

Föreläsning

Datastrukturer (DAT036)

Nils Anders Danielsson

2013-10-28

Motivation

Varför studera datastrukturer?

- ▶ Snabbt.
- ▶ Lite minne.
- ▶ Mindre energiförbrukning.
- ▶ Billigare.
- ▶ Och korrekt!

Exempel

```
public class Bits {  
  
    public static void main(String[] args) {  
  
        int n = Integer.parseInt(args[0]);  
  
        String bits = new String();  
  
        for (int i = 0; i < n; i++) {  
            bits += i & 1;  
        }  
  
        System.out.print(bits);  
  
    }  
}
```

Hur lång tid tar det att köra "java Bits 200000", jämfört med "java Bits 100000"?

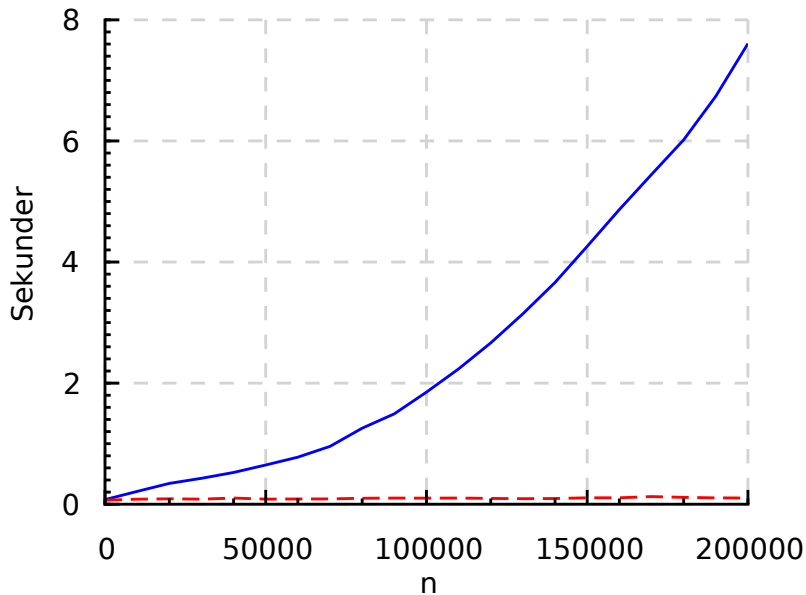
- ▶ Lika snabbt, 2, 3 eller 4 ggr långsammare?
- ▶ Fundera ett tag.
- ▶ Gå sedan till <http://pingo.upb.de/>.
- ▶ Ange en kod som jag ger er.
- ▶ Svara på frågan.

Dåligt val av datastruktur

```
public class Bits {  
  
    public static void main(String[] args) {  
  
        int n = Integer.parseInt(args[0]);  
  
        String bits = new String();  
  
        for (int i = 0; i < n; i++) {  
            bits += i & 1;  
        }  
  
        System.out.print(bits);  
  
    }  
}
```

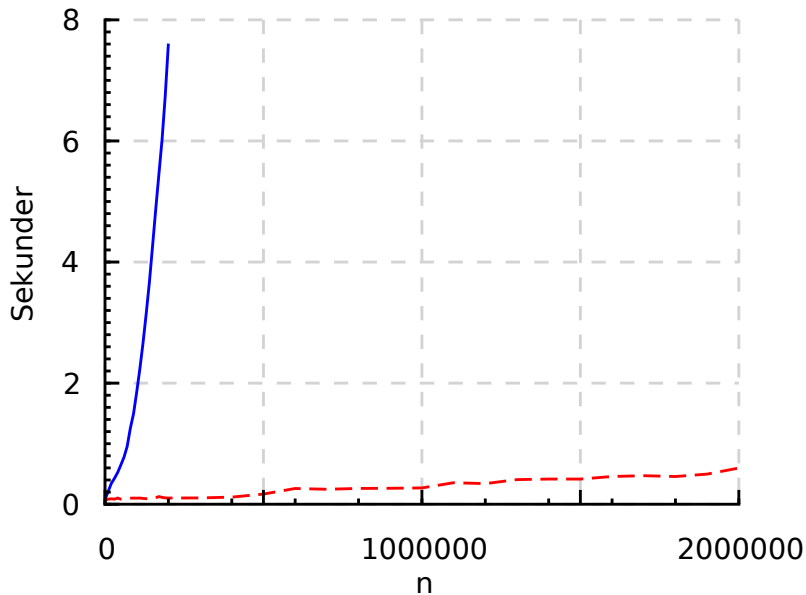
Bättre val av datastruktur

```
public class FastBits {  
  
    public static void main(String[] args) {  
  
        int n = Integer.parseInt(args[0]);  
  
        StringBuffer bits = new StringBuffer();  
  
        for (int i = 0; i < n; i++) {  
            bits.append(i & 1);  
        }  
  
        System.out.print(bits);  
  
    }  
}
```



— Bits

- - - FastBits



— Bits

- - - FastBits

Motivation

I kursen får ni verktyg för att:

- ▶ Snabba upp era program på motsvarande sätt.
- ▶ Förutse (ungefär) hur snabbt era program kommer att gå.

