Programátorské strategie

Ivana Kolingerová
Centrum počítačové grafiky
a vizualizace dat
Západočeská univerzita, Plzeň

http://iason.zcu.cz/~kolinger

Úvod do algoritmizace

Ivana Kolingerová

Obsah:

1. Algoritmizace: proč se tím vůbec zabývat?
2. Správnost a účinnost algoritmu
3. Robustnost algoritmu
4. Analýza algoritmů
5. Hledání řešení neznámého problému

1. Algoritmizace: proč se tím vůbec zabývat?

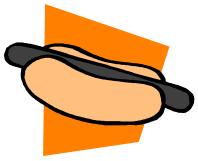
„Nechci se živit programováním, tak to nepotřebuji.“

Ale: algoritmizace rozvíjí abstraktní a logické myšlení, potřebné zejména, ale nejen, pro techniky

„Všechno je hotovo a v knihovnách.“

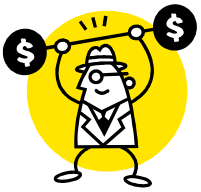
Ale: vždycky člověk narazí na problém, který je "trochu jiný"

Je váš cíl být opravdový programátor ?



Šéf dá úkol:

- Dány současné ceny vepřového, obilí, hořčice, …
- Dána omezení, z čeho se skládá hotdog.
- Vytvořte co nejlevnější hotdog.



Reálný život klade takové úkoly každý den.

Vaše odpověď:

- Hm? Řekněte mi, co mám naprogramovat.

S pokrokem v softwarových inženýrských systémech bude potřeba opravdových programátorů klesat.

Budoucnost patří počítačovým odborníkům, kteří

- znají základní problémy a jejich řešení na současném stavu poznání
- ovládají principy a techniky k řešení obrovské oblasti neznámých problémů, které se v rychle se měnící oblasti objevují

Obsah předmětu

- Přehled technik řešení problémů a návrhu algoritmů
- Abstraktní myšlení
- Návrh algoritmů pro nové problémy
- Šedá je teorie, zelený strom života: něco - snad spíše více než méně - zajímavých problémů k praktickému procvičení

2. Správnost a účinnost algoritmu

Algoritmus
postup k vyřešení určitého úkolu,
musí řešit obecný, dobře specifikovaný problém

Př.: Řazení
Input: Posloupnost klíčů $a_1,\ldots,a_n$
Output: Permutace (přeuspořádání) vstupní posloupnosti:
$a_1' \leq a_2' \leq a_n'$

Rozlišovat mezi problémem a jeho instancemi

Př.: instance řazení - pole jmen

Algoritmus - procedura, která libovolná možná vstupní data (instance) transformuje na požadovaný výstup

Algoritmus - vždy vede k cíli, heuristika - nemusí

Cíl snažení: algoritmy správné, efektivní, snadno implementovatelné (nemusí jít vše najednou)

• Teoret. návrh x průmyslová praxe

důraz na efektivitu a správnost
spíše než na jednoduchost implementace

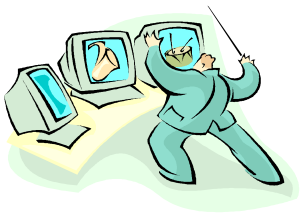
jednoduchost implementace

Doporučení: pozornost obojímu

Make it simple!

• Nutnost důkladného vyzkoušení nového algoritmu (důkaz = obtížné, "it is obvious" - nestačí)

- Potřebujeme větší zlepšení výkonu - víc pomůže lepší algoritmus než lepší počítač !
- Kdy nehledat účinnější řešení: poběží jen několikrát nebo výpočet přes noc je OK
- Optimalizovat "bottleneck" - kritické místo (obvykle 90% času v 10% kódu)



3. Robustnost algoritmu

Cíl snažení: algoritmy odolné vůči numerickým aj. chybám

- Singulární případy: při první úvaze ignorovat, pak zahrnout
- Robustnost: malá chyba při výpočtu nesmí vést k selhání - algoritmy často odvozeny např. pro nekonečnou přesnost reálných čísel (Populární “eps” technika - zdroj nekonzistence (a=b, b=c, a<>c)

4. Analýza algoritmů

• Cíl snažení: hodnocení a porovnávání algoritmů nezávislé na typu počítače a na jazyku, ale také typické chování pro očekávaná vstupní data

• Nástroj: asymptotická analýza složitosti a experimentální otestování na reprezentativních vstupních datech

• Složitost – paměťová x časová x předzpracování

• Nejčastěji nás zajímá nejhorší případ a očekávaný případ => složitost v nejhorším případě, očekávaná složitost

Užívaná symbolika složitostí:

- $O(f(n))$ horní mez
- $\Omega(f(n))$ dolní mez
- $\theta(f(n))$ omezeno shora i zdola (optimální algoritmus)

