Vijfde college algoritmiek

8 maart 2019

Exhaustive Search

Opdracht 1

- partner?
- deadline: 28 maart / 2 april 2019
- vragenuur: vanmiddag, 15.30 uur

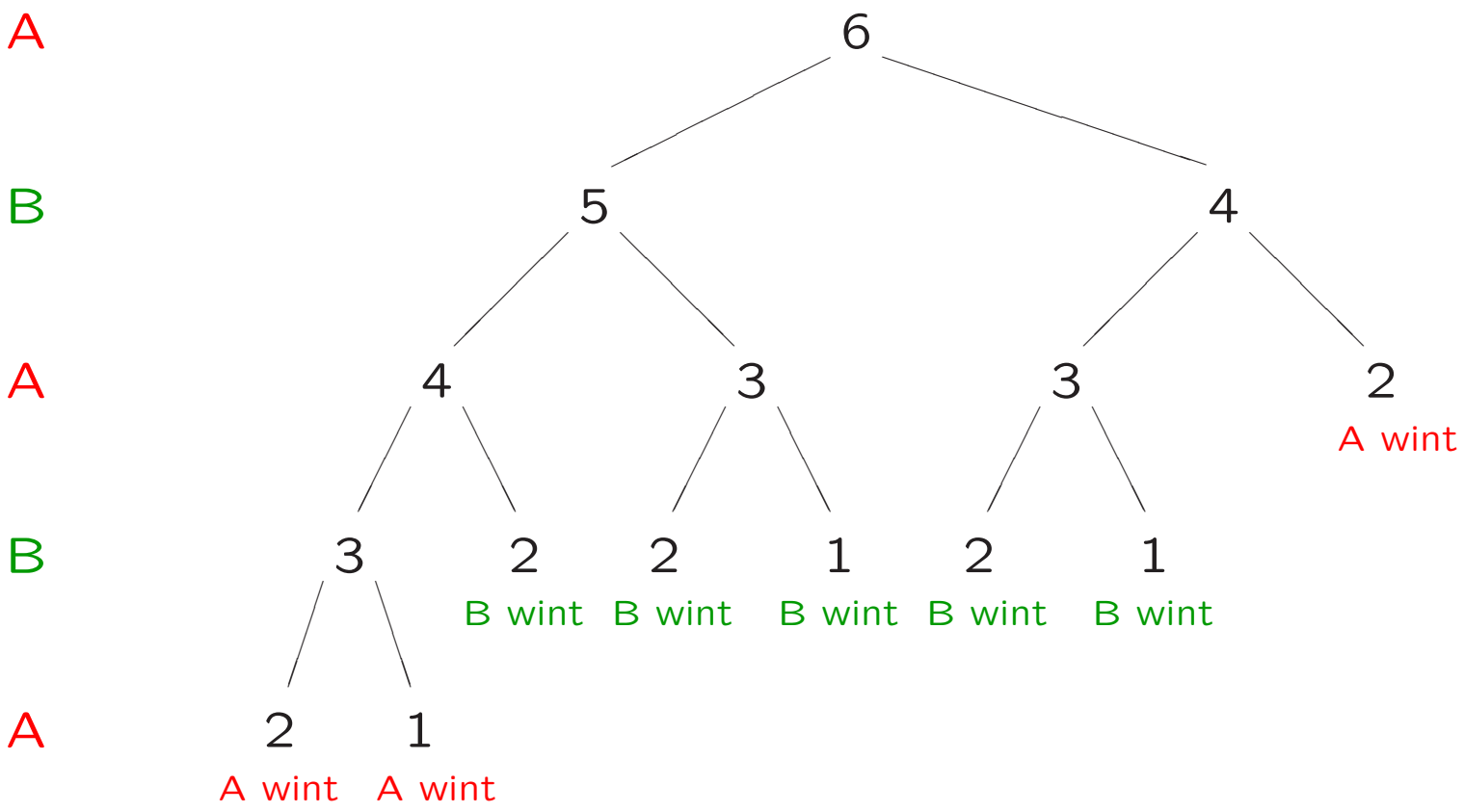
Exhaustive search: brute force benadering voor problemen die te maken hebben met het vinden van een element met een speciale eigenschap binnen een verzameling van bijv. permutaties of deelverzamelingen of toestanden of ...

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)



