

Vierde college algoritmiek

2 maart 2018

Toestand-actie-ruimte

Exhaustive Search

Voorbeeld 4: Kannenprobleem

We hebben twee kannen: een grote met een inhoud van 8 liter, en een kleine met een inhoud van 5 liter. Op de kannen staat geen maatverdeling. Verder hebben we de beschikking over een waterkraan en een afvoer. Bij aanvang zijn beide kannen leeg.

Vraag: Hoe krijgen we precies 4 liter water in een van de twee kannen? En liefst zo snel mogelijk.



Een Intermezzo

https://www.youtube.com/watch?v=5_MoNu9Mkm4

We onderscheiden toestanden en zinvolle (!) acties:

Toestand: Een paar (x, y) met $0 \leq x \leq 8$ en $0 \leq y \leq 5$. Hierin is x de inhoud van de grote kan en y de inhoud van de kleine kan.

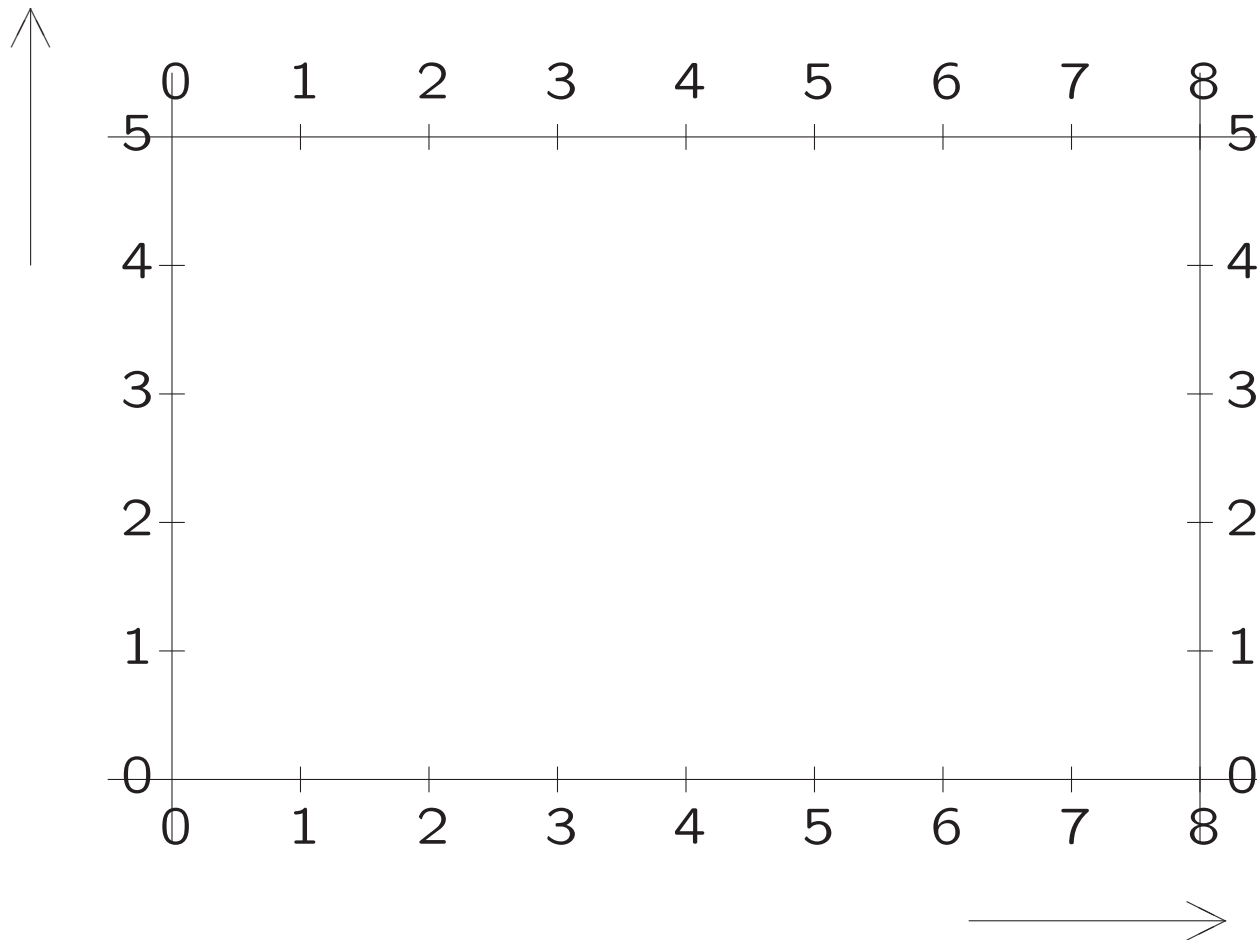
Begintoestand: beide kannen leeg, dus $(0,0)$

Eindtoestanden: alle toestanden met 4 liter in een van beide kannen, dus $(4, y)$ en $(x, 4)$

