Vijfde college algoritmiek

5 maart 2025

Exhaustive Search

Backtracking

Opdracht 1

- partner?
- deadline: dinsdag 1 april, 23.59 uur
- practicumbijeenkomst: dinsdag, 09.00-10.45 uur

Werkcollege: vrijdag, 11.00-12.45 uur of liever 13.15-15.00 uur?

- brute force: optimale score/zet
- heuristiek voor ‘goede’ zet: grootste cluster
- heuristiek voor ‘goede’ zet: Monte Carlo
- experimenten: heuristieken tegen optimaal en tegen elkaar
- skeletprogramma: `TODO` vs `aqualin.h`

Mogelijke verdeling werk:

- Week 1: leesInSpel, drukAf, constructor, doeZet, unDoeZet
- Week 2: optScore, bepaalZetGrootsteCluster, bepaalZetMonteCarlo, speelUitScore
- Week 3: doeExperiment1/2/3, verslag

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 ...

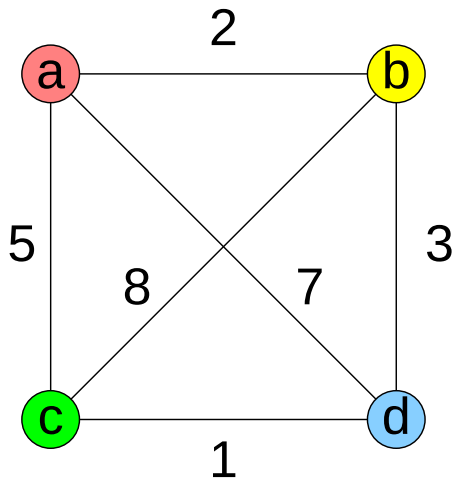
Methode:

. 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



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.

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

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

  ...
  ...
  ...
  ...
  ...
  ...
  ...
  ...
  ...
  ...
  ...
  ...
  ...
}

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[], 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
    ...
    ...
    ...
    ...
    ...
  }
}

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

Code voor TSP met ES, 4

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

Complexiteit. . .

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:**

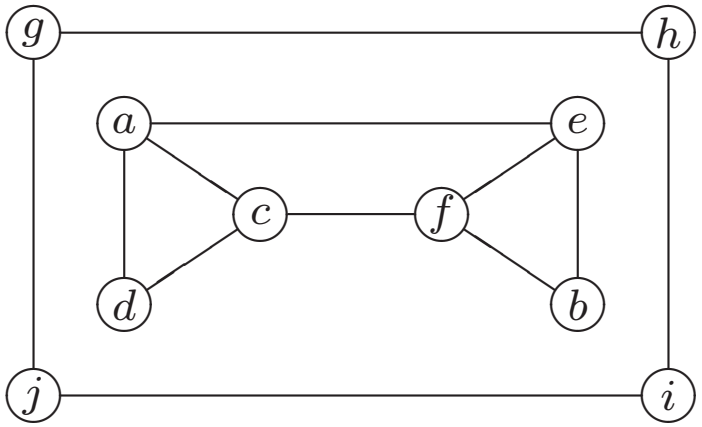
  1. Depth-first-search: te vergelijken met WLR-wandeling bij bomen

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

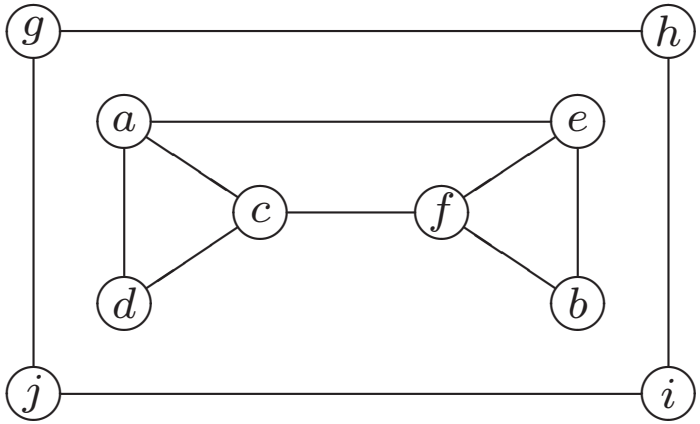
Depth-first-search

- 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.

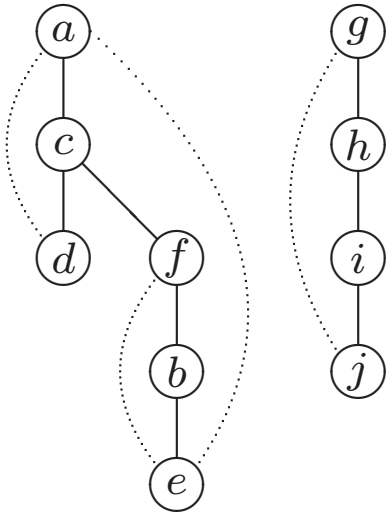
Depth-First Search



Depth-First Search



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



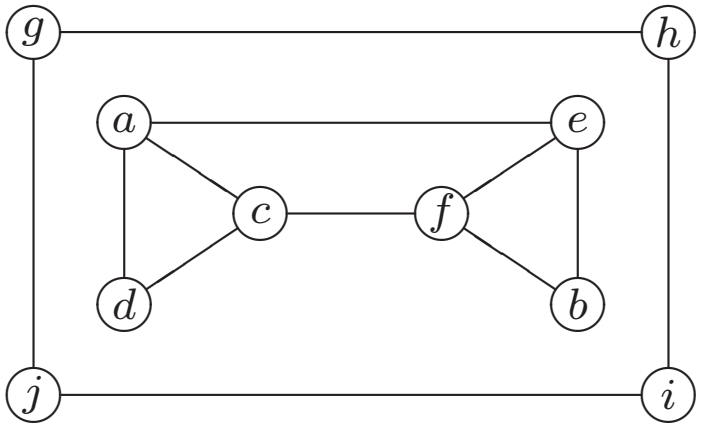
```
ALGORITME DFS (G)
    // Implementeert DFS wandeling door gegeven graaf
    // Invoer: Graaf G = (V,E)
    // Uitvoer: Graaf G met zijn knopen genummerd in de volgorde
    //          waarin ze bij DFS wandeling voor het eerst worden ontdekt

{
    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
}
```

Pseudo-code DFS (vervolg)

```
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

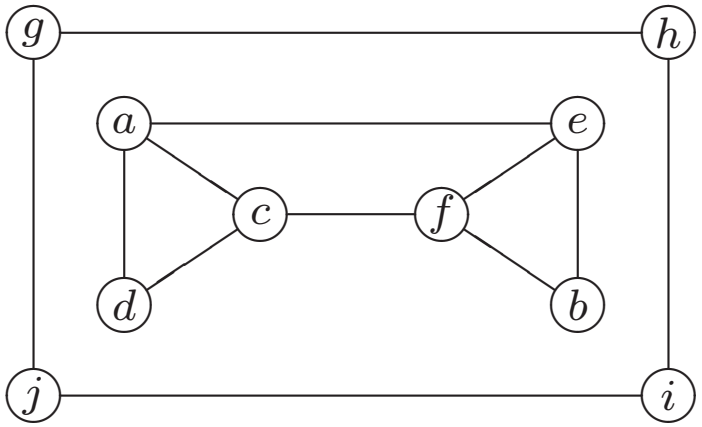
```
dfs (v)
    // Bezoekt recursief alle nog onbezochte knopen die via een pad
    // met v zijn verbonden, en nummert deze in de volgorde waarin
    // ze worden ontdekt, met globale variabele 'teller'

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

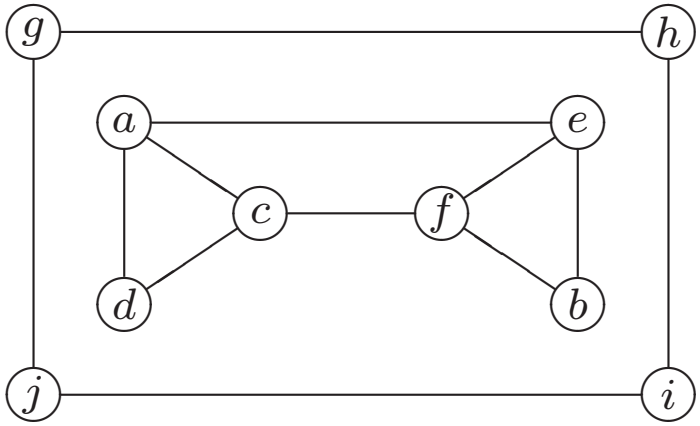
Complexiteit, bij adjacency matrix. . .

Breadth-first-search

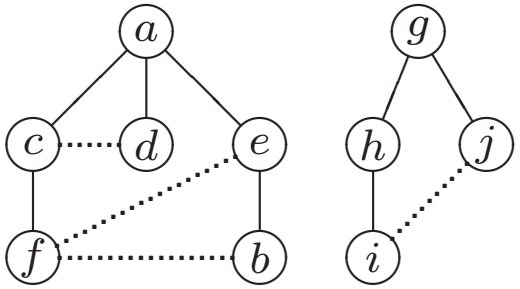
- De wandeling begint in een gegeven knoop v van de graaf.
- Vanuit een zojuist bezochte knoop worden eerst alle aangrenzende -nog onbezochte- knopen bezocht, dan de daaraan grenzende knopen (voor zover nog niet eerder bezocht), en zo verder totdat alle bereikbare knopen bezocht zijn.
- Knopen worden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