Časová složitost **algoritmu**:

- nejdelší doba výpočtu potřebná pro vstup velikosti n

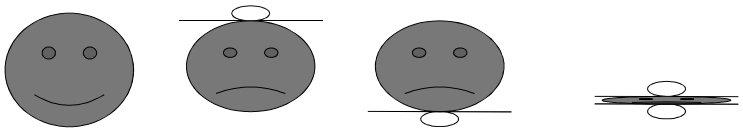
- O(n^2)*:* Dokázat, že pro každý vstup velikosti n algoritmus potřebuje nejvýše čas cn^2
- Ω(n^2): Najít aspoň jeden vstup velikosti n , pro který algoritmus potřebuje nejméně tento čas
- θ(n^2)*:* Obojí.

Pro nás: u algoritmů nám obvykle stačí O() hodnocení

Časová složitost **problému**:

- čas. složitost nejrychlejšího algoritmu řešícího tento problém

- *O(n^2):* Najít algoritmus, který řeší tento problém v čase nejvýše cn^2
- *Ω(n^2):* Dokázat, že žádný algoritmus neřeší tento problém rychleji
- *θ(n^2):* Obojí.



Problém *O()* *Ω()* *θ()*

Experimentální ověření složitosti algoritmu:

- Spotřeba času a paměti jako funkce velikosti vstupu
- Pro zmenšení chyby měřit více opakování, bez I/O, pro více dat. množin, pro více typů dat
- Spočítat *t/f(n),* zkoumat průměrné a nejhorší chování

Očekávaná složitost (Expected complexity) - odhad pozorovaného chování algoritmu,
často odhad podle implementace,
závisí též na očekávaném typu vstupních dat

Standardní třídy složitosti problému

- P - problém z této třídy řešitelný výpočty v polynomiálním čase

- NP - řešitelný v polynomiálním čase nedeterministicky (tj. nevíme, jak řešit polynomiálně, ale umíme v polyn. čase ověřit řešení) dnes známo asi 3000 problémů

- Nejtěžší problémy z NP - NP-úplné, pro ně není známo polyn. řešení, ale nebylo dokázáno, že neexistuje (tyto problémy všechny ekvivalentní - - buď v P všechny nebo žádný)

Chcete se proslavit v computer science?

- Dokažte nebo vyvraťte pro nějaký NP-úplný problém, že je nebo není v P

Př.: TSP je NP-úplný problém.
Jako NP problém by se formulovalo:
Najděte cestu max. délky B

Rychlost růstu složitostí:

- konstantní *O(1)*
- logaritmická *O(log n)*
- polylogaritmická $O(\log^k n)$, k = konst.
- polynomy $O(n^k)$, k = konst.
- exponenciální $O(2^{kn})$, k = konst.
- exp $O(2^{n^k})$, k = konst.
- dvojitý exp $O(2^{2^{(kn)}})$, k = konst.

:-(

Logaritmická:

- $\log_{10} n$ = # číslic pro zápis n
- $\log_2 n$ = # bitů pro zápis n
  $= 3.32 \log_{10} n$
- $\log(n^{1000}) = 1000 \log(n)$

