UPPSALA UNIVERSITET

MA1: L04

Tom Smedsaas

Asymptotisk notation
Ordo ($\mathcal{O}$), Omega (Ω) och Theta (Θ)

UPPSALA UNIVERSITET

Asymptotisk notation

Exempel på formuleringar:

- *Den enkla algoritmen för att beräkna x^n är $\Theta(n)$*
- *Kvadreringsalgorimen för att beräkna x^n är $\Theta(\log n)$*
- *Binärsökningsalgoritmen är $\Theta(\log n)$*

- *Linjärsökningsalgoritmen är $\mathcal{O}(n)$ alltid*
- *Linjärsökningsalgoritmen är $\mathcal{O}(1)$ i bästa fall*
- *Linjärsökningsalgoritmen är $\mathcal{O}(n)$ i genomsnitt*

Asymptotisk notation - definition

En funktion $t(n)$ sägs vara $\mathcal{O}(f(n))$ om det existerar två konstanter c och n_0 sådana att

$$|t(n)| < c|f(n)| \quad \text{för alla} \quad n > n_0$$

Observera att detta är en *matematisk* definition!

Asymptotisk notation

Påstående: $f(n) = n^2 + 10n + 100$ är $\mathcal{O}(n^2)$

Varför?

Om $n > 1$ så

$$|f(n)| = n^2 + 10n + 100 < n^2 + 10n \cdot n + 100 \cdot n \cdot n = 111n^2$$

Alltså duger konstanterna $c = 111$ och $n_0 = 1$

Observera att det är *existensen* som är viktig – inte värdena!

UPPSALA UNIVERSITET

Asymptotisk notation

Eftersom $\mathcal{O}$ är en *övre* gräns behöver det inte innebära så mycket.

Funktionen på förra bilden är t.ex. automatiskt $\mathcal{O}(n^3)$ och $\mathcal{O}(2^n)$.

För att ange en *undre* gräns används Ω en liknande definition med > i stället för <.

Exempelfunktionen är alltså också $\Omega(n^2)$.

En funktion som är både $\mathcal{O}(f(n))$ och $\Omega(f(n))$ sägs vara $\Theta(f(n))$.

I datavetenskapen är det vanligt att säga $\mathcal{O}$ när man menar Θ.

UPPSALA UNIVERSITET

Användning i algoritmanalys

Funktionerna som vi uppskattar är *tid* eller *antal gånger* någon central operation utförs.

Exempel: Antal multiplikationer som utförs i kvadreringsalgoritmen för att beräkna x^n är $\Theta(\log n)$.

Behöver vi inte ange bas i logaritmen?

Nej, eftersom

$$\log_a x = \frac{1}{\log_b a} \cdot \log_b x$$

UPPSALA UNIVERSITET

Användning i algoritmanalys

Exempel:

$\mathcal{O}(1)$	indexering i array (Python-lista)
$\Theta(\log n)$	binär sökning, kvadreringsalgoritmen
$\Theta(n)$	`insert(0, x)` i en lista med n lagrade värden
$\Theta(n \log n)$	snabba sorteringsmetoder
$\Theta(n^2)$	enkla sorteringsmetoder
$\Theta(n^3)$	multiplikation av $n \times n$ - matriser

UPPSALA UNIVERSITET

I algoritmanalys

När man analyserarar algoritmer kan man behöva skilja på bästa, värsta och genomsnittlig komplexitet.

Exempel:

Att sortera n element med *insticksortering* kräver

- $\Theta(n)$ operationer i bästa fall
- $\Theta(n^2)$ operationer i värsta fall
- $\Theta(n^2)$ operationer i genomsnitt

Vad menar man med ”*i genomsnitt*”?

UPPSALA UNIVERSITET

I algoritmanalys

När man diskuterar sökning behöver man ofta skilja mellan ”lyckad” och ”misslyckad” sökning.

Till exempel, funktionen

```
def search(x, lst):
    for e in lst:
        if e == x:
            return True
    return False
```

kräver för *lyckad sökning*

- $\Theta(1)$ operationer i bästa fall
- $\Theta(n)$ operationer i genomsnitt
- $\Theta(n)$ operationer i värsta fall

medan misslyckad sökning alltid kräver $\Theta(n)$ operationer.

UPPSALA UNIVERSITET

Tidsuppskattningar

Antag att vi vet att tiden för en viss algoritm är $\Theta(f(n))$. Då kan vi uppskatta tiden för stora värden på n med funktionen

$$t(n) = c \cdot f(n)$$

Konstanten c beror på många saker (dator, os, programmerare, …).

För en viss implementation och dator kan den uppskattas genom att göra en eller flera tidsmätningar.

Exempel

Antag att tiden för ett program som implementerar en $\Theta(n \log n)$ - algoritm har uppmätts till 1 sek för en $n = 10^3$.
Hur lång tid kommer programmet då ta för $n = 10^6$?

$$t(n) = c\, n \log n$$

$$t(10^3) = c \cdot 10^3 \log 10^3 = 1$$

$$c = 1/3000$$

$$t(10^6) = \frac{1}{3000} \cdot 10^6 \log 10^6 = 2000 \text{ sek } \approx 33 \text{ min}$$

UPPSALA UNIVERSITET

Exempel

Antag samma mätning men att det är en $\Theta(n^2)$ - algoritm.
Hur lång tid kommer programmet då ta för $n = 10^6$?

$$t(n) = c \cdot n^2$$

$$t(10^3) = c \cdot (10^3)^2 = 1$$

$$c = 10^{-6}$$

$$t(10^6) = 10^{-6} \cdot (10^6)^2 = 10^6 \text{ sek} \approx 12 \text{ dagar}$$

Slutsats: En $\Theta(n \log n)$ - algoritm är *mycket* snabbare än en $\Theta(n^2)$ - algoritm för stora värden på n.

UPPSALA UNIVERSITET

The end

.