Complexiteit 2023 — college 3

21 februari 2023

Ondergrenzen

Vorige keer

- 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)

Vandaag

Technieken om ondergrenzen aan te tonen:

- beslissingsbomen
- adversary-argument

En:

- toernooimethode

Optimaliteit binair zoeken

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 $X =, < A[i]$

Binaire bomen

De volgende stelling geeft een verband aan 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 (d.w.z. het hoogste niveau dat voorkomt in de boom) is dan gelijk aan het aantal niveaus $-\,1$.

Stelling

Gegeven een binaire boom met n knopen, b bladeren en hoogte h. Er geldt:

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

Algoritme → boom: type 1

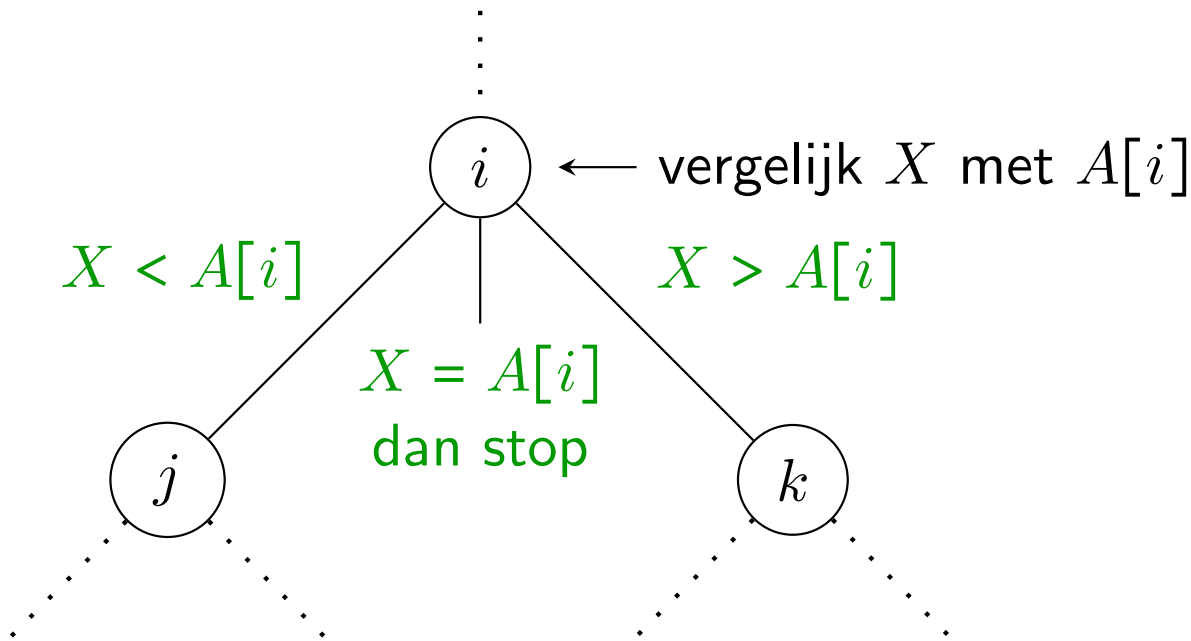
algoritme gebaseerd op het doen van sleutelvergelijkingen
$X =, < A[i]$

$$\Downarrow$$

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 sleutelvergelijkingen dat het algoritme doet op de betreffende invoer

Beslissingsboom type 1



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

Beslissingsboom type 1

- 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)?

Beslissingsbomargument

Bewijs stelling ondergrens zoeken

- Het zoekalgoritme werkt voor elke mogelijke invoer, dus in het bijzonder voor het geval dat A allemaal verschillende waarden bevat.
- Merk nu op dat elke $A[i]$ door het algoritme moet kunnen worden gevonden; 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 moeten hebben in de beslissingsboom. Derhalve is $N \geq n$.
- Daaruit volgt (zie eerdere slides) 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 $\lfloor \lg n \rfloor + 1 \in \Theta(\lg n)$ sleutelvergelijkingen moeten doen.

