Derde college complexiteit

21 februari 2025

Beslissingsbomen, selectie

- 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: $\lfloor \lg n \rfloor + 1 \in \Theta(\lg n)$ sleutelvergelijkingen worst case, $\Theta(\lg n)$ average case

Zou het nog sneller kunnen dan dat?

*namelijk $\frac{n}{k} + k$ met $k = \lceil \sqrt{n} \rceil$ en dat geeft ruwweg $2\sqrt{n}$

Nog even: BZ average case

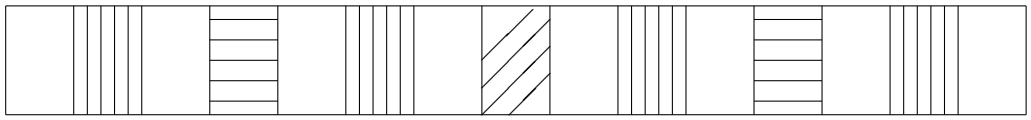
Average case (voor $n = 2^k - 1$): gemiddeld aantal vergelijkingen nodig om X te vinden in A is

- $\frac{1}{n}\Sigma_{i=0}^{k-1}(i+1)2^i$ 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

$n = 2^k - 1$: bekijk voor elk array-element hoeveel vergelijkingen het kost om dit te vinden.



4 3 4 2 4 3 4 1 4 3 4 2 4 3 4

$n = 2^4 - 1$

# elementen	# vergelijkingen	grootte zoekgedeelte
1	1	$2^k - 1$
2	2	$2^{k-1} - 1$
2^2	3	$2^{k-2} - 1$
...	...	...
2^{k-1}	k	$2^{k-(k-1)} - 1 = 1$

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 $\lfloor \lg n \rfloor + 1 = \lceil \lg(n+1) \rceil \in \Theta(\lg n)$ 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 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 $X =, < A[i]$

- Technieken om ondergrenzen te bewijzen
  - ▷ beslissingsboomargument (vandaag)
  - ▷ adversary-argument (volgende week)
- Selectieprobleem, toernooimethode

De volgende stelling geeft een verband tussen de hoogte van een binaire boom en het aantal knopen (resp. bladeren). We kiezen de definitie van niveau zo, dat de wortel op niveau 0 zit. De hoogte van de boom (= het hoogste niveau dat in de boom voorkomt) is dan gelijk aan het aantal niveaus - 1.

Stelling

Gegeven een binaire boom met n knopen (en b bladeren) en hoogte h. Dan geldt:

1. $h \geq \lceil \lg b \rceil$

2. $h \geq \lceil \lg(n+1) \rceil - 1 = \lfloor \lg n \rfloor$

Algoritme $\rightarrow$ boom: type 1

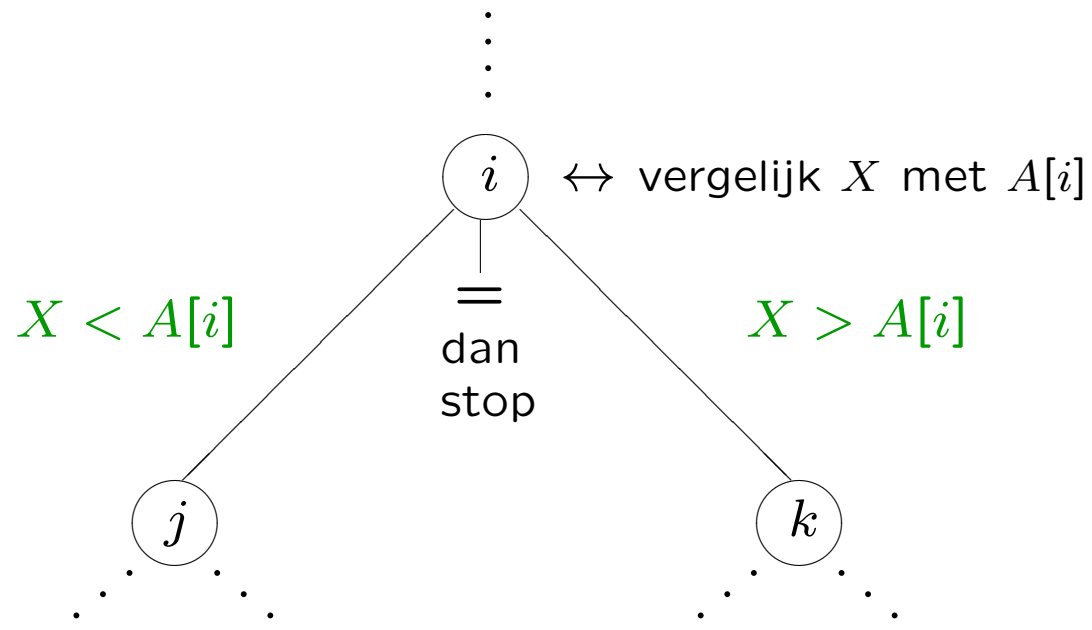
algoritme gebaseerd op het doen van sleutelvergelijkingen $X =, < A[i]$

$\Downarrow$

beslissingsboom:

- binaire boom waarin knopen corresponderen met sleutelvergelijkingen
- een pad vanaf de wortel naar een willekeurige knoop correspondeert met een executie van het algoritme, d.w.z. de achtereenvolgende vergelijkingen van het algoritme op zekere invoer
- het aantal knopen op zo'n pad is het aantal sleutel-vergelijkingen dat het algoritme doet op de betreffende invoer

Beslissingsboom: type 1



Beslissingsboom voor algoritmen gebaseerd op **sleutelvergelijkingen:** beschrijft de werking op elke mogelijke invoer

- ▷ wortel van de boom op niveau 0
- ▷ $h =$ hoogte (hoogste niveau dat voorkomt)
- ▷ knopen: sleutelvergelijkingen
- ▷ $N =$ aantal knopen
- ▷ $b =$ aantal bladeren (nu nog niet relevant)
- ▷ in een binaire boom: $h \geq \lceil \lg(N+1) \rceil - 1 = \lfloor \lg N \rfloor$ (**)

Wat stelt h voor en wat weten we van N bij zoekalgoritmen gebaseerd op sleutelvergelijkingen (corresponderend met deze beslissingsbomen)?

Bewijs ondergrens zoeken

- Het zoekalgoritme werkt voor elke mogelijke invoer, dus in het bijzonder voor het geval dat A allemaal verschillende waarden bevat.
- Merk op dat elke $A[i]$ door het algoritme gevonden moet kunnen worden; X kan immers op elke positie in het array voorkomen. Dat betekent dat er voor elke waarde $A[i]$ (en dus voor elke index i) ten minste één corresponderende knoop moet zijn, dus dat we ten minste n verschillende knopen in de beslissingsboom moeten hebben. Derhalve is $N \geq n$.
- Volgens ongelijkheid (**) volgt daaruit dat $h \geq \lfloor \lg n \rfloor$.
- Omdat het aantal vergelijkingen in de worst case gelijk is aan $h + 1$, geldt dat we (in de worst case) gegarandeerd ten minste $\lfloor \lg n \rfloor + 1$ sleutelvergelijkingen moeten doen.