Administrativt

Kurshemsidan.

Tillbaka till exemplet

```
private char[] string;

public void append(char c) {
    char[] temp = new char[string.length + 1];

    for (int i = 0; i < string.length; i++) {
        temp[i] = string[i];
    }
    temp[string.length] = c;

    string = temp;
}
```

Tillbaka till exemplet

```
private char[] string;

public void append(char c) {
    char[] temp = new char[string.length + 1]; // ~ n

    for (int i = 0; i < string.length; i++) { // ~ 3n
        temp[i] = string[i];
    }
    temp[string.length] = c; // 1

    string = temp; // 1
}
```

Totalt (för $n = \text{string.length}$): $\sim 4n$.

Tillbaka till exemplet

```
private char[] string;

public void append(char c) {
    char[] temp = new char[string.length + 1]; // ~ n

    for (int i = 0; i < string.length; i++) { // ~ 3n
        temp[i] = string[i];
    }
    temp[string.length] = c; // 1

    string = temp; // 1
}
```

Totalt (för $n = \text{string.length}$): $\sim n$.

Tillbaka till exemplet

```
String bits = new String();           // 1

for (int i = 0; i < n; i++) {        // ???
    bits += i & 1;                   // ~ i
}

System.out.print(bits);              // ~ n
```

???

$$\begin{aligned}\sum_{i=0}^{n-1} i &= 0 + 1 + 2 + \dots + (n - 1) \\ &= n \frac{n - 1}{2} \\ &\sim n^2\end{aligned}$$

Tillbaka till exemplet

```
String bits = new String();           // 1

for (int i = 0; i < n; i++) {        //  $\sim n^2$ 
    bits += i & 1;                    //  $\sim i$ 
}

System.out.print(bits);              //  $\sim n$ 

Totalt:  $\sim n^2$ .
```


Tillbaka till exemplet

```
String bits = new String();           // 1

for (int i = 0; i < n; i++) {        //  $\sim n^2$ 
    bits += i & 1;                    //  $\sim i$ 
}

System.out.print(bits);              //  $\sim n$ 
```

Totalt: $\sim n^2$.

Anta att vi kan utföra $\sim 10^6$ "instruktioner"/s.

$n = 10^5 \Rightarrow \sim 3$ timmar

$n = 10^6 \Rightarrow \sim 10$ dagar

$n = 10^7 \Rightarrow \sim 3$ år

Analys

- ▶ Verkar onödigt att kopiera arrayen varje gång.
- ▶ När vi kopierar kan vi lägga till några extra “tomma” element i slutet av arrayen, så att vi inte behöver kopiera lika ofta.
- ▶ Låt oss göra arrayen dubbelt så stor.

Dubbelt så stor

```
private char[] string;
private int    length;

public void append(char c) {
    if (length == string.length) {
        char[] temp = new char[2 * string.length];

        for (int i = 0; i < string.length; i++) {
            temp[i] = string[i];
        }

        string = temp;
    }

    string[length++] = c;
}
```

Dubbelt så stor

```
String bits = new String();           // 1

for (int i = 0; i < n; i++) {        // n + ???
    bits += i & 1;
}

System.out.print(bits);              // ~ n
```

???

Anta att n är en jämn tvåpotens, låt $n = 2^k$:

$$\begin{aligned}\sum_{i=0}^{k-1} 2^i &= 1 + 2 + 4 + \dots + 2^{k-1} \\ &= 2^k - 1 \\ &\sim n\end{aligned}$$

Dubbelt så stor

```
String bits = new String();           // 1  
  
for (int i = 0; i < n; i++) {        // ~ n  
    bits += i & 1;  
}  
  
System.out.print(bits);              // ~ n
```

Totalt: $\sim n$.

Anta att vi kan utföra 10^6 "instruktioner"/s.

$n = 10^6 \Rightarrow \sim 1$ sekund

$n = 10^9 \Rightarrow \sim 20$ minuter

$n = 10^{12} \Rightarrow \sim 10$ dagar

Datastrukturer

- ▶ `String`: + kopierar varje gång.
Kan ej ändras: “immutable”.
- ▶ `StringBuffer`: append kopierar sällan.
Dynamisk array.

Blir programmet effektivt om vi lägger till 10 element istället för att göra arrayen dubbelt så stor?

- ▶ Fundera ett tag (ensam).
- ▶ Svara sedan på <http://pingo.upb.de/>.