Algoritme → boom: type 2

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

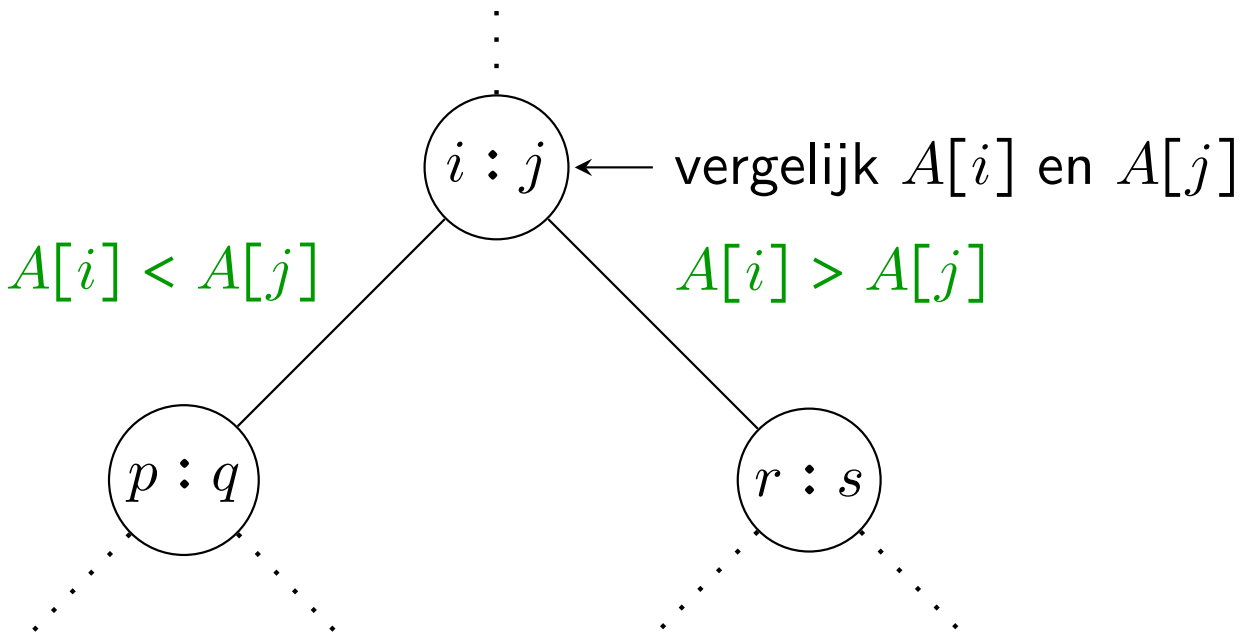
$$\Downarrow$$

beslissingsboom:

- binaire boom waarin de interne knopen corresponderen met arrayvergelijkingen en de bladeren 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.v.v.a. aannemen dat alle $A[i]$ verschillend zijn

Beslissingsboom type 2



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

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

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

Beslissingsboom type 2

- 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 is h

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

*inclusief de speciale bladeren

Maximum

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.

Merk op

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

Maximum en minimum

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.

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

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!

Opgave 28

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 bewijzen voor de worst case voor het probleem met dit soort invoerrijtjes. Wat verandert er dan in het bewijs bij **b.** en welke ondergrens levert dat op?

Selectieprobleem

Probleem

Gegeven een array $A = A[1], \ldots, A[n]$ met n verschillende getallen. Gegeven 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 zijn gebaseerd op het doen van arrayvergelijkingen.

Selectie: complexiteit

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). Selectie is dus $\Omega(\lg n)$.

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

Beide grenzen kunnen scherper. We bekijken hierna steeds (het precieze aantal vergelijkingen in) de worst case.

Speciale gevallen 1

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

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

2. (Variant, met $n \geq 2$) Het maximum en minimum beide opsporen.

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

Maximum en minimum: een optimaal algoritme

```
1  if A[1] > A[2] then                                   // n ≥ 2
2  |   grootste := A[1]; kleinste := A[2];
3  else
4  |   grootste := A[2]; kleinste := A[1];
5  i := 3;                                  // voor het gemak: n even
6  while i < n do                          // anders kleine aanpassing
7  |   if A[i] > A[i + 1] then
8  |   |   gr := A[i]; kl := A[i + 1];
9  |   else
10 |   |   gr := A[i + 1]; kl := A[i];
11 |   if gr > grootste then
12 |   |   grootste := gr;
13 |   if kl < kleinste then
14 |   |   kleinste := kl;
15 |   i := i + 2;
```

Speciale gevallen 2

3. $k = n - 1$: de op één 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 (toernooimethode). Dit is optimaal. (Hier onbewezen, 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 zeggen over de optimaliteit van dat algoritme. Zie ook opgave 23.

