Digitální učební materiál

Číslo projektu:	CZ.1.07/1.5.00/34.0548
Název školy:	Gymnázium, Trutnov, Jiráskovo náměstí 325
Název materiálu:	VY_32_INOVACE_148_IVT
Autor:	Ing. Pavel Bezděk
Tematický okruh:	Algoritmy
Datum tvorby:	srpen 2013
Ročník:	4. ročník a oktáva
Anotace:	Algoritmus VIII. – Polynomiální a nepolynomiální algoritmy
Metodický pokyn:	Při výuce nutno postupovat individuálně.

esf evropský sociální fond v ČR
EVROPSKÁ UNIE
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY
OP Vzdělávání pro konkurenceschopnost
INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Autor	**Ing. Pavel Bezděk**		
Vytvořeno dne	**1. 8. 2013**		
Odpilotováno dne	**3. 2. 2014**	**ve třídě**	**8.Y**
Vzdělávací oblast	**Informatika a informační a komunikační technologie**		
Vzdělávací obor	**Informatika a výpočetní technika**		
Tematický okruh	**Algoritmus**		
Téma	**Algoritmus VIII. - Polynomiální algoritmus**		
Klíčová slova	**Algoritmus, polynomiální algoritmy**		

Polynomiální a nepolynomiální algoritmy

Druhy asymptotických složitostí

Časové složitosti algoritmů (ty paměťové jsou obdobné) jsme seřadili od nejrychlejších ke nejpomalejším a ke každé připsali příklad algoritmu.

O(1) – konstantní (třeba zjištění, jestli je číslo sudé)

O(log (log N)) – dvojitě logaritmická, obecně **O((log^{x}N) - polylogaritmická,**

O(N$^{(1/2)}$) odmocninová

O(log N) – logaritmická (binární vyhledávání); všimněte si, že na základu logaritmu nezáleží, protože platí $\log_a n = \log_b n / \log_b a$, takže logaritmy o různých základech se liší jen konstanta-krát, což se „schová do O-čka“ .

O(N) – lineární (hledání maxima z N čísel)

*O(N * log N)* – lineárně-logaritmická (nejlepší algoritmy na třídění pomocí porovnávání), obecně O(N*log^x(N))

O(N^2) – kvadratická (Bubble Sort)

O(N^3) – kubická (násobení matic podle definice)

O(2^N) – exponenciální (nalezení všech posloupností délky N složených z nul a jedniček; pokud je chceme i vypsat, dostaneme O(N * 2^N))

O(N!) – faktoriálová, N ! = 1 * 2 * 3 * . . . * N (nalezení všech permutací N prvků, tedy třeba všech přesmyček slova o N různých písmenech)

O(N^N) – exponenciální zobrazeni do sebe: je potřeba alespoň N*N kroků

Polynomiální algoritmy

Polynomiální algoritmy říkáme těm, které patří do $O(N^k)$ pro přirozené hodnoty k (hodnoty **k** 1, **2, 3,..**).

1. **Čas, který potřebuji ke zpracovaní** vstupních dat algoritmy považované za rychlé, bývá zpravidla **shora omezen funkcemi typu n, n.log (n), n^2, …**

2. ***Jako horní hranici lze vždy použit polynom***

(Algoritmus je polynomiální, pokud ho lze shora omezit n^k)

3. Jeho výpočetní složitost lze definovat jako $f(n)=O(n^k)$, k je přirozené číslo

4. Nepolynomiální algoritmus - když neexistuje přirozené k

5. ***Těmto algoritmům říká polynomiální*** nebo také prakticky použitelné

6. Do polynomiálních algoritmů patří například i algoritmus se složitostí O(log N). A to proto, že $O(\log N) \subset O(N)$. Proto každý algoritmus, který proběhne v čase O(log N), proběhne i v O(N).

Nepolynomiální algoritmy

Nepolynomiální (exponenciální) algoritmy

1. Pomalé (prakticky nepoužitelné) algoritmy užívají metodu „hrubé síly“ (brute force), kterou **probírají všechny možnosti**.

2. Mají tu vlastnost, že funkci jejich časové složitosti nelze shora ohraničit žádným polynomem

3. Jsou použitelné jen pro malé objemy dat, pro větší objemy dat je výpočet v nereálném čase (např. stovky, tisíce, milióny let by trval výpočet i na super rychlých počítačích).