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

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
```

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,
bijvoorbeeld alle permutaties van de getallen 1 t/m n

. 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

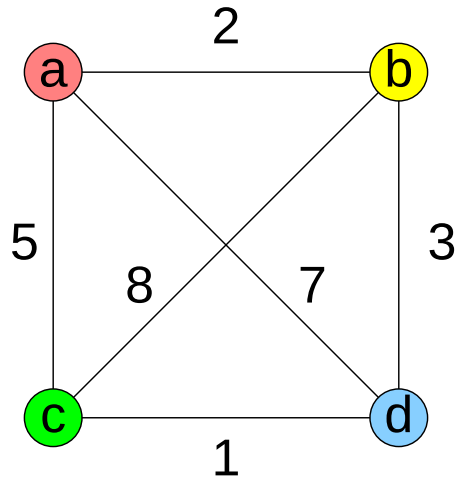
Traveling Salesman Problem (handelsreizigersprobleem)

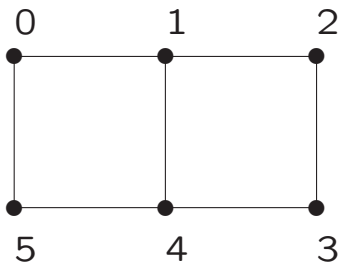
Gegeven n steden waarvan alle onderlinge afstanden bekend zijn.

Gevraagd: de/een kortste route die elke stad precies één keer aandoet, en weer terugkeert in het vertrekpunt.

Ofwel: vind de/een kortste Hamiltonkring in een samenhangende gewogen (volledige) graaf.

Voorbeeld:

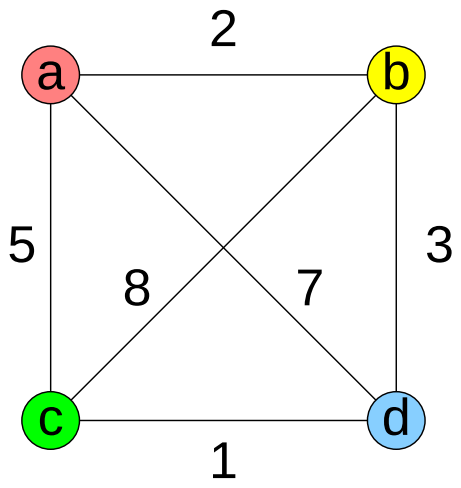




1.

$V = \{0, 1, 2, 3, 4, 5\}$;

$E = \{(0, 1), (1, 2), (2, 3), (3, 4), (4, 5), (5, 0), (4, 1)\}$



Route	**Lengte**
$a \to b \to c \to d \to a$	$2+8+1+7=18$
$a \to b \to d \to c \to a$	$2+3+1+5=11$
$a \to c \to b \to d \to a$	$5+8+3+7=23$
$a \to c \to d \to b \to a$	$5+1+3+2=11$
$a \to d \to b \to c \to a$	$7+3+8+5=23$
$a \to d \to c \to b \to a$	$7+1+8+2=18$

Complexiteit: $\Omega((n-1)!)$,
immers alle $(n-1)!$ mogelijke Hamiltonkringen worden bekeken.

```
void bepaalroute(int n, int pos, int route[], int &best) {
  int lengte;

  if (pos == n) {  // route afsluiten ('eindstand')
    route[pos] = route[0];
    lengte = berekenlengte (n, route)
    if (lengte < best)
      best = lengte;
  }
  else {  // pos < n
    for (int i=1; i<=n; i++)  // 'for alle mogelijke zetten'
      if ('i nog niet in route') {
        route [pos] = i;
        bepaalroute (n, pos+1, route, best);
      }
  }
}

int main ( ) {
  int best = MAXINT;

  route[0] = 1;
  bepaalroute (n, 1, route, best);
  cout << "Kortste afstand: " << best << endl;
}
```

```
void bepaalroute(int n, int pos, int route[], bool gehad[], int &best) {
  int lengte;
  if (pos == n) {  // route afsluiten (‘eindstand’)
    route[pos] = route[0];
    lengte = berekenlengte (n, route)
    if (lengte < best)
      best = lengte;
  }
  else {  // pos < n
    for (int i=1; i<=n; i++)  // ‘for alle mogelijke zetten’
      if (! gehad[i]) {
        route [pos] = i;      gehad[i] = true;
        bepaalroute (n, pos+1, route, gehad, best);
        gehad[i] = false;  // ‘undoezet’
      }
  }
}

int main ( ) {
  bool gehad[n+1];  // TODO: false initialiseren
  int best = MAXINT;
  route[0] = 1;      gehad[1] = true;
  bepaalroute (n, 1, route, gehad, best);
  cout << "Kortste afstand: " << best << endl;
}
```

N	10	50	100	300	1000
$\log_2 N$	3	5	6	8	9
$5N$	50	250	500	1500	5000
$N \cdot \log_2 N$	33	282	665	2469	9966
N^2	100	2500	10.000	90.000	7 cijfers
N^3	1000	125.000	7 cijfers	8 cijfers	10 cijfers
2^N	1024	16 cijfers	31 cijfers	91 cijfers	302 cijfers
$N!$	7 cijfers	65 cijfers	161 cijfers	623 cijfers	onvoorstelbaar
N^N	11 cijfers	85 cijfers	201 cijfers	744 cijfers	onvoorstelbaar

Ter vergelijking:

het aantal protonen in het heelal is een getal met 79 cijfers
het aantal microseconden sinds de Big Bang heeft 24 cijfers

Knapzakprobleem

Gegeven n objecten, met gewicht $w_1, \ldots, w_n$ en waarde $v_1, \ldots, v_n$, en een knapzak met capaciteit W.

Gevraagd: de meest waardevolle deelverzameling der objecten die in de knapzak past (dus met totaalgewicht $\leq W$).

Voorbeeld:

object	gewicht	waarde
1	8	42
2	3	14
3	4	40
4	5	27

knapzakcapaciteit 12

deelverzameling	gewicht	waarde
$\emptyset$	0	0
$\{1\}$	8	42
$\{2\}$	3	14
$\{3\}$	4	40
$\{4\}$	5	27
$\{1,2\}$	11	56
$\{1,3\}$	12	82
$\{1,4\}$	13	te zwaar
$\{2,3\}$	7	54
$\{2,4\}$	8	41
$\{3,4\}$	9	67
$\{1,2,3\}$	15	te zwaar
$\{1,2,4\}$	16	te zwaar
$\{1,3,4\}$	17	te zwaar
$\{2,3,4\}$	12	81
$\{1,2,3,4\}$	20	te zwaar

Complexiteit: $\Omega(2^n)$,
immers alle 2^n deelverzamelingen van n objecten worden bekeken.

Hoe? Zien we later: stap voor stap opbouwen

- steeds alle mogelijkheden proberen voor het eerstvolgende object
- steeds het volgende object wel-of-niet kiezen

Assignmentproblem (toewijzingsprobleem)

Gegeven n personen en n taken (jobs). Persoon i kan taak j doen voor $\texttt{kosten}[i][j]$ euro.

Gevraagd: de/een toewijzing van de personen aan de jobs (één persoon per job en één job per persoon) met minimale kosten.

Voorbeeld:

	job 1	job 2	job 3	job 4
Anna	9	2	7	8
Bob	6	4	3	7
Carla	5	8	1	8
David	7	6	9	4

$n = 4$

	job 1	job 2	job 3	job 4
Anna	9	2	7	8
Bob	6	4	3	7
Carla	5	8	1	8
David	7	6	9	4

$n = 4$