Toernooimethode

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

Terminologie:

wedstrijd ↔ arrayvergelijking;

winnaar ↔ grootste van de twee;

speler ↔ 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 toernooi weergeeft (opgave 29).

---

*Jósef Schreier, 1932

†S. S. Kislitsin, 1964

Toernooimethode

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 ($\ell = \lg n$).

Toernooimethode

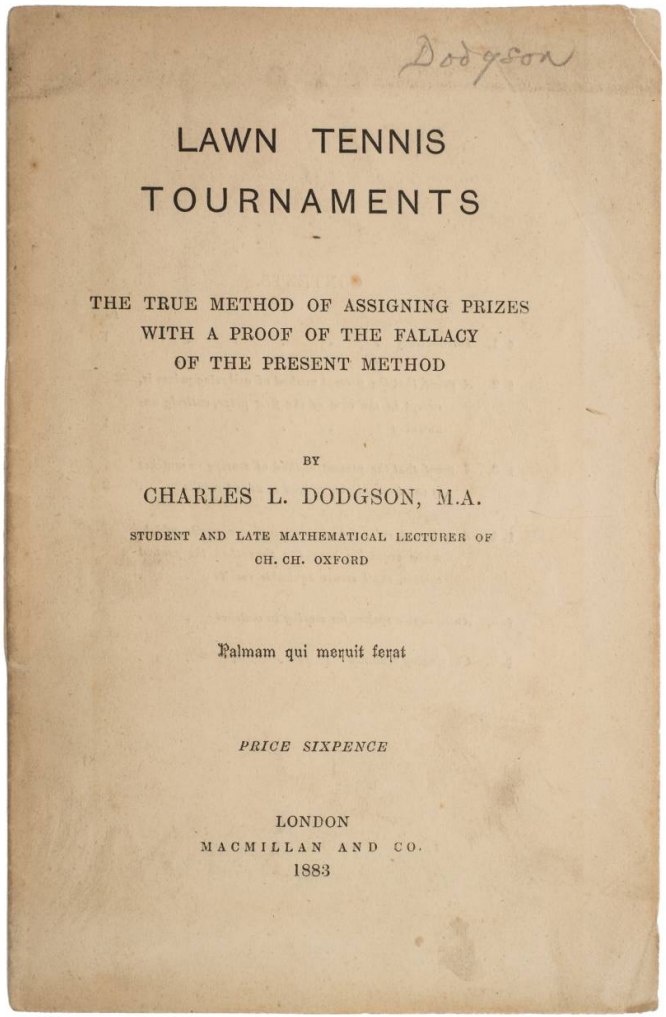
Algoritme (vervolg):

- nu moet de op één na grootste nog worden gevonden.
- dit moet een van de ℓ spelers zijn die in het toernooi heeft verloren van de grootste, en wel de grootste van die ℓ.
- het kost $\ell - 1$ 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 $n + \lceil \lg n \rceil - 2$.

Even tussendoor



Dodgson

LAWN TENNIS
TOURNAMENTS

THE TRUE METHOD OF ASSIGNING PRIZES
WITH A PROOF OF THE FALLACY
OF THE PRESENT METHOD

BY

CHARLES L. DODGSON, M.A.

