Derde college complexiteit

19 februari 2019

Zoeken

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)

Probleem

Gegeven een waarde X en een array A, bestaande uit n elementen $A[1], \ldots, A[n]$. 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
- Kost dus $\Theta(1)$ tijd per aanroep
- De drie-weg vergelijking kost in orde van grootte evenveel tijd als de vergelijking ($X = A[i]$)
- We tellen beide soorten sleutelvergelijkingen dan ook als één vergelijking

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

Complexiteit van het probleem (opgave 3.e.)
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*.

Algoritme en complexiteit
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 op alle mogelijke invoerrijtjes (en te zoeken X) werken. 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.

Algoritme (zie opgave 3):

(1) $index := 1$;
(2) **while** $index \leq n$ **and** $A[index] \neq X$ **do**
(3) $index := index + 1$; ↑ basisoperatie
(4) **od**
(5) **if** $index \leq n$ **then**
(6) **return** $index$;
(7) **else**
(8) **return** -1;
(9) **fi**

Best case

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

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 $q * \frac{1}{2} * (n+1) + (1-q) * n \in \Theta(n)$ sleutelvergelijkingen gedaan

Worst case

- Als X niet in A zit of helemaal achteraan: $\Theta(n)$

Probleem

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

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
```

De complexiteit van geordend lineair zoeken*.

Worst case:

n sleutelvergelijkingen

(Bij welke invoer gebeurt dit? En bij ongeordend zoeken?)

Average case:

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

met q de kans 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

*zie ook werkcollege, opgave 4

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 $- k$] $< X \leq A[n]$
    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 ≤ Rechts do
    Midden := ⌊(Links + Rechts)/2⌋;
    if X = A[Midden] then
        return Midden;
    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 3-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).

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 zeker in A zit*
- 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 (en 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.

Gevolg. Binair zoeken is optimaal voor wat betreft de worst case: er bestaat geen algoritme 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:
  dinsdag 26 februari, 11.00 – 12.45, zaal 174

- Eerste werkcollege:
  dinsdag 19 februari, 13.30 – 15.15, zaal 174
  Opgaven 1, 2, 4, 6, 7

- www.liacs.leidenuniv.nl/~graafjmde/COMP/