```
1,2,3,4 -> 9+4+1+4 = 18        2,3,1,4 -> ..        3,4,1,2 -> ..
1,2,4,3 -> 9+4+8+9 = 30        2,3,4,1 -> ..        3,4,2,1 -> ..
1,3,2,4 -> 9+3+8+4 = 24        2,4,1,3 -> ..        4,1,2,3 -> ..
1,3,4,2 -> 9+3+8+6 = 26        2,4,3,1 -> ..        4,1,3,2 -> ..
1,4,2,3 -> 9+7+8+9 = 33        3,1,2,4 -> ..        4,2,1,3 -> ..
1,4,3,2 -> 9+7+1+6 = 23        3,1,4,2 -> ..        4,2,3,1 -> ..
2,1,3,4 -> 2+6+1+4 = 13        3,2,1,4 -> ..        4,3,1,2 -> ..
2,1,4,3 -> 2+6+8+9 = 25        3,2,4,1 -> ..        4,3,2,1 -> ..
```

De goedkoopste toewijzing is hier `2,1,3,4`, met kosten 13.

Complexiteit: $\Omega(n!)$,
immers alle $n!$ mogelijke toewijzingen worden bekeken.

Eindconclusie Exhaustive Search

* Veel exhaustive search algoritmen werken **alleen voor kleine probleeminstanties** in acceptabele tijd

* Voor veel problemen zijn er veel efficiëntere algoritmen bekend (Eulerkring, kortste paden, toewijzingsprobleem)

* Voor andere problemen is exhaustive search (of varianten daarop) in essentie de enig bekende oplossing (handelsreizigersprobleem, knapzakprobleem)

Graafwandelingen

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

- **Graafwandelingen:**

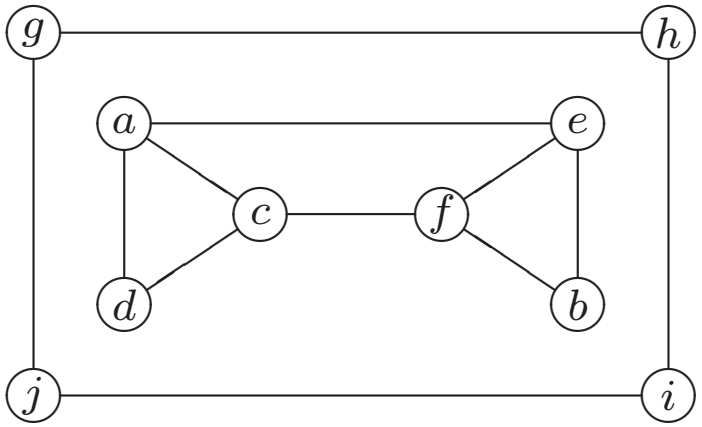
  1. Depth-first-search: te vergelijken met WLR-wandeling bij bomen

  2. Breadth-first-search: te vergelijken met nivo-orde wandeling bij bomen

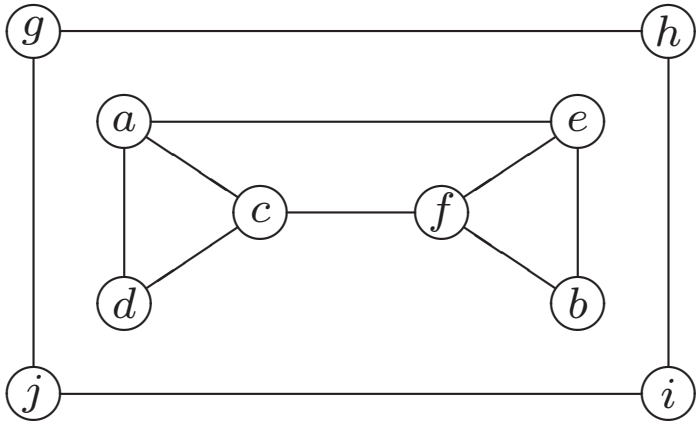
Depth-first-search

- 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.

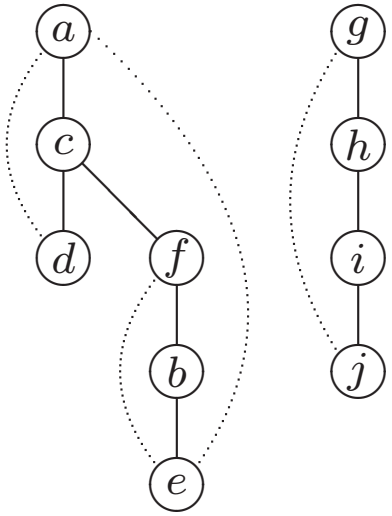
Depth-First Search



Depth-First Search



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

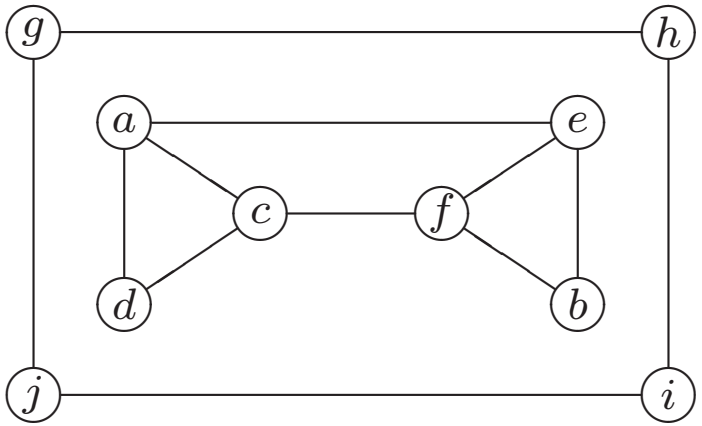