algoritme gebaseerd op het doen van arrayvergelijkingen $A[i] < A[j]^*$

$\Downarrow$

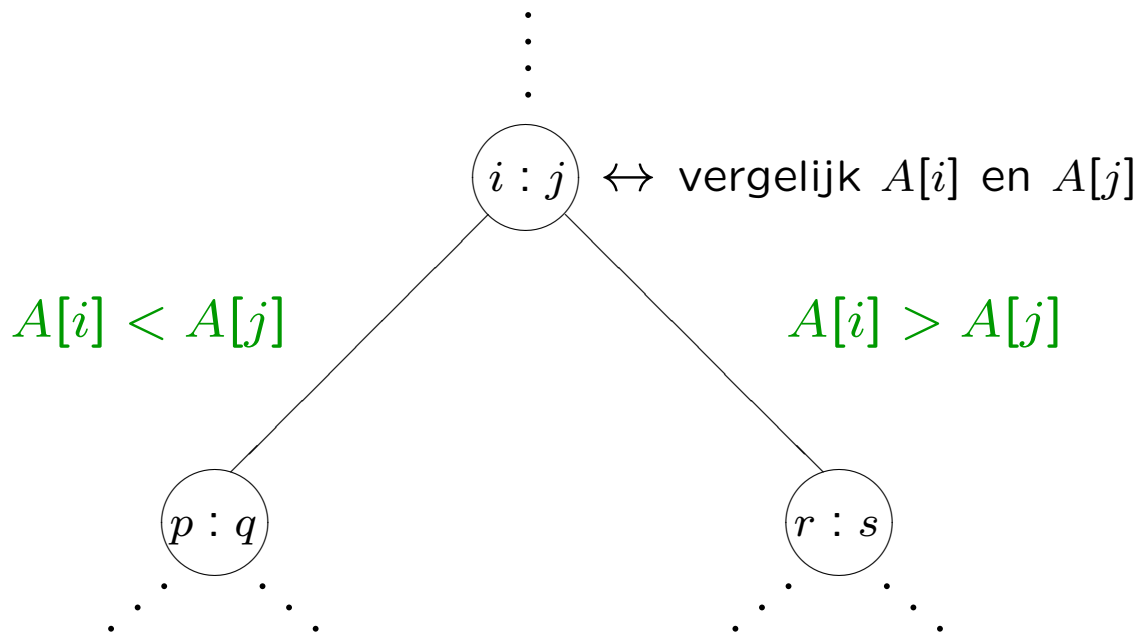
beslissingsboom:

- binaire boom waarin de interne knopen corresponderen met arrayvergelijkingen en de bladeren/externe knopen met het eindresultaat

- een pad vanaf de wortel naar een blad correspondeert met een executie van het algoritme, dus de achtereenvolgende vergelijkingen die het algoritme doet voor zekere invoer

*we mogen z.b.d.a. aannemen dat alle $A[i]$ verschillend zijn

Beslissingsboom: type 2



vergelijk vervolgens $A[p]$ en $A[q]$

vergelijk vervolgens $A[r]$ en $A[s]$

Beslissingsboom voor algoritmen gebaseerd op **arrayvergelijkingen:** beschrijft de werking op alle mogelijke invoer

- ▷ wortel van de boom op niveau 0
- ▷ $h =$ hoogte (hoogste niveau dat voorkomt*)
- ▷ interne knopen: arrayvergelijkingen
- ▷ bladeren: eindresultaten; algoritme stopt hier
- ▷ $b =$ aantal bladeren
- ▷ in een binaire boom: $h \geq \lceil \lg b \rceil$
- ▷ aantal vergelijkingen in de worst case $= h$

Wat weten we van b bij algoritmen gebaseerd op arrayvergelijkingen (corresponderend met deze beslissingsbomen)?

*inclusief de speciale bladeren (externe knopen)

Stelling

Elk algoritme dat de (index van de) grootste (of de kleinste) waarde bepaalt uit een array met n elementen, en dat uitsluitend is gebaseerd op het doen van arrayvergelijkingen, doet ten minste $\lceil \lg n \rceil$ vergelijkingen in de worst case.

Bewijs met behulp van een beslissingsboomargument.

Merk op: we hadden voor het opsporen van het maximum (of het minimum) al een scherpere ondergrens gevonden, namelijk $n - 1$.