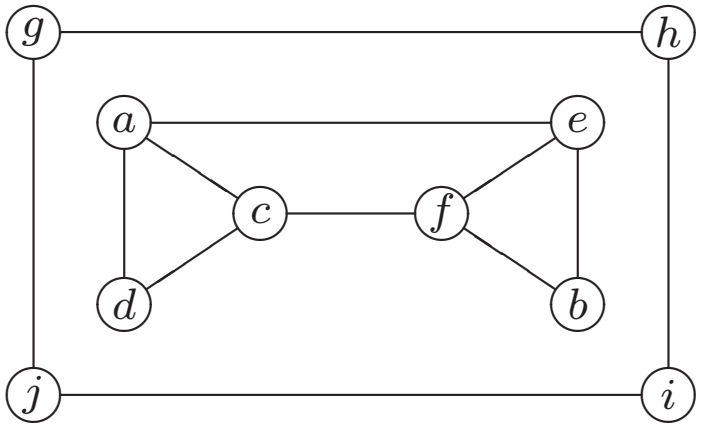


```
ALGORITME BFS (G)
    // Implementeert BFS wandeling door gegeven graaf
    // Invoer: Graaf G = (V,E)
    // Uitvoer: Graaf G met zijn knopen genummerd in de volgorde
    //          waarin ze bij BFS wandeling worden bezocht

{
    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

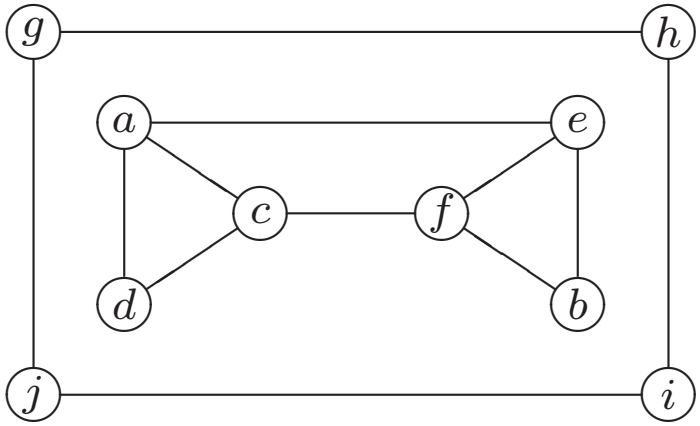


Met adjacency matrix

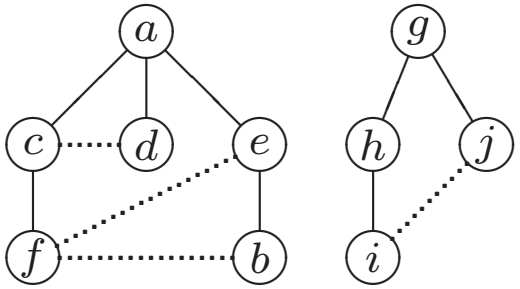
Met adjacency list

DFS vs BFS

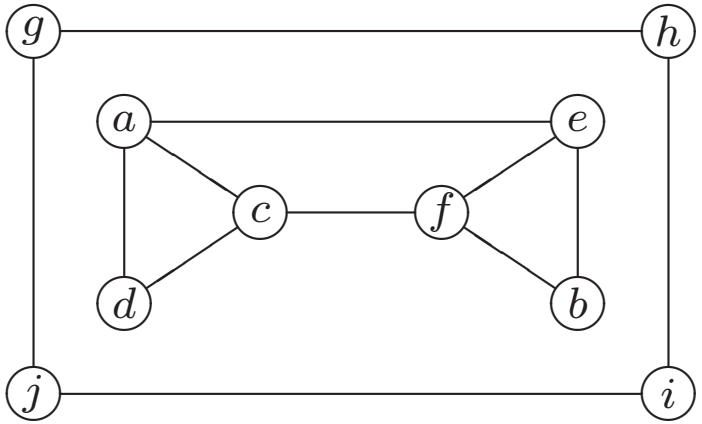
	DFS	**BFS**
Data structuur	een stapel	een queue
Aantal volgordes knopen	twee volgordes	één volgorde
Soorten takken (onger. grf)	tree en back edges	tree en cross edges
Toepassingen	samenhang, acycliciteit, 'articulation points'	samenhang acycliciteit minimum-tak pad
Complexiteit voor adj. matrix	$\Theta(\|V\|^2)$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



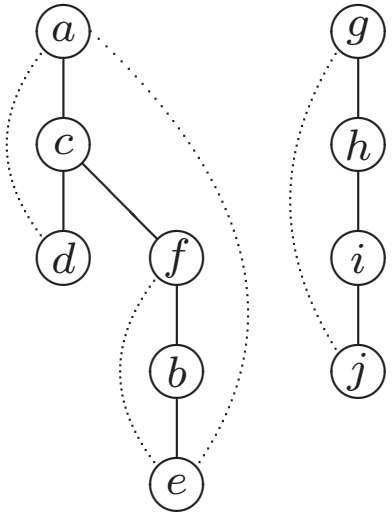
a_1 c_2 d_3 e_4 f_5 b_6
g_7 h_8 j_9 i_{10}

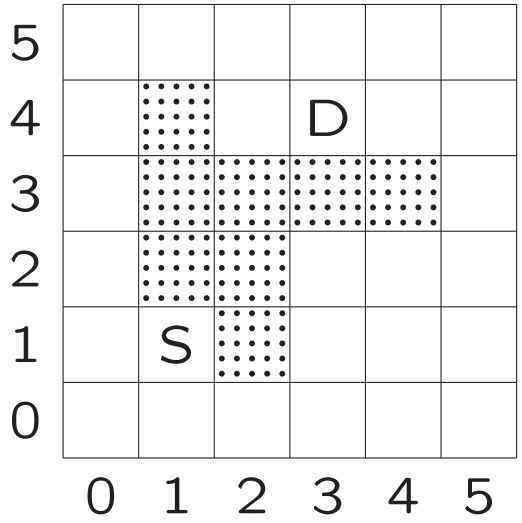


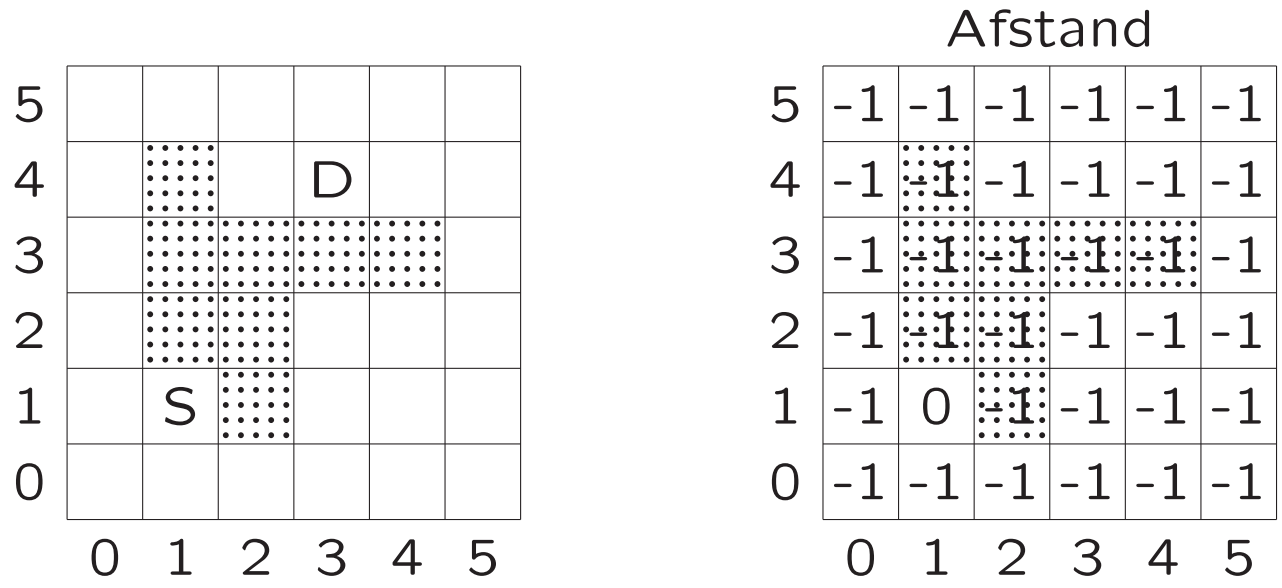
Depth-First Search



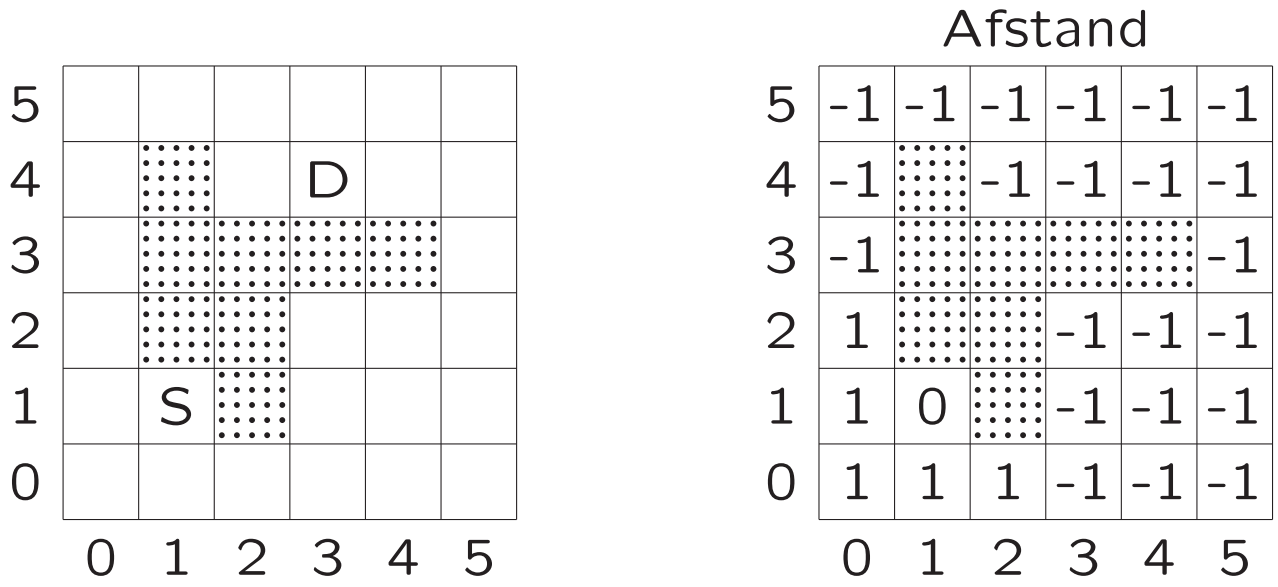
	$e_{6,2}$	
	$b_{5,3}$	$j_{10,7}$
$d_{3,1}$	$f_{4,4}$	$i_{9,8}$
$c_{2,5}$		$h_{8,9}$
$a_{1,6}$		$g_{7,10}$



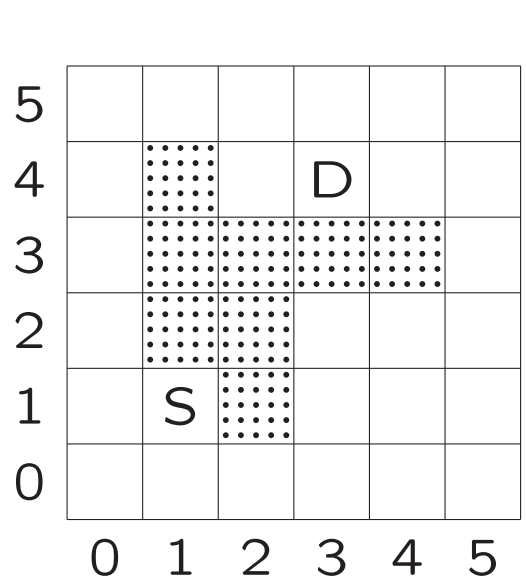

5
4
3
2
1
0
S
D
0 1 2 3 4 5


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: $(1, 1)$



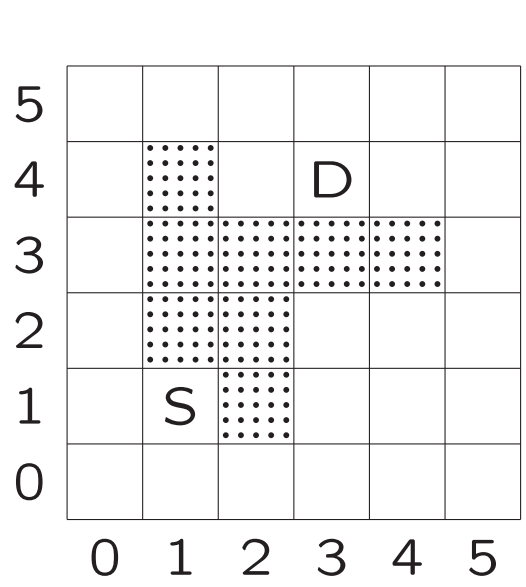
Queue: $(0,2),(0,1),(0,0),(1,0),(2,0)$



Afstand

