Tweede college Complexiteit

14 februari 2025

Restant college 1

Zoeken

- **(Tijd)complexiteit**: hoeveelheid werk
- Complexiteit van algoritmen
  - ▷ Basisoperatie
  - ▷ Worst case, average case, best case
  - ▷ O (grote-O), Θ en Ω
  - ▷ Optimaliteit
- Complexiteit van problemen
  - ▷ Ondergrenzen
- Complexiteitstheorie
  - ▷ Handelbaar ($\mathcal{P}$) versus onhandelbaar
  - ▷ $\mathcal{P}$, $\mathcal{NP}$, $\mathcal{NPC}$

Bestaat er een efficiënter algoritme voor het probleem?

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

- 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 (namelijk door dat algoritme)

- moeilijker is het bepalen van een (niet-triviale) ondergrens

- een simpele (meestal niet al te scherpe) ondergrens voor de complexiteit van een probleem kan soms verkregen worden door het aantal invoer/uitvoer-elementen te tellen

- een betere ondergrens op de complexiteit van een probleem vind je i.h.a. door te kijken naar de basisoperatie

- je bewijst zo (een ondergrens op) de complexiteit van een probleem binnen een bepaalde klasse van algoritmen; bijv. 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)

- technieken voor het bewijzen van ondergrenzen zijn o.a. beslissingsboomargumenten en adversary-argumenten

- soms is een ad-hoc argument voldoende, zoals het bewijs uit het ongerijmde voor het vinden van het maximum van n getallen

Gegeven een willekeurig array $A = A[1], A[2], \ldots, A[n]$ dat $n \geq 1$ gehele getallen bevat.
Gevraagd: het maximum van deze getallen.

Bewering: *elk* algoritme voor dit probleem –dat gebaseerd is op het doen van arrayvergelijkingen*– doet in de *worst case* ten minste $n - 1$ arrayvergelijkingen.

Bewijs: uit het ongerijmde. Zie college.

*dus vergelijkingen van de vorm $A[i] < A[j]$

Polynomiaal: (meestal) binnen redelijke tijd klaar $\rightarrow$ handelbaar

Exponentieel: zeker niet in acceptabele tijd klaar $\rightarrow$ onhandelbaar

n	10	20	50	100	300
n^2	$\frac{1}{10000}$ sec	$\frac{1}{2500}$ sec	$\frac{1}{400}$ sec	$\frac{1}{100}$ sec	$\frac{9}{100}$ sec
n^5	$\frac{1}{10}$ sec	3,2 sec	5,2 min	2,8 uur	28,1 dag
2^n	$\frac{1}{1000}$ sec	1 sec	35,7 jaar	400 biljoen eeuwen	75 cijfers eeuwen
n^n	2,8 uur	3,3 biljoen jaar	70 cijfers eeuwen	185 cijfers eeuwen	728 cijfers eeuwen

executietijd bij miljoen stappen per seconde

Merk op: de oerknal was ruim 10 miljard jaar geleden: dat is een getal van 11 cijfers.

We onderscheiden drie soorten problemen:

- ▷ handelbare problemen $\rightarrow$ in het algemeen (min of meer) efficiënt op te lossen (polynomiale bovengrens): $\mathcal{P}$
- ▷ onhandelbare problemen $\rightarrow$ in het algemeen niet efficiënt op te lossen (bijv. exponentiële ondergrens)
- ▷ onbeslisbare problemen $\rightarrow$ in het algemeen niet op te lossen: zie Computability

Complicatie: van veel bekende problemen weten we niet of ze wel of niet handelbaar zijn: $\mathcal{NPC}$. Hiervoor is geen polynomiaal algoritme bekend, maar ook geen niet-polynomiale ondergrens: $\mathcal{P} \stackrel{?}{=} \mathcal{NP}$

(https://www.claymath.org/millennium-problems)

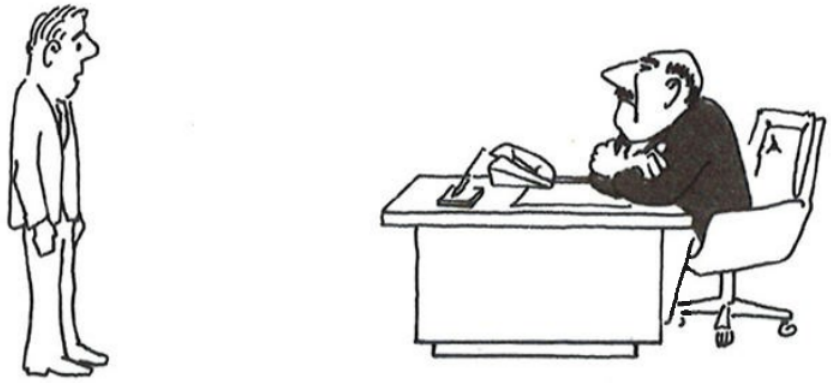
$\mathcal{NP}$ betekent NIET: niet-polynomiaal

WEL: niet-deterministisch polynomiaal

Intuïtief:

- $\mathcal{P}$: oplossing van het probleem in polynomiale tijd te *vinden*
- $\mathcal{NP}$: oplossing van het probleem in polynomiale tijd te *verifiëren/controleren*

$\mathcal{NPC}$: in zekere zin de moeilijkste problemen uit $\mathcal{NP}$.
Hiervoor is geen polynomiaal algoritme bekend, maar ook geen niet-polynomiale ondergrens.



