Achtste college Complexiteit

4 april 2025

Shellsort restant

optimaal sorteren

$O(n)$-sorteren

- Quicksort
  - verdeel-en-heers: partitioneert het array in twee delen en sorteert deze *recursief*
  - worst case $\in \Theta(n^2)$, average case $\in \Theta(n \lg n)$
- Shellsort
  - gebruikt een rijtje stapgroottes $h_t, h_{t-1}, \ldots, h_2, h_1 = 1$
  - h_i-sorteert deelrijtjes van eltn op afstand h_i van elkaar
  - sorteert de deelrijtjes m.b.v. Insertion sort
  - met $h_i = \lfloor \frac{h_{i+1}}{2} \rfloor$ (Shell): worst case $\in \Theta(n^2)$
  - complexiteit hangt af van de gekozen stapgroottes
  - analyse extreem moeilijk en incompleet

Definitie

Een rij $A[1], A[2], \ldots, A[n]$ heet k-gesorteerd als geldt:
$A[i] \leq A[i+k]$ voor elke $i = 1, 2, \ldots, n-k$.

Voorbeeld: Het rijtje

$$5, 8, 24, 13, 7, 18, 31, 19, 44, 63, 82, 29$$

is 4-gesorteerd (en overigens ook 6-gesorteerd).

Merk op: 1-gesorteerd = gesorteerd.

```
(1)   h = n div 2; // h_t = ⌊n/2⌋ dus
(2)   while h > 0 do
(3)       for i := h + 1 to n do
(4)           temp := A[i]; j := i;
(5)           while j − h > 0 do
              // invoegen op de juiste plek in je eigen deelrijtje
(6)               if temp < A[j − h] then
(7)                   A[j] := A[j − h];              // schuif
(8)                   j := j − h;
(9)               else
(10)                  "exit binnenste while";
(11)              fi
(12)          od
(13)          A[j] := temp;                          // zet neer
(14)      od
          // de rij is nu h-gesorteerd
(15)      h := h div 2; // oorspronkelijke keuze van Shell: h_i = ⌊h_{i+1}/2⌋
(16)  od
```

Merk op: regel (4) t/m (13) is Insertion sort op deelrijtjes

Stapgroottes: $h_t = \lfloor \frac{n}{2} \rfloor$, $h_i = \lfloor \frac{h_{i+1}}{2} \rfloor$ voor $i = t-1, \ldots, 1$.
Het aantal rondes is dan $t = \lfloor \lg n \rfloor$.

- Stelling A
  Het aantal arrayvergelijkingen dat Shellsort met deze serie stapgroottes doet is in de worst case $\Omega(n^2)$.

  Vorige keer aangetoond voor $n = 2^k$ door een bad case invoer te geven waarvoor het aantal vergelijkingen $\Omega(n^2)$ is.

- Stelling B
  Het aantal arrayvergelijkingen dat Shellsort doet met deze serie stapgroottes is in de worst case $O(n^2)$.

  Bewijs: zie college.

Stapgroottes (Hibbard): $2^k - 1, 2^{k-1} - 1, \ldots, 7, 3, 1$.
Het aantal rondes is hier $k = \lfloor \lg n \rfloor$.

Merk op: opeenvolgende stapgroottes zijn hier relatief priem (hebben geen gemeenschappelijke deler). Dit volgt uit: $h_{i+1} = 2h_i + 1$.

- Stelling. In de worst case doet Shellsort met Hibbard's serie stapgroottes $O(n\sqrt{n})$ vergelijkingen.

Voorbeeld. Na 8-sorteren heeft 3-sorteren i.h.a. veel meer effect dan 4-sorteren.

8-gesorteerd, 36 inversies

$5, 2, 11, 7, 15, 3, 16, 6, 12, 9, 14, 8, 19, 13, 20, 10$

$\downarrow$ 4-sorteren

31 inversies

$5, 2, 11, 6, 12, 3, 14, 7, 15, 9, 16, 8, 19, 13, 20, 10$

8-gesorteerd, 36 inversies

$5, 2, 11, 7, 15, 3, 16, 6, 12, 9, 14, 8, 19, 13, 20, 10$

$\downarrow$ 3-sorteren

7 inversies

$5, 2, 3, 6, 7, 8, 9, 12, 11, 10, 13, 15, 14, 16, 20, 19$

Stapgroottes (Hibbard): $2^k - 1, 2^{k-1} - 1, \ldots, 7, 3, 1$, $k = \lfloor \lg n \rfloor$.