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

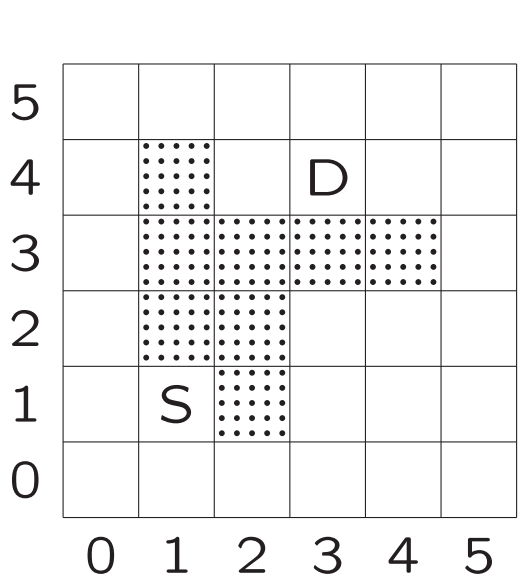
Queue: $(0,3),(3,0),(3,1)$



Afstand

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

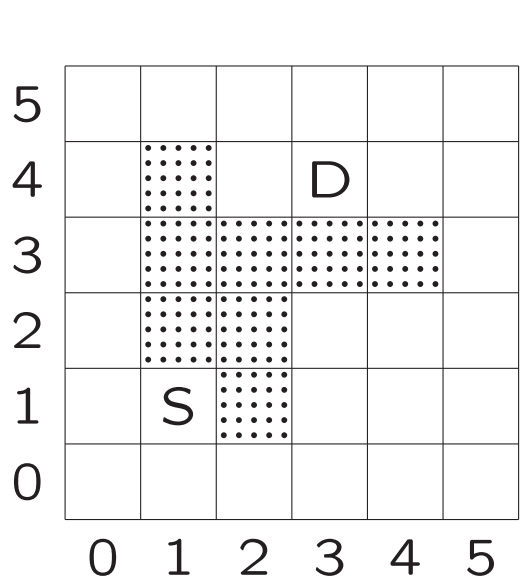
Queue: $(0,4),(4,0),(4,1),(4,2),(3,2)$



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

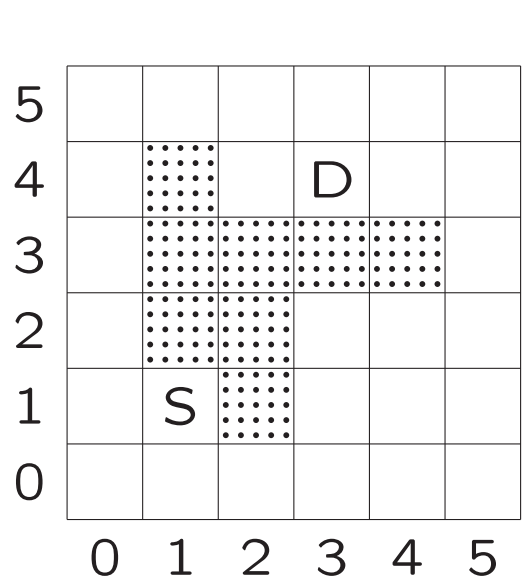
Queue: $(1,5),(0,5),(5,0),(5,1),(5,2),(5,3)$



Afstand

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
	0	1	2	3	4	5

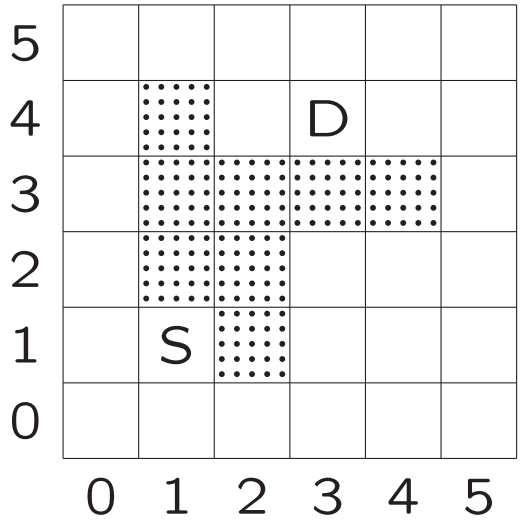
Queue: $(2,4),(2,5),(5,4),(4,4)$



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

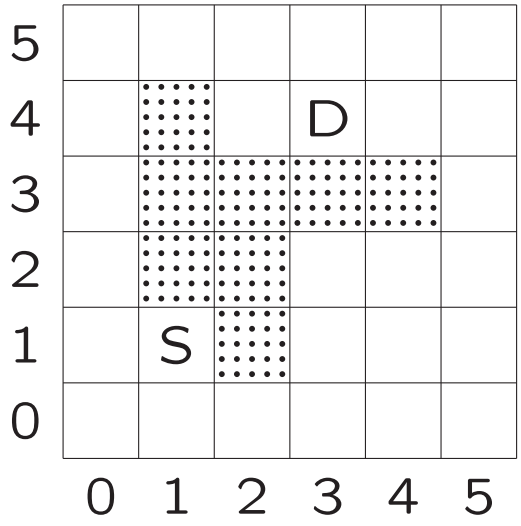
5	4	4	5	6	6	6
4	3		5	6	5	5
3	2					4
2	1			3	3	4
1	1	0		2	3	4
0	1	1	1	2	3	4
	0	1	2	3	4	5

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

Floodfill

Stop zodra D bereikt is

Bepaal route(s) door . . .



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

Bij veel problemen gaat het erom een element met een speciale eigenschap te vinden binnen een ruimte die exponentieel groeit als functie van de invoergrootte. Dan wordt meestal backtracking gebruikt als goed alternatief voor ES.

Exhaustive search genereert alle kandidaatoplossingen en haalt daar het speciale element tussenuit.

Backtracking

- bouwt kandidaatoplossingen component voor component op,
- kijkt al tijdens de constructie of de deeloplossing nog tot een oplossing kan leiden en
- zo niet, breidt dan de deeloplossing niet verder uit