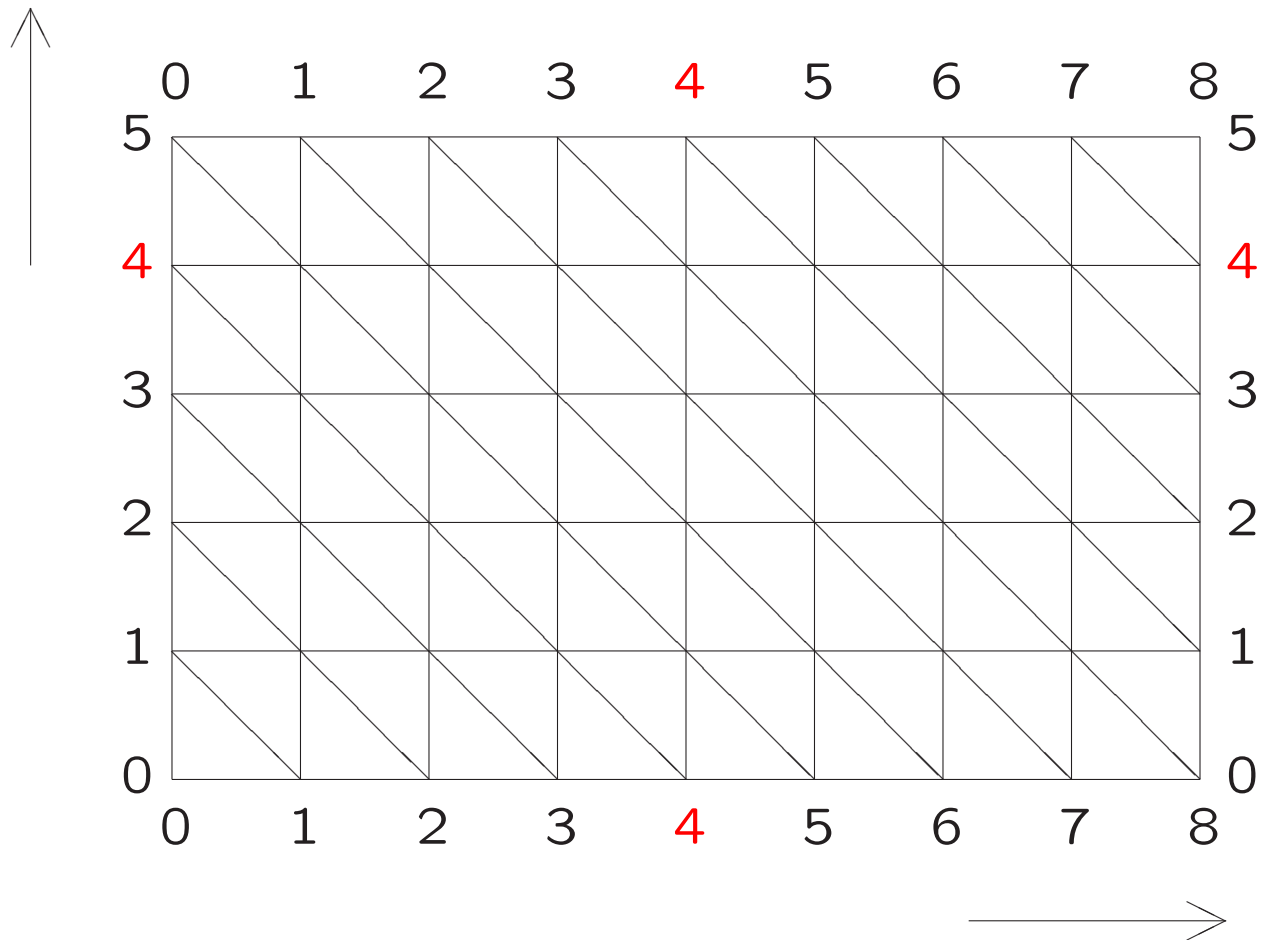
Acties: vullen, legen en overgieten

- een kan geheel (aan)vullen
- een kan geheel leeggooien
- de ene kan leeggooien in de andere
- van de ene kan in de andere gieten totdat deze vol is

inhoud kleine kan

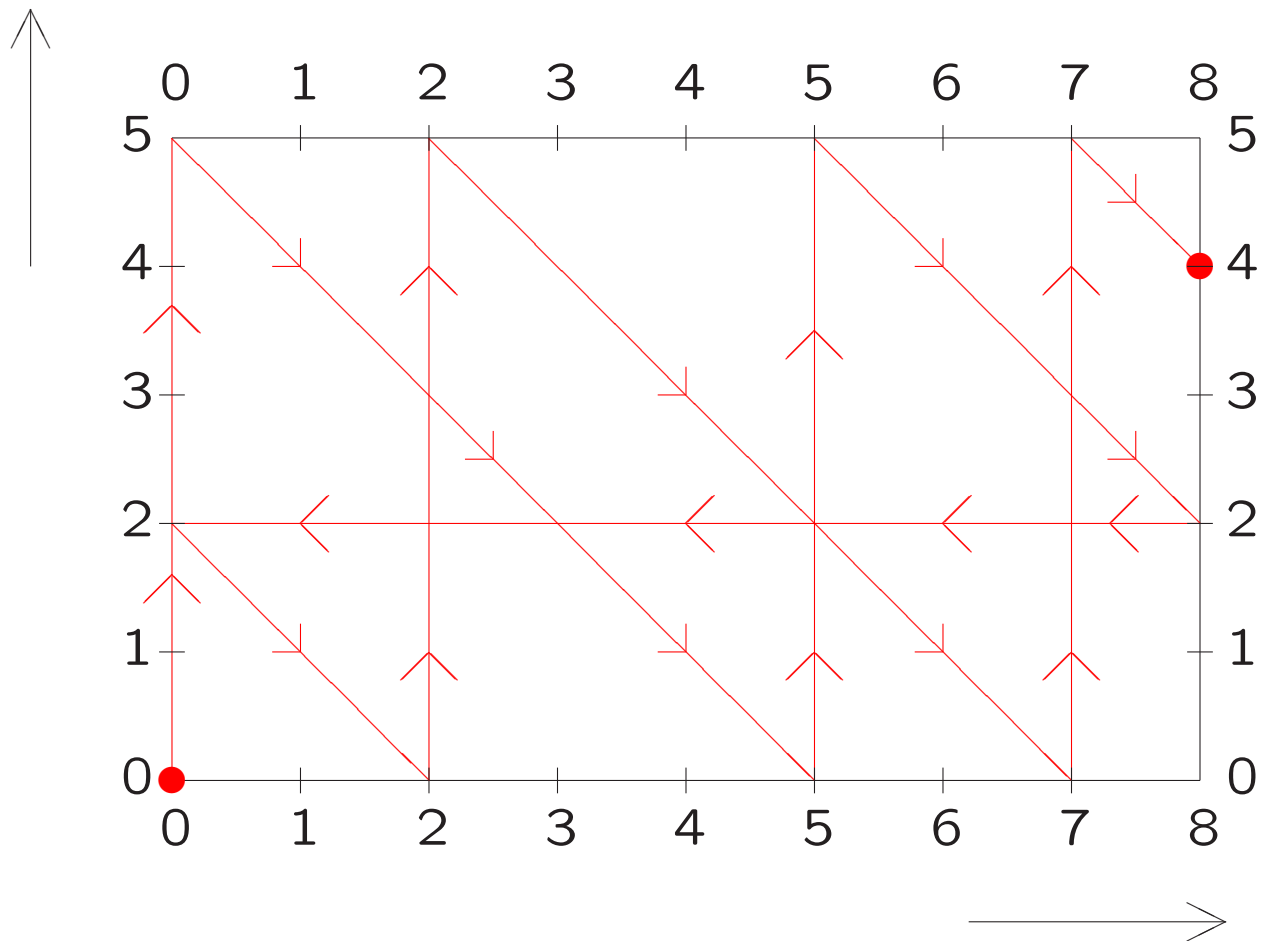


inhoud kleine kan



inhoud grote kan

inhoud kleine kan



inhoud grote kan

De snelste oplossing gebruikt de volgende **strategie** en zorgt voor 4 liter in de kleine kan. Er is overigens ook een (iets) langere oplossing, die 4 liter in de grote kan achterlaat.

