Zložitosť algoritmov

IB110

Zložitosť algoritmov

- koreknosť algoritmu sama o sebe nezaručuje jeho použiteľnosť
- dĺžka výpočtu a jeho pamäťová náročnosť
- časová a priestorová zložitosť
- zložitosť výpočtu závisí na vstupnej inštancii
- zložitosť algoritmu vyjadrujeme ako funkciu dĺžky vstupnej inštancie

Optimalizácia zložitosti algoritmu

- na úrovni kompilácie
- programátorská optimalizácia

Optimalizácia - príklad 1

Vstup zoznam študentov a počet bodov, ktoré získali v záverečnom teste predmetu IB110
$L(1), \ldots, L(N)$

Výstup normalizované body
(1) Nájdi maximálny počet bodov, *MAX*
(2) každý bodový zisk vynásob hodnotou 100 a vydeľ hodnotou *MAX*

Implementácia (1) štandartne

(2) **for** I **from** 1 **to** N **do** $L(I) \leftarrow L(I) \times 100/MAX$ **od**

pre každé $L(I)$ *potrebujeme 1 násobenie a 1 delenie*

Optimalizácia (1) štandartne

(2) $FAKTOR \leftarrow 100/MAX$

(3) **for** I **from** 1 **to** N **do** $L(I) \leftarrow L(I) \times FAKTOR$ **od**

zlepšenie o cca 50%

Optimalizácia - príklad 2

- vyhľadávanie prvku X v neusporiadanom zozname
- implementácia pomocou cyklu, v ktorom sa realizujú dva testy: (1) našli sme X? a (2) prehľadali sme celý zoznam?
- optimalizácia: na koniec zoznamu pridáme prvok X a v cykle testujeme len podmienku (1)
- po ukončení cyklu overujeme, či nájdeny prvok X sa nachádza vo vnútri zoznamu alebo na jeho konci
- zlepšenie o cca 50%

Zložitosť

- je zlepšenie o 50% (60%, 90% . . .) dostačujúce?
- ako charakterizovať zložitosť algoritmu?
- ako porovnať zložitosť dvoch algoritmov?

zložitosť algoritmu ako funkcia dĺžky vstupnej inštancie

zložitosť v najhoršom prípade

asymptotická zložitosť, O-notácia

Asymptotická zložitosť

- jednoduchá charakterizácia efektivity algoritmu
- umožňuje porovnať relatívnu efektivitu rôznych algoritmov
- charakterizuje, ako rastie zložitosť algoritmu s rastúcou dĺžkou vstupnej inštancie

O-notácia

Symbolom $\mathcal{O}(g(n))$ označujeme množinu fukcií t.ž.

$$\mathcal{O}(g(n)) = \{f(n) \mid \text{existuje kladná konštanta } c \text{ a } n_0 \text{ také, že } 0 \leq f(n) \leq cg(n) \text{ pre všetky } n \geq n_0\}.$$

Rovnosť $f(n) = \mathcal{O}(g(n))$ vyjadruje, že $f(n)$ je prvkom množiny $\mathcal{O}(g(n))$.

Asymptotická zložitosť - príklad

Binárne vyhľadávanie položky Y v telefónnom zozname s N položkami $X_1, X_2, \ldots, X_N$ pre $N = 1000000$

lineárne vyhľadávanie až 1 000 000 porovnaní

zložitosť $\mathcal{O}(N)$

binárne vyhľadávanie postup: v prvom kroku porovnaj Y s X_{500000}

podľa výsledku porovnaj v druhom kroku Y buď s X_{250000} alebo s X_{750000}

v najhoršom prípade 20 porovnaní

zložitosť $\mathcal{O}(\log_2(N))$

Prečo?

sme ako zložitosť vyhľadávania sme uvažovali len počet porovnaní?

Robustnosť O-notácie

Fakt

O-notácia zakrýva konštantné faktory

- časová zložitosť algoritmu je relatívny pojem
- zložitosť je relatívna voči fixovanej množine elementárnych inštrukcií
- každý programovací jazyk resp. kompilátor môže mať inú množinu elementárnych inštrukcií
- pokiaľ používajú štandartné inštrukcie, tak rozdiel v časovej zložitosti je práve o konštantný faktor
- O-notácia je invariantná voči takýmto implementačným detailom

Nevýhoda O-notácie: skryté konštanty, hraničná hodnota n_0

Príklad — bubblesort

```
1 procedure BUBBLESORT(A, n)
2 for i = 1 to n − 1 do
3     for j = 1 to n − i do
4         if A[j] > A[j + 1] then  vymeň A[j] s A[j + 1] fi
5     od
6 od
```

riadok 4 čas $\mathcal{O}(1)$

for cyklus 3 - 5 čas $\mathcal{O}(n - i)$

for cyklus 2 - 6 čas $\mathcal{O}(\sum_{i=1}^{n-1}(n - i))$

$$= \mathcal{O}(n(n-1) - \sum_{i=1}^{n-1} i) = \mathcal{O}(n^2)$$

celková časová zložitosť je $\mathcal{O}(n^2)$

Príklad — vnorené cykly

```
1 r ← 0
2 for i = 1 to n do
3     for j = 1 to i do
4         for k = j to i + j do
5             r ← r + 1
6         od
7     od
8 od
```

cyklus 4 - 6 $(i+j)-j+1 = i+1$ priradení

cyklus 3 - 7 i opakovaní cyklu 4 - 6, tj. $i(i+1) = i^2 + i$ priradení

cyklus 2 - 8 $n-1$ opakovaní cyklu 3 - 7, tj. $\sum_{i=1}^{n-1} i^2 + i$ priradení

$$\sum_{i=1}^{n-1} i^2 + i = \frac{(n-1)n(2n-1)}{6} + \frac{(n-1)n}{2} = \frac{n^3 - n}{3} = \mathcal{O}(n^3)$$

Príklad — maximálny a minimálny prvok

Problém nájdenia maximálneho a minimálneho prvku postupnosti $S[1..n]$.
Zložitostné kritérium - počet porovnaní prvkov.