Op deze manier spaar je soms veel werk uit en kun je dus grotere probleeminstanties oplossen.

Backtracking versus exhaustive search

Exhaustive search bekijkt *alle* volledige kandidaatoplossingen.
Backtracking controleert telkens van deeloplossingen of ze nog aan de eisen/restricties voldoen; zo niet, dan weet je zeker dat alle uitbreidingen van deze oplossing ook niet voldoen, dus die hoef je dan niet meer expliciet te bekijken.

Basisidee backtracking

- bouw een oplossing stap voor stap op en controleer steeds of de deeloplossing in conflict komt met de restricties (en nog wel tot een oplossing kan leiden)

- op elk moment kun je kiezen uit een aantal mogelijke vervolgstappen; maak een keuze en ga langs die weg verder met het opbouwen van de oplossing

- als een keuze op niets uitloopt, herzie je deze keuze en probeer je een andere mogelijkheid

Vergelijk

- het vinden van de uitgang in een doolhof: loop steeds verder en als je bij het zoeken vastloopt, ga terug op je pad om het laatste open alternatief te proberen. Straks!

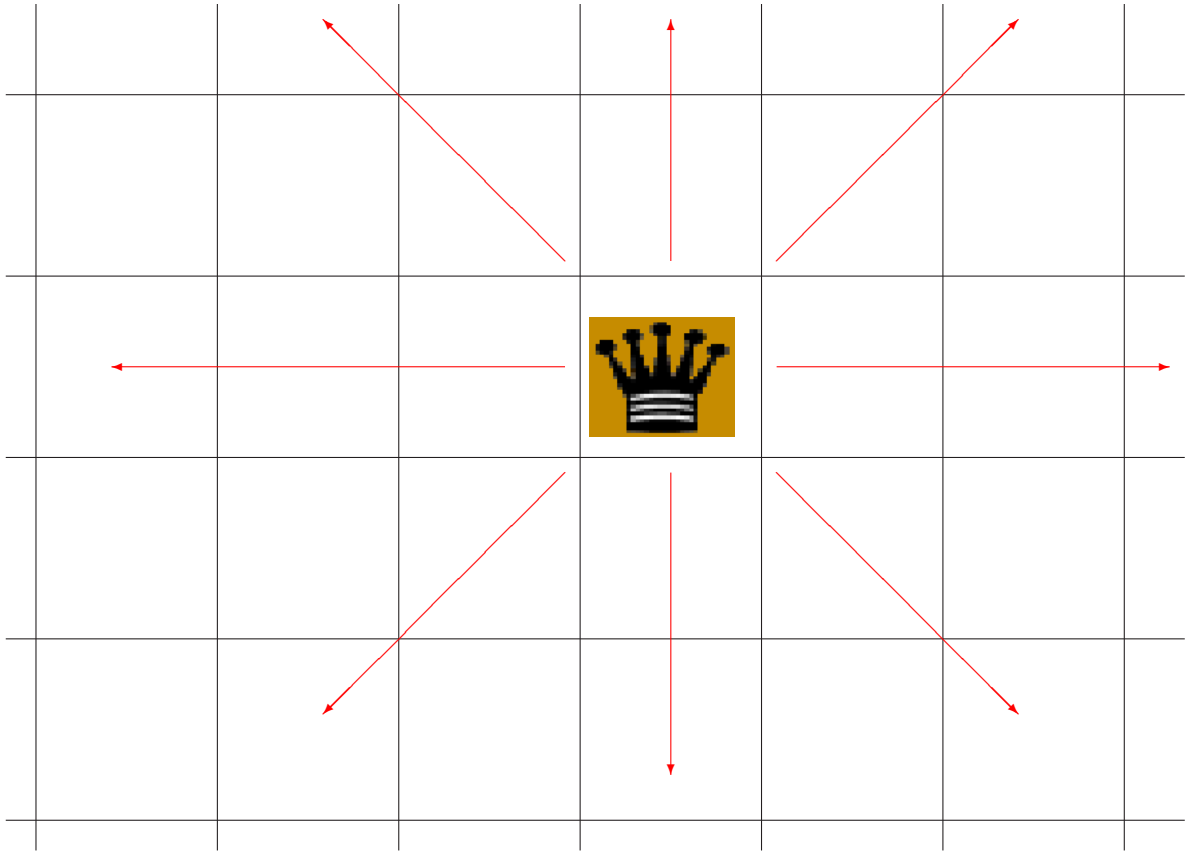
Het acht koninginnenprobleem luidt als volgt:

1. Kun je 8 dames (koninginnen) op een 8 bij 8 schaakbord zetten zonder dat zij elkaar aanvallen (= in één keer kunnen slaan)?
2. Zo ja, op hoeveel verschillende manieren kan dat?

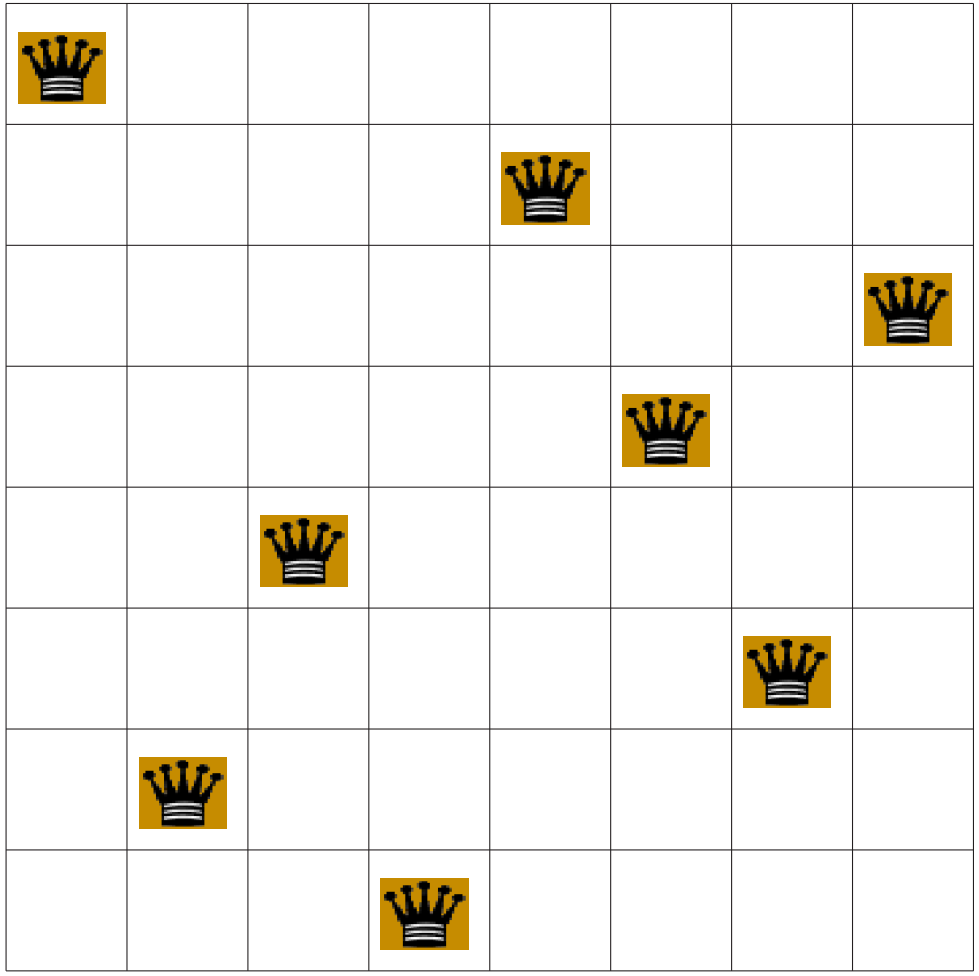
En nu **algemeen**:

Op hoeveel manieren kun je n dames op een n bij n bord plaatsen zonder dat zij elkaar aanvallen?