"I can't find an efficient algorithm, I guess I'm just too dumb."

"I can't find an efficient algorithm, because no such algorithm is possible!"

"I can't find an efficient algorithm, but neither can all these famous people."

We bekijken een aantal zoekalgoritmen, waarvan we de complexiteit vergelijken. Met behulp van beslissingsbomen bewijzen we later* een ondergrens op de complexiteit.

- ▷ Zoeken met behulp van sleutelvergelijkingen
- ▷ Zoekalgoritmen:
  - ongeordend lineair zoeken (OLZ; opgave 3)
  - geordend lineair zoeken (GLZ; opgave 4)
  - jump search (JS)
  - binair zoeken (BZ; opgave 5)

*derde college

Probleem

Gegeven een waarde X en een array $A = A[1], \ldots, A[n]$. Bepaal of X in A zit. Zo ja, geef de index terug, zo nee, geef -1 terug.

We maken gebruik van sleutelvergelijkingen van de vorm:

```
if (X = A[i]) then          // binaire vergelijking
    gevonden;
else
    ...;
```

of

```
if (X = A[i]) then          // drie-weg vergelijking
    gevonden;
else
    if (X < A[i]) then
        ....;
    else
        ....;
```

- een sleutelvergelijking doet voor iedere aanroep maximaal twee ja/nee vergelijkingen
- dat kost dus $\Theta(1)$ tijd per aanroep
- de drie-weg vergelijking kost in orde van grootte evenveel tijd als de binaire vergelijking $X = A[i]$
- we tellen beide soorten sleutelvergelijkingen dan ook als één vergelijking

Probleem

Zoek X in een willekeurig array $A = A[1], \ldots, A[n]$.

Algoritme (zie opgave 3):

```
(1)  index := 1;
(2)  while index ≤ n and A[index] ≠ X do
(3)      index := index + 1;        ↑ basisoperatie
(4)  od
(5)  if index ≤ n then
(6)      return index;
(7)  else
(8)      return −1;
(9)  fi
```

Best case

Je kunt al bij de eerste vergelijking prijs hebben: $A[1] = X$.
1 sleutelvergelijking $\rightarrow \Theta(1)$

Worst case

Als X niet in A voorkomt of helemaal achteraan staat.
n sleutelvergelijkingen $\rightarrow \Theta(n)$

Average case

- Laat q de kans zijn dat X in A voorkomt
- Iedere positie in A is even waarschijnlijk
- Dan worden er in de average case $q \cdot \frac{1}{2}(n+1) + (1-q) \cdot n$ $\in \Theta(n)$ sleutelvergelijkingen gedaan

Complexiteit van het probleem (opgave 3e)
Elk algoritme dat een waarde X zoekt in een (willekeurig) array A met n elementen, en dat alleen gebruik maakt van sleutelvergelijkingen ($X =, < A[i]$), doet in de worst case ten minste n vergelijkingen*. Bewijs uit het ongerijmde, te vinden op de website.

Algoritme en optimaliteit
Ongeordend lineair zoeken voldoet aan de eisen van de stelling en doet in de worst case n vergelijkingen. Het is dus optimaal.

*De stelling gaat over algoritmen die werken op alle mogelijke invoerrijtjes (en te zoeken X). In het bewijs wordt expliciet gebruikt dat je geen informatie hebt over de invoer, behalve wat je uit de sleutelvergelijkingen leert. Het bewijs gaat niet op voor algoritmen die voor een speciaal soort invoer zijn gemaakt en daarvan gebruik maken.

Probleem

Zoek X in een oplopend gesorteerd array $A = A[1], \ldots, A[n]$.

Algoritme (zie opgave 4):

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

Best case: is nog steeds 1 sleutelvergelijking (dus $\Theta(1)$), maar voor meer invoerrijtjes dan OLZ: 1 vergelijking $\Leftrightarrow X \leq A[1]$.

Worst case: is nog steeds n sleutelvergelijkingen (dus $\Theta(n)$). Je kunt echter eerder stoppen t.o.v. OLZ (namelijk zodra $A[index] \geq X$). Dat geeft minder worst-case-gevallen dan OLZ: n vergelijkingen $\Leftrightarrow X > A[n-1]$.