Stelling. In de worst case doet Shellsort met Hibbard's reeks stapgroottes $O(n\sqrt{n})$ arrayvergelijkingen.

Bewijsschets: Je kunt inzien dat je bij stapgrootte h_i voor elk van de $n - h_i$ getallen maar met hoogstens $8h_i + 4$ getallen hoeft te vergelijken*. Met $h_i = 3$, $h_{i+1} = 7$ en $h_{i+2} = 15$ weet je bij het h_i-sorteren bijvoorbeeld al zeker dat $A[40] \leq A[92]$ omdat $A[40] \leq A[47] \leq A[62] \leq A[77] \leq A[92]$. Dat is dus $O(nh_i)$ werk. We gebruiken dit voor $h_i < \sqrt{n}$. Voor $h_i > \sqrt{n}$ als voorheen (zie bewijs Stelling B): $O(n^2/h_i)$. Totaal geeft dat uiteindelijk $O(n\sqrt{n})$.

*dit volgt uit de speciale eigenschappen van de h_i

Bij bewijs Hibbard*

We gebruiken een Stelling uit de getaltheorie

Als ggd$(h, \ell) = 1$, dan is het grootste gehele getal m dat niet als $a \cdot h + b \cdot \ell$ ($a, b \geq 0$, geheel) geschreven kan worden: $m = h\ell - h - \ell$

Gevolg

Alle gehele getallen $\geq h\ell - h - \ell = (h-1)(\ell-1)$ kunnen als lineaire combinatie van h en ℓ worden geschreven.
Voorbeeld: $h = 3, \ell = 7$ dus $m = 12$: $1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, \ldots$

Als we dit toepassen op h_{i+1} en h_{i+2} vinden we: alle integers j met $j \geq (h_{i+1} - 1)(h_{i+2} - 1) = 8{h_i}^2 + 4h_i$ kunnen geschreven worden als $a \cdot h_{i+1} + b \cdot h_{i+2}$. Voor deze j geldt dus bij het h_i-sorteren dat $A[p-j] = A[p - a \cdot h_{i+1} - b \cdot h_{i+2}] \leq A[p]$, want het array is op dat moment al h_{i+1}- en h_{i+2}-gesorteerd.
Voor elk van de $n - h_i$ in te voegen array-elementen hoeven we dus maar met maximaal $(8{h_i}^2 + 4h_i)/h_i = 8h_i + 4$ elementen te vergelijken.

*dit is extra voor de liefhebbers

Betere stapgrootteseries:

- Twee (!) doorgangen: $h_2 \approx 1,72 \cdot \sqrt[3]{n},\ h_1 = 1$
  Gemiddeld $O(n^{5/3})$

- Heel veel doorgangen: increments van de vorm $2^i \cdot 3^j$:
  $1, 2, 3, 4, 6, 8, 9, 12, 16, 18, 24, 27, 32, 36, 48, 54, 72, 81, \ldots$
  Worst case $O(n(\lg n)^2)$

- Increments van de vorm $4^{j+1} + 3 \cdot 2^j + 1$:
  $1, 8, 23, 77, 281, 1073, 4193, 16577, \ldots$
  Worst case $O(n^{4/3})$

- $\cdots$ (optimale serie stapgroottes is [nog] onbepaald)

Voor sorteeralgoritmen gebaseerd op arrayvergelijkingen is bewezen: het aantal arrayvergelijkingen* in de worst case is ten minste $\lceil \lg n! \rceil$

Vraag:
hoe dicht kan men in de buurt van deze ondergrens komen?

Tabel:

n	2	3	4	5	6	7	8	9	10	11	12	...	21
$\lceil \lg n! \rceil$	1	3	5	7	10	13	16	19	22	26	29	...	66
$M(n)$	1	3	5	8	11	14	17	21	25	29	33	...	74
$B(n)$	1	3	5	8	11	14	17	21	25	29	33	...	74

*voor het vinden van de juiste ordening

$M(n)$ is het aantal arrayvergelijkingen dat Mergesort doet in de worst case voor een rij met n elementen.
$B(n)$ is het aantal arrayvergelijkingen dat Binary Insertion sort in de worst case doet voor een rij met n elementen.
Merk op dat $\lceil \lg 1! \rceil = M(1) = B(1) = 0$.