Een dame kan in één zet een willekeurig aantal vakjes naar links, rechts, onder, boven of diagonaal schuiven.

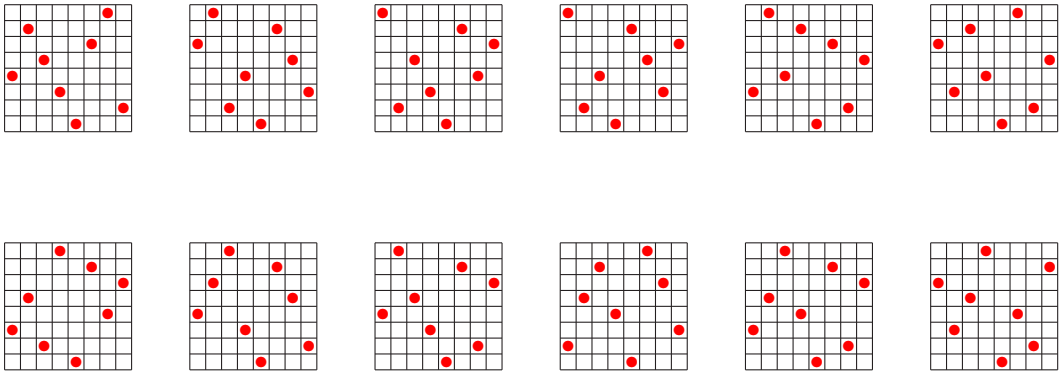


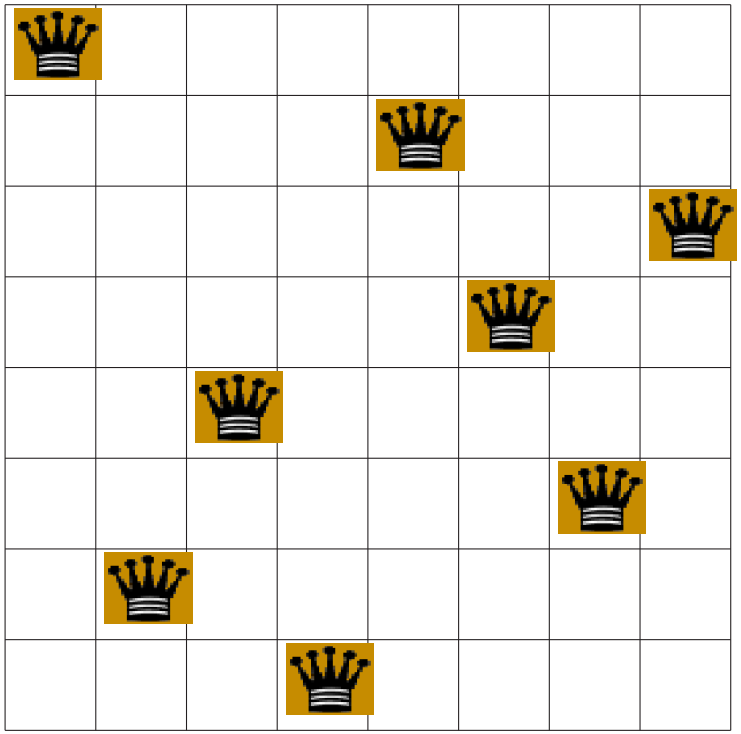
Een oplossing is onderstaande configuratie:



dit correspondeert met
de volgende permutatie:
1 5 8 6 3 7 2 4

Op het 8×8 schaakbord zijn er 92 oplossingen. In essentie zijn er 12 verschillende oplossingen, waaruit je door draaien en spiegelen (8 mogelijkheden) ze allemaal kunt maken. Er is één wat meer symmetrische oplossing.





Aantallen oplossingen

n	aantal	echt aantal	$n!$
1	1	1	1
2	0	0	2
3	0	0	6
4	2	1	24
5	10	2	120
6	4	1	720
7	40	6	5040
8	92	12	40.320
9	352	46	362.880
10	724	92	3.628.800
11	2680	341	39.916.800
12	14.200	1787	479.001.600
13	73.712	9233	
14	365.596		

W. Ahrens, 1910

MATHEMATISCHE
UNTERHALTUNGEN UND SPIELE

VON

DR. W. AHRENS
IN MAGDEBURG

ZWEITE, VERMEHRTE UND
VERBESSERTE AUFLAGE

ERSTER BAND

MIT 200 FIGUREN IM TEXT

LEIPZIG UND BERLIN
DRUCK UND VERLAG VON B. G. TEUBNER
1910

Kapitel IX.

Das Achtköniginnenproblem.

[illegible]
Jean Paul.
„Die unsichtbare Loge". Erster Sektor.

[illegible]

[illegible]
Aus einem Gedicht des Muhammed ibn Scheraph.
[illegible]

§ 1. Historische Einleitung.

