Eerste college complexiteit

4 februari 2008

Introductie

- docent: dr. J.M. (Jeannette) de Graaf
- werkgroepleider: Thijs van Ommen
- website: http://www.liacs.nl/home/graaf/COMP/
- college: dinsdag 11.15 – 13.00 in zaal 174
- werkcollege: maandag 11.15 – 13.00 in zaal 174

- eerste college: maandag 4 februari 2008
- tweede college: dinsdag 5 februari 2008
- eerste werkcollege: maandag 11 februari 2008
- tentamen: donderdag 5 juni, 14.00–17.00
- hertentamen: dinsdag 12 augustus, 10.00–13.00

Huiswerk:

- Er komen drie of vier huiswerkopgaven
- Deze dienen individueel ingeleverd te worden
- Samenwerken mag (met mate), overschrijven niet
- Voor elke huiswerkopgave krijg je een cijfer
- Het gemiddelde hiervan wordt opgeteld bij het aantal punten dat je voor het tentamen haalt
- Delen door 10 levert je eindcijfer

- Programmeren en Correctheid: correctheid van algoritmen
- Complexiteit:
  - (tijd)complexiteit: hoeveelheid werk
  - ruimtecomplexiteit: hoeveelheid geheugen
  - optimaliteit: kan het nog beter?

Opmerking: we zullen hier altijd sequentiële algoritmen bekijken.
Vraag: hoe meten we de hoeveelheid werk die een algoritme doet?

Complexiteit = tijdcomplexiteit = hoeveelheid werk

- een maat voor de hoeveelheid werk moet iets vertellen over de efficiëntie van de methode

- die maat moet onafhankelijk zijn van de gebruikte computer, programmeertaal, implementatiedetails etc.

- de complexiteit hangt gewoonlijk af van de grootte van de invoer: hoe groter de invoer, hoe hoger de complexiteit

Gegeven een array A ($A[1], A[2], \ldots, A[n]$, ongesorteerd) met $n \geq 1$ gehele getallen. **Gevraagd** het maximum van deze getallen.

Een algoritme dat het maximum vindt:

```
(1)   max := A[1];
(2)   index := 2;
(3)   while index ≤ n do
(4)       if max < A[index] then
(5)           max := A[index];
(6)       fi
(7)       index := index + 1;
(8)   od
(9)   return max;
```

(1)	$max := A[1];$	1 x
(2)	$index := 2;$	1 x
(3)	**while** $index \leq n$ **do**	n x
(4)	**if** $max < A[index]$ **then**	$n-1$ x
(5)	$max := A[index];$	$\leq n-1$ x
(6)	**fi**	
(7)	$index := index + 1;$	$n-1$ x
(8)	**od**	
(9)	**return** $max;$	1 x
		$(\leq)\ 4n$

aantal vergelijkingen $(max < A[index]) = n - 1 \approx n =$ aantal vergelijkingen $(index \leq n) \approx 4n$ $\Theta(n)$

Om de complexiteit van een algoritme te bepalen tellen we het aantal keer dat een geschikte basisoperatie wordt uitgevoerd.

- identificeer een operatie die fundamenteel is voor het algoritme (dus geen boekhoudoperaties zoals tellerophogingen)
- het totale aantal uitgevoerde operaties moet ruwweg evenredig zijn (in orde van grootte) aan het aantal basisoperaties, ofwel:
- alle andere operaties worden (in orde van grootte) hooguit even vaak uitgevoerd als de basisoperatie

Basisoperatie: voorbeelden

probleem	basisoperatie
X zoeken in een array	vergelijking van X met een array-element (en vergelijking tussen array-elementen onderling)
twee polynomen vermenigvuldigen	vermenigvuldiging van twee getallen (en/of optelling)
een array sorteren	vergelijking van twee array-entries
graafprobleem	bezoek aan een knoop en/of doorlopen van een tak

De complexiteit van een algoritme hangt af van de grootte van de invoer: $f(n)$

- bepaal een basisoperatie

- tel het aantal basisoperaties $\Longrightarrow f(n)$

- van belang is de orde van grootte van $f(n)$ voor grote $n \Longrightarrow O, \Theta, \Omega$

De complexiteit hangt af van de grootte van de invoer. Wat wordt bedoeld met invoergrootte?

probleem	grootte van de invoer
X zoeken in een array	aantal array-elementen
twee polynomen vermenigvuldigen	graad van de polynomen (=aantal coëfficiënten)
een array sorteren	aantal array-elementen
graafprobleem	aantal knopen en/of takken

De complexiteit van een algoritme hangt af van de soort invoer: worst case, average case, best case

- worst case complexiteit is $g(n)$: het algoritme doet voor elke mogelijke invoer hooguit $g(n)$ stappen
- best case complexiteit is $h(n)$: het algoritme doet voor elke mogelijke invoer minstens $h(n)$ stappen
- worst case geeft dus een bovengrens, best case een ondergrens
- average case complexiteit is de complexiteit gemiddeld over alle mogelijke invoeren
- verschillende algoritmen voor hetzelfde probleem: vergelijk complexiteiten $\Longrightarrow$ optimaliteit

- bestaat er een efficiënter algoritme voor het probleem?

- heeft te maken met de complexiteit van het probleem: soms kan het niet beter

- je bewijst complexiteit van een probleem altijd binnen een bepaalde klasse van algoritmen, bijvoorbeeld de klasse van algoritmen gebaseerd op het doen van arrayvergelijkingen

- bewijs stellingen die een ondergrens opleveren voor het aantal (basis)operaties dat nodig is om het probleem op te lossen, d.w.z.

- laat zien dat elk algoritme voor het probleem minstens ... elementaire (basis-) stappen moet doen in de ... case (meestal worst case)

- om een ondergrens te bewijzen kunnen we bijvoorbeeld een beslissingsboomargument of een adversary-argument gebruiken

- de worst case complexiteit van een algoritme dat het probleem oplost levert een bovengrens voor de complexiteit van het probleem: het probleem kan opgelost worden in hooguit ... stappen

Complexiteit van recursieve algoritmen:

- in het algemeen is hier het aantal recursieve aanroepen een goede maat voor de hoeveelheid werk

- een en ander leidt tot recurrente betrekkingen

Als voorbeeld bekijken we een recursief algoritme voor het bepalen van het maximum van n getallen, opgeslagen in een array A.

```
int grootste(int A[ ], n) {
    if n = 1 then
        return A[n];
    else
        max := grootste(A, n − 1);
        if A[n] > max then
            max := A[n];
        fi
    fi
    return max;
}
```

werk per aanroep ($n > 1$)

Laat $C(n)$ = aantal recursieve aanroepen op n elementen, dan geldt:

$$C(n) = \begin{cases} 1 & n = 1 \\ C(n-1) + 1 & n > 1 \end{cases}$$

Oplossing: $C(n) = n$

```
(1)   A[0] := 0; i := 1;
(2)   while i < n do
(3)       while i < n and A[i] < A[i + 1] do
(4)           i := i + 1;
(5)       od
(6)       if i < n then
(7)           wissel(A[i], A[i + 1]);
(8)           i := i − 1;
(9)       fi
(10)  od
```

Dit algoritme sorteert A (met $A[i] > 0$ voor $i = 1, \ldots, n$ en $n > 1$ en alle $A[i]$ verschillend) oplopend.

a. Leg uit waarom het vergelijken van array-elementen (regel 3, tweede test) een goede basisoperatie is.
b. Bekijk worst case en best case.