor osobního počítače vykoná jednu naši základní operaci vyššího jazyka za $1\mu\text{s}$ (10^{-6} vteřiny). To znamená, že **za jednu vteřinu mikroprocesor vykoná milion operací**. Tímto předpokladem určitě nijak nepřevyšujeme realitu. Naopak, moderní více-jádrové procesory s vysokým taktem zvládnou i vícenásobně více. U více jádrových procesorů však přichází problém, kdy musí být program pro více jader optimalizován, a tak zůstaneme u našeho původního předpokladu, který řádově odpovídá skutečnosti.

Když máme dán předpoklad, můžeme se podívat na samotný výpočet asymptotické složitosti. Možná už tušíte, že vzhledem k vysokému výkonu procesorů, budeme mnohé operace zanedbávat a opravdu je tomu tak. Nemá smysl počítat jednotlivé operace, ale ani desítky, stovky až deseti-tisíce operací (obvykle si můžeme dovolit zanedbat i sta-

tisíce operací). Ve výsledku **zanedbáme všechny operace nezávislé na datech, dílčí jednodušší části algoritmu a multiplikační konstanty**. Podíváme se na výpočet asymptotické složitosti na [našem příkladu](#).

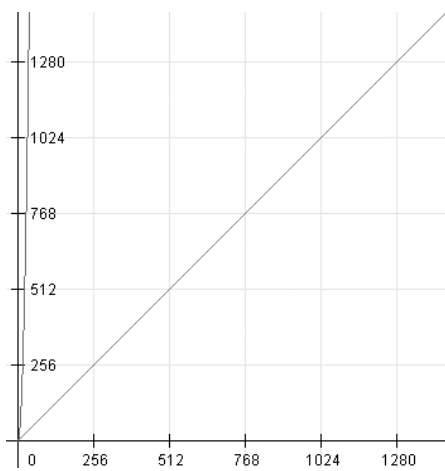
Na prvním řádku máme operaci nezávislou na datech – zanedbáváme. Následuje cyklus, který prochází celá data. Žádnou složitější část v našem krátkém algoritmu nevidíme, takže toto ponecháváme. V těle cyklu máme další dvě (nejhorší případ) operace. Ty tedy ve výsledné rovnici znázorňují multiplikační konstanty – zanedbáváme. Sečteno podtrženo, **složitost našeho algoritmu je N – tedy lineární**. Chybí nám ukázka zanedbání dílčí jednodušší části algoritmu. Můžeme si tedy představit, že jsou v příkladu ještě dva cykly for – v sobě vnořené, které oba procházejí celé data. Pak bychom náš lineární cyklus mohli zanedbat, protože ty vnořené cykly by vytvořily kvadratickou složitost, která lineární převyšuje hned $N \times$.



Graf 1: lineární a kvadratický algoritmus pro malá data

Co to ale znamená, lineární asymptotická složitost? **Asymptotická složitost algoritmu je řád růstu času v závislosti na datech**. Když máme složitost lineární, znamená to, že nám s růstem dat bude délka běhu programu růst lineárně. To je velmi pěkný algoritmus. Pro příklad a lepší pochopení si srovnáme složitost lineární a kvadratickou (N^2). Nejprve si vezmeme data o velikosti 10. U lineáry nám vyjde 10 operací, u kvadrátu 100 – oboje je pro procesor takřka zanedbatelné ([viz. Obrázek výše](#)).

Na tomto je také vidět, kdy asymptotickou složitost počítáme. Určitě ne, u běžným uživatelských částí programu, ale např. u řadících algoritmů s vysokým vstupem už ano. Nyní si zvětšíme vstupní data na milion a podíváme se, co se stane. U lineárního algoritmu je vše v pořádku. Program proběhne (při podmínkách dle našeho předpokladu) za 1s a my jeho výpočetní dobu takřka nepostřehneme. U kvadratického však už vidíme markantní rozdíl. Vyjde nám totiž milion milionů operací, což už našemu procesoru zabere 11,4 dne.



Graf 2: lineární a kvadratický algoritmus pro velká data

[Na obrázku](#) je vidět, jak se pro velká N začíná lineára značně odchylovat od paraboly. Graf je pouze pro $N = \text{cca } 1000$. Pro milionové N by už parabola nebyla vzhledem k lineáre vůbec vidět.