In der „Illustrirten Zeitung" vom 1. Juni 1850 (Nr. 361, 14. Bd., p. 352) findet sich unter der Rubrik „Schach" „Eine in das Gebiet der Mathematik fallende Aufgabe von Herrn Dr Nauck in Schleusingen" folgenden Inhalts: „**Man kann 8 Schachfiguren, von denen jede den Rang einer Königin hat, auf dem Brett so aufstellen, daß keine von einer anderen geschlagen werden kann.**"*) †) In der Nummer vom 21. September

*) „Vesir" für unser „Königin". Ich entnehme dies Wort aus A. v. d. Linde, „Geschichte u. Litteratur des Schachspiels", II, p. 257.

†) Irrtümlicherweise wird diese Stelle zumeist als das erste Vorkommen unseres Problems zitiert. Die Aufgabe ist jedoch bereits in der Schachzeitung, herausgegeben von der Berliner Schachgesellschaft, Bd. III, 1848, p. 363 von einem ungenannten „Schachfreund" gestellt worden, und zwar war, wie Max Lange („Handbuch der Schachaufgaben", Leipzig 1862, p. 30, Anm. 6) „nach einer direkten persönlichen Mittheilung" angibt, dieser „Schachfreund" Max Bezzel in Ansbach. — Wenn wir trotzdem oben die Geschichte des Problems an jene Nauck'sche Behandlung anknüpfen, so bestimmt uns hierbei der Umstand, daß die Fragestellung in der „Schachzeitung" zunächst nur 2 spezielle Lösungen (s. Schachzeitung IV, 1849, p. 40) gezeitigt und anscheinend überhaupt kein sonderliches Interesse für unser Problem

14*

	Stammlösungen				
n	doppelt-symmetrische	einfach-symmetrische	unsymmetrische	zusammen	Gesamtzahl aller Lösungen
2				0	0
3				0	0
4	1			1	2
5	1		1	2	10
6		1		1	4
7		2	4	6	40
8		1	11	12	92
9		4	42	46	352
10		3	89	92	724
11		12	329	341	2680
12	4	18	1744	1766	14032

Michael Simkin, 2021