```
ALGORITME DFS (G)
    // Implementeert DFS wandeling door gegeven graaf
    // Invoer: Graaf G = (V,E)
    // Uitvoer: Graaf G met zijn knopen genummerd in de volgorde
    //          waarin ze bij DFS wandeling voor het eerst worden ontdekt

{
    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



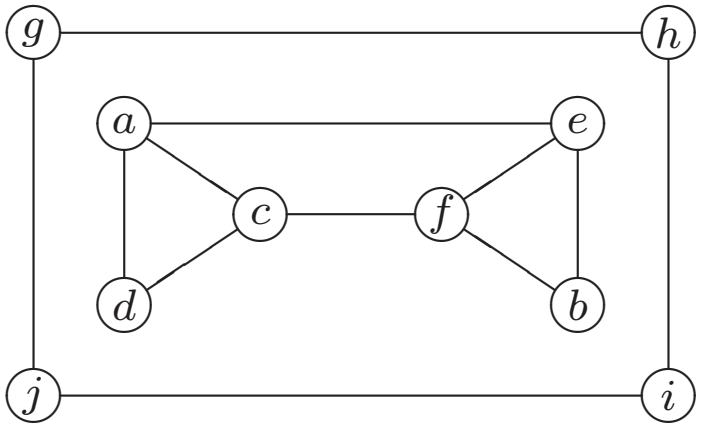
Met adjacency matrix

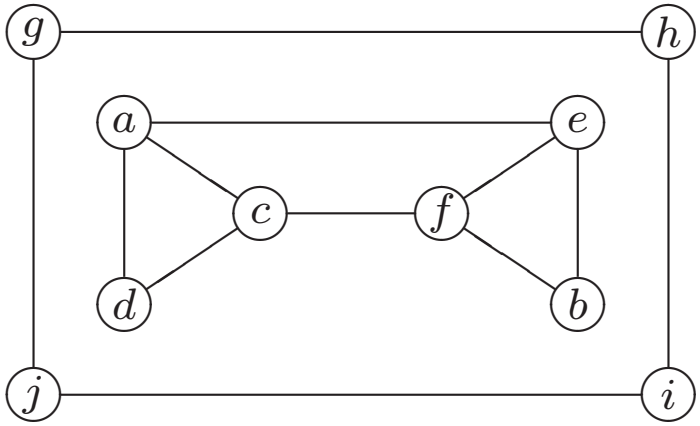
Met adjacency list

Breadth-first-search

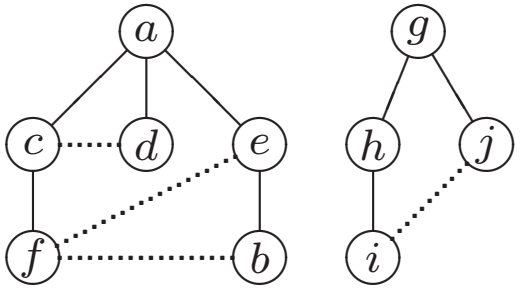
- De wandeling begint in een gegeven knoop v van de graaf.
- Vanuit een zojuist bezochte knoop worden eerst alle aangrenzende -nog onbezochte- knopen bezocht, dan de daaraan grenzende knopen (voor zover nog niet eerder bezocht), en zo verder totdat alle bereikbare knopen bezocht zijn.
- Knopen worden 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 buren 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}

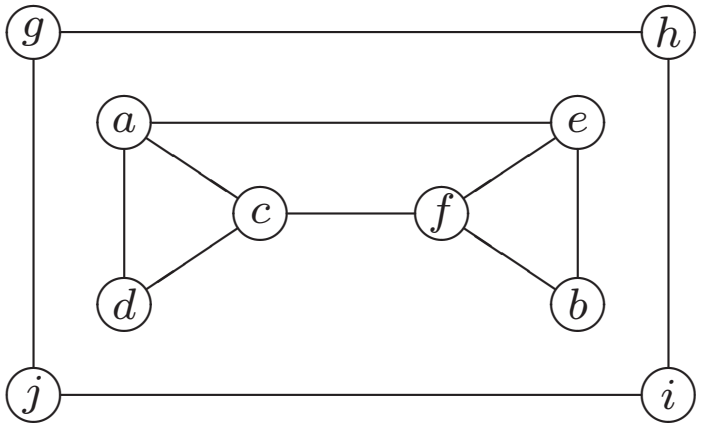


```
ALGORITME BFS (G)
    // Implementeert BFS wandeling door gegeven graaf
    // Invoer: Graaf G = (V,E)
    // Uitvoer: Graaf G met zijn knopen genummerd in de volgorde
    //          waarin ze bij BFS wandeling worden bezocht

{
    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
}
```

