Derde college complexiteit

12 februari 2008

Zoeken en beslissingsbomen

Probleem
Zoek X in een (willekeurig) array A, bestaande uit n elementen $A[1], \ldots, A[n]$.

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
Lineair zoeken (zie opgave 3) doet in de worst case n vergelijkingen, en is dus optimaal.
Overigens worden gemiddeld (average case) $q * \frac{1}{2} * (n+1) + (1-q) * n$ sleutelvergelijkingen gedaan, met q de kans dat X in A voorkomt.

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 ($A[index] < X$)

Average case:

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

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 opgave 4

Neem weer aan dat A oplopend gesorteerd is en 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 sort in het slechtste geval $\Theta(\sqrt{n})$ sleutelvergelijkingen. Dat is beter dan 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;
```

We tellen de achtereenvolgende tests $X = A$[Midden] en $X < A$[Midden] als 1 vergelijking.

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

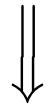
Average case (voor $n = 2^k - 1$): gemiddeld aantal vergelijkingen nodig om X te vinden in A is

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

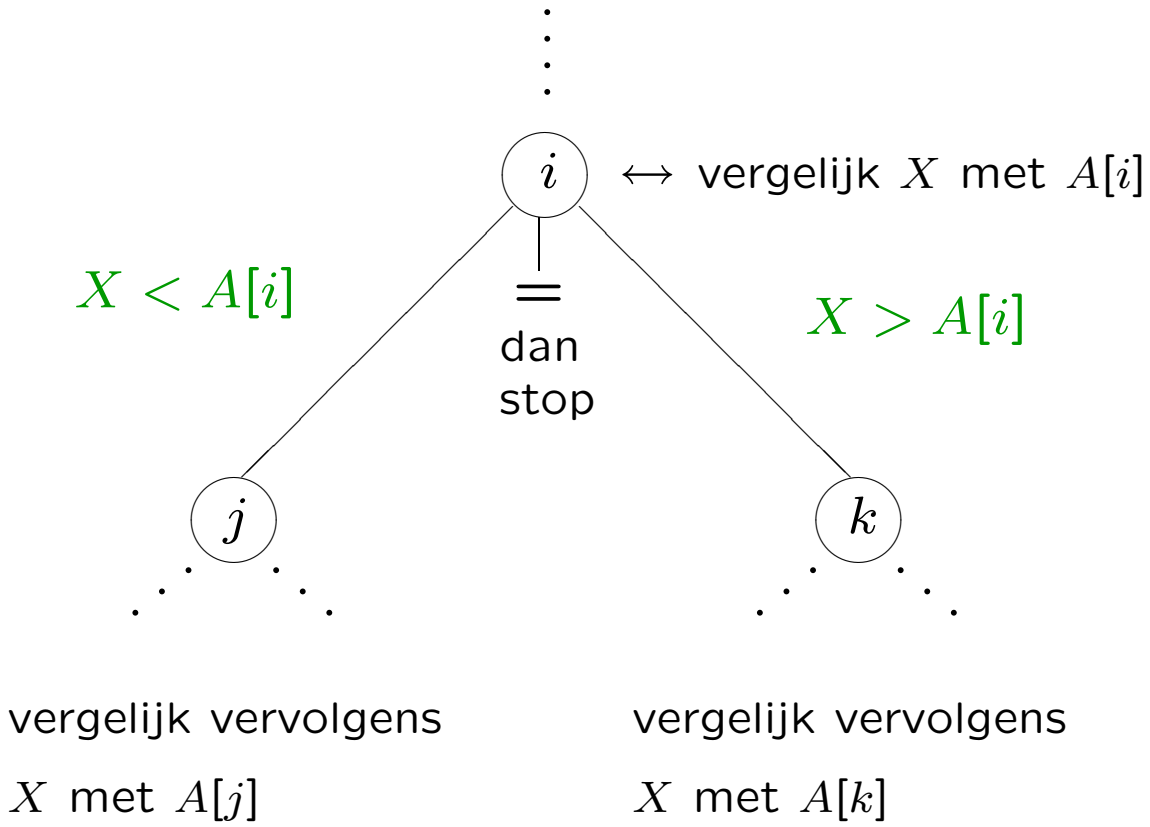
*onder de aanname dat elke positie even waarschijnlijk is

†het kost altijd k vergelijkingen om dat te constateren

algoritme gebaseerd op het doen van sleutelvergelijkingen $X =, < A[i]$



beslissingsboom: binaire boom waarin knopen corresponderen met sleutelvergelijkingen; een pad vanaf de wortel naar een willekeurige knoop correspondeert met een executie van het algoritme



Beslissingsboom voor algoritmen gebaseerd op sleutelvergelijkingen

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 $\lceil \lg(n+1) \rceil = \lfloor \lg n \rfloor + 1$ vergelijkingen in de worst case.

Gevolg

Binair zoeken is optimaal (voor wat betreft de worst case)

*van de vorm $X =, < A[i]$