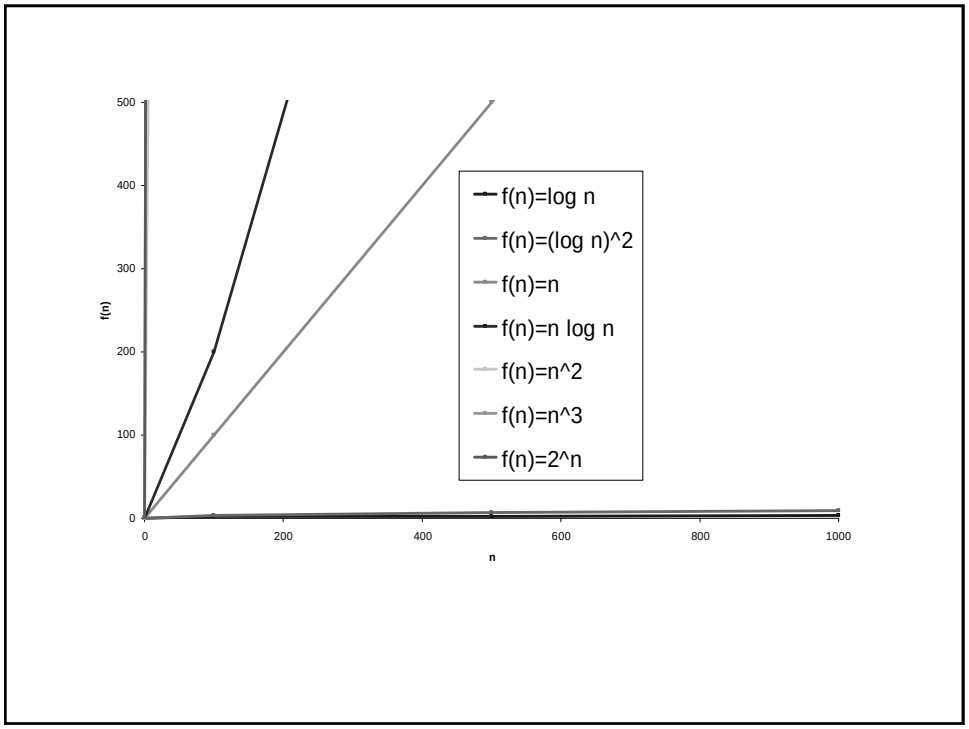
Liší se pouze násobící konstantou

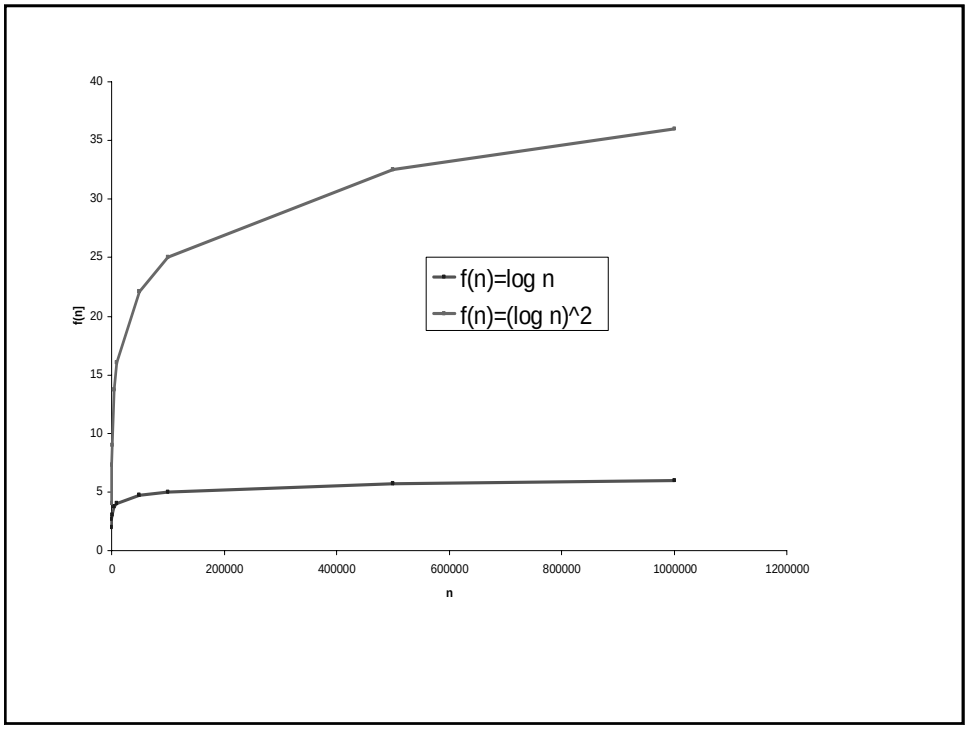
Rychlost růstu složitostí:

- konstantní $O(1)$
- logaritmická $O(\log n)$
- polylogaritmická $O(\log^k n)$, k = konst.
- polynomy $O(n^k)$, k = konst.

práh praktické použitelnosti

- exponenciální $O(2^{kn})$, k = konst.
- exp $O(2^{n^k})$, k = konst.
- dvojitý exp $O(2^{2^{(kn)}})$, k = konst.


500
400
300
200
100
0
f(n)
0
200
400
600
800
1000
n
f(n)=log n
f(n)=(log n)^2
f(n)=n
f(n)=n log n
f(n)=n^2
f(n)=n^3
f(n)=2^n


40
35
30
25
20
15
10
5
0
f(n)
0
200000
400000
600000
800000
1000000
1200000
n
f(n)=log n
f(n)=(log n)^2

Rychlost růstu složitostí:

- log n, n, n log n, n^2, n^3, 2^n, $n!$ - pro $n < 10$ zhruba stejné
- $n!$ bezcenné pro n asi 20
- 2^n bezcenné pro n asi 40
- n^2 bezcenné pro n kolem 1000
- n, n log n - OK asi do miliardy
- log n - OK vždy

5. Hledání řešení neznámého problému

(S. Skiena: The Algorithm Design Manual, Spring-Verlag, New York Berlin Heidelberg, 1998)

Pomáhá klást si správné otázky (v uvedeném pořadí):

1. Opravdu problému rozumím?
   a) Co je vstupem?
   b) Jaký přesně má být výstup?
   c) Umím sestavit malý příklad a vyřešit ručně? Co se stane, když to zkusím?
   d) Jak moc je pro mou aplikaci důležité najít vždy přesné, optimální řešení? Nestačí něco, co obvykle funguje „docela dobře“?
   e) Jak velká je typická instance mého problému? 10? 1000? 1 mil.?
   e) Jak důležitá je pro moji aplikaci rychlost?

f) Musí být problém vyřešen za 1s? 1 min? 1 h? 1 den?
g) Kolik času a úsilí mohu dát do implementace?
h) Budu řešit numerický problém? Grafový? Geometrický? Se znakovým řetězcem? Více možných formulací? Která se zdá nejlehčí?