Binary Insertion sort werkt als Insertion sort, maar gebruikt voor het zoeken van de plek waar $A[i]$ moet komen binair zoeken in plaats van lineair zoeken. (Zie ook opgave 34.)

Om $A[i]$ op de juiste plek te kunnen invoegen in het deelarray $A[1], \ldots, A[i-1]$ zijn nu in het slechtste geval $\lceil \lg i \rceil$ arrayvergelijkingen nodig*. Dus:

$$B(n) = \sum_{i=2}^{n} \lceil \lg i \rceil \geq \lceil \sum_{i=2}^{n} \lg i \rceil = \lceil \lg n! \rceil$$

*voor het vinden van de juiste positie

Er geldt zelfs:

$$\sum_{i=2}^{n} \lceil \lg i \rceil = n\lceil \lg n \rceil - 2^{\lceil \lg n \rceil} + 1,$$

en dat is ook precies het aantal vergelijkingen dat Mergesort doet. Dus $B(n) = M(n)$.

Vraag: Kan het nog beter?
Antwoord: Ja !
Voorbeeld: 5 elementen kunnen worden gesorteerd met 7 vergelijkingen (Howard Demuth, 1956). Zie ook opgave 25b.

De methode van Demuth kan gegeneraliseerd worden tot het algoritme Merge Insertion sort (Ford & Johnson, 1959)*.

*Lester Ford, Selmer Johnson

5 elementen kunnen worden gesorteerd met hooguit 7 vergelijkingen

3 vergelijkingen: $a < b < d$; $c < d$; e

2 vergelijkingen: e invoegen in $a < b < d$

max 2 vergelijkingen: c invoegen in $e < a < b < d$ of

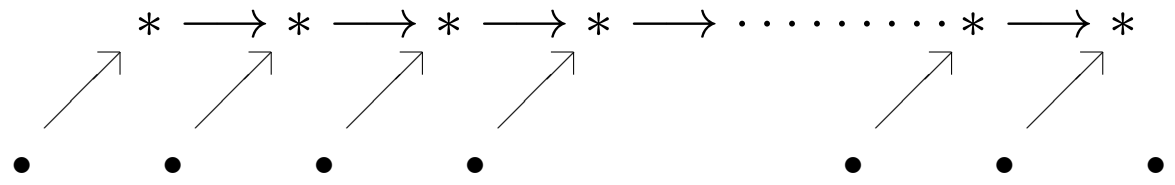
$a < e < b < d$ of

$a < b < e < d$ of

$a < b < d < e$

1. Vergelijk de n elementen twee aan twee.

2. Sorteer de $\lfloor \frac{n}{2} \rfloor$ winnaars $*$ (de grootsten dus) recursief. Dit levert iets op als:



Hierin betekent $*_1 \longrightarrow *_2$ dat $*_1 < *_2$ en $\bullet \longrightarrow *$ dat $\bullet < *$

3. Voeg nu de $\lfloor \frac{n}{2} \rfloor$ verliezers (en de losse waarde als n oneven is) in op de juiste plek **in een ingenieuze volgorde**.

Het aantal vergelijkingen dat Merge Insertion sort doet is voor $n \leq 11$ en $n = 20, 21$ gelijk aan de theoretische ondergrens $\lceil \lg n! \rceil$ voor sorteren.

Merge Insertion sort sorteert bijvoorbeeld:

- 10 elementen in 22 vergelijkingen (optimaal)
- 21 elementen in 66 vergelijkingen (optimaal)
- 12 elementen in 30 vergelijkingen: niet gelijk aan de theoretische ondergrens, wel (optimaal)

Voor kleine n (zoals $n = 12$) is het mogelijk om, via exhaustive search, de echte ondergrens *empirisch* te bepalen: probeer alle mogelijke combinaties van vergelijkingen op alle permutaties van n getallen en concludeer dat ℓ vergelijkingen niet voldoende zijn. De ondergrens is dan dus $\geq \ell + 1$.
Inmiddels is zo ook aangetoond dat Merge Insertion sort optimaal is voor $n = 13, 14, 15, 22$ (Peczarski, 2004/2006). In 2023 is bewezen dat dit ook geldt voor $n = 16, 17, 18$ (Stober, Weiss).