ongeveer $(0.143n)^n$ oplossingen voor grote n

`https://www.quantamagazine.org/mathematician-answers-chess-problem-about-`

Een brute force (exhaustive search) aanpak:

> Genereer alle mogelijke configuraties van n dames op een n bij n bord, en controleer van elk daarvan of de dames elkaar al dan niet aanvallen.

Het aantal te controleren kandidaatoplossingen is hier exponentieel:

- n^n onder de aanname: één dame per rij
- $n!$ onder de aanname: één dame per rij en één per kolom; dit zijn gewoon alle permutaties van 1 t/m n

Basisidee backtracking

- bouw een oplossing stap voor stap op en controleer steeds of de deeloplossing in conflict komt met de restricties (en nog wel tot een oplossing kan leiden)

- op elk moment kun je kiezen uit een aantal mogelijke vervolgstappen; maak een keuze en ga langs die weg verder met het opbouwen van de oplossing

- als een keuze op niets uitloopt, herzie je deze keuze en probeer je een andere mogelijkheid

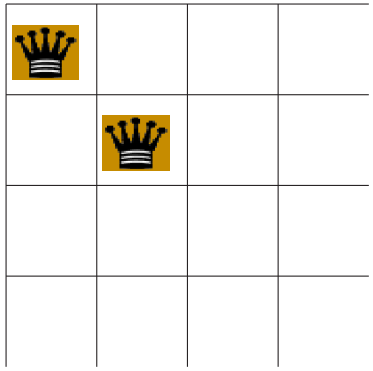
Vergelijk

- het vinden van de uitgang in een doolhof: loop steeds verder en als je bij het zoeken vastloopt, ga terug op je pad om het laatste open alternatief te proberen. Straks!

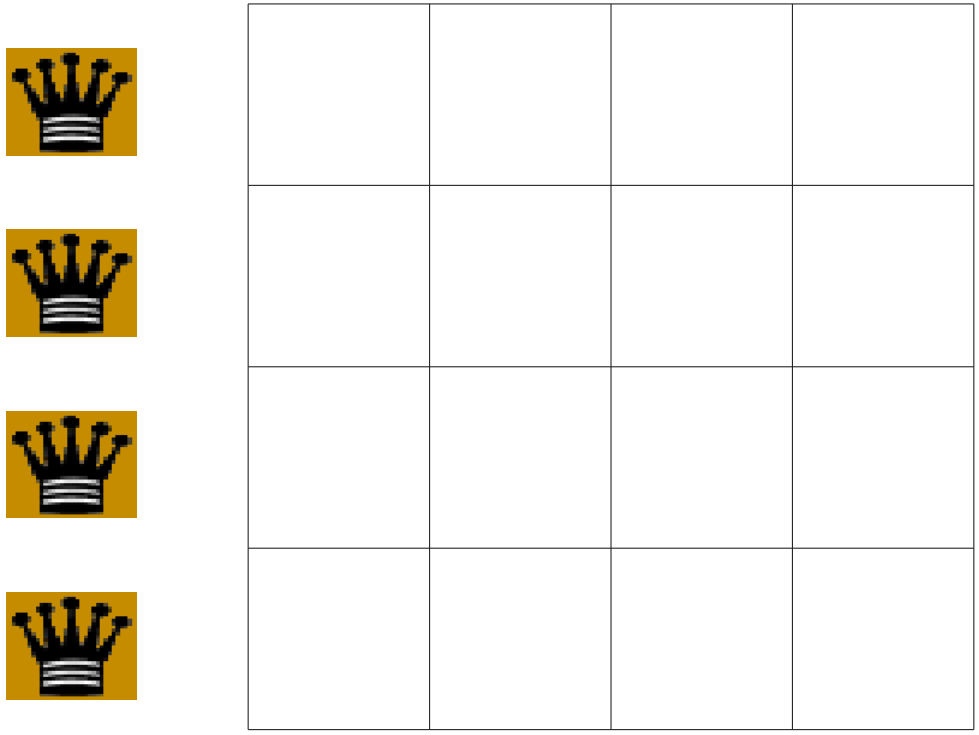
Backtracking versus exhaustive search

Exhaustive search bekijkt *alle* volledige kandidaatoplossingen. Dat zijn hier alle permutaties van 1 t/m n.