2. Umím najít jednoduchý algoritmus nebo heuristiku řešící daný problém?
   a) Umím najít algoritmus správně řešící můj problém prohledáním všech podmnožin a výběrem nejlepší? (Pokud ano, jsem si jist správností odpovědi? Umím změřit kvalitu nalezeného řešení? Stačí časově? Pokud ne, mám problém dostatečně definovaný, aby se dal vyřešit?)
   b) Umím problém vyřešit opakovaným výběrem nejlepšího? Opakovaným náhodným výběrem? (Pokud ano, pro jaký vstup to funguje dobře, špatně? Je to rychlé?)

3. Není můj problém v katalogu algoritmických problémů?
   a) Pokud ano, co je o problému známo? Není k mání implementace řešení?
   b) Pokud ne, je to správné místo? Umím hledat v knihách? ;-)
   c) Co Web?
4. Existující speciální případy, které umím řešit přesně?
   a) Umím to, pokud ignoruji některé parametry?
   b) Co se stane, když některé parametry nastavím na triviální hodnoty, jako je 0,1?
   c) Umím problém zjednodušit na případ, který je možné řešit přesně? Je teď triviální nebo stále zajímavý?
   d) Pokud už umím řešit spec. případ, proč nejde řešení použít pro obecnější problém?
   e) Je můj problém spec.případ některého obecného problému?

5. Které ze standardních paradigmat návrhu algoritmů je nejvhodnější pro můj problém?
   a) Je možné položky setřídit? Ulehčí to řešení? *(SORT)*
   b) Je možné problém rozdělit na 2 nebo více podproblémů? Malý a velký? Levý a pravý? 2 stejně velké? *(D&C)*
   c) Má vstup nebo řešení přirozené pořadí zpředu dozadu, zleva doprava (řetězce, permutace, listy stromu)? *(DP)*
   d) Opakuje se stejná operace nad stejnými daty (hledání), kterou by urychlila pomocná dat. struktura? *(slovník, hašovací tabulka, hromada, prioritní fronta…)*
   e) Lze použít náhodné vzorkování pro výběr dalšího objektu? Výběr nejlepší z náhodně vybraných konfigurací? *(řízená náhodnost, např.SA)?*
   f) Problém pro *lineární programování*? *Celočíselné programování*?

   g) Vypadá problém podobně jako obchodní cestující nebo jiný NP-úplný problém? *(NP-úplný)*
6. Jsem stále ztracena(a)?
   a) Nenajmu si na to experta?
   b) Nezkusím otázky projít znova? (Změnily se moje odpovědi?)

Vylepšování řešení: přes algoritmy a/nebo
přes datové struktury, pomáhá znalost problému
a vhodných technik

Př.: Hledat v *n*-prvkovém neseřazeném poli

Nejhorší případ? O(*n*)
Nejlepší? O(1)
Očekávaný? Podle očekávaných vstup. dat,
zjistit z implementace.
Opakované hledání? => předzpracování:

- Strom, hromada apod.
- Seřadit a půlit interval , tj. předzpr. cca O(*n* log *n*),
  1 dotaz O(log *n*)

Literatura k předmětu :

- S. Skiena: The Algorithm Design Manual,
  Spring-Verlag, New York Berlin Heidelberg, 1998

- Podklady k přednáškám a další zdroje od přednášející

Další možné zajímavé zdroje:

- J.Hromkovič: Algorithms for Hard Problems, Introduction to Combinatorial Optimization, Randomization, Approximation and Heuristics, Springer Verlag Berlin Heidelberg New York, 2000
- Z.Michalewicz, D.B. Fogel: How to Solve It: Modern Heuristics, Springer-Verlag, Berlin Heidelberg New York, 2000
- Problems from ACM Programming Contest - http://contest.felk.cvut.cz

- S. Dvořák: Dekompozice a rekurzivní algoritmy, Grada, 1992
- G. Gonnet, R. Baeza – Yates: Handbook of Algorithms and Data Structures, Addison-Wesley, Wokingham, UK, 1991 (2nd edition)
- G. Rawlins: Compared to What ? Computer Science Press, New York, 1992
- B. Moret, H. Shapiro: Algorithm from P to NP: Design and Efficiency, Benjamin/Cummings, Redwood City, US, 1991