Vierde college complexiteit

26 februari 2019

Beslissingsbomen en selectie
Toernooimethode
Adversary argument

- 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 $X =, < A[i]$

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 (= 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:

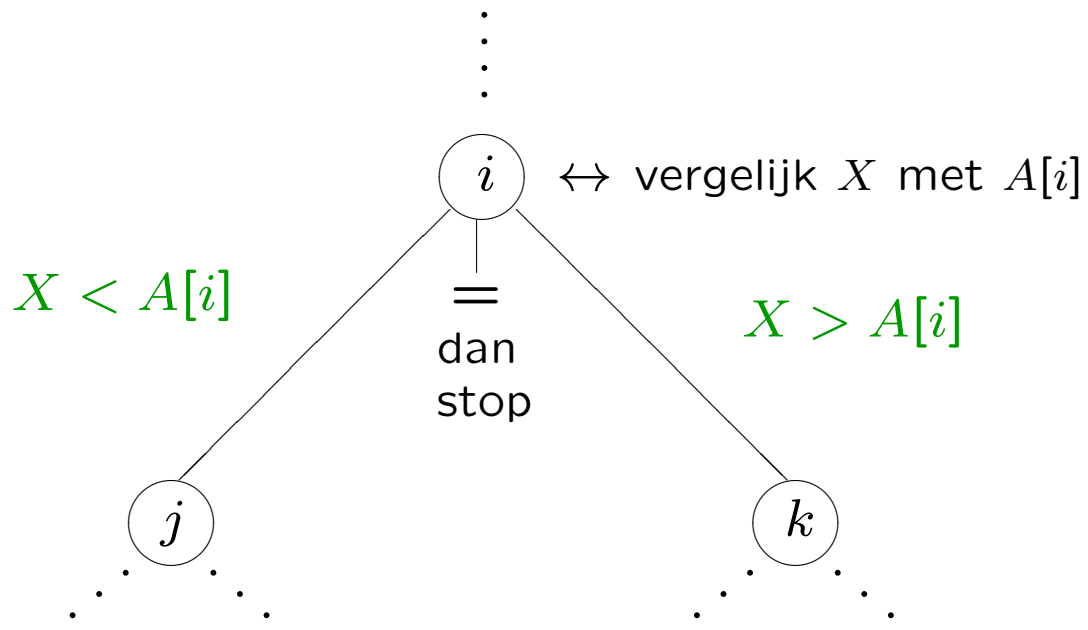
1. $h \geq \lceil \lg b \rceil$

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

algoritme 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 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 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 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 (**) en $N \geq n$ geldt dan 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.

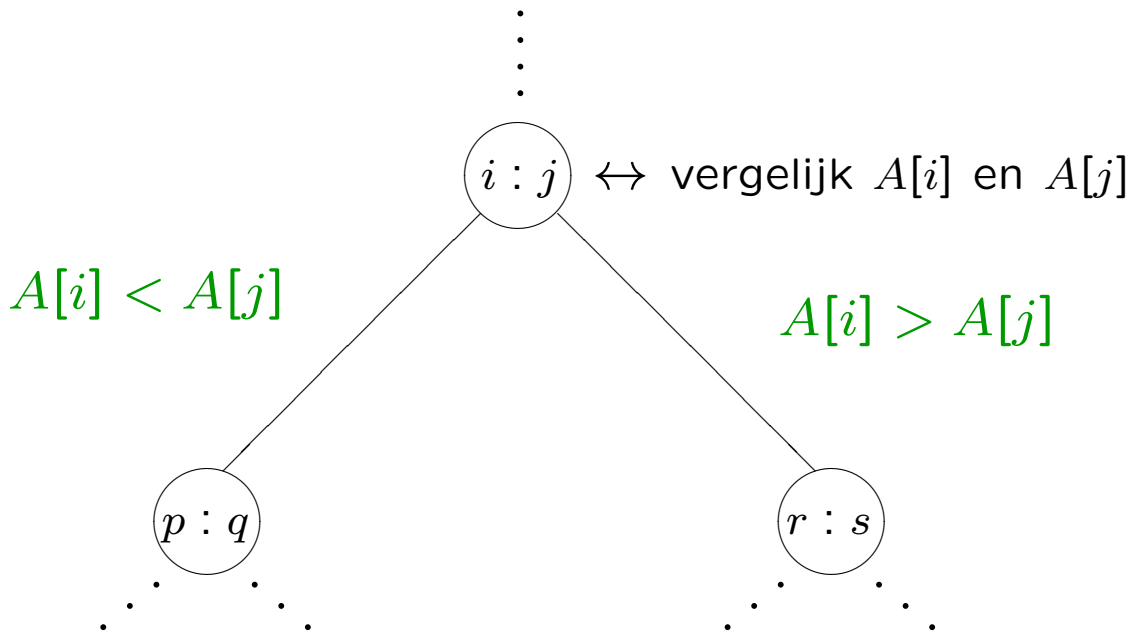
algoritme gebaseerd op het doen van arrayvergelijkingen $A[i] < A[j]$*