Herhaal

Herhaal

Vul de kleine kan;

Giet over in de grote kan;

totdat (de grote kan vol is) of (oplossing gevonden)

Als nog geen oplossing gevonden

Grote kan leeggooien;

Giet uit de kleine kan over in de grote kan;

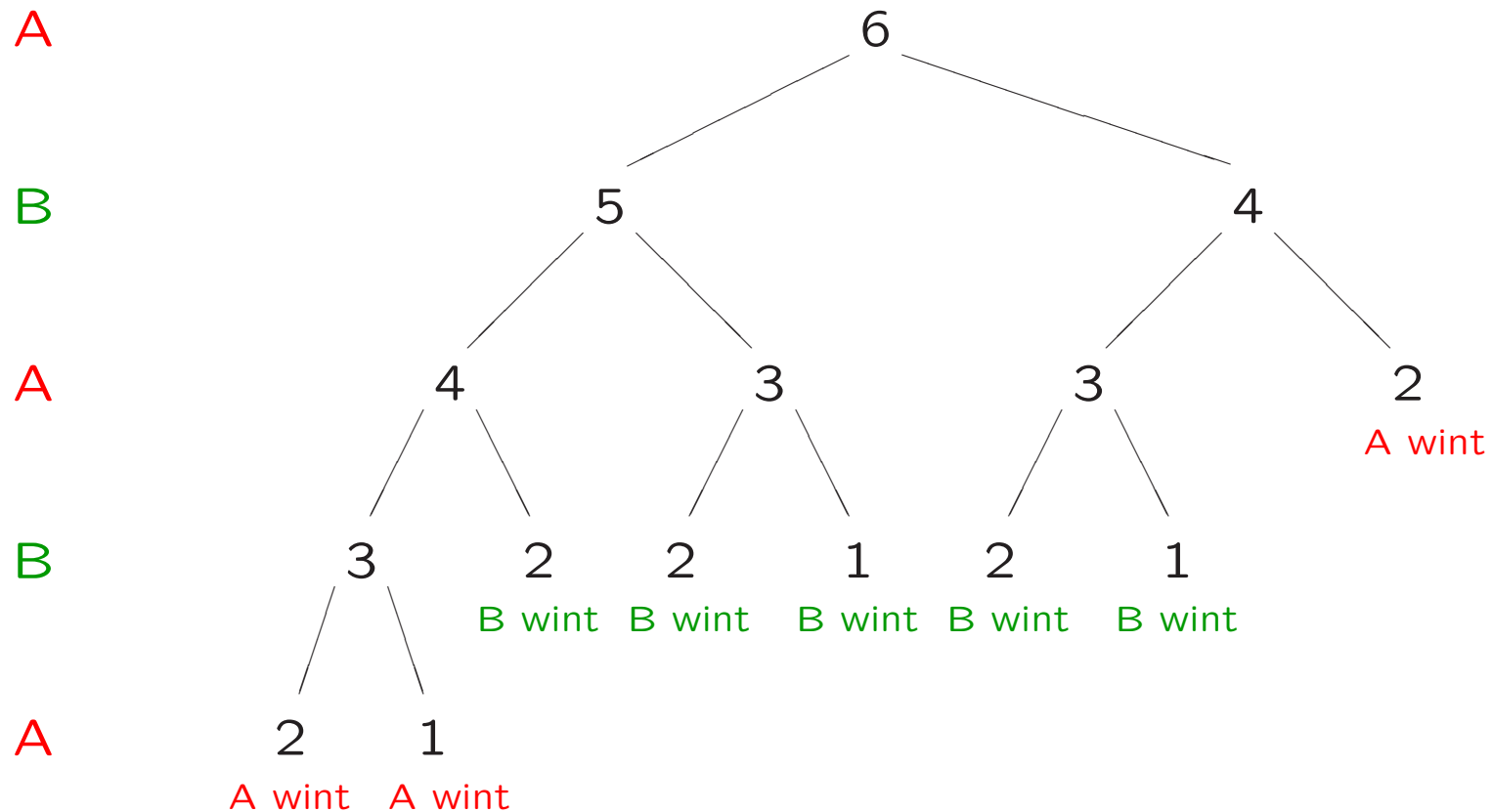
totdat oplossing gevonden

Exhaustive search: brute force benadering voor problemen die te maken hebben met het vinden van een element met een speciale eigenschap binnen een verzameling van bijv. permutaties of deelverzamelingen of toestanden of ...

Methode:

- . construeer op een systematische manier alle kandidaatoplossingen
- . evalueer elk van deze mogelijke oplossingen
- . retourneer een/de kandidaatoplossing met de gevraagde eigenschap (als die bestaat) (*)

(*) soms, zoals bij optimalisatieproblemen, *moet* je daartoe alle kandidaatoplossingen gezien hebben



Exhaustive search: doorloop (*als het ware*) de hele spelboom om te bepalen of een stand winnend is. Alle toestanden/alle spelverlopen worden zo bekeken. Je kunt stoppen zodra je een winnende zet gevonden hebt.

Belangrijke observatie:

een stand is **winnend** voor degene die aan de beurt is, dan en slechts dan als ten minste één van zijn directe vervolgstanden **niet winnend** is voor de tegenstander

\Rightarrow RECURSIE

Met kopie:

```
winnend(stand)::  
  
    if eindstand(stand) then  
        // makkelijk; bijv return false;  
    else  
        for alle mogelijke zetten i do  
            kopie := stand;  
            doezet(kopie,i);  
            if not winnend(kopie) then  
                return true;  
        od  
        return false;  
    fi
```

Zie ook Programmeermethoden (college over recursie)

Met terugzetten:

```
winnend(stand)::  
  
    if eindstand(stand) then  
        // makkelijk; bijv return false;  
    else  
        for alle mogelijke zetten i do  
            doezet(stand,i);  
            if not winnend(stand) then  
                undoezet(stand,i);  
                return true;  
            undoezet(stand,i);  
        od  
        return false;  
    fi
```

Zie ook Programmeermethoden (college over recursie)

Graafwandelingen

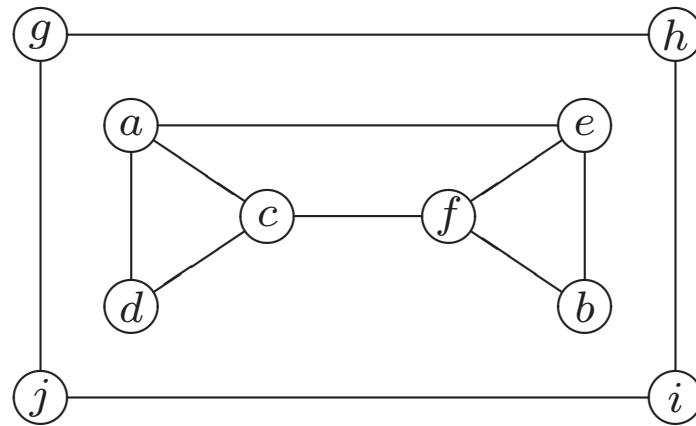
- Bij veel (graaf)problemen is het nodig om alle knopen van de graaf op een systematische manier te bezoeken