Vraag: is Merge Insertion sort optimaal?

Antwoord: nee, bijvoorbeeld voor $n = 47$ is een algoritme bekend (Schulte Mönting, 1981) dat één vergelijking minder nodig heeft (200) dan Merge Insertion sort (201).

Als je helemaal niets weet over het array A is de enige manier om A correct te ordenen het stellen van vragen van de vorm $A[i] < A[j]$. In dat geval geldt de theoretische ondergrens $\lceil \lg n! \rceil \in \Theta(n \lg n)$. Soms kun je echter sneller sorteren dan $\Theta(n \lg n)$ (namelijk in $O(n)$ tijd) door handig gebruik te maken van voorkennis over de invoer.

Dit is niet in strijd met de theoretische ondergrens: die geldt immers voor sorteeralgoritmen die *alle* mogelijke inputs kunnen sorteren (en gebaseerd zijn op arrayvergelijkingen).

We bekijken ter afsluiting van het gedeelte over sorteren twee methoden die efficiënt sorteren als de invoer aan zekere voorwaarden voldoet: Counting sort en Radix sort.

Invoer: een array $A = A[1], \ldots, A[n]$
Aanname: elke $A[i]$ is een geheel getal tussen 1 en k (voor zekere k)
Uitvoer: een array B, voorstellende het array A oplopend gesorteerd

Het basisidee (als alle $A[j]$'s verschillen): bepaal voor elke $X = A[j]$ het aantal array-elementen kleiner dan X. Deze informatie kan dan gebruikt worden om X meteen op de juiste positie in het uitvoerarray te zetten: als er m elementen kleiner zijn komt $A[j]$ op plek $m + 1$. Pas dit idee aan voor de situatie waarin waarden vaker kunnen voorkomen in A. Onder bovenstaande aanname over de invoer kan dat zonder (array)vergelijkingen in $O(n)$ stappen.

```
for i := 1 to k do
      C[i] := 0;        // initialisatie
od
for j := 1 to n do
      C[A[j]] := C[A[j]] + 1;
      // telt hoe vaak de waarde A[j] in A voorkomt
od
for i := 2 to k do
      C[i] := C[i] + C[i − 1];
od
// C[i] bevat nu het aantal getallen ≤ i uit A
for j := n downto 1 do      // !!
      B[C[A[j]]] := A[j];
      C[A[j]] := C[A[j]] − 1;
od
```

A	$=$	3	6	4	1	3	4	1	4
C	$=$	0	0	0	0	0	0		
C	$=$	2	0	2	3	0	1		
C	$=$	2	2	4	7	7	8		

$n = 8$

$k = 6$

$\longleftarrow$ na tweede for

$C[i] =$ aantal keer dat i in A voorkomt

$\longleftarrow$ na derde for

$C[i] =$ aantal array-elementen $\leq i$

De complexiteit van Counting sort wordt bepaald door het aantal toekenningen (:=) aan array-elementen, en dat zijn er $\Theta(n+k)$. Indien $k \in O(n)$ is de complexiteit dus $O(n)$.

Extra geheugenruimte is ook $\Theta(n+k)$.

Counting sort werkt voor invoerrijtjes waarvan de waarden begrensd zijn (bijvoorbeeld tussen 1 en k liggen).

Counting sort is een stabiele sorteermethode, dat wil zeggen: gelijke waarden uit het invoerarray A komen in precies dezelfde volgorde in het uitvoerarray B te staan als ze in A stonden.