$max \leftarrow S[1]$
for i **from** 2 **to** n **do**
    if $S[i] > max$ **then** $max \leftarrow S[i]$ **fi**
od

Minimum nájdeme medzi zvyšnými $n-1$ prvkami podobne.
Celkove $(n-1)+(n-2)$ porovnaní.

Maximálny a minimálny prvok

Prístup Rozdeľ a panuj

1. pole rozdeľ na dve (rovnako veľké) podpostunosti
2. nájdi minimum a maximum oboch podpostupností
3. maximálny prvok postupnosti je väčší z maximálnych prvkov podpostupností; podobne minimálny prvok

function MAXMIN(x, y)
if $y - x \leq 1$ **then return** $(\max(S[x], S[y]), \min(S[x], S[y]))$
else $(max1, min1) \leftarrow$ MAXMIN$(x, \lfloor (x+y)/2 \rfloor)$
$(max2, min2) \leftarrow$ MAXMIN$(\lfloor (x+y)/2 \rfloor + 1, y)$
return $(\max(max1, max2) \min(min1, min2))$
fi

Maximálny a minimálny prvok

Zložitosť: (počet porovnaní)

$$T(n) = \begin{cases} 1 & \text{pre } n = 2 \\ T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 2 & \text{pre } n > 2 \end{cases}$$

Indukciou k n overíme, že $T(n) \leq \frac{5}{3}n - 2$.

1. Pre $n = 2$ platí $\frac{5}{3} \cdot 2 - 2 > 1 = T(2)$.
2. Predpokladajme platnosť nerovnosti pre všetky hodnoty $2 \leq i < n$, dokážeme jej platnosť pre n.

$$\begin{aligned} T(n) &= T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 2 && \text{indukčný predp.} \\ &\leq \frac{5}{3}\lfloor n/2 \rfloor - 2 + \frac{5}{3}\lceil n/2 \rceil - 2 + 2 = \frac{5}{3}n - 2 \end{aligned}$$

Priemerná a očakávaná zložitosť

Priemerná zložitosť priemer zložitostí výpočtov na všetkých vstupoch danej dĺžky
Quicksort - zložitosť v najhoršom prípade je $\mathcal{O}(n^2)$, *priemerná zložitosť je* $\mathcal{O}(n \log n)$

Očakávaná zložitosť zložitosť jednotlivých výpočtov je vážená frekvenciou výskytu príslušných vstupných inštancií

Výhody: presnejšia informácia o efektivite algoritmu
v prípade očakávanej zložitosti je relevancia voči aplikačnej oblasti
Nevýhody: obtiažna analýza
v prípade očakávanej zložitosti nutnosť poznať presnú frekvenciu vstupných inštancií

Horné a dolné odhady zložitosti

Zložitosť algoritmu

- v najlepšom prípade
- v najhoršom prípade
- priemerná zložitosť
- očakávaná zložitosť

Zložitosť problému

- dolný odhad zložitosti problému
- horný odhad zložitosti problému — zložitosť konkrétneho algoritmu pre problém
- zložitosť problému

Dolný odhad zložitosti problému - techniky

Informačná metóda riešenie problému v sebe obsahuje isté množstvo informácie a v každom kroku výpočtu sme schopní určiť len časť tejto informácie (*násobenie matíc, cesta v grafe, triedenie*)

Metóda sporu *Varianta A:* za predpokladu, že algoritmus má zložitosť menšiu než uvažovanú hranicu, vieme skonštruovať vstup, na ktorom nedá korektné riešenie.
Varianta B: za predpokladu, že algoritmus nájde vždy korektné riešenie, vieme skonštruovať vstup, pre ktorý zložitosť výpočtu presiahne uvažovanú hranicu.

Redukcia

Dolný odhad zložitosti pre problém maximálneho prvku postupnosti

Dolný odhad

Nech algoritmus $\mathcal{A}$ je algoritmus založený na porovnávaní prvkov a nech $\mathcal{A}$ rieši problém maximálneho prvku. Potom $\mathcal{A}$ musí na každom vstupe vykonať aspoň $n-1$ porovnaní.

Dôkaz

Nech $x = (x_1, \ldots, x_n)$ je vstup dĺžky n, na ktorom $\mathcal{A}$ vykoná menej než $n-1$ porovnaní a nech x_r je maximálny prvok v x.
Potom v x musí existovať prvok x_p taký, že $p \neq r$ a v priebehu výpočtu x_p nebol porovnávaný so žiadnym prvkom väčším než on sám. Existencia takého prvku plynie z počtu vykonaných porovnaní.
Ak v x zmeníme hodnotu prvku x_p na $x_r + 1$, tak $\mathcal{A}$ určí ako maximálny prvok x_r – spor.

Dolný odhad zložitosti pre problém vyhľadávania v telefónnom zozname

binárny strom a jeho hĺbka

Výzkum v oblasti zložitosti problémov

- optimalizácia dátových štruktúr
- dolné odhady zložitosti a dôkaz optimality
- priestorová zložitosť
- vzťah medzi priestorovou a časovou zložitosťou