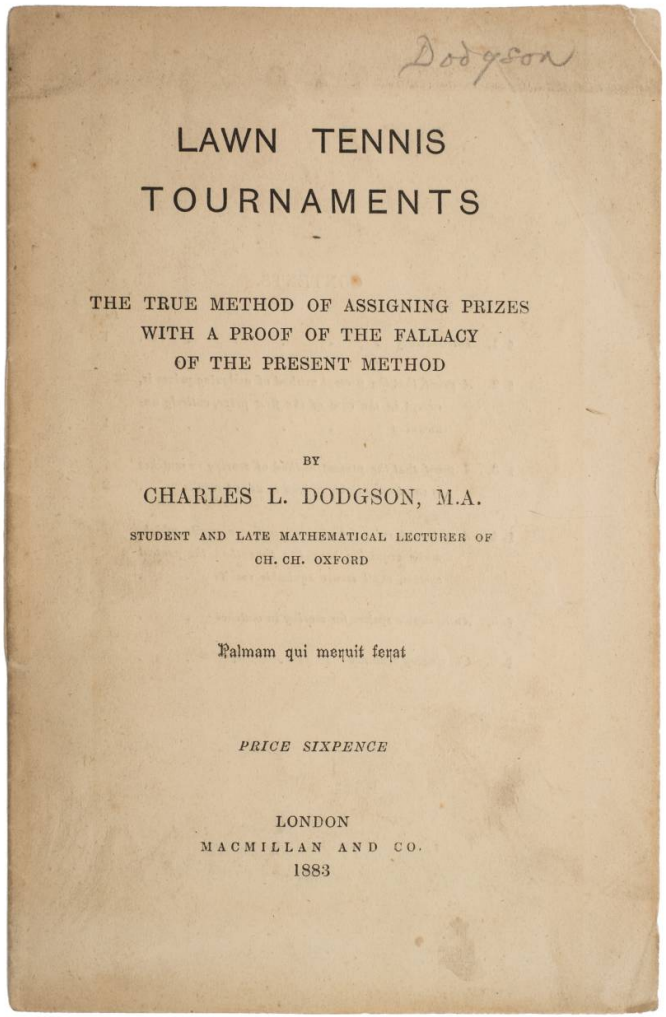
STUDENT AND LATE MATHEMATICAL LECTURER OF
CH. CH. OXFORD

Palmam qui meruit ferat

PRICE SIXPENCE

LONDON
MACMILLAN AND CO.
1883

Even tussendoor


Dodgson

LAWN TENNIS
TOURNAMENTS

THE TRUE METHOD OF ASSIGNING PRIZES
WITH A PROOF OF THE FALLACY
OF THE PRESENT METHOD

BY

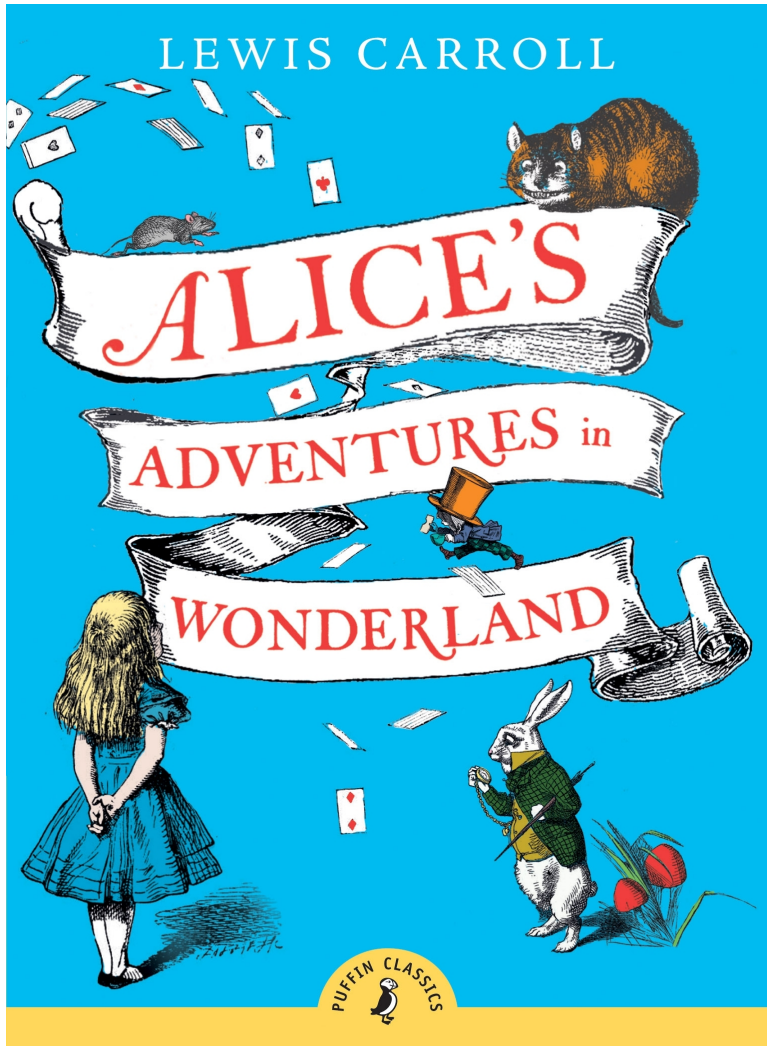
CHARLES L. DODGSON, M.A.

STUDENT AND LATE MATHEMATICAL LECTURER OF
CH. CH. OXFORD

Palmam qui meruit ferat

PRICE SIXPENCE

LONDON
MACMILLAN AND CO.
1883



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 ... 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 ... stappen doet
- een manier om dit voor elkaar te krijgen is met behulp van een adversary-argument

Adversary-argument

Een algoritme speelt een vraag-en-antwoord-spel tegen een adversary (tegenstander).