$\Downarrow$

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

Beslissingsboom type 2

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

*inclusief de speciale bladeren

Stelling

Elk algoritme dat de (index van de) grootste (of de kleinste) waarde bepaalt uit een array met n elementen, en dat uitsluitend gebaseerd is 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 $n - 1$.

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 $\lceil \frac{3n}{2} \rceil - 2$.

- 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 !

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 n verschillende getallen, opgeslagen in een array A: $A[1], A[2], \ldots, A[n]$. Laat verder een geheel getal k met $1 \leq k \leq n$ gegeven zijn.
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

Zowel de bovengrens als de ondergrens kunnen scherper.

We bekijken hierna steeds (het precieze aantal vergelijkingen in) de worst case.

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

   Het kan met $n - 1$ vergelijkingen. Dit 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 (toernooimethode.) Dit is optimaal. Bewijzen we 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 werkcollege opgave 23

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 toernooi 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: $n + \lceil \lg n \rceil - 2$.

- 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
- 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

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

- algoritme: wil zo veel mogelijk informatie krijgen om met zo weing 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:

- datum raden (maand + dag)
- galgje: woord raden (bijvoorbeeld binnen 10 stappen)
- X zoeken in een rij met n verschillende elementen (zie college)
- sorteren

- 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 op de gestelde vragen 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

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 we te maken hebben met een rijtje bestaande uit de getallen 1 t/m n).

We bewijzen de ondergrens via een adversary argument.

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

- Sorteren komt neer op het vinden van de juiste (=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 $A[i] < A[j]$? van het algoritme geeft de adversary antwoord: JA of NEE

- De adversary gooit vervolgens alle kaarten weg die inconsistent zijn met dit antwoord

- 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 1 ordening overblijft ?

- Omdat de adversary telkens de meeste kaarten over probeert 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

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 als het ware tegen elk algoritme een bad case invoer op

Het doel is een ondergrens $f(n)$ af te leiden voor het aantal stappen dat elk algoritme tegen de adversary nodig heeft. 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.

Stelling

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

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	$n-2$	

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 (constructie bad case)
N-N	kies "frisse" waarden zodat x>y
W-N	kies "frisse" waarde voor y (<x) je mag de waarde van x ook ongestraft verhogen; dat beïnvloedt eerdere uitslagen namelijk niet
W-V	x mag ongestraft verhoogd worden (of y verlaagd) zodat x > y wordt
W-W	laat winnen welk van de twee groter is
WV-V	y mag ongestraft verlaagd worden

- Om van beginsituatie naar eindsituatie te komen *moeten* er $n-2$ vergelijkingen van type 2 gedaan worden, want dat is de enige manier om $n-2$ WV's te krijgen

- Tevens *moeten* er ten minste $\lceil \frac{n}{2} \rceil$ van type 1 gedaan worden, 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

- Volgende college:
  dinsdag 5 maart, 11.00 – 12.45, zaal 174

- Tweede werkcollege:
  dinsdag 26 februari, 13.30 – 15.15, zaal 174
  Opgaven 7, 8, 9, 10, 20

- Huiswerkopgave 1:

  * deadline: vrijdag 8 maart; LaTeX, print $\rightarrow$ kamer 159

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