Algoritmer, datastrukturer och komplexitet

Övning 1

Anton Grensjö
grensjo@csc.kth.se

14 september 2015

Kursplanering

F1: Introduktion, algoritmanalys

F2: Sortering

F3: Datastrukturer

Ö1: Algoritmanalys

F4: Datastrukturer

F5: Grafsökning

Ö2: Datastrukturer, grafer

Översikt

- Algoritmanalys
- Varför?
- Idag:
  - Definitioner av $\mathcal{O}, \Theta, \Omega$
  - Repetition av matematik
  - Exempel på ordoräkning
  - Exempel på algoritmanalys

Kostnadsmått

Enhetskostnad

- Varje operation tar en tidsenhet
- Varje variabel tar en minnesenhet
- Används när algoritmen räknar med begränsade tal

Bitkostnad

- Varje bitoperation tar en tidsenhet
- Varje bit tar upp en minnesenhet
- Används när algoritmen räknar med godtyckligt stora tal

Modell för algoritmen

- Låt $T(n)$ vara antalet operationer (alt. tiden) en algoritm behöver för att lösa problemet för indata av storleken n.
- Undersök hur $T(n)$ beter sig för stora n.
  - Varför är vi endast intresserade av detta?
- Det vi bryr oss om är främst flaskhalsen i vår algoritm.

Asymptotisk notation

Vi fokuserar på den dominerande termen och struntar i konstanta faktorer.

Definition

$$\lim_{n\to\infty}\frac{g(n)}{f(n)} = \begin{cases} 0 & \text{om } g(n) \in o(f(n)) \\ c > 0 & \text{om } g(n) \in \Theta(f(n)) \\ \infty & \text{om } g(n) \in \omega(f(n)) \end{cases}$$

$$\left.\begin{matrix} o(f(n)) \\ \Theta(f(n)) \end{matrix}\right\} \mathcal{O}(f(n)), \qquad \left.\begin{matrix} \Theta(f(n)) \\ \omega(f(n)) \end{matrix}\right\} \Omega(f(n))$$

Gränsvärden

Metod då $n \to \infty$: Bryt ut nämnarens dominerande term.

Exempel

$$\lim_{n\to\infty} \frac{5n^4 - 2n^3 + 5}{2n^6 - n^2 + n} = \lim_{n\to\infty} \frac{n^6 \overbrace{\left(\frac{5}{n^2} - \frac{2}{n^3} + \frac{5}{n^6}\right)}^{\to 0}}{n^6 \underbrace{\left(2 - \frac{1}{n^4} + \frac{1}{n^5}\right)}_{\to 2}} = 0$$

Logaritmlagar

Definition

Logaritmen definieras som inversen till exponentialfunktionen, dvs:

$$\log 2^n = n$$

$$2^{\log n} = n$$

I den här kursen använder vi oss av notationen $\log n = \log_2 n$.

Logaritmlagar

Sats

$$\log a + \log b = \log ab \qquad \log a - \log b = \log \frac{a}{b}$$

$$\log a^n = n \log a \qquad \log_a(n) = \frac{\log_b(n)}{\log_b(a)}$$

Följdsats

För alla reella tal $a, b > 1$ gäller att

$$\log_a n \in \theta(\log_b n)$$

Uppgift 1: Ordo

Jämför följande par av funktioner med avseende på hur dom växer då n växer. Tala i varje fall om ifall $f(n) \in \Theta(g(n))$, $f(n) \in O(g(n))$ eller $f(n) \in \Omega(g(n))$.

	$f(n)$	$g(n)$
a)	$100n + \log n$	$n + (\log n)^2$
b)	$\log n$	$\log n^2$
c)	$\frac{n^2}{\log n}$	$n(\log n)^2$
d)	$(\log n)^{\log n}$	$\frac{n}{\log n}$