- **Graafwandelingen:**
 1. **Depth-first-search**: te vergelijken met WLR-wandeling bij bomen
 2. **Breadth-first-search**: te vergelijken met nivo-orde wandeling bij bomen

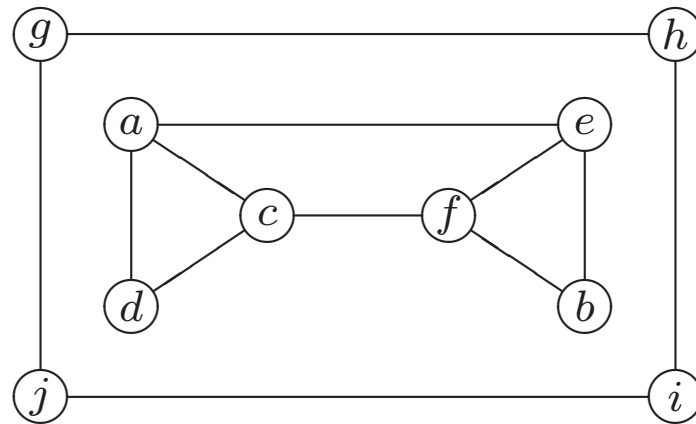
Depth-first-search

- De wandeling begint in een gegeven knoop v van de graaf.
- Vanuit een zojuist bezochte knoop wordt vervolgens steeds een aangrenzende -nog onbezochte- knoop bezocht, en vandaaruit op dezelfde manier verder gelopen tot je niet verder kan.
- In dat geval wordt teruggegaan naar de knoop waar je net vandaan kwam, en wordt een andere aangrenzende knoop daarvan bezocht, en zo verder tot je weer bij v terug bent.
- Aangrenzende knopen kunnen bijvoorbeeld altijd in alfabetische volgorde bezocht worden.
- Een knoop wordt steeds als reeds bezocht gemarkeerd op het moment dat deze voor de eerste keer bekeken wordt.
- Alle knopen die vanuit v bereikbaar zijn worden zo precies één keer bezocht. Voor niet-samenhangende grafen moet bovenstaande telkens herhaald worden vanuit een resterende, nog niet bezochte knoop.
- Depth-first-search kan recursief of met behulp van een stapel worden geïmplementeerd.

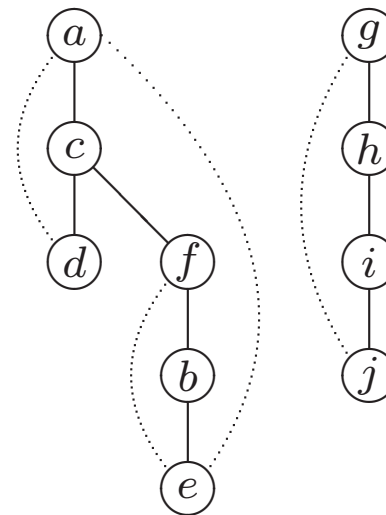
Depth-First Search



Depth-First Search



	$e_{6,2}$	
	$b_{5,3}$	$j_{10,7}$
$d_{3,1}$	$f_{4,4}$	$i_{9,8}$
$c_{2,5}$		$h_{8,9}$
$a_{1,6}$		$g_{7,10}$



ALGORITME DFS (G)

```
// Implementeert DFS wandeling door gegeven graaf
// Invoer: Graaf  $G = (V,E)$ 
// Uitvoer: Graaf  $G$  met zijn knopen genummerd in de volgorde
//          waarin ze bij DFS wandeling voor het eerst worden ontdekt

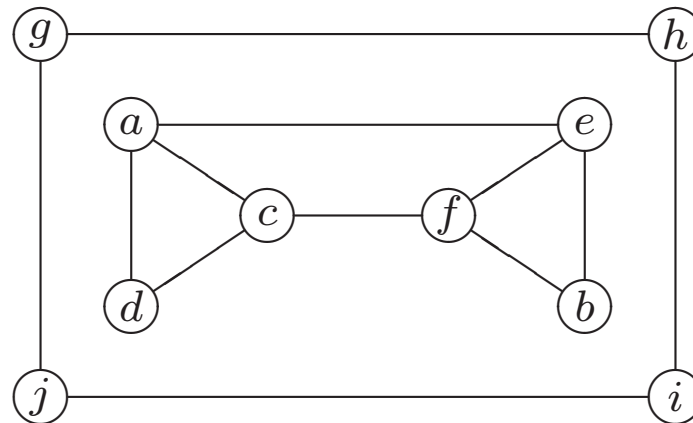
{
  for elke knoop  $v$  in  $V$  do
    mark[v] = 0; // nog niet bezocht
  od
  teller = 0;
  for elke knoop  $v$  in  $V$  do
    if mark[v] == 0 then
      dfs (v);
    fi
  od
}
```

```
dfs (v)
  // Bezoekt recursief alle nog onbezochte knopen die via een pad
  // met v zijn verbonden, en nummert deze in de volgorde waarin
  // ze worden ontdekt, met globale variabele 'teller'

{
  teller ++;
  mark[v] = teller;
  for elke buurknoop w van v do
    if mark[w] == 0 then
      dfs (w);
    fi
  od
}
```