Asymptotisk komplexitet

Kursen fokuserar på *asymptotisk* komplexitet:
vad händer när storleken går mot oändligheten?

$$\begin{array}{rcl} 8n + 3 & \Rightarrow & O(n) \\ 4n^2 + 3n + 6 & \Rightarrow & O(n^2) \end{array}$$

Ordo-notation (en variant)

$$T(n) = O(f(n))$$

om och endast om

det finns ett naturligt tal n_0

och ett reellt tal $c > 0$

så att $T(n) \leq cf(n)$ för alla $n \geq n_0$.

$$T(n) = \Omega(f(n))$$

om och endast om

det finns ett naturligt tal n_0

och ett reellt tal $c > 0$

så att $T(n) \geq cf(n)$ för alla $n \geq n_0$.



$$T(n) = \Theta(f(n))$$

om och endast om

$$T(n) = O(f(n)) \text{ och } T(n) = \Omega(f(n))$$

Exempel

Kan använda Θ för att uttrycka resultatet av tidigare analyser:

- ▶ Bits: $\Theta(n^2)$.
- ▶ FastBits: $\Theta(n)$.

Exempel

Kan använda Θ för att uttrycka resultatet av tidigare analyser:

- ▶ Bits: $\Theta(n^2)$.
- ▶ FastBits: $\Theta(n)$.

Vad är bäst: $\Theta(n)$ eller $\Theta(n^2)$?

Vilka påståenden är korrekta?

- ▶ $n^2 = O(n)$
- ▶ $n^2 = \Omega(n)$
- ▶ $n = O(n^2)$
- ▶ $n = \Omega(n^2)$
- ▶ $10n + 7 = \Theta(3n + 1)$
- ▶ $10n + 7 = \Theta(13n + 12)$

Svara på <http://pingo.upb.de/>.

Asymptotisk notation

Notera:

$$1 = O(n)$$

$$1 = O(n^2)$$

$$n = O(n \log n)$$

$$n = O(n^2)$$

Ordonotation: regler

Om $T(n)$ är ett polynom av grad k :

$$T(n) = O(n^k).$$

- ▶ $7n^2 + 3n + 2 = O(n^2)$.
- ▶ $0.1n^3 + 1000 = O(n^3)$.

Ordonotation: regler

Om $k > 0$ är en *konstant*:

$$\begin{aligned}(\log_2 n)^k &= O(n) \quad (\text{ej } \Theta(n)), \\ \log_2(n^k) &= k \log_2 n = O(\log n).\end{aligned}$$

För konstant $a > 1$:

$$\log_a n = \log_2 n / \log_2 a = O(\log_2 n) = O(\log n).$$

- ▶ $(\log_2 n)^{10000} = O(n)$.
- ▶ $\log_2(n^{10000}) = O(\log n)$.

Ordonotation: regler

Om $T(n) = O(f(n))$ och $U(n) = O(g(n))$:

$$\begin{aligned}T(n) + U(n) &= O(f(n) + g(n)) \\ &= O(\max(f(n), g(n))),\end{aligned}$$

$$T(n)U(n) = O(f(n)g(n)).$$

- ▶ $7n^2 + 3n^2 = O(n^2 + n^2) = O(n^2)$.
- ▶ $2n^3 + (\log_2 n)^{1000} = O(n^3 + n) = O(n^3)$.
- ▶ $2n^3 \log_7 5n^2 = O(n^3 \log n)$.

Vilka analyser är korrekta?

A

```
for (int i = 0; i < n; i++) {  
    for (int j = 0; j < n; j++) {  
        a[i][j] = i*j;  
    }  
}
```

$\Theta(n)$
 $\Theta(n^2)$

B

```
for (int i = 0; i < 7; i++) {  
    for (int j = 0; j < n; j++) {  
        a[i][j] = i*j;  
    }  
}
```

$\Theta(n)$
 $\Theta(n^2)$

Svara på <http://pingo.upb.de/>.

Tidskomplexitet

Hur analyserar man tidskomplexitet?

- ▶ *Mäta.*

Nackdelar: Kan vara tidskrävande,
inget bra stöd för design.

- ▶ *Räkna instruktioner.*

Nackdelar: Komplicerat.

- ▶ *Förenklad modell.*

Nackdelar: Inte tillämpligt i alla lägen.

Uniform kostnadsmodell

- ▶ Enkel dator.
- ▶ Varje instruktion tar en tidsenhet.
Bara enkla instruktioner,
men godtyckligt stora tal.
- ▶ Oändligt minne.

Uniform kostnadsmodell

- ▶ Enkel dator.
- ▶ Varje instruktion tar en tidsenhet.
Bara enkla instruktioner,
men godtyckligt stora tal.
- ▶ Oändligt minne.
- ▶ Inte realistisk.
- ▶ Fungerar ganska bra om man är försiktig.
- ▶ Används ofta i kursen.

Logaritmisk kostnadsmodell

- ▶ Enkel dator.
- ▶ Tid beräknas i termer av indatas storlek, räknat i antal bitar.
- ▶ Oändligt minne.
- ▶ Lite mer realistisk, lite krångligare.

Begränsningar

- ▶ I/O.
- ▶ Cacheminnen.

Sammanfattning

- ▶ Dynamiska arrayer.
- ▶ Asymptotisk komplexitet/notation.
- ▶ Modeller.