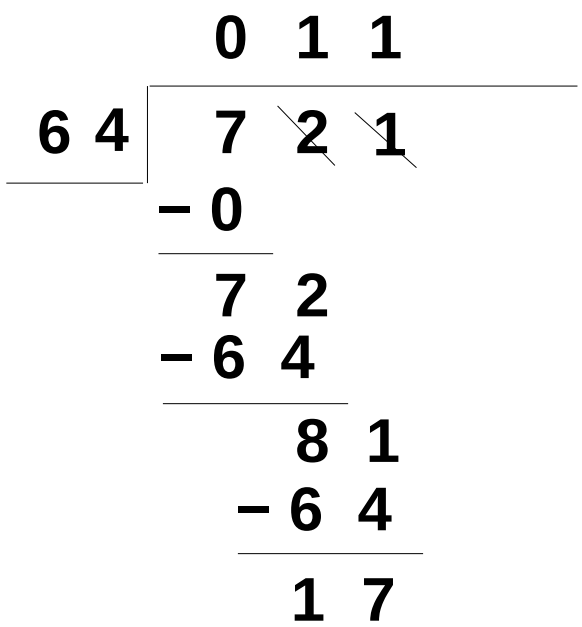
Uppgift 1: Ordo

Jämför följande par av funktioner med avseende på hur dom växer då n växer. Tala i varje fall om ifall $f(n) \in \Theta(g(n))$, $f(n) \in O(g(n))$ eller $f(n) \in \Omega(g(n))$.

	$f(n)$	$g(n)$
e)	$\sqrt{n}$	$(\log n)^5$
f)	$n2^n$	3^n
g)	$2^{\sqrt{\log n}}$	$\sqrt{n}$

Lösningar finns på kurshemsidan, under Kursinnehåll/Detaljschema.

Uppgift 2: Division



Uppgift 2: Division

Följande algoritm bestämmer q och r så att $a \cdot q + r = b$, givet två n-bitstal a och b i basen B $a = a_{n-1}a_{n-2}\cdots a_0, \quad b = b_{n-1}b_{n-2}\cdots b_0$. Bestäm tidskomplexiteten.

$Div(a, b, n) =$

PRE: $a > 0, b \geq 0$, a *och* b *lagras med* n *bitar.*

POST: $qa + r = b, 0 \leq r < a$

$r \leftarrow 0$

for $i \leftarrow n - 1$ **to** 0 **do**

INV: $(q_{n-1}...q_{i+1}) \cdot a + r = (b_{n-1}...b_{i+1}), 0 \leq r < a$

$r \leftarrow (r \ll 1) + b_i$ /* *Byt till nästa siffra* */

$q' \leftarrow 0$

$a' \leftarrow 0$

while $a' + a \leq r$ **do** /* *Hitta max* q' *så att* $q'a \leq r$ */

INV: $a' = q'a \leq r$

$a' \leftarrow a' + a$

$q' \leftarrow q' + 1$

$q_i \leftarrow q'$

$r \leftarrow r - a'$

return (q, r) /* *kvot och rest* */

Uppgift 3: Euklides algoritm

Analysera Euklides algoritm, som hittar största gemensamma delaren mellan två heltal, både med avseende på enhetskostnad och bitkostnad. Vad är skillnaden?

Vi förutsätter att $a \geq b$.

$gcd(a, b)$=
    if $b|a$ **then**
        $gcd \leftarrow b$
    else
        $gcd \leftarrow gcd(b, a \bmod b)$

Uppgift 3: Euklides algoritm

$gcd(a, b)$=
    if $b|a$ **then**
        $gcd \leftarrow b$
    else
        $gcd \leftarrow gcd(b, a \bmod b)$

1. Terminerar algoritmen?
   - Betrakta variabeln a.
   - Eftersom $a \geq b$ måste a minska i varje anrop.
   - Men a kan aldrig bli mindre än 1.
   - $\implies$ algoritmen terminerar.
2. Bestäm tidskomplexitet.

Uppgift 3: Euklides algoritm

Tidskomplexitet

$gcd(a, b)$=
  if $b|a$ **then**
    $gcd \leftarrow b$
  else
    $gcd \leftarrow gcd(b, a \bmod b)$

- Antalet anrop verkar bero på parametrarnas storlek.
- Låt oss beräkna hur a minskar.
- Låt a_i vara värdet på a i det i:te anropet.

Uppgift 3: Euklides algoritm

Tidskomplexitet

$gcd(a, b)$=
    if $b|a$ **then**
        $gcd \leftarrow b$
    else
        $gcd \leftarrow gcd(b, a \bmod b)$

Lemma

$$a_{i+2} \leq a_i/2$$

Bevis.