Er is ook niet-recursieve implementatie, met expliciete stapel

Complexiteit Depth-First Search



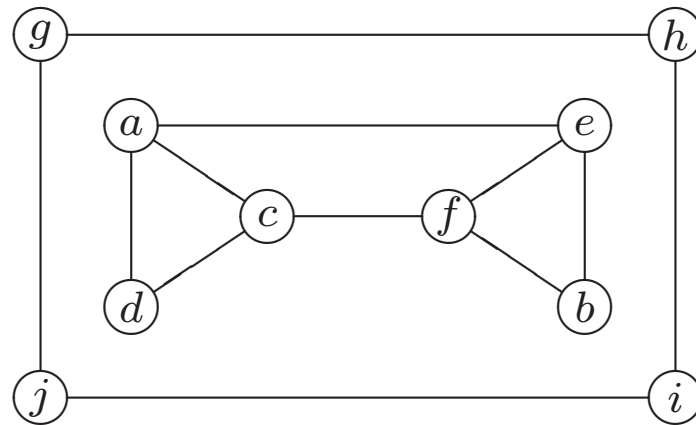
Met adjacency matrix

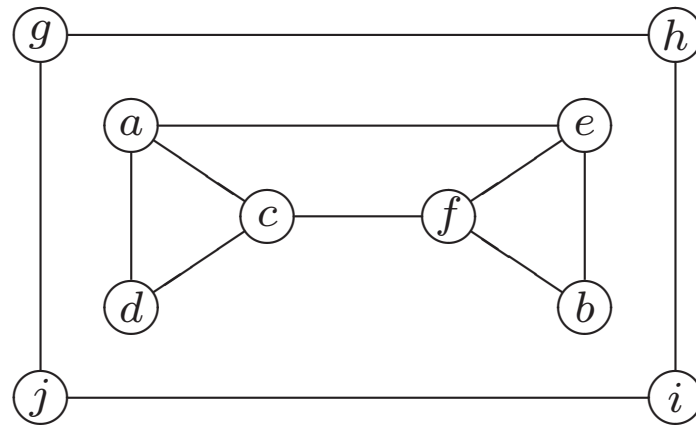
Met adjacency list

Breadth-first-search

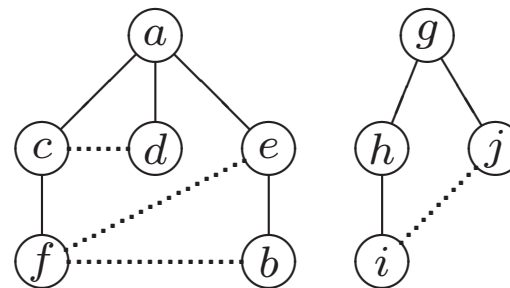
- De wandeling begint in een gegeven knoop v van de graaf.
- Vanuit een zojuist bezochte knoop worden eerst alle aangrenzende -nog onbezochte- knopen bezocht, dan de daaraan grenzende knopen (voor zover nog niet eerder bezocht), en zo verder totdat alle bereikbare knopen bezocht zijn.
- Knopen worden zo bezocht in volgorde van hun afstand vanaf v .
- Aangrenzende knopen kunnen bijvoorbeeld altijd in alfabetische volgorde bezocht worden.
- Bij de implementatie gebruiken we een rij. In de eerste stap wordt v gemarkeerd als bezocht en in de rij gezet. In elke volgende stap wordt de voorste knoop uit de rij gehaald, en worden diens burens gemarkeerd als bezocht en in de rij geplaatst.
- Alle knopen die vanuit v bereikbaar zijn worden zo precies één keer bezocht. Voor niet-samenhangende grafen moet bovenstaande telkens herhaald worden vanuit een resterende, nog niet bezochte knoop.

Breadth-First Search





$a_1 c_2 d_3 e_4 f_5 b_6$
 $g_7 h_8 j_9 i_{10}$



ALGORITME BFS (G)

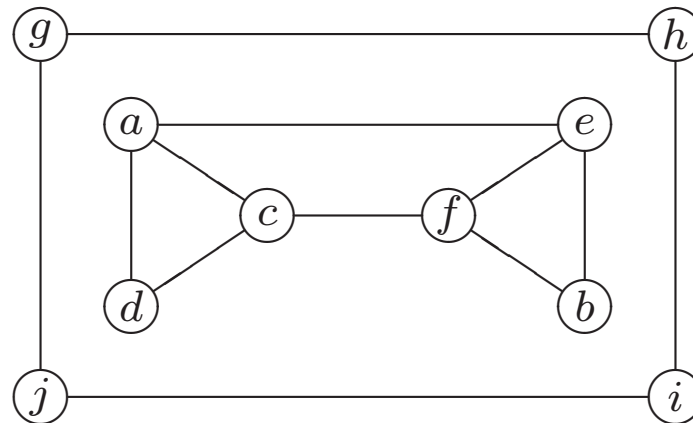
```
// Implementeert BFS wandeling door gegeven graaf
// Invoer: Graaf  $G = (V,E)$ 
// Uitvoer: Graaf  $G$  met zijn knopen genummerd in de volgorde
//          waarin ze bij BFS wandeling worden bezocht
```

```
{
  for elke knoop  $v$  in  $V$  do
    mark[v] = 0; // nog niet bezocht
  od
  teller = 0;
  for elke knoop  $v$  in  $V$  do
    if mark[v] == 0 then
      bfs (v);
    fi
  od
}
```



```
bfs (v)
  // Bezoekt alle nog onbezochte knopen die via een pad
  // met v zijn verbonden, en nummert deze in de volgorde waarin
  // ze worden bezocht, met globale variabele 'teller'
{
  teller ++;
  mark[v] = teller;
  initialiseer queue met v erin;
  while queue is niet leeg do
    for elke buurknoop w van voorste-knoop-in-queue do
      if mark[w] == 0 then
        teller ++;
        mark[w] = teller;
        voeg w toe aan queue; // achteraan
      fi
    od
  verwijder voorste knoop uit queue;
od
}
```

Complexiteit Breadth-First Search

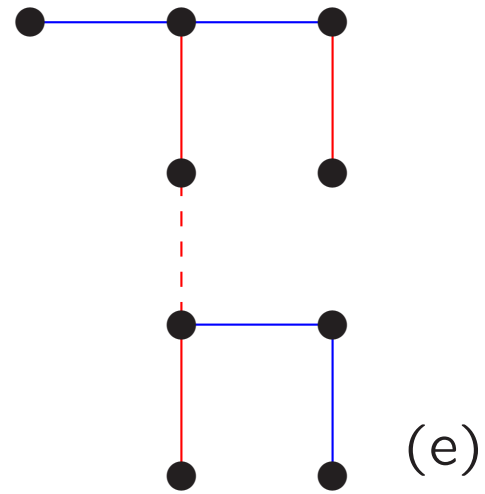


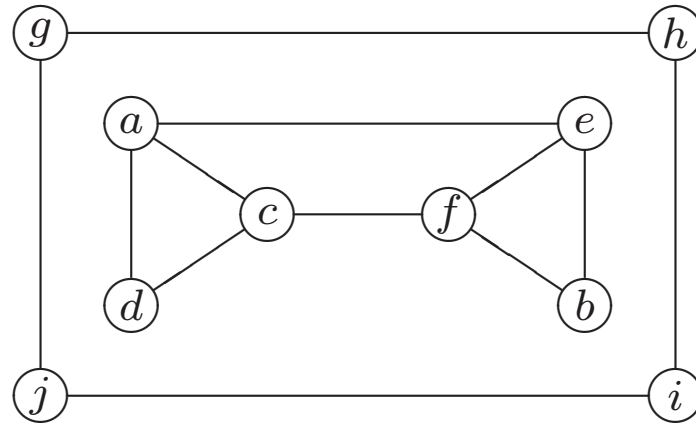
Met adjacency matrix

Met adjacency list

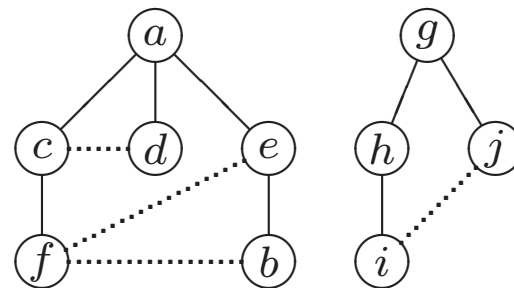
DFS vs BFS

	DFS	BFS
Data structuur	een stapel	een queue
Aantal volgordes knopen	twee volgordes	één volgorde
Soorten takken (onger. grf)	tree en back edges	tree en cross edges
Toepassingen	samenhang, acycliciteit, 'articulation points'	samenhang acycliciteit minimum-tak pad
Complexiteit voor adj. matrix	$\Theta(V ^2)$	$\Theta(V ^2)$
Complexiteit voor adj. list	$\Theta(V + E)$	$\Theta(V + E)$