- algoritme: wil zo veel mogelijk informatie krijgen om met zo weinig mogelijk vragen het probleem op te lossen
- adversary: wil zo weinig mogelijk informatie prijsgeven om ervoor te zorgen dat het algoritme zo veel mogelijk vragen moet stellen om de/een oplossing te vinden
- belangrijkste spelregel: consistentie. De adversary geeft alleen antwoorden die consistent zijn met eerder gegeven informatie.

Voorbeelden:

- Wie is het?, Galgje, Mastermind, Zeeslag, ...
- X zoeken in een rij met n verschillende elementen (zie college)
- sorteren

Adversary-argument

- de adversary beantwoordt de vragen van het algoritme volgens een of andere adversary-strategie
- deze strategie wil het algoritme dwingen zo veel mogelijk vragen te stellen
- de adversary bouwt zo tijdens de uitvoering van een algoritme a.h.w. een bad case -invoer op
- de adversary(-strategie) zorgt ervoor dat hij op elk moment een invoer kan geven die consistent is met de al gegeven antwoorden
- het aantal stappen (vragen) dat *elk willekeurig* algoritme *ten minste* tegen de adversary(-strategie) moet uitvoeren om het juiste antwoord te krijgen geeft een ondergrens op de worst case-complexiteit van het probleem

Ondergrens sorteren

Stelling

Elk algoritme gebaseerd op het doen van arrayvergelijkingen dat een array van n elementen (oplopend) sorteert, heeft in de worst case ten minste $\lceil \lg n! \rceil \in \Theta(n \lg n)$ vergelijkingen nodig.

Opmerking: we bewijzen een ondergrens voor de worst case. We mogen daarom aannemen dat alle array-elementen (bijv. getallen) verschillend zijn (of zelfs dat het gaat om 1 t/m n).

We bewijzen de ondergrens via een adversary-argument.

Bovengenoemde ondergrens is scherp: sorteren kán ook in $O(n \lg n)$.

Adversary en sorteren

- sorteren komt neer op het vinden van de juiste (d.w.z. oplopende) ordening van de array-elementen (bijvoorbeeld $A[3] < A[2] < A[4] < A[1]$)
- er zijn voor een rijtje met n verschillende waarden precies $n!$ mogelijke ordeningen
- de adversary heeft $n!$ kaarten in zijn hand, met op iedere kaart een mogelijke ordening
- na iedere vraag $A[i] \stackrel{?}{<} A[j]$ van het algoritme geeft de adversary antwoord: ja of nee
- de adversary gooit vervolgens alle kaarten weg die inconsistent zijn met dit antwoord

Adversary en sorteren

- de adversary probeert het spelletje zo lang mogelijk te rekken, opdat het algoritme zo veel mogelijk vragen moet stellen om het antwoord (zeker) te weten
- strategie van de adversary: kies altijd het antwoord dat de meeste kaarten overlaat; kies ja als het niet uitmaakt
- het algoritme is klaar als er nog maar één kaart (ordening) over is
- wat is dan het *minimale* aantal vragen dat de adversary het algoritme kan dwingen te stellen, zodanig dat er maar één ordening overblijft?

Adversary en sorteren

- omdat de adversary telkens de meeste kaarten probeert over te laten, kan het algoritme bij iedere vraag maximaal de *helft* van de kaarten wegspelen
- het aantal vragen dat het algoritme moet stellen is daarom gegarandeerd minimaal $\lceil \lg n! \rceil \in \Theta(n \lg n)$

Conclusie: voor iedere volgorde van vragen van het algoritme kunnen we zo een bijbehorende *bad case-invoer*[*] construeren die ten minste $\Theta(n \lg n)$ vragen vereist

[*] namelijk een rijtje corresponderend met de ordening die als laatste overbleef tegen de adversary

Adversary-argument

Een adversary-argument wordt gebruikt om een ondergrens te vinden voor de worst case-complexiteit van een probleem. De adversary

- "speelt" tegen een algoritme volgens een adversary-strategie
- probeert het algoritme zo veel mogelijk vragen te laten stellen
- antwoordt altijd consistent met eerdere antwoorden
- bouwt zo tegen elk algoritme een bad case-invoer op

Het doel is om een ondergrens $f(n)$ af te leiden voor het aantal stappen dat elk algoritme nodig heeft tegen de adversary. Dat betekent dat er voor *elk algoritme* een invoer bestaat waarop minstens $f(n)$ stappen nodig zijn. In dat geval is het aantal stappen in de worst case voor elk algoritme dus $\geq f(n)$.

Merk op dat je niet hoeft te weten hoe zo'n bad case er uitziet, alleen dat hij bestaat.

Maximum en minimum

Stelling

Elk algoritme gebaseerd op arrayvergelijkingen dat het minimum en het maximum vindt van n (verschillende) waarden, doet in het slechtste geval ten minste $\lceil \frac{3n}{2} - 2 \rceil$ vergelijkingen.

Bewijs

De status van een array-element geven we aan met:

1. W: ≥ 1 keer gewonnen, nooit verloren
2. V: ≥ 1 keer verloren, nooit gewonnen
3. WV: ≥ 1 keer gewonnen en ≥ 1 keer verloren
4. N: nog nooit "gespeeld"

De adversary-strategie is gebaseerd op de status van de array-elementen op het moment dat die worden vergeleken door het algoritme.

Adversary-strategie

wedstrijd x–y	uitslag	N	W	V	WV	type
N–N	$x > y$	-2	$+1$	$+1$	—	1
V–N	$x < y$	-1	$+1$	—	—	1
W–N	$x > y$	-1	—	$+1$	—	1
W–W	consistent	—	-1	—	$+1$	2
V–V	consistent	—	—	-1	$+1$	2
W–V	$x > y$	—	—	—	—	
WV–WV	consistent	—	—	—	—	
WV–N	$x > y$	-1	—	$+1$	—	1
WV–W	$x < y$	—	—	—	—	
WV–V	$x > y$	—	—	—	—	
	begin	n	0	0	0	
	eind	0	1	1	$n-2$	

Toelichting

Merk op dat de uitslagen altijd kunnen, want de waarde van een V-element (resp. W-element) mag altijd ongestraft omlaag (resp. omhoog); de eerder gegeven antwoorden blijven dan geldig.

vergelijking x–y	actie van de adversary
N–N	kies "frisse" waarden zodat $x > y$
W–N	kies "frisse" waarde voor y ($< x$)
W–V	x mag omhoog of y omlaag zodat $x > y$
W–W	laat de grootste winnen
WV–V	y mag omlaag

Bewijs ondergrens

- om van de beginsituatie naar de eindsituatie te komen *moeten* er $n-2$ vergelijkingen van type 2 worden gedaan, want dat is de enige manier om $n-2$ WV's te krijgen
- tevens *moeten* er minstens $\lceil \frac{n}{2} \rceil$ vergelijkingen van type 1 worden gedaan om het aantal N's op 0 te krijgen
- aangezien vergelijkingen van type 1 niets doen met het aantal WV's en die van type 2 niets met het aantal N's, zijn er in totaal (je mag ze dus optellen) ten minste $n-2+\lceil \frac{n}{2} \rceil$ vergelijkingen nodig tegen deze strategie

Om te onthouden van vandaag

- beslissingsboomargument:
  - hoogte boom h, worst case $h + 1$
  - n knopen: $h \geq \lceil \lg(n + 1) \rceil - 1 = \lfloor \lg n \rfloor$
  - b bladeren: $h \geq \lceil \lg b \rceil$
  - "makkelijk", niet altijd even scherp
- adversary-argument
  - zie het probleem als een twee-spelerspel
  - adversary (tegenstander) houdt jou zo lang mogelijk bezig
  - adversary-strategie geeft ondergrens op worst case
- toernooimethode: optimaal algoritme voor het vinden van de op één na grootste

(Werk)college

Volgende college: dinsdag 28 februari, 9u00–10u45, zaal C1 (Gorlaeus)

Opnamen van vorig jaar komen binnenkort op Brightspace.

Werkcollege: zodadelijk van 11u00 tot 12u45, computerzalen 302–304, 306–308 en 303 (Snellius)
Opgaven uit het dictaat: 20, 5d, 27, 22, 32

Extra werkcollege/vragenuur: zodadelijk van 13u15 tot 14u00, computerzaal 302–304 (Snellius)

Opmerking: hoeveelheid en volgorde opgaven zijn een advies, het is niet de verwachting dat iedereen dit elke week af krijgt.

Huiswerk

Inleveren op Brightspace, vóór hoorcollege 4 (28 februari 9u00).
Daarna nog wel feedback maar geen cijfer.