- $a_{i+2} = b_{i+1}, \quad b_{i+1} = a_i \bmod b_i \implies a_{i+2} = a_i \bmod b_i$
- Antag att $a_{i+2} > a_i/2$. Av detta följer att $b_i > a_i/2$. Detta är en motsägelse, ty $a_i = a_{i+2} + cb_i > a_i/2 + a_i/2 = a_i$

□

Uppgift 3: Euklides algoritm

Tidskomplexitet

$gcd(a, b)$=
  if $b|a$ **then**
    $gcd \leftarrow b$
  else
    $gcd \leftarrow gcd(b, a \bmod b)$

- Lemmat ($a_{i+2} \leq a_i/2$) ger att

$$\lceil \log a_{i+2} \rceil \leq \lceil \log \frac{a_i}{2} \rceil = \lceil \log a_i - \log 2 \rceil = \lceil \log a_i \rceil - 1$$

- Storleken på a minskas alltså åtminstone med ett vartannat anrop. Totala antalet anrop $\leq 2\lceil \log a \rceil = 2n$, där n är antalet bitar i a.
- Enhetskostnad: Varje anrop sker på konstant tid $\implies \mathcal{O}(n)$.
- Bitkostnad: Varje enskilt anrop har komplexiteten $\mathcal{O}(n^2)$ vilket ger total tidskomplexitet $\mathcal{O}(n^3)$.
- Vad är lämpligast när?

Upgift 4: Potenser med upprepad kvadrering

Följande algoritm beräknar tvåpotenser när exponenten själv är en tvåpotens.

Indata: $m = 2^n$
Utdata: 2^m

$power(m) =$
    $pow \leftarrow 2$
    for $i \leftarrow 1$ **to** $\log m$ **do**
        $pow \leftarrow pow \cdot pow$
    return pow

Analysera denna algoritm både med avseende på enhetskostnad och bitkostnad och förklara vilken kostnadsmodell som är rimligast att använda.

Upgift 4: Potenser med upprepad kvadrering

Indata: $m = 2^n$
Utdata: 2^m

$power(m) =$
    $pow \leftarrow 2$
    for $i \leftarrow 1$ **to** $\log m$ **do**
        $pow \leftarrow pow \cdot pow$
    return pow

- Enhetskostnad:
  - Slingan går $\log m = n$ varv.
  - Varje varv tar konstant tid.
  - Tidskomplexiteten är: $\mathcal{O}(n)$.

Upgift 4: Potenser med upprepad kvadrering

Indata: $m = 2^n$

Utdata: 2^m

$power(m) =$

$pow \leftarrow 2$

for $i \leftarrow 1$ **to** $\log m$ **do**

$pow \leftarrow pow \cdot pow$

return pow

- Bitkostnad:
  - Multiplikation av två l-bitstal tar $\mathcal{O}(l^2)$ tid.
  - Vid varv i så är $pow = pow_i = 2^{2^i}$.
  - Tidskomplexiteten för varv i:

    $$\mathcal{O}\left((\log pow_i)^2\right) = \mathcal{O}\left((\log 2^{2^i})^2\right) = \mathcal{O}\left((2^i)^2\right) = \mathcal{O}\left(2^{2i}\right)$$

  - Summera över alla varv:

    $$\sum_{i=1}^{\log m} c2^{2i} = c\sum_{i=1}^{\log m} 4^i = 4c\sum_{i=0}^{\log m - 1} 4^i = 4c\frac{4^{\log m} - 1}{4 - 1} \in O(4^{\log m}) = O(4^n).$$

Uppgift 4: Potenser med upprepad kvadrering

Indata: $m = 2^n$
Utdata: 2^m

$power(m) =$
    $pow \leftarrow 2$
    for $i \leftarrow 1$ **to** $\log m$ **do**
        $pow \leftarrow pow \cdot pow$
    return pow

Slutsatser:

- $\mathcal{O}(n)$ vid enhetskostnad, $\mathcal{O}(4^n)$ vid bitkostnad!
- Vilket mått är bäst?
- Kan man använda denna algoritm för att beräkna tvåpotenser för godtyckliga heltalsexponenter?
- Andra baser än 2? Matriser?