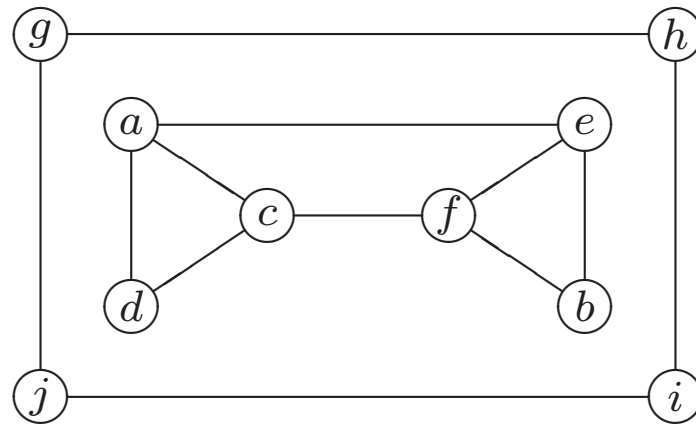




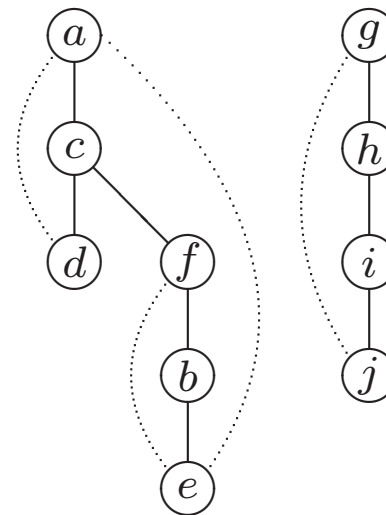
$a_1 c_2 d_3 e_4 f_5 b_6$
 $g_7 h_8 j_9 i_{10}$

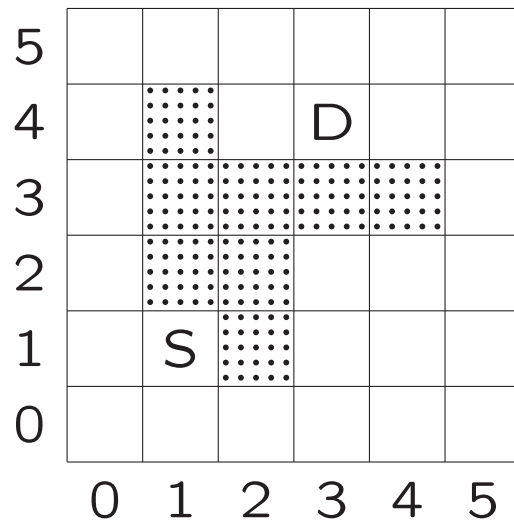


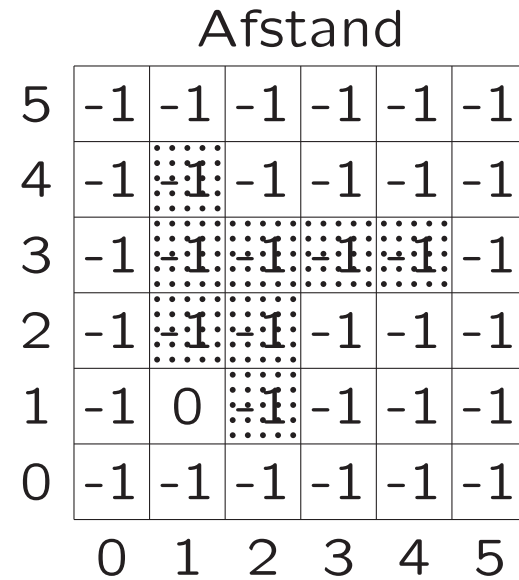
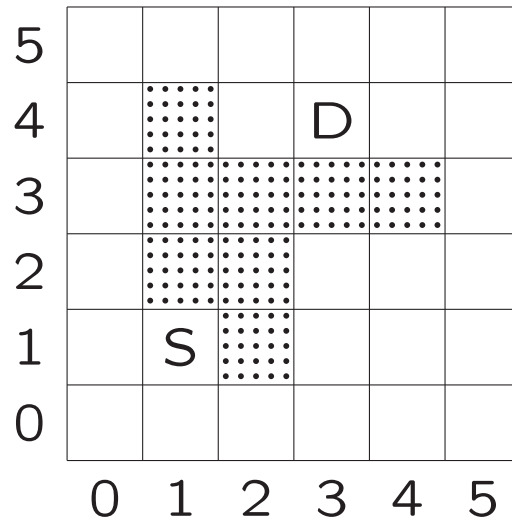
Depth-First Search



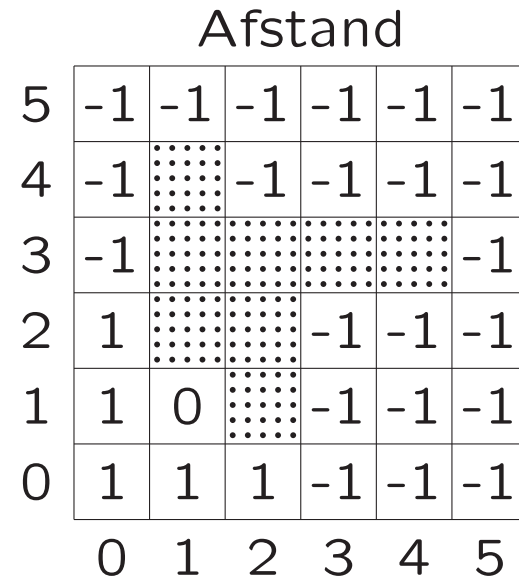
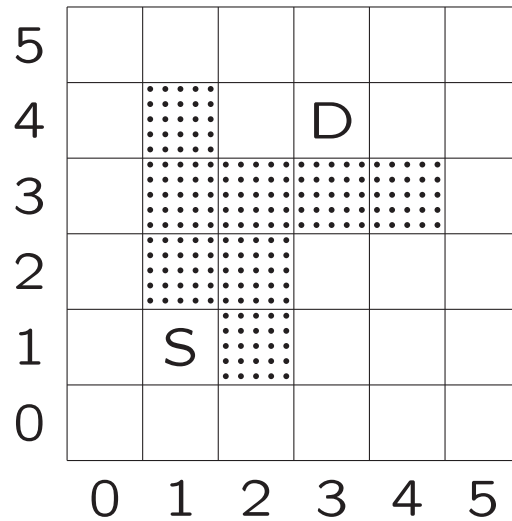
	$e_{6,2}$	
	$b_{5,3}$	$j_{10,7}$
$d_{3,1}$	$f_{4,4}$	$i_{9,8}$
$c_{2,5}$		$h_{8,9}$
$a_{1,6}$		$g_{7,10}$