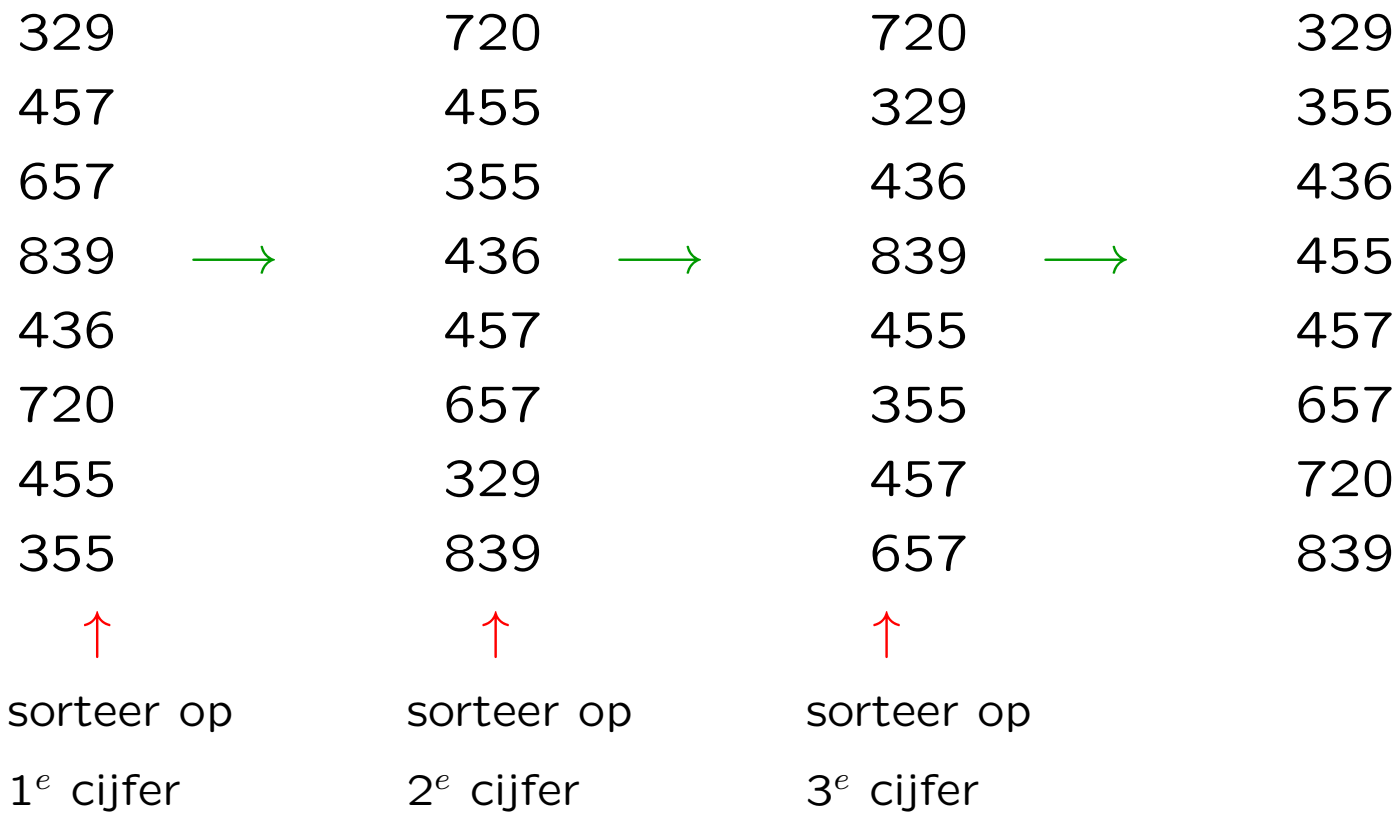
De sorteermethode Radix sort sorteert n getallen, elk van d cijfers, waarbij elk cijfer een waarde heeft tussen 0 en $k-1$ (bijvoorbeeld $k = 2$, of $k = 10$).
De getallen worden gesorteerd door ze achtereenvolgens op het i-de cijfer te sorteren, te beginnen bij het minst significante cijfer. Het gebruik van een *stabiele* sorteermethode* voor het sorteren op het i-de cijfer is essentieel.

Radixsort(A, d):

