Complexiteit 2023 — college 2

14 februari 2023

Zoeken

Vorige keer

- basisoperaties
- ↳ complexiteit van algoritmen
  - O, Θ en Ω
  - best, worst en average case
- complexiteit van problemen

Vandaag

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 (opgave 3)
  - geordend lineair zoeken (opgave 4)
  - jump search
  - binair zoeken (opgave 5)

Zoeken

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.

Basisoperatie: sleutelvergelijkingen van de vorm

```
1 if X = A[i] then                    /* binaire vergelijking */
2 |   gevonden;
3 else
4 |   ...;
```

```
1 if X = A[i] then                   /* drie-weg vergelijking */
2 |   gevonden;
3 else if X < A[i] then
4 |   ...;
5 else
6 |   ...;
```

Sleutelvergelijking

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

Ongeordend lineair zoeken (OLZ)

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    // basisoperatie
3 |   index := index + 1;
4 od
5 if index ≤ n then
6 |   return index;
7 else
8 |   return −1;
9 fi
```

OLZ: complexiteit

Beste case

- je kunt al bij de eerste vergelijking prijs hebben: $1 \in \Theta(1)$

Worst case

- als X niet voorkomt in A of helemaal achteraan: $n \in \Theta(n)$

Average case

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

OLZ: optimaliteit

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 leert uit de sleutelvergelijkingen. Het bewijs gaat niet op voor algoritmen die zijn gemaakt voor een speciaal soort invoer en daarvan gebruik maken.

Geordend lineair zoeken (GLZ)

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    // basisoperatie
3 |   index := index + 1;
4 od
5 if index ≤ n and X = A[index] then
6 |   return index;
7 else
8 |   return −1;
9 fi
```

GLZ: complexiteit

Best case is nog steeds $1 \in \Theta(1)$

Worst case is nog steeds $n \in \Theta(n)$

- wel kan je vaak eerder stoppen t.o.v. ongeordend zoeken: alleen n vergelijkingen als $X > A[n-1]$

Average case

- laat q de kans zijn dat X voorkomt in A
- als $X \in A$: alle n posities even waarschijnlijk
- als $X \notin A$: alle $n+1$ gaten even waarschijnlijk

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

Jump search (JS)

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

```
1 index := k;
  // index is altijd een k-voud
  // vergelijk X met A[k], A[2k], A[3k], ...
2 while index ≤ n and A[index] < X do
3 |   index := index + k;
4 od
5 if index ≤ n then
  |   // A[index − k] < X ≤ A[index]
6 |   lineair zoeken van X in A[index − k + 1], ..., A[index];
7 else
  |   // A[index − k] < X ≤ A[n] of X > A[n]
8 |   lineair zoeken van X in A[index − k + 1], ..., A[n];
9 fi
```

Vergelijk: zoeken in een woordenboek

JS: complexiteit

Worst case

- $\lfloor \frac{n}{k} \rfloor + k$ 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.

Binair zoeken (BZ)

```
1  Links := 1; Rechts := n;
2  while Links ≤ Rechts do
3  |   Midden := ⌊(Links+Rechts)/2⌋;
4  |   if X = A[Midden] then
5  |   |   return Midden ;                          // gevonden
6  |   else if X < A[Midden] then
7  |   |   Rechts := Midden − 1 ;    // verder in linkerstuk
8  |   else
9  |   |   Links := Midden + 1 ;    // verder in rechterstuk
10 |   fi
11 od
12 return −1;
```

BZ: worst case

Zoals besproken tellen we de achtereenvolgende tests $X = A[Midden]$ en $X < A[Midden]$ als één sleutelvergelijking.

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) en is te vinden op de website.

BZ: average case

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

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

*onder de aanname dat elke positie even waarschijnlijk is

†het kost altijd k vergelijkingen om dat te constateren

Zoeken: samengevat

- 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
- Binair zoeken: $\Theta(\lg n)$ sleutelvergelijkingen worst case, $\Theta(\lg n)$ average case

Zou het nog sneller kunnen dan dat?

Optimaliteit

Stelling. Elk algoritme* dat X opspoort in een array met n elementen, en dat uitsluitend is gebaseerd 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.

Gevolg. Binair zoeken is optimaal wat betreft de worst case.

---

*ook als dat speciaal is toegespitst op reeds gesorteerde arrays

†van de vorm $X =, < A[i]$

Om te onthouden van vandaag

- ongeordend lineair zoeken: $\Theta(n)$ (worst case), optimaal (voor sleutelvergelijkingen en willekeurige arrays)
- geordend lineair zoeken: $\Theta(n)$ (worst case)
- jump search: $\Theta(\sqrt{n})$ (worst case)
- binair zoeken: $\Theta(\lg n)$ (worst case), optimaal (voor sleutelvergelijkingen)

(Werk)college

Volgende college: dinsdag 21 februari, 9u00–10u45, zaal C1 (Gorlaeus)

Werkcollege: zodadelijk van 11u00 tot 12u45, computerzalen 302–304, 306–308 en 303 (Snellius)
Opgaven uit het dictaat: 6, 17, 8, 20, 19

De website heeft uitwerkingen van o.a. opgave 6. Gebruik die kennis verstandig.

Huiswerk

- $\lg n$ in pgfplots: `log2(x)`
- deadline (vóór hoorcollege 4, 28 februari 9u00) is strikt (daarna nog wel feedback, maar geen cijfer)