Beslissingsboom type 2

- voor algoritmen gebaseerd op arrayvergelijkingen
- $h =$ aantal vergelijkingen in de worst case
- $b =$ aantal bladeren
- de bladeren bevatten de eindresultaten/eindantwoorden van het algoritme voor alle mogelijke invoeren
- in een binaire boom: $h \geq \lceil \lg b \rceil$
- $b \geq$ aantal eindantwoorden dat kan voorkomen !

Stelling (zie opgave 27a.)
Elk algoritme dat de (indices van de) grootste en de kleinste waarde bepaalt uit een array met n elementen, en dat uitsluitend gebaseerd is op het doen van arrayvergelijkingen, doet ten minste $\lceil 2 \lg(n-1) \rceil$ vergelijkingen in de worst case.

Bewijs met behulp van een beslissingsboomargument.

Opmerking
Met behulp van een adversary-argument kunnen we een scherpere ondergrens vinden, namelijk $\lceil \frac{3n}{2} \rceil - 2$.

Gegeven twee oplopend gesorteerde even lange rijen A en B met in totaal $2n$ verschillende getallen. Gevraagd wordt het n-de getal (in volgorde *van klein naar groot*) van de in totaal $2n$ elementen van A en B.

b. Bewijs met behulp van een beslissingsboomargument dat *elk* algoritme dat dit probleem voor het gegeven soort rijtjes oplost m.b.v. arrayvergelijkingen ten minste $\lceil \lg n \rceil + 1$ vergelijkingen moet doen in de worst case.

c. We zoeken nu de n-de waarde in grootte zoals boven, maar de ene gesorteerde rij bevat $\frac{n}{2}$ elementen en de andere $\frac{3n}{2}$ (n is hier even). Net als in **b.** kunnen we een ondergrens voor de worst case bewijzen voor het probleem met dit soort invoerrijtjes. Wat verandert er dan in het bewijs bij **b.** en welke ondergrens levert dat op?

Probleem

Gegeven een array $A = A[1], A[2], \ldots, A[n]$ met n verschillende getallen. Gegeven is verder een geheel getal k met $1 \leq k \leq n$. Gevraagd de $A[i]$ die groter is dan precies $k-1$ andere $A[j]$'s. M.a.w.: we zoeken de k-de in grootte (in volgorde van klein naar groot).

Klasse van algoritmen

We bekijken algoritmen die uitsluitend gebaseerd zijn op het doen van arrayvergelijkingen.

De **complexiteit** van het **probleem**:

Ondergrens

Elk algoritme gebaseerd op arrayvergelijkingen doet voor het selectieprobleem in de worst case altijd ten minste $\lceil \lg n \rceil$ vergelijkingen (beslissingsboomargument).

Bovengrens

Het probleem kan worden opgelost door het array eerst oplopend te sorteren. De k-de in grootte is dan $A[k]$.
Sorteren kan met $\Theta(n \lg n)$ vergelijkingen (zie later), dus selectie is $O(n \lg n)$.

Opmerkingen

(1) Zowel de bovengrens als de ondergrens kan scherper. (2) We bekijken hierna steeds (het precieze aantal vergelijkingen in) de worst case.

1. $k = n$: het maximum, of
   $k = 1$: het minimum.

   Dit kan met $n - 1$ vergelijkingen. Dat is optimaal (al gezien)!

2. (Variant) Het maximum en minimum beide opsporen.

   Voor de hand ligt een algoritme met $2n - 3$ vergelijkingen, maar het kan met $\lceil \frac{3n}{2} \rceil - 2$. Dit is optimaal (adversary-argument, zie later).

   Voor deze variant geldt natuurlijk $n \geq 2$.