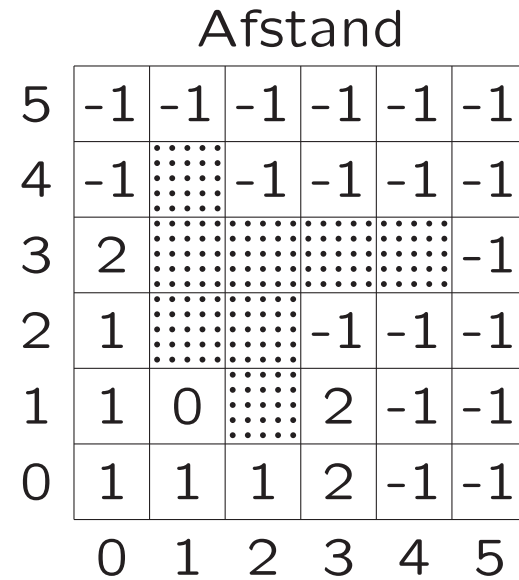
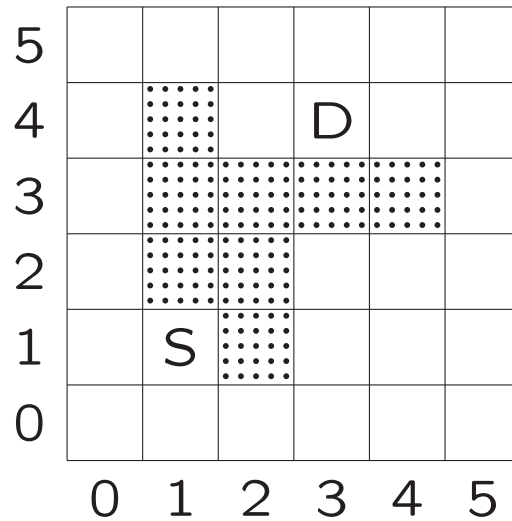




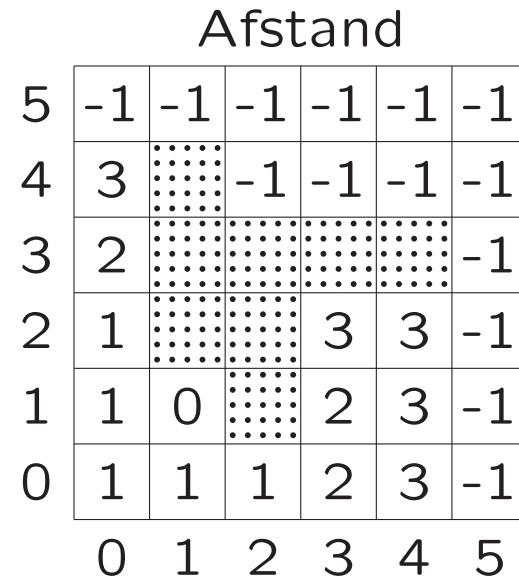
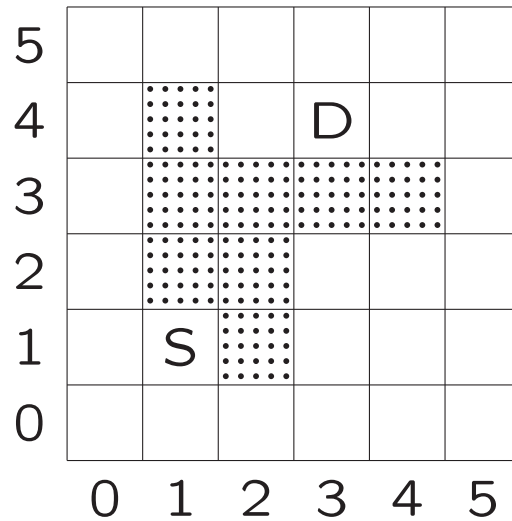
Queue: (1, 1)



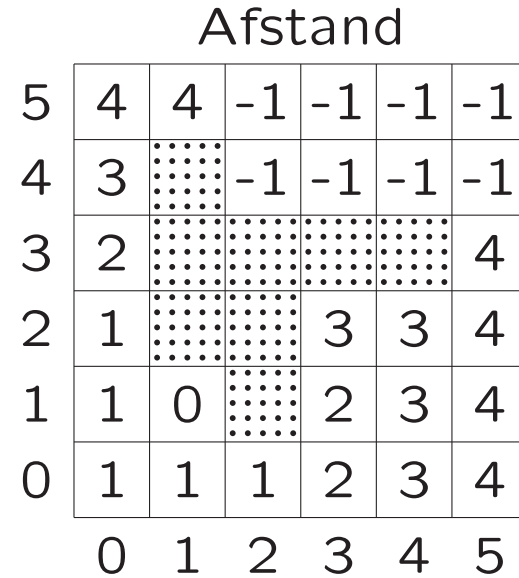
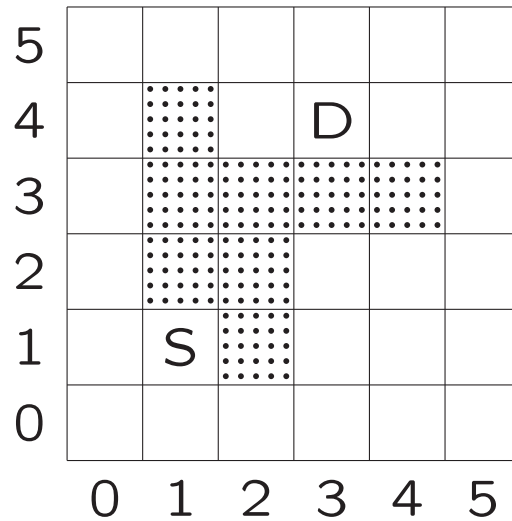
Queue: (0, 2), (0, 1), (0, 0), (1, 0), (2, 0)