Average case:

$$\frac{n}{2} + \frac{n}{n+1} + q \cdot (\frac{1}{2} - \frac{n}{n+1}) \in \Theta(\frac{n}{2}),$$

waarin q de kans is dat X in A voorkomt, en:

1. als X in A: alle n posities in A even waarschijnlijk
2. als X niet in A: alle $n+1$ gaten even waarschijnlijk

A is weer oplopend gesorteerd; kies k met $1 \leq k < n$

$index := k$; // $index$ is altijd een k-voud
// vergelijk X met $A[k], A[2k], A[3k], \ldots$
while $index \leq n$ **and** $A[index] < X$ **do**
    $index := index + k$;
od
if $index \leq n$
    // $A[index - k] < X \leq A[index]$
    lineair zoeken van X in $A[index - k + 1], \ldots, A$[index];
else
    // $A[index - k] < X \leq A[n]$ of $X > A[n]$
    lineair zoeken van X in $A[index - k + 1], \ldots, A[n]$;
fi

Worst case:

$$\lfloor \frac{n}{k} \rfloor + k \text{ sleutelvergelijkingen}$$

Beste keus: $k = \lceil \sqrt{n} \rceil$.
Dan doet jump search in het slechtste geval $\Theta(\sqrt{n})$ sleutelvergelijkingen. Dat is beter dan geordend lineair zoeken.

Vraag: kan zoeken in een geordend array nog beter?

Antwoord: Ja, namelijk binair zoeken.

$Links := 1;\ Rechts := n;$
while $Links \leq Rechts$ **do**
    $Midden := \lfloor \frac{Links + Rechts}{2} \rfloor;$
    if $X = A[Midden]$ **then**
        return $Midden;$ // gevonden
    else
        if $X < A[Midden]$ **then**
            $Rechts := Midden - 1;$ // linkerstuk
        else
            $Links := Midden + 1;$ // rechterstuk
        fi
    fi
od
return $-1;$

Zoals besproken tellen we de achtereenvolgende tests $X = A[Midden]$ en $X < A[Midden]$ als 1 (sleutel)vergelijking.

Worst case: $\lceil \lg(n+1) \rceil = \lfloor \lg n \rfloor + 1$ vergelijkingen

Zij $W(n)$ het aantal drie-weg-vergelijkingen van de vorm $X =, < A[Midden]$ dat het algoritme doet in het *slechtste* geval. Dan geldt:

- $W(n) = 1 + W(\lfloor n/2 \rfloor)$

- De oplossing van deze recurrente betrekking*, met $W(1) = 1$, wordt gegeven door: $W(n) = \lceil \lg(n+1) \rceil = \lfloor \lg n \rfloor + 1$. Het bewijs gaat met volledige inductie (opgave 5).

*We zullen later nog bestuderen hoe we dit soort recurrente betrekkingen kunnen aanpakken/oplossen.

Average case (voor $n = 2^k - 1$): gemiddeld aantal vergelijkingen nodig om X te vinden in A is

- $\frac{1}{n}\Sigma_{i=0}^{k-1}(i+1)2^i$ als X voorkomt in A*, met $k = \lg(n+1)$
- en dit is gelijk aan $\frac{1}{n}((k-1)2^k+1)$ (opgave 12)
- k als X niet aanwezig†
- in totaal dus $\frac{q}{n}\Sigma_{i=0}^{k-1}(i+1)2^i + (1-q)k \in \Theta(\lg(n+1))$, met q de kans dat X in A voorkomt.

*onder de aanname dat elke positie even waarschijnlijk is

†het kost altijd k vergelijkingen om dat te constateren

- Ongeordend lineair zoeken: $\Theta(n)$ sleutelvergelijkingen worst case, $\Theta(n)$ average case
- Geordend lineair zoeken: $\Theta(n)$ sleutelvergelijkingen worst case, $\Theta(\frac{n}{2})$ average case
- Jump search: $\Theta(\sqrt{n})$ sleutelvergelijkingen worst case
- Binary search: $\Theta(\lg n)$ sleutelvergelijkingen worst case, $\Theta(\lg n)$ average case

Zou het nog sneller kunnen dan dat?

Stelling. Elk algoritme* dat X opspoort in een array met n elementen, en dat uitsluitend gebaseerd is op het doen van sleutelvergelijkingen†, doet ten minste $\lfloor \lg n \rfloor + 1 = \lceil \lg(n+1) \rceil$ vergelijkingen in de worst case.

Bewijs met behulp van een beslissingsboomargument.
$\longrightarrow$ volgende week

Gevolg. Binair zoeken is optimaal voor wat betreft de worst case: er bestaat geen zoekalgoritme gebaseerd op sleutelvergelijkingen dat in het slechtste geval minder vergelijkingen doet, dus minder dan $\lfloor \lg n \rfloor + 1$.

*ook als dat speciaal op reeds gesorteerde arrays is toegespitst

†van de vorm $X =, < A[i]$

- **Volgende college:** vrijdag 21 februari, 11.00 – 12.45,
  Gorlaeusgebouw zaal BW.0.40

- **Straks werkcollege:** vrijdag 14 februari, 13.15 – 15.00,
  Gorlaeusgebouw zalen BW.0.29 en BW.0.30
  Opgaven: 3e, 8, 10, 17, 20

- **Huiswerk:**
  eerste huiswerkopgave: waarschijnlijk volgende week

- **Website:**
  liacs.leidenuniv.nl/~graafjmde/COMP/