Pseudo-code BFS (vervolg)

```
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

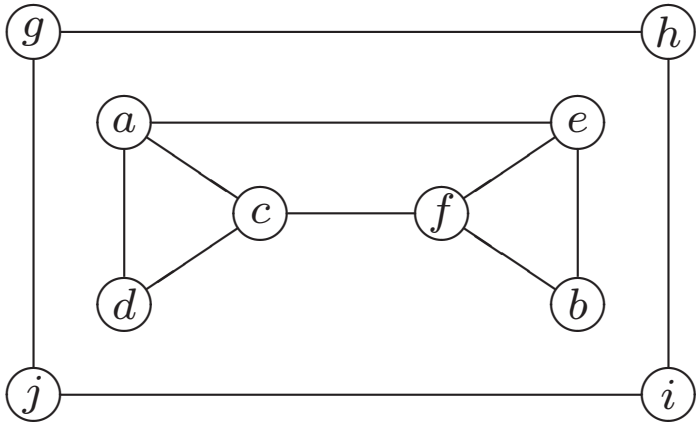


Met adjacency matrix

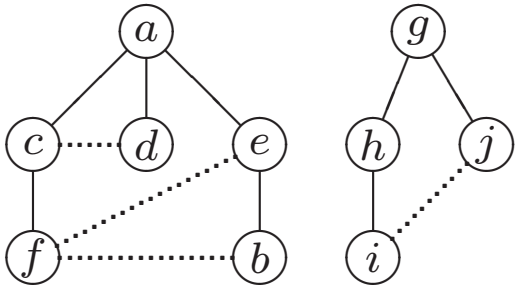
Met adjacency list

DFS vs BFS

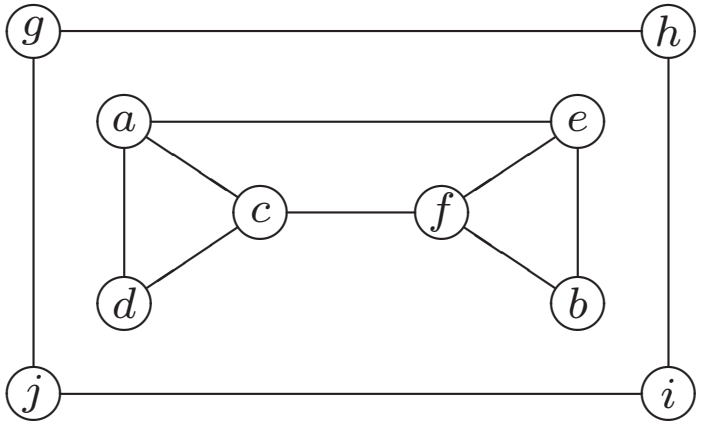
	DFS	**BFS**
Data structuur	een stapel	een queue
Aantal volgordes knopen	twee volgordes	één volgorde
Soorten takken (onger. grf)	tree en back edges	tree en cross edges
Toepassingen	samenhang, acycliciteit, 'articulation points'	samenhang acycliciteit minimum-tak pad
Complexiteit voor adj. matrix	$\Theta(\|V\|^2)$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



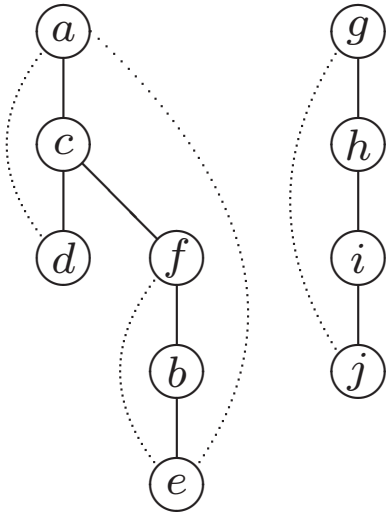
a_1 c_2 d_3 e_4 f_5 b_6
g_7 h_8 j_9 i_{10}

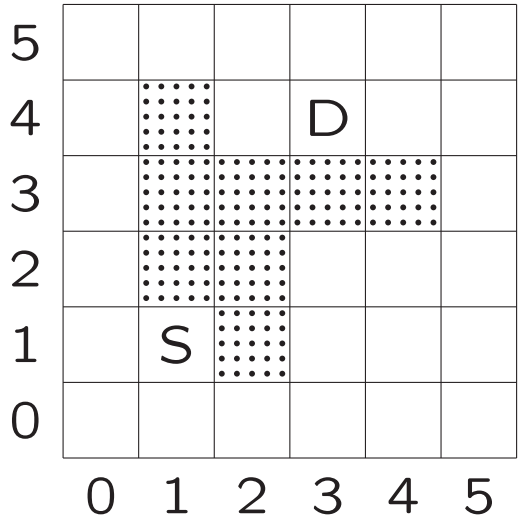