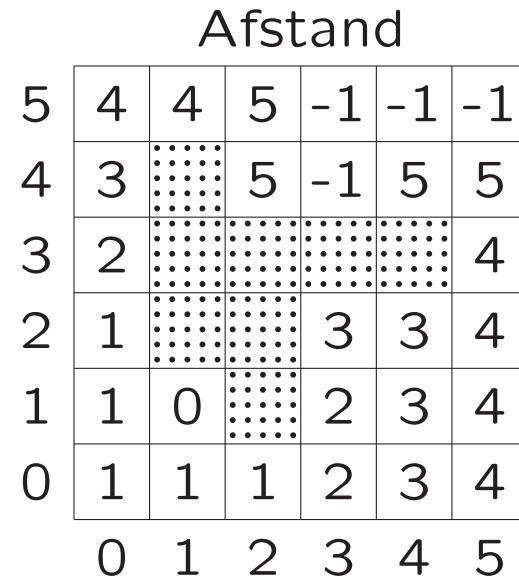
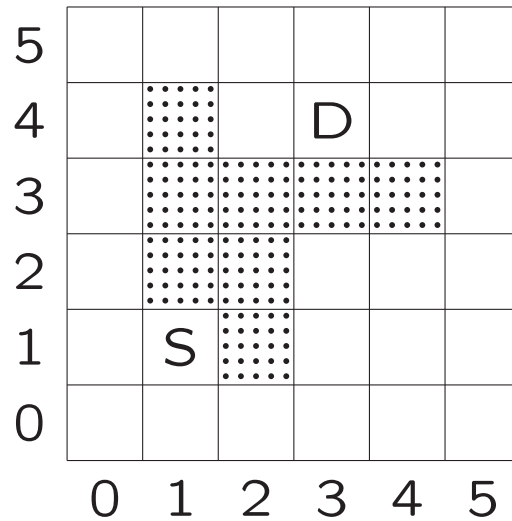
Queue: (0,3), (3,0), (3,1)



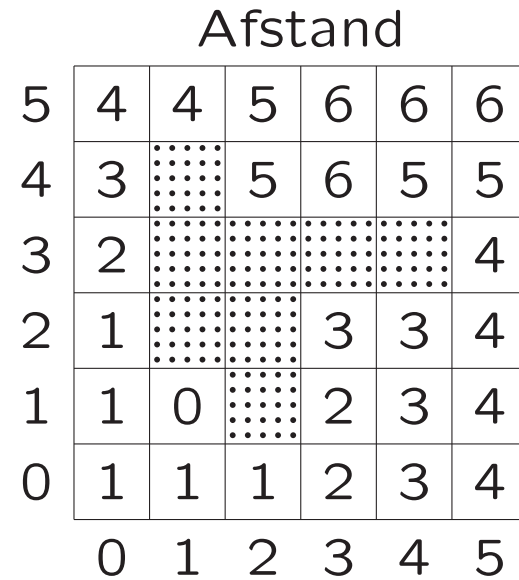
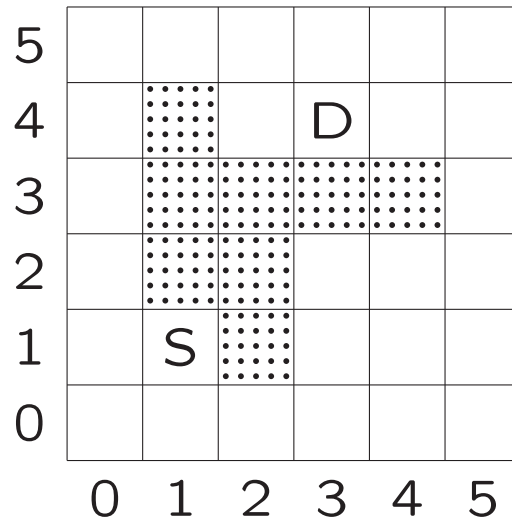
Queue: (0, 4), (4, 0), (4, 1), (4, 2), (3, 2)



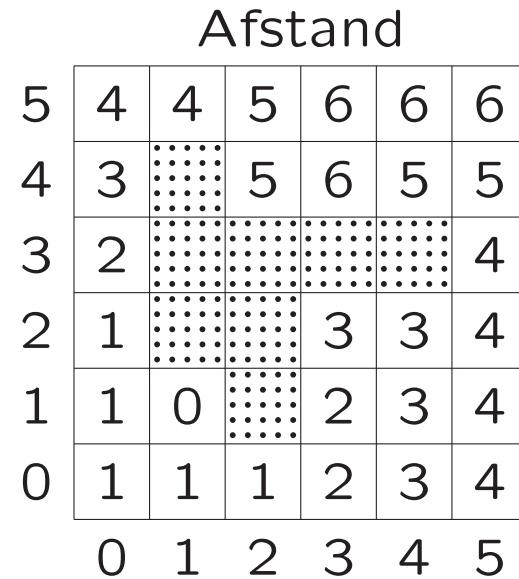
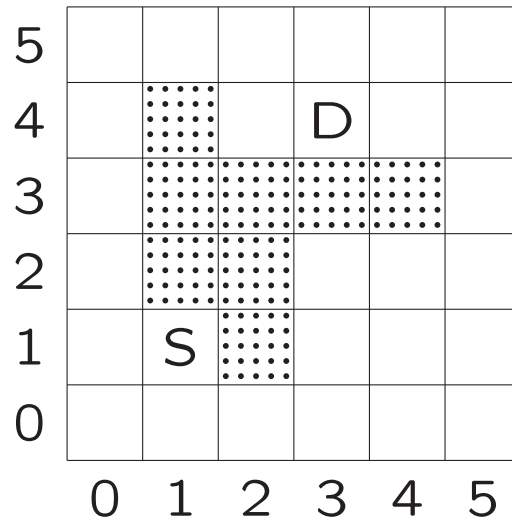
Queue: (1, 5), (0, 5), (5, 0), (5, 1), (5, 2), (5, 3)



Queue: (2, 4), (2, 5), (5, 4), (4, 4)



Queue: (3, 4), (3, 5), (4, 5), (5, 5)



Queue: (3, 4), (3, 5), (4, 5), (5, 5)

Floodfill

Stop zodra D bereikt is

Bepaal route(s) door terug te lopen vanaf D

- **Lezen/leren bij dit college:**
slides
Paragraaf 3.5
- **Practicumbijeenkomst** programmeeropdracht 1:
vanmiddag, 15.30–17.15, computerzalen
- **Volgend college:**
9 maart 2018, 11.00–12.45, De Sitterzaal

Your **opinion** is important.

What do you think of your
study programme?

Fill in the **NSE**

Check your mail, click the link and give
your opinion!

donderdag 8 maart, 15.30 uur, Gorlaeus C1

AlphaGo



<https://is.gd/liacsalphago>