Backtracking controleert telkens van deeloplossingen of ze nog aan de eisen/restricties voldoen; zo niet, dan weet je zeker dat alle uitbreidingen van deze deeloplossing ook niet voldoen, dus die hoef je dan niet meer expliciet te bekijken. *Soms* spaar je zo heel veel uit.

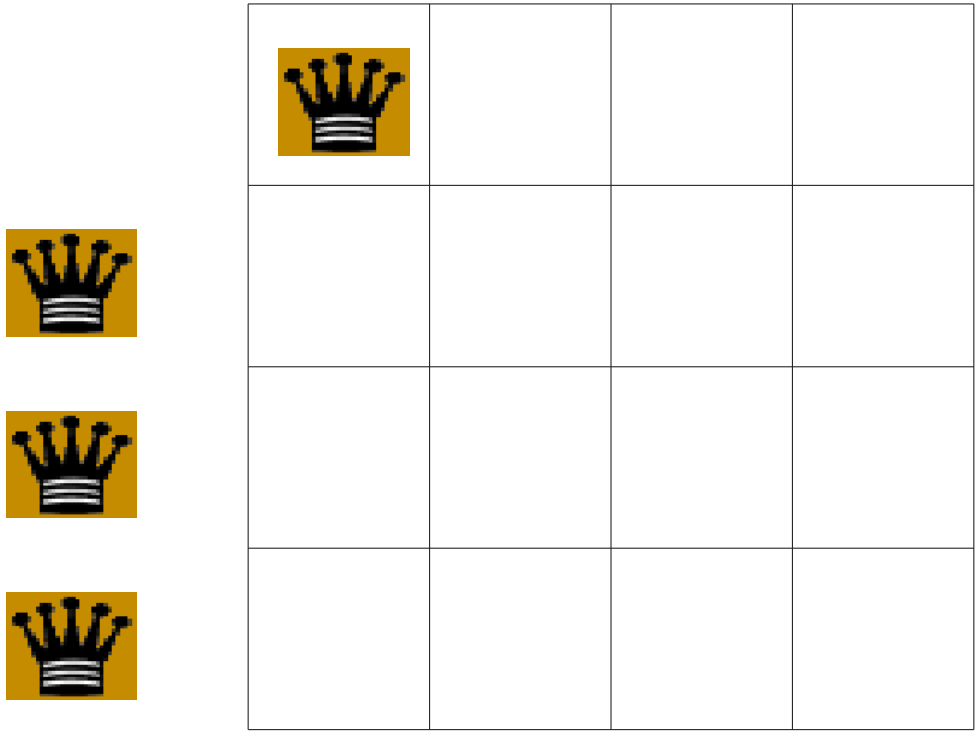
Voorbeeld:

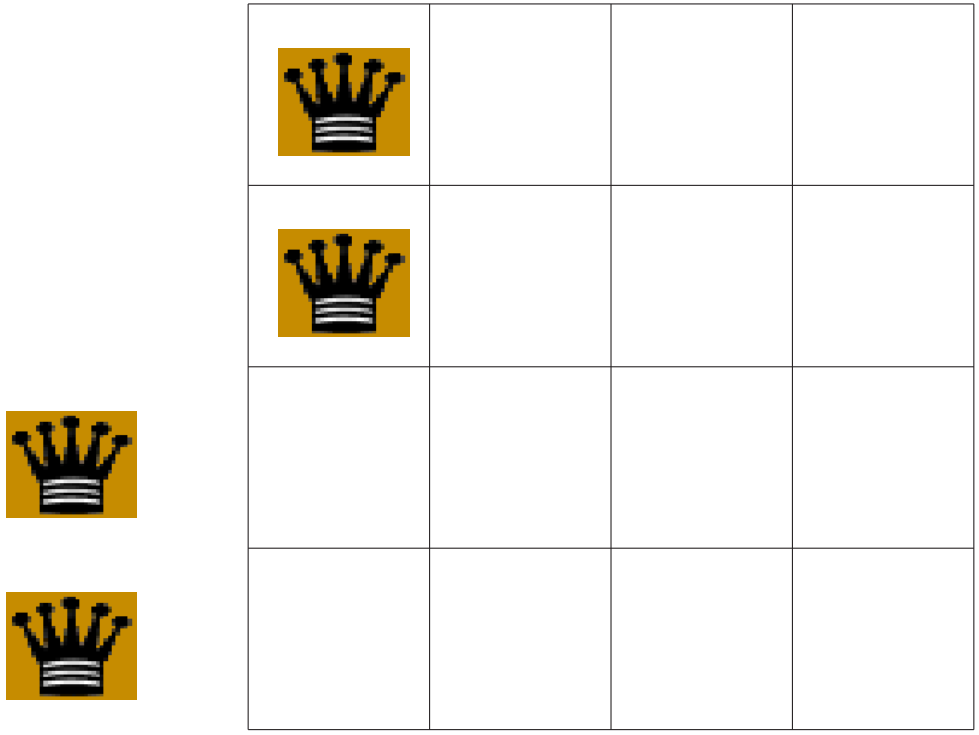


Alle $(n-2)!$ kandidaatoplossingen met de eerste twee dames op de aangegeven posities behoeven niet verder onderzocht te worden!

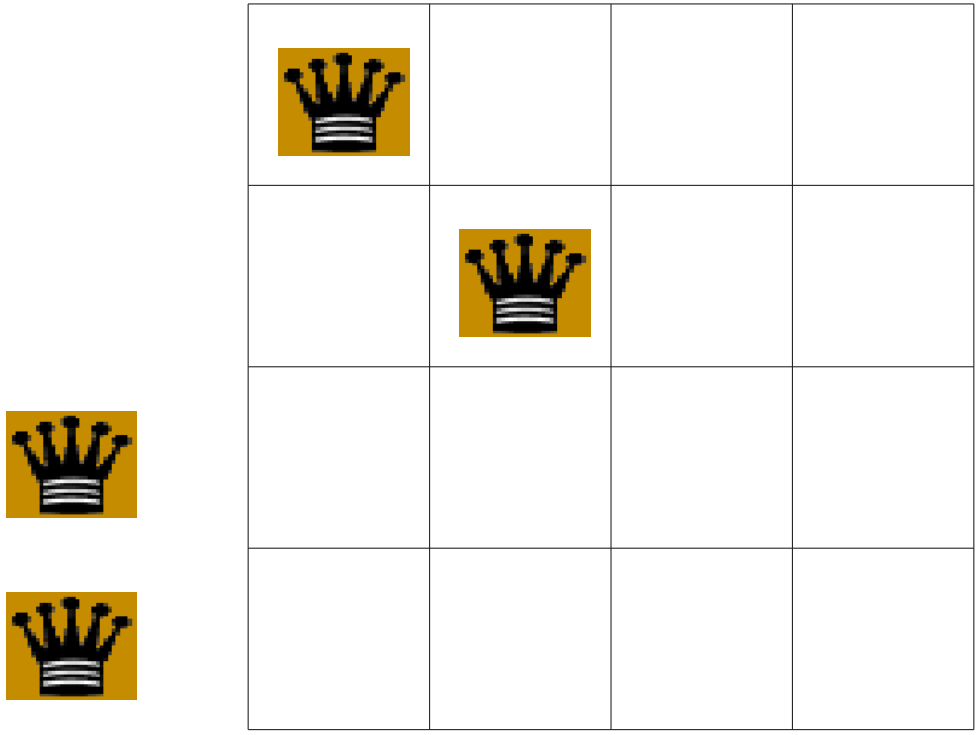


Dames $n = 4$

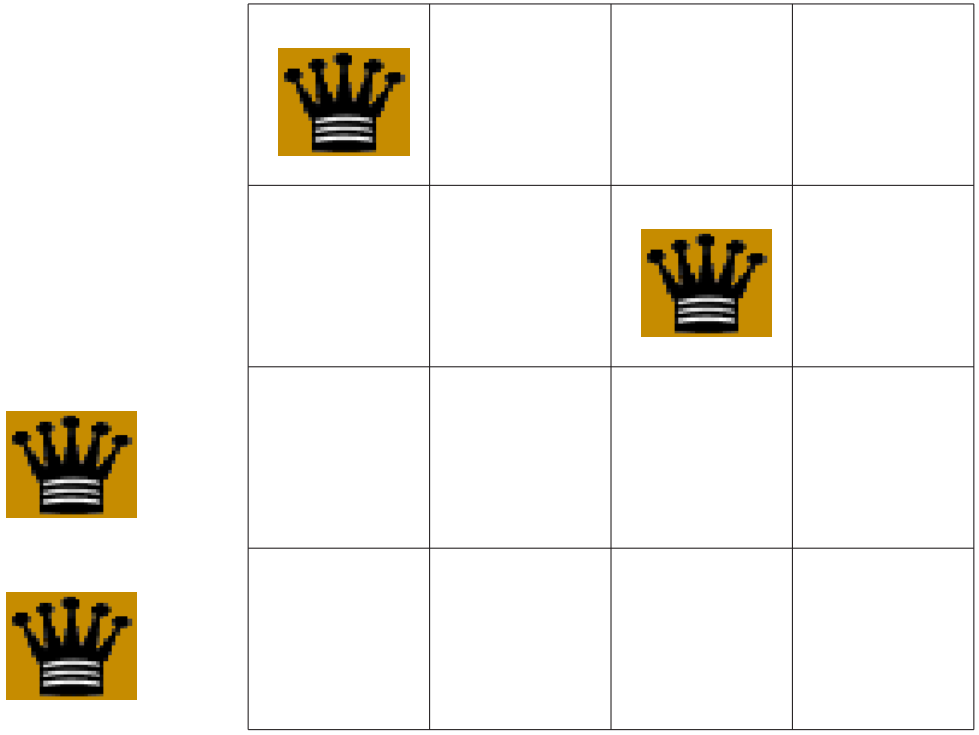




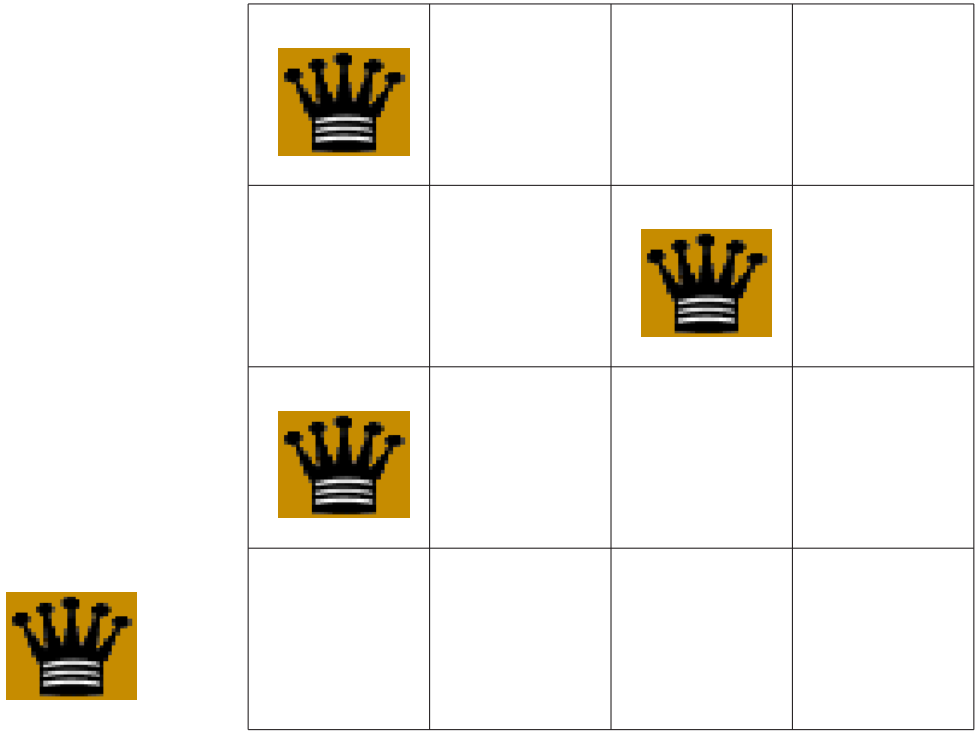
Dames $n = 4$



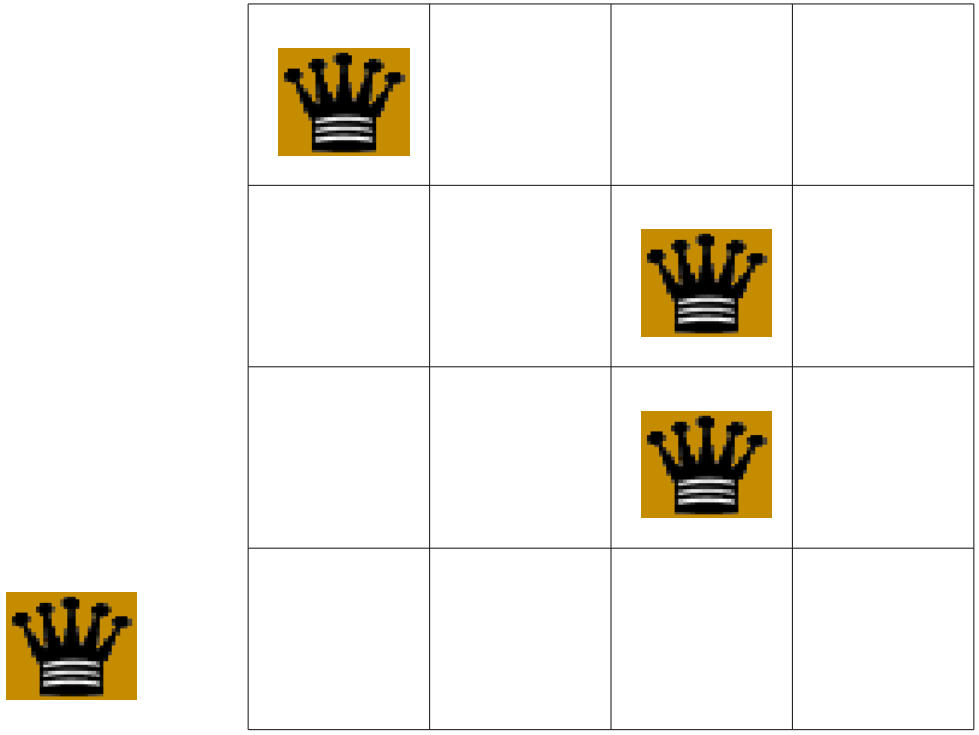
Dames $n = 4$

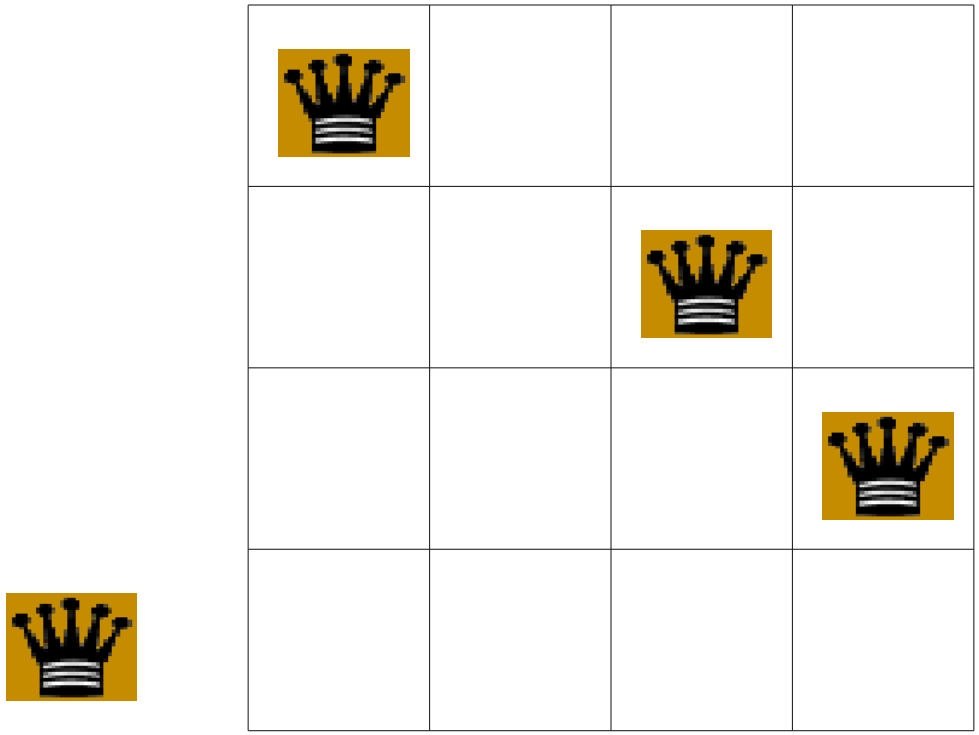


Dames $n = 4$



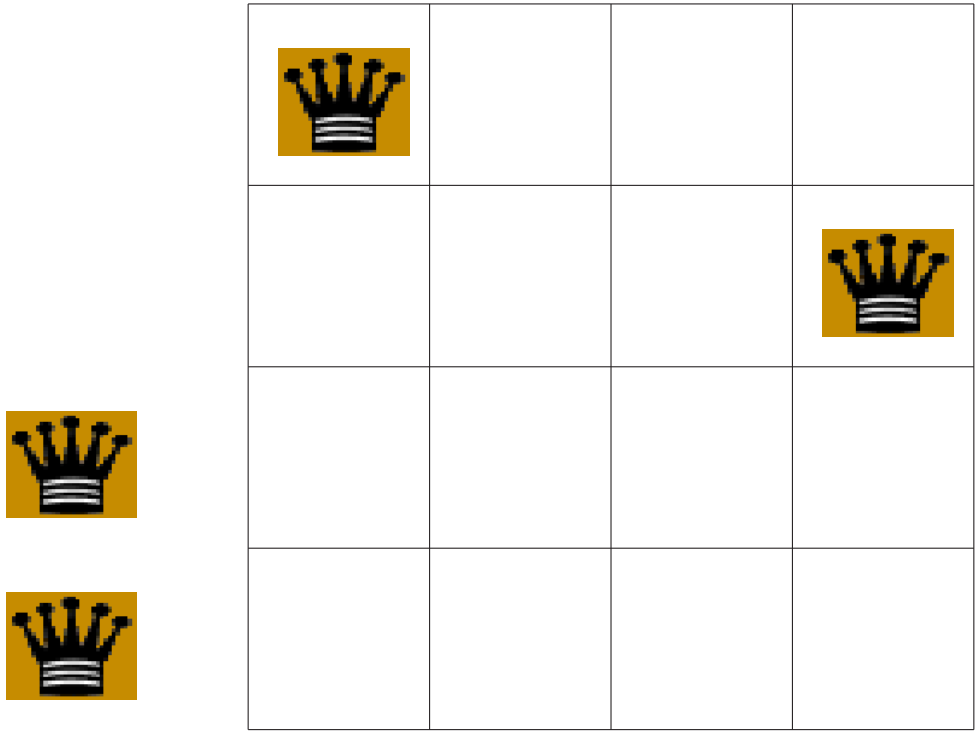
Dames $n = 4$



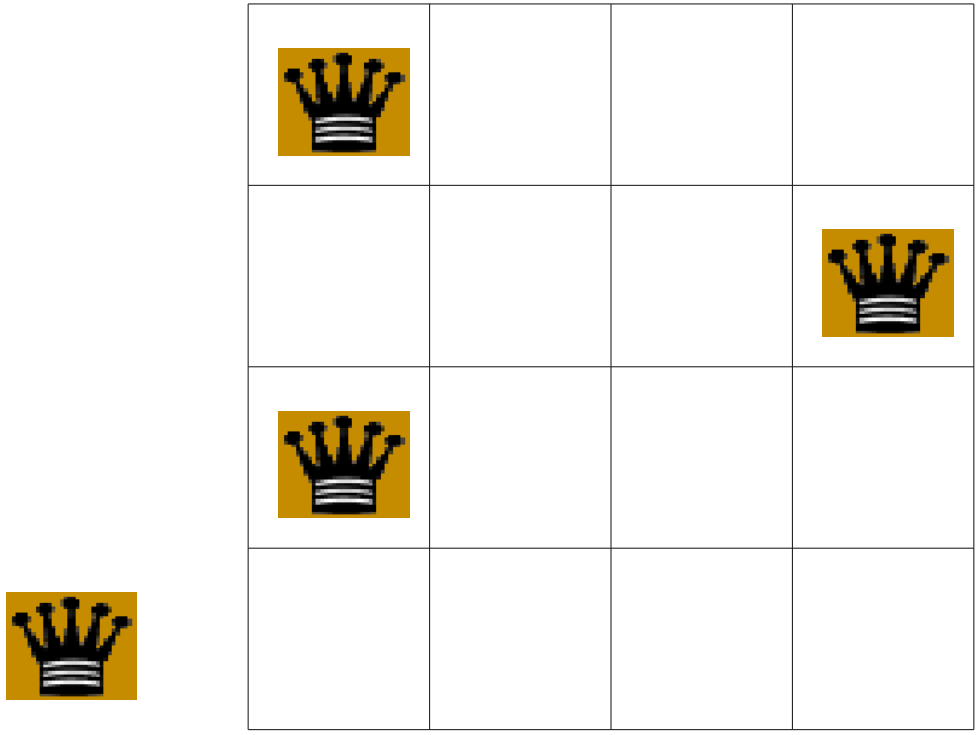


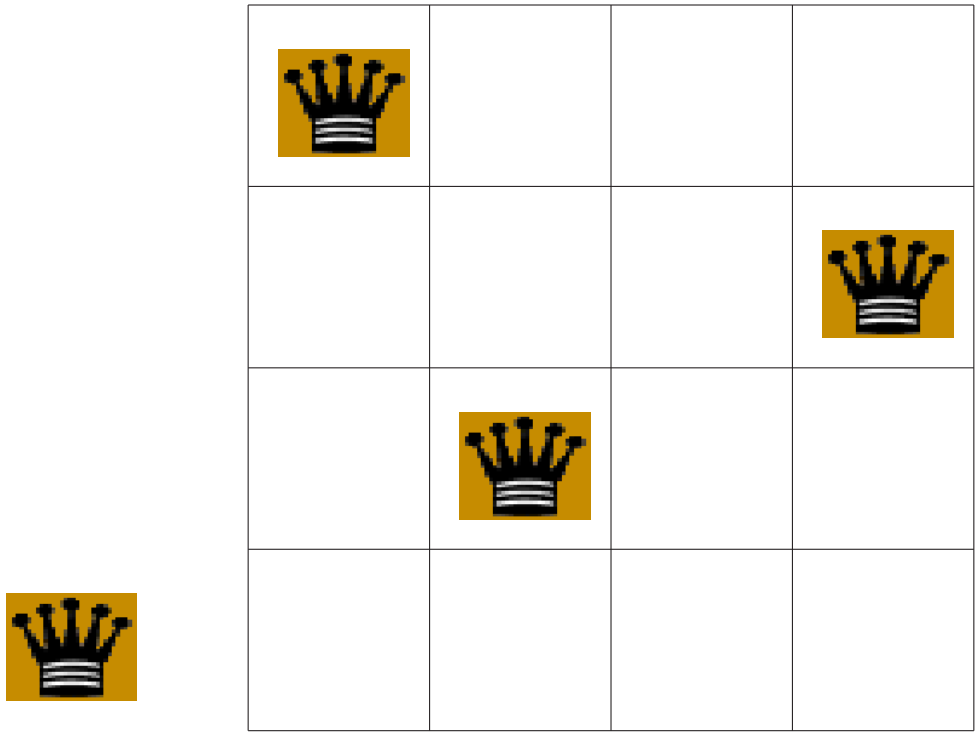
dame 3 kan niet geplaatst worden; verschuif daarom de dame uit de vorige rij (keuze herzien)

Dames $n = 4$



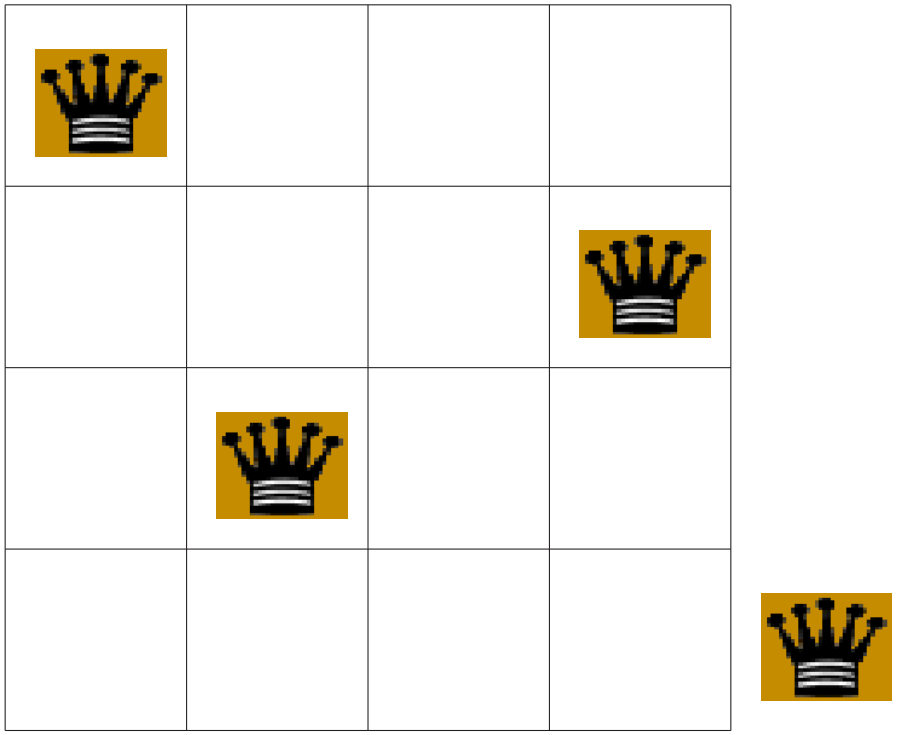
Dames $n = 4$





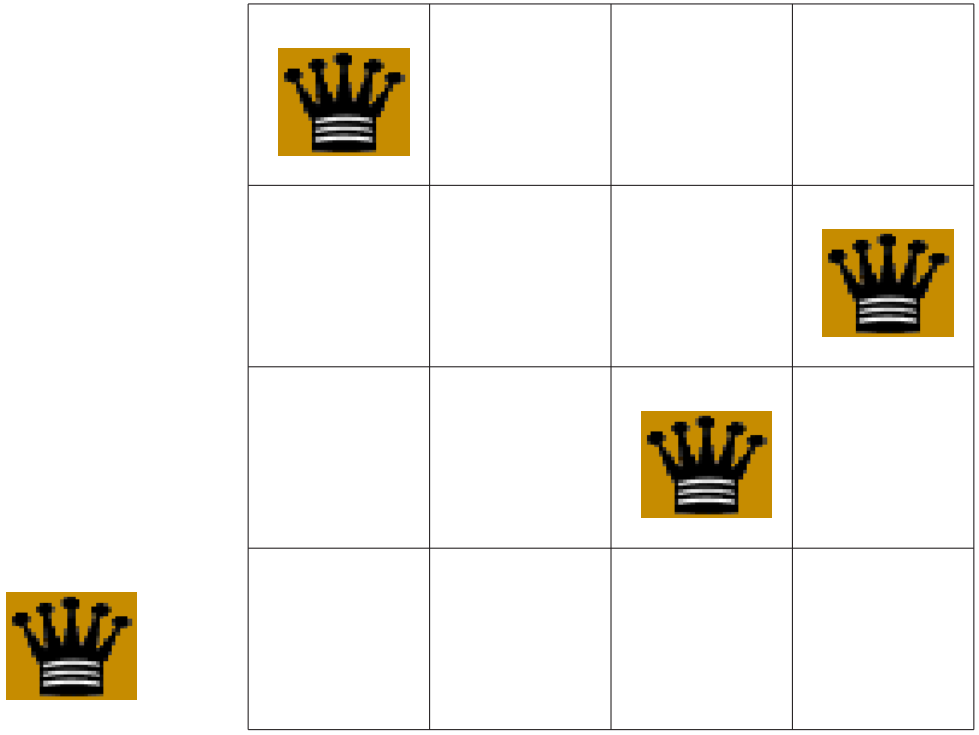
de eerste drie dames zijn goed geplaatst

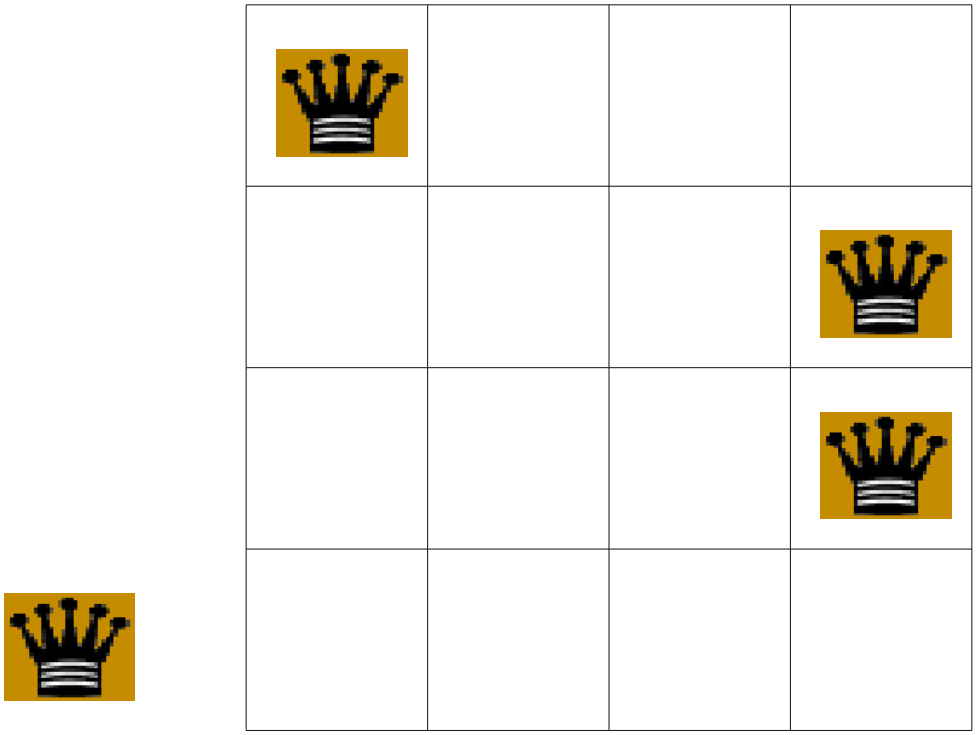
probeer nu de vierde in kolom 1 (nee), 2 (nee), 3 (nee), 4 (nee)



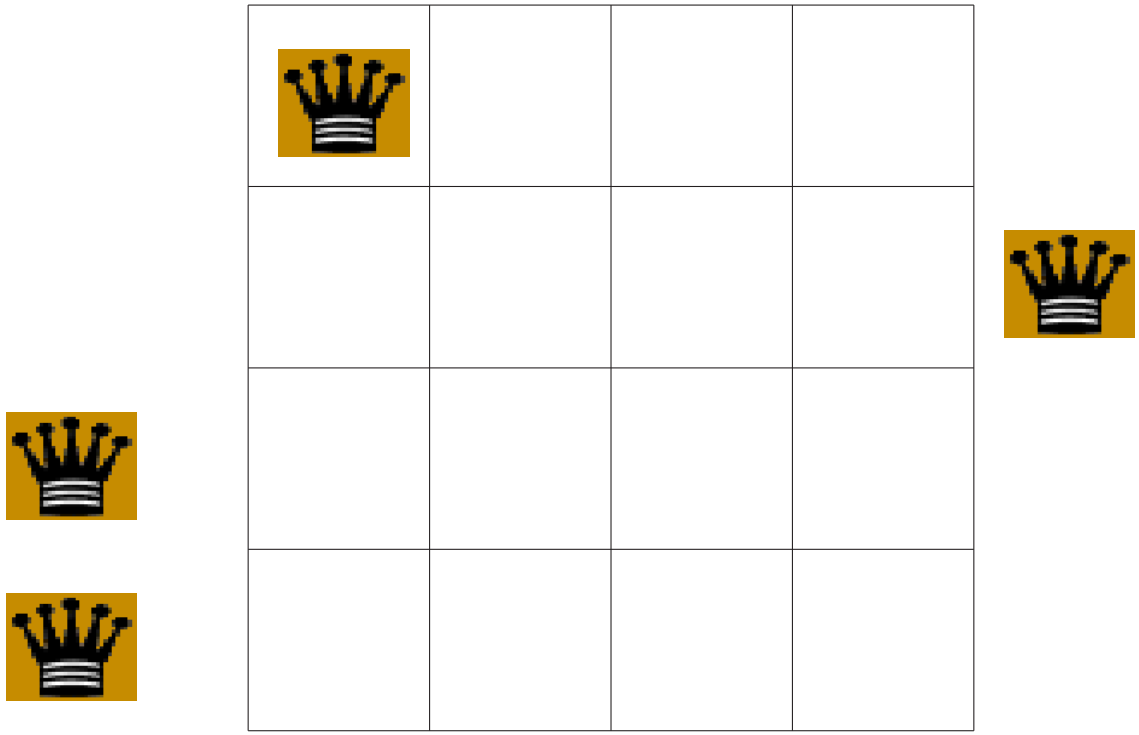
de vierde dame kan niet geplaatst worden; verschuif dus de vorige dame

Dames $n = 4$

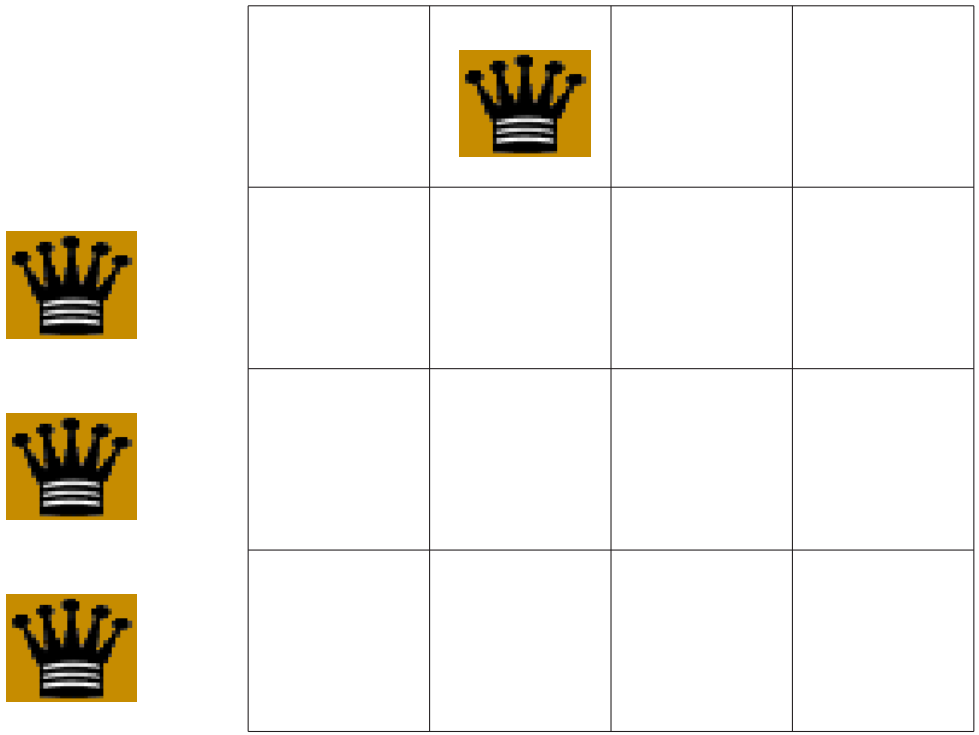




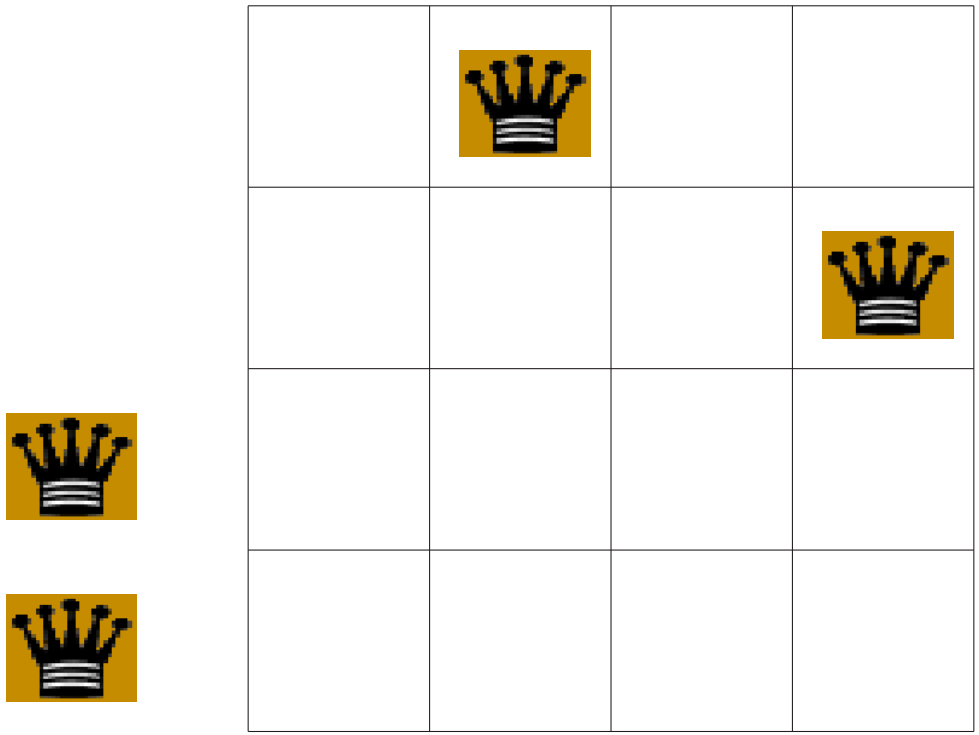
de derde dame kan niet geplaatst worden; verschuif de tweede dame