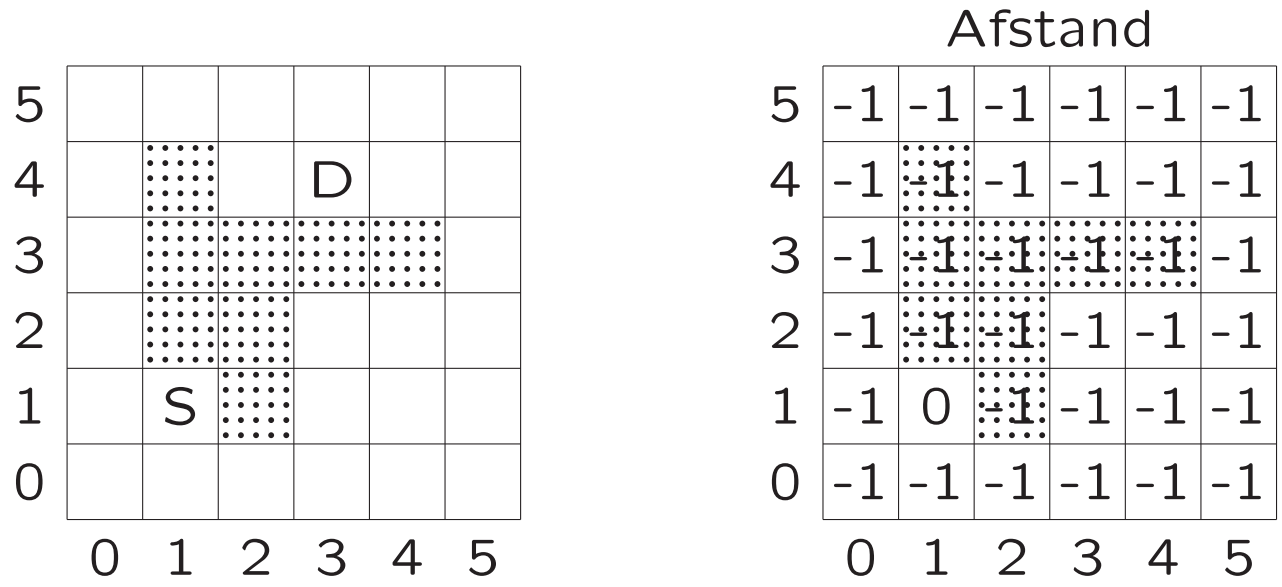
Depth-First Search



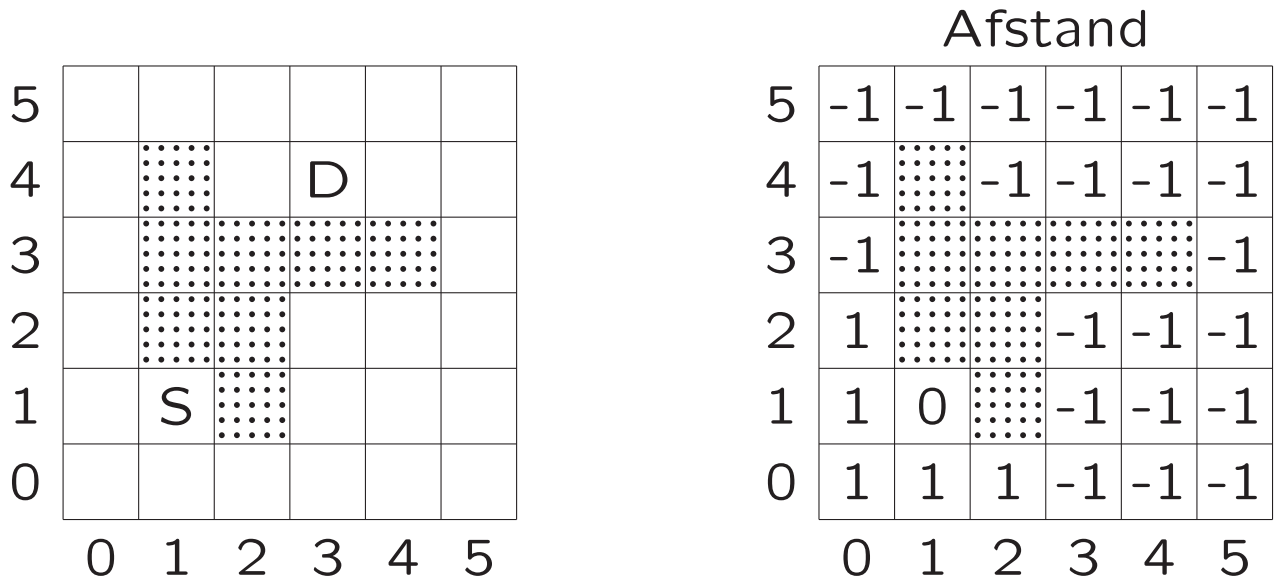
	$e_{6,2}$	
	$b_{5,3}$	$j_{10,7}$
$d_{3,1}$	$f_{4,4}$	$i_{9,8}$
$c_{2,5}$		$h_{8,9}$
$a_{1,6}$		$g_{7,10}$



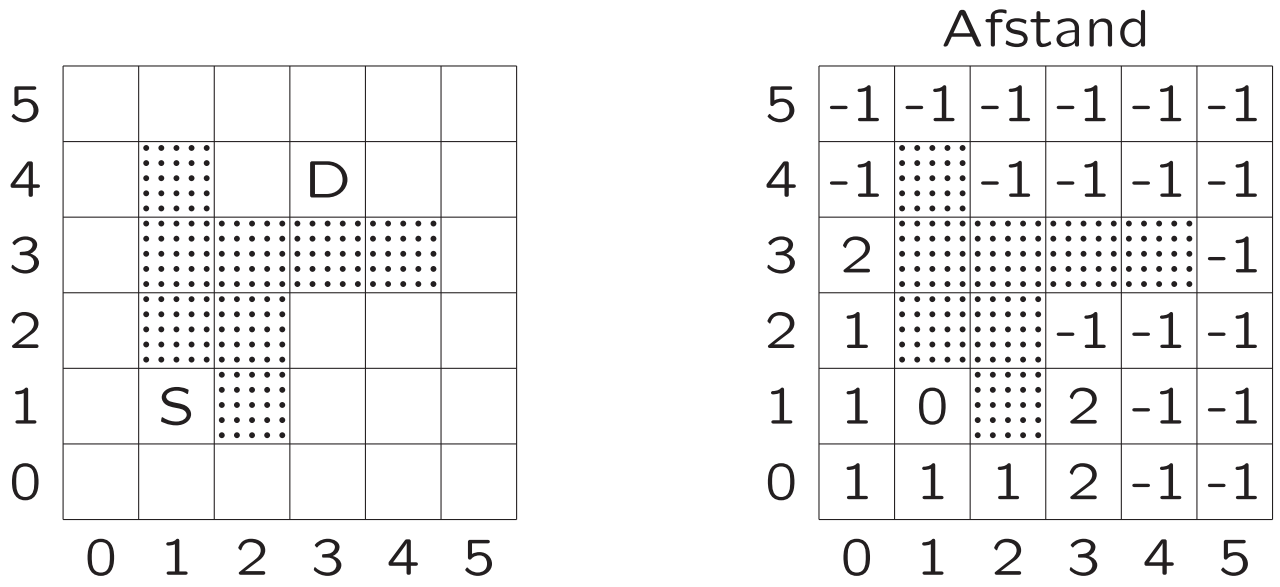

5
4
3
2
1
0
D
S
0 1 2 3 4 5


Afstand
5
4
3
2
1
0
S
D
0 1 2 3 4 5
-1 -1 -1 -1 -1 -1
-1 1 -1 -1 -1 -1
-1 1 1 1 1 -1
-1 1 1 -1 -1 -1
-1 0 1 -1 -1 -1
-1 -1 -1 -1 -1 -1

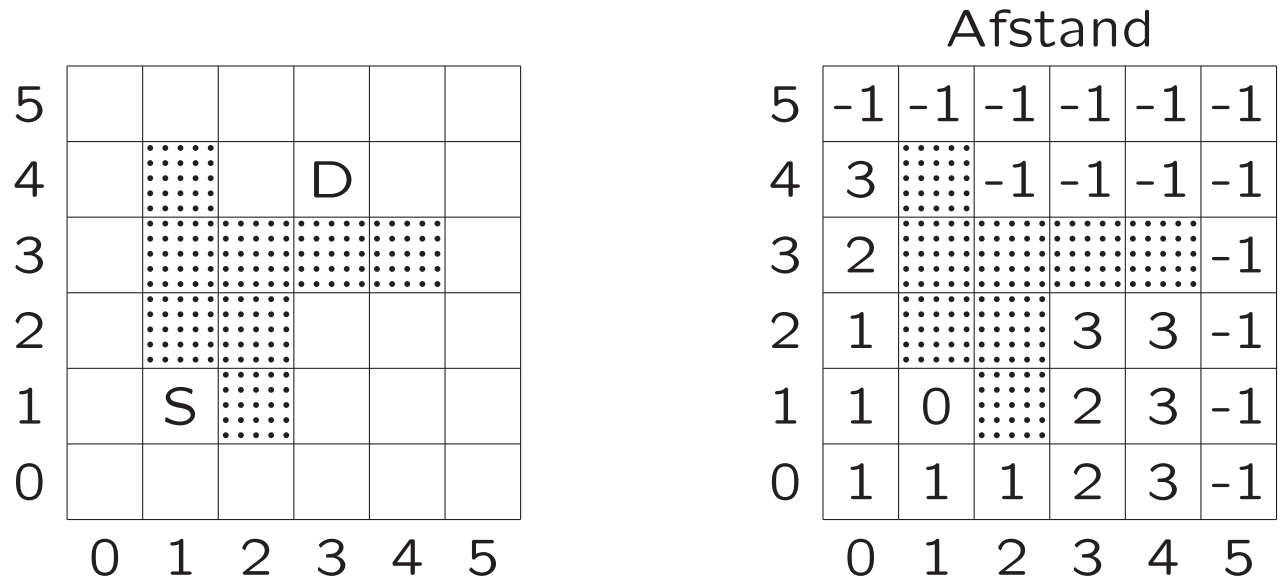
Queue: $(1, 1)$



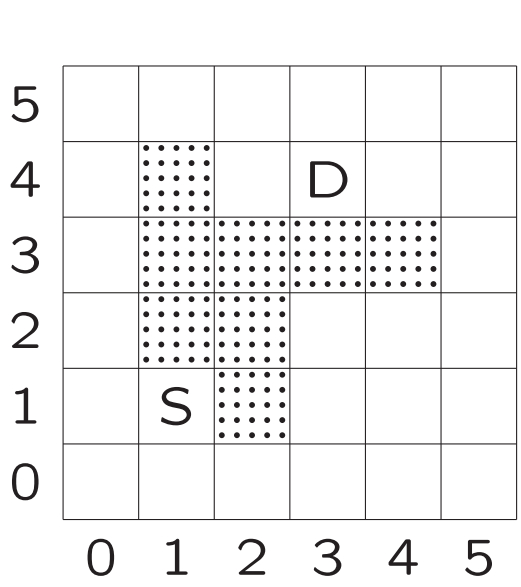
Queue: $(0,2),(0,1),(0,0),(1,0),(2,0)$



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



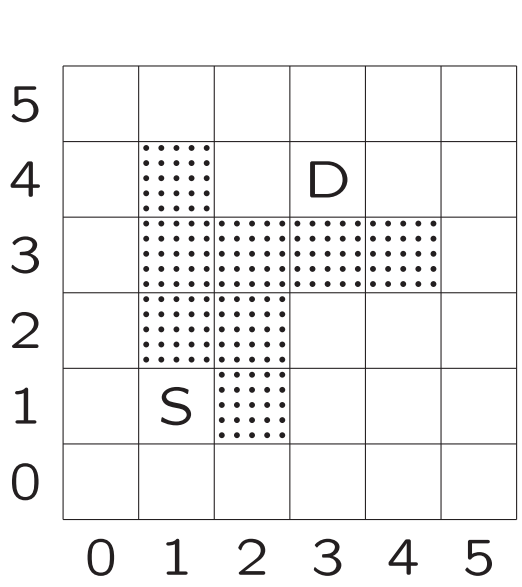
Queue: $(0,4),(4,0),(4,1),(4,2),(3,2)$



Afstand

5	4	4	-1	-1	-1	-1
4	3		-1	-1	-1	-1
3	2					4
2	1			3	3	4
1	1	0		2	3	4
0	1	1	1	2	3	4