Een optimaal algoritme:

```
if A[1] > A[2] then                          // n ≥ 2
    grootste := A[1]; kleinste := A[2];
else
    grootste := A[2]; kleinste := A[1];
i := 3;                                      // voor het gemak: n even;
while i < n do                               // anders kleine aanpassing
    if A[i] > A[i + 1] then                                        *
        gr := A[i]; kl := A[i + 1];
    else
        gr := A[i + 1]; kl := A[i];
    fi
    if gr > grootste then                                          *
        grootste := gr;
    fi
    if kl < kleinste then                                          *
        kleinste := kl;
    fi
    i := i + 2;
od
```

3. $k = n - 1$: de op een na grootste (dus $n \geq 2$).

   Voor de hand ligt een algoritme met $2n - 3$ vergelijkingen, maar het kan met $n + \lceil \lg n \rceil - 2$ vergelijkingen (toernooi-methode). Dit is optimaal.

   Bewijzen we hier niet, maar kan met een adversary-argument.

4. $k = \lceil \frac{n}{2} \rceil$: de mediaan (= de middelste in grootte).

   Er zit een gat tussen de best bekende ondergrens (ongeveer $2n$) en het best bekende algoritme. We kunnen dus niets over de optimaliteit van dat algoritme zeggen.

   Zie ook opgave 23, 24

De toernooimethode is een *optimaal* algoritme voor het vinden van de op een na grootste.

Terminologie:

wedstrijd $\longleftrightarrow$ arrayvergelijking;
winnaar $\longleftrightarrow$ grootste van de twee;
speler $\longleftrightarrow$ array-element; etcetera.

Complexiteit:

De toernooimethode doet $n + \lceil \lg n \rceil - 2$ arrayvergelijkingen. Dit is optimaal (worst case).

Geschikte implementatie:

Met behulp van een heap-achtige structuur die de "uitslagen" van het toornooi weergeeft (opgave 29).

Algoritme (voor het gemak met $n = 2^\ell$):

- Laat de spelers twee aan twee tegen elkaar spelen ($\frac{n}{2}$ wedstrijden).
- Laat de $\frac{n}{2}$ winnaars weer twee aan twee tegen elkaar spelen; de $\frac{n}{4}$ winnaars daarvan weer, etcetera.
- Herhaal dit totdat je één speler overhoudt: dit is de eindwinnaar, dus de grootste van allemaal.
- Er zijn nu $n-1$ wedstrijden gespeeld, en er waren ℓ rondes nodig ($\ell = \lg n$).

Algoritme (vervolg):

- Nu moet de op een na grootste nog gevonden worden.
- Dit moet een van de ℓ spelers zijn die in het toernooi van de grootste verloren heeft, en wel de grootste van die ℓ.
- Het kost nog $\ell - 1$ vergelijkingen om deze te bepalen.

Als $n = 2^\ell$ doet de toernooimethode dus $n + \lg n - 2$ arrayvergelijkingen in totaal. Dit is optimaal (worst case).

Opmerking: als n geen tweemacht is, is een kleine aanpassing nodig. Het aantal vergelijkingen wordt daarmee: $n + \lceil \lg n \rceil - 2$.

Ondergrens complexiteit

- Doel: bewijzen van een ondergrens op de worst case complexiteit van een probleem
- Dus: bewijzen dat elk algoritme voor het probleem ten minste $\cdots$ stappen nodig heeft in de worst case
- Dat kan met behulp van beslissingsbomen, maar het kan ook anders
- Het is voldoende om voor elk algoritme een "bad case" invoer te geven (of te weten dat er een bestaat !) waarop dat algoritme minstens $\cdots$ stappen doet
- Een manier om dit voor elkaar te krijgen is met behulp van een adversary argument

- **Volgende college:** vrijdag 28 februari, 11.00 – 12.45,
  Gorlaeusgebouw zaal BW.0.40

- **Straks werkcollege:** vrijdag 21 februari, 13.15 – 15.00,
  Gorlaeusgebouw zalen BW.0.29 en BW.0.30
  Opgaven: 20, 19, 22, 27, 23abc, 24

- **Huiswerk:**
  dit weekend, op de website

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