IV. Zanedbávání dílčích jednodušších částí algoritmu a multiplikačních konstant

Někomu, kdo zcela asymptotickou složitost algoritmů nepochopil, se nemusí zdát skutečné, že se dají zanedbat multiplikační konstanty nebo celé jednodušší části algoritmu. Vždyť pokud budu mít v programu lineární cyklus, v jehož těle bude 10 operací, doba běhu

programu musí být $10\times$ větší, než u programu s cyklem o jedné operaci. To je sice pravda, ale jak je psáno v definici, u asymptotické složitosti nám jde „pouze“ o **řád růstu času**.

Dejme tomu, že máme program, ve kterém je lineární průchod daty se 100 operacemi v každé iteraci. Kromě toho ale ještě máme nad stejnými daty průchod kvadratický. Je jasné, že můžeme lineární část zanedbat, protože i přes 100 operací v každém průchodu je tato část kódu provedena ve zlomku času, vzhledem ke kvadratickému průchodu. Samozřejmě vše za předpokladu, že pracujeme nad velkými daty, protože jak už jsme si řekli, asymptotickou složitost nemá jinde význam počítat.

Jinak řečeno, u asymptotické složitosti nás ani tak nezajímá, zda program poběží 5 nebo 10 vteřin, ale jestli výsledek dostaneme např. právě v řádu vteřin nebo si na něj budeme muset počkat několik minut, hodin či dní.

V. Množiny růstů funkcí

Pro porovnávání složitosti algoritmu používáme množiny růstů, které jsou tři. Pozorný čtenář si jistě všimnul, že jsem již před spojení složitost algoritmu nepředradil slovo asymptotická. Je to proto, že i při běžné diskuzi o složitosti algoritmu obvykle automaticky míníme složitost asymptotickou. Jak jsme si již řekli, počítat složitost přesně, na jednotlivé operace, postrádá smyslu.

Ale zpět k množinám funkcí. Množina omega (Ω) obsahuje všechny funkce, které od určitého bodu rostou rychleji, než naše $f(x)$ (nebo její násobek). Dále množina omikron (O) obsahuje všechny funkce, které od určitého bodu rostou pomaleji než $f(x)$ (nebo její násobek). A nakonec je tu množina theta (Θ) obsahující funkce rostoucí stejně rychle, jako $f(x)$. Zápis pak vypadá např. takto:

$$\log(N) \in o(N) \quad (1)$$

$$N \times \log(N) \in \Omega(N) \quad (2)$$

Tyto vzorce se dají vyjádřit slovně: $\log(N)$ roste pomaleji než N , ale $N \times \log(N)$ roste rychleji než N .

VI. Řadící algoritmy a asymptotická složitost

Jeden z nejčastějších případů, kdy se setkáváme s asymptotickou složitostí algoritmů, je u algoritmů řadících.

Řadící algoritmy se, jak název napovídá, starají o seřazení prvků v poli. Jedná se o operaci, kterou potřebujeme snad v každém složitějším programu, a tak je velmi vhodné, naučit se používat efektivní algoritmus co nejdříve. Bubble sorter je sice velmi jednoduchý na pochopení, ale jak už nyní

víme, kvadratická složitost se nedá použít u velkých vstupních dat.

Na závěr se podívejme na tabulku doby běhů různě složitých algoritmů a posuďte sami, jaké je vhodné používat a jaké ne. Ještě složitější než exponenciála je faktoriál, nicméně troufám si tvrdit, že napsat tak špatný algoritmus, je skoro umění. V rámečku jsou zvýrazněny složitosti dvou dosti známých algoritmů. **$N \times \log(N)$ odpovídá řadícímu algoritmu Heap sort, kvadratický již zmíněnému Bubble sort.** Závěrem snad už pouze jedna důležitá věc. Pokud je špatně napsaný algoritmus, velmi těžko se takový problém vyrovnává kvalitnějším hardwarem.

Tab. 1: doby běhů různě složitých algoritmů

Vstupní data	Asymptotická složitost					
	1	N	$N \times \log(N)$	N^2	N^3	2^N
1	1 μ s	1 μ s	1 μ s	1 μ s	1 μ s	2 μ s
10	1 μ s	10 μ s	10 μ s	100 μ s	1 ms	1 ms
100	1 μ s	100 μ s	200 μ s	10 ms	1 s	3×10^{11} dob ledových
1000	1 μ s	1 ms	3 ms	1 s	16,6 min	...
1000000	1 μ s	1 s	6 s	11,4 dnů	31 709 let	

VII. Zdroje

- [1] Přednáškové snímky pro předmět Algoritmizace (RNDr. Marko Genyk-Berezovskij)
- [2] Wikipedia: http://cs.wikipedia.org/wiki/Asymptotická_složitost (7. dubna 2009)