	0	1	2	3	4	5

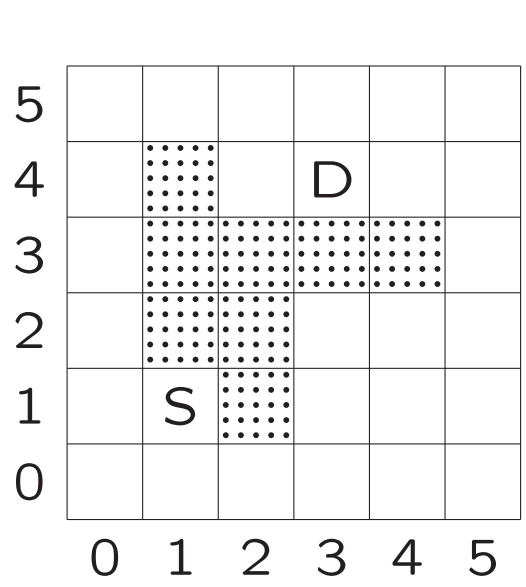
Queue: $(1,5),(0,5),(5,0),(5,1),(5,2),(5,3)$



Afstand

	0	1	2	3	4	5
5	4	4	5	-1	-1	-1
4	3		5	-1	5	5
3	2					4
2	1			3	3	4
1	1	0		2	3	4
0	1	1	1	2	3	4

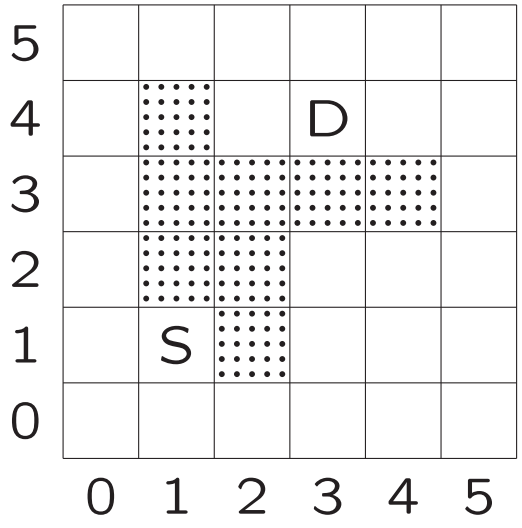
Queue: $(2,4),(2,5),(5,4),(4,4)$



Afstand

5	4	4	5	6	6	6
4	3		5	6	5	5
3	2					4
2	1			3	3	4
1	1	0		2	3	4
0	1	1	1	2	3	4
	0	1	2	3	4	5

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



Afstand

5	4	4	5	6	6	6
4	3		5	6	5	5
3	2					4
2	1			3	3	4
1	1	0		2	3	4
0	1	1	1	2	3	4
	0	1	2	3	4	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:**
  3.4, 3.5, slides

- **Werkcollege** brute force en exhaustive search

  vanmiddag, 13.30–15.15, in deze zaal

- **Opgaven:**

  zie `http://www.liacs.leidenuniv.nl/~vlietrvan1/algoritmiek/`

- **Vragenuur bij programmeeropdracht 1:**

  vanmiddag, 15.30-17.15, in kleine computerzalen Snellius

- **Volgend college:**

  vrijdag 22 maart 2019, 11.00–12.45, De Sitterzaal

NSE 2019

Jouw mening is belangrijk!

Wat?

- **De *N*ationale *S*tudenten *E*nquête**

Waarom?

- **Omdat je graag je mening wilt geven & wilt meehelpen je opleiding te verbeteren**
- **Omdat bij 25% respons studenten koeken krijgen**
- **Omdat er per ingevulde enquête 25 cent wordt gedoneerd aan stichting vluchteling-student (UAF)**
- **Omdat de studievereniging met de hoogste respons een gratis sportactiviteit mag organiseren**

Hoe?

- **Via de persoonlijke link in de uitnodigingsmail**
- **Link kwijt? Vul je uMail-ibox e-mailadres in op www.nse.nl**

Gemakkelijk via je telefoon!