4. Zejména sem patří úlohy na matematických grafech. Odhad počtu kroků je alespoň exponenciální.

Je-li např. daná n prvková množina a probírají-li se všechny její:

a) podmnožiny: je potřeba alespoň 2^N kroků

b) permutace: je potřeba alespoň N! kroků

c) zobrazeni do sebe: je potřeba alespoň N*N kroků

5. Nepolynomiální jsou třídy **$O(2^N)$, …** a **O(N!)** a **$O(N^N)$.** Takové algoritmy jsou extrémně pomalé a snažíme se jim co nejvíce vyhýbat.

6. **Nemožnost tyto praktické úlohy řešit výpočtem algoritmů této třídy, *se v praxi řeší sestavením algoritmů s přijatelnou (polynomickou) časovou složitostí pro tyto úlohy*, *které neřeší danou úlohu přesně, ale jen přibližně*, čímž typicky dávají o něco horší výsledky, než by dal přesný algoritmus, který ovšem má nepolynomickou časovou složitost.**

Převody „pomalých“ algoritmů na „rychlejší“ algoritmy

Řešíme-li n rovnic o n neznámých používáme k tomu **Gaussovu eliminační metodu**. Převedeme matici soustavy na trojúhelníkový tvar - diagonální matice (pod hlavní diagonálou jsou nuly).

Popř. můžeme i použít Gaussova-Jordanova metoda – řádkové úpravy matice soustavy, při nichž redukovaná matice po všech n-1 fázích eliminace je diagonální nebo dokonce jednotková.

Když ale řešíme na počítači 20 rovnic o 20 neznámých, přestává již být čas výpočtu reálný a při vyšším počtu rovnic a neznámých je již zcela nereálný. Proto zde již nemůžeme používat tyto přímé metody řešení soustav lineárních rovnic jako je Gaussova nebo Gaussova - Jordanova metoda řešení soustav lineárních rovnic, ale musíme použít méně přesné rychlejší metody, které nám ale poskytují dostatečnou přesnost, popř. si můžeme zvolit přesnost výpočtu. Menší přesnost - velmi rychlý výpočet, větší přesnost – výpočet je o něco méně rychlejší, ale pořád ještě pro naše potřeby rychlý, stále se pohybujeme v reálných časech. Řešíme soustavu lineárních rovnic postupným přibližováním se k přesnému řešení. Existuje řada způsobů konstrukce takové posloupnosti aproximací (přibližných hodnot), jejíž limitou $\mathbf{x_i}$ je přesné řešení dané soustavy rovnic.

Např. **Jacobiho metoda** nebo **Gaussova- Seidelova metoda**

(Problémy převodu matematických úloh neřešitelných v reálných časech na úlohy jednodušší řešitelné na počítači v reálném čase se zabývá **numerická matematika**.) Těmto metodám bude věnován samostatný DUM.

Třídy složitostí

Asymptotická složitost rozděluje algoritmy do tříd složitostí, u kterých platí, *že* ***od určité velikosti dat, je algoritmus dané třídy vždy pomalejší než algoritmus třídy předchozí, bez ohledu na to, jestli je některý z počítačů c-násobně výkonnější (c je konstanta).***

Škála v nekonečnu slouží k rozlišení jednotlivých tříd. Říká, že **pokud se *n* blíží k nekonečnu, tak neexistuje reálná konstanta taková, aby byl algoritmus z vyšší třídy rychlejší, než ten z třídy přechozí.**

$O(1) \ll O(\log(N)) \ll O(N) \ll O(N*\log(N)) \ll O(N^k) \ll O(k^N) \ll O(N!) \ll O(N^N)$

Pro představu o tom, jak se složitost projevuje na opravdovém počítači, se podíváme, jak dlouho poběží algoritmy na počítači, který provede 10^9 (miliardu) operací za sekundu. Tento počítač je srovnatelný s těmi, které dnes běžně používáme. Podívejme se, jak dlouho na něm poběží algoritmy s následujícími složitostmi:

Počítač provede min. 10^9 operací za sekundu

reálný čas | nereálný čas | několik sekund

množství vstupních dat		2	10	100	1000	10^{+4}	10^{+5}	10^{+6}	10^{+7}
O(1)	počet operací	20	20	20	20	20	20	20	20
	čas	$2*10^{-8}$ s	$2*10^{-8}$ s	$2*10^{-8}$ s	$2*10^{-8}$ s	$2*10^{-8}$ s	$2*10^{-8}$ s	$2*10^{-8}$ s	$2*10^{-8}$ s
O(logN)	počet operací	1	1	2	3	4	5	6	7
	čas	10^{-9} s	10^{-9} s	$2*10^{-9}$ s	$3*10^{-9}$ s	$4*10^{-9}$ s	$5*10^{-9}$ s	$6*10^{-9}$ s	10^{-8} s
O(N)	počet operací	2	10	100	1000	10^{+4}	10^{+5}	10^{+6}	10^{+7}
	čas	$2*10^{-9}$ s	10^{-8} s	10^{-7} s	10^{-6} s	10^{-5} s	10^{-4} s	0,001 s	1 s
O(N* log N)	počet operací	1	10	200	3000	$4*10^{+4}$	$5*10^{+5}$	$6*10^{+6}$	$7*10^{+7}$
	čas	10^{-9} s	10^{-8} s	$2*10^{-7}$ s	$3\text{-}10^{-6}$ s	$4*10^{-5}$ s	0,0005 s	0,006 s	9 s
$O(N^2)$	počet operací	4	100	10000	10^{+6}	10^{+8}	10^{+10}	10^{+12}	10^{+14}
	čas	$4*10^{-9}$ s	10^{-7} s	10^{-5} s	0,001 s	0,1 s	10 s	17 min	32 let
$O(N^3)$	počet operací	8	1000	10^{+6}	10^{+9}	10^{+12}	10^{+15}	10^{+18}	10^{+21}
	čas	$8*10^{-9}$ s	10^{-6} s	0,001 s	1 s	17 min	12 dní	32 let	$3*10^{+13}$let
$O(2^N)$	počet operací	4	1024	10^{+30}	10^{+301}	∞	∞	∞	∞
	čas	$4*10^{-9}$ s	10^{-6} s	$4*10^{+13}$ let	∞	∞	∞	∞	∞

Vliv objemu dat na reálnost výpočtového času

Povšimneme si, že horší algoritmy jsou lepší než optimální algoritmy, jen pro min. data (1 a 2), pro větší data jsou již velmi pomalé

(a s rostoucími vstupy – nepoužitelné).

Optimální algoritmy jsou stále rychlé i při velmi rostoucích vstupech!!!

Modrá pole - časy blížící se jedné nebo několika sekundám.

Červená pole – zcela nereálné časy, algoritmy nepoužitelné

Asymptotická složitost konstantní O(1), logaritmická O(log N), lineární O(N) a lineárně-logaritmická O(N*logN),{ popř. i odmocninová složitost $O(N^{(1/2)})$ }, **zcela vyhovují i pro extrémně velká vstupní data**.

Asymptotická kvadratická složitost sice ještě zvládá i větší data, ale u extrémně velkých dat je nepoužitelná.

Asymptotická složitost kubická je u běžně velkých dat pomalá a při větších vstupech již nepoužitelná.

Asymptotická složitost exponenciální je nepoužitelná i pro běžné velikosti dat, lze ji použít jen pro malé vstupy.

Závěr

Z toho pro nás plyne, že se **vždy snažíme vymyslet algoritmy s asymptotickou časovou složitostí menší než kvadratickou, (pokud to nevede k extrémním paměťovým nárokům, které by počítač nezvládl).**

Pokud jsme nuceni použít algoritmus s **asymptotickou časovou kvadratickou složitostí**, musíme počítat s tím, že **pro extrémně velká data se nehodí.**

Algoritmy se složitostí vyšší než je kubická, jsou použitelné jen pro malé vstupy dat.

Pokud ale musíme řešit úlohy, pro které máme algoritmy se časovou složitostí kvadratickou a vyšší, i pro velké objemy dat, musíme tyto algoritmy převést na algoritmy, které sice poskytují trochu méně přesný výsledek (ale v rámci nám vyhovující přesnosti), ale mají nižší časovou složitost, a proto jsou použitelné i pro vyšší objemy dat.

Použité zdroje

BÖHM, Martin. *Programátorská kuchařka: Recepty z programátorské kuchařky* [online]. Praha: KSP MFF UK Praha, 2011/2012 [cit. 2013-08-01]. KSP, Korespondenční seminář z programování: Programátorské kuchařky, 24. ročník KSP. Dostupné z: http://ksp.mff.cuni.cz/tasks/24/cook1.html