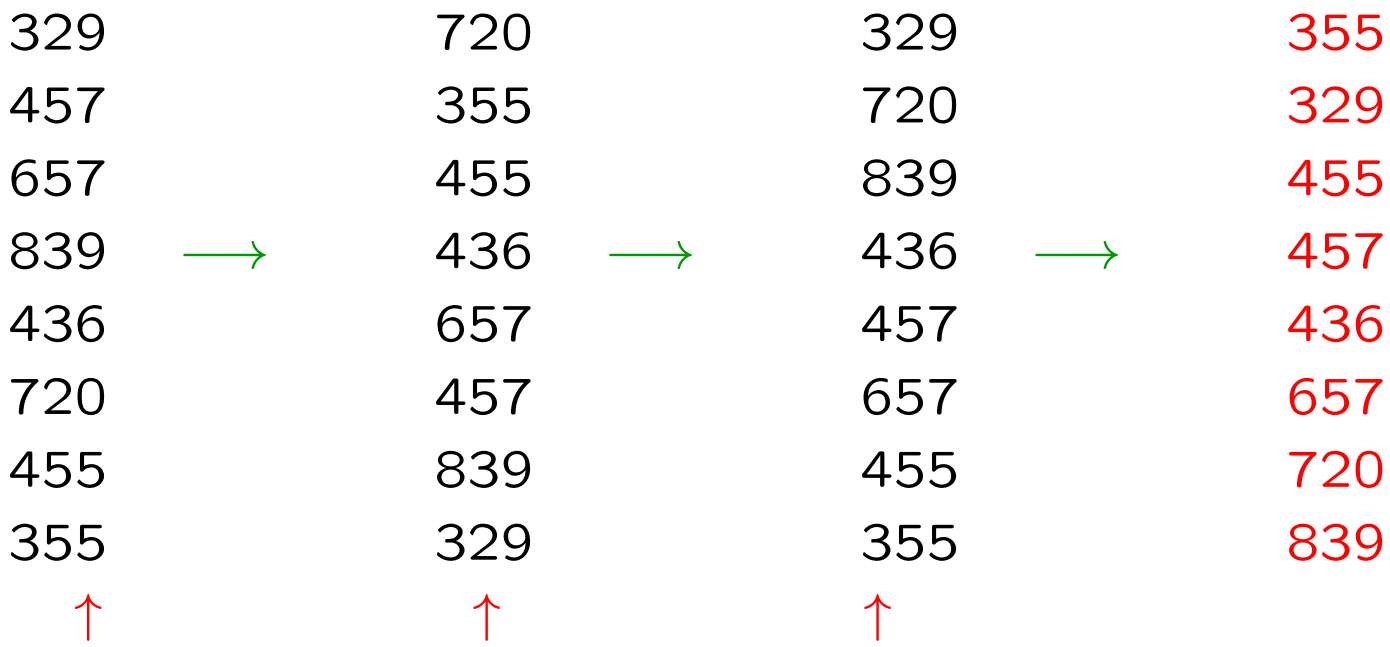
for $i := 1$ **to** d **do**
// minst significante cijfer eerst,
// dus van rechts naar links
sorteer A op het i-de cijfer
// met een stabiele (!!) sorteermethode
od

*deze behoudt de volgorde van gelijke elementen

Voorbeeld


329
457
657
839
436
720
455
355
sorteer op
1e cijfer
720
455
355
436
457
657
329
839
sorteer op
2e cijfer
720
329
436
839
455
355
457
657
sorteer op
3e cijfer
329
355
436
455
457
657
720
839

Indien de gebruikte sorteermethode voor het sorteren per cijfer niet stabiel is kan het fout gaan:



Gebruikte sorteermethode per cijfer is hier Mergesort

- Als sorteermethode per cijfer kunnen we Counting sort gebruiken, aangezien de cijfers tussen 0 en $k-1$ zitten.
- Bovendien is Counting sort *stabiel* (zie eerder).
- Als we Counting sort gebruiken kost elke ronde $\Theta(k+n)$ stappen. In totaal (d rondes) dus $\Theta(dk+dn)$. En dat is $O(n)$ als d een constante is en $k \in O(n)$.
- Een nadeel van deze methode is dat er net als bij Counting sort $\Theta(n+k)$ extra geheugenruimte nodig is.
- Radix sort kun je bijvoorbeeld ook gebruiken om woorden alfabetisch te sorteren (met $k = 26$ voor het Nederlandse alfabet).

- **Volgende college:** vrijdag 11 april, 11.00 – 12.45, Gorlaeusgebouw zaal BM.1.33

- **Straks geen werkcollege, wel:** mogelijkheid om te werken aan de tweede huiswerkopgave en stellen van vragen
  Gorlaeusgebouw zalen BW.0.29 en BW.0.30

- **Tweede huiswerkopgave:**
  deadline maandag 7 april

- **Derde huiswerkopgave (Shellsort):**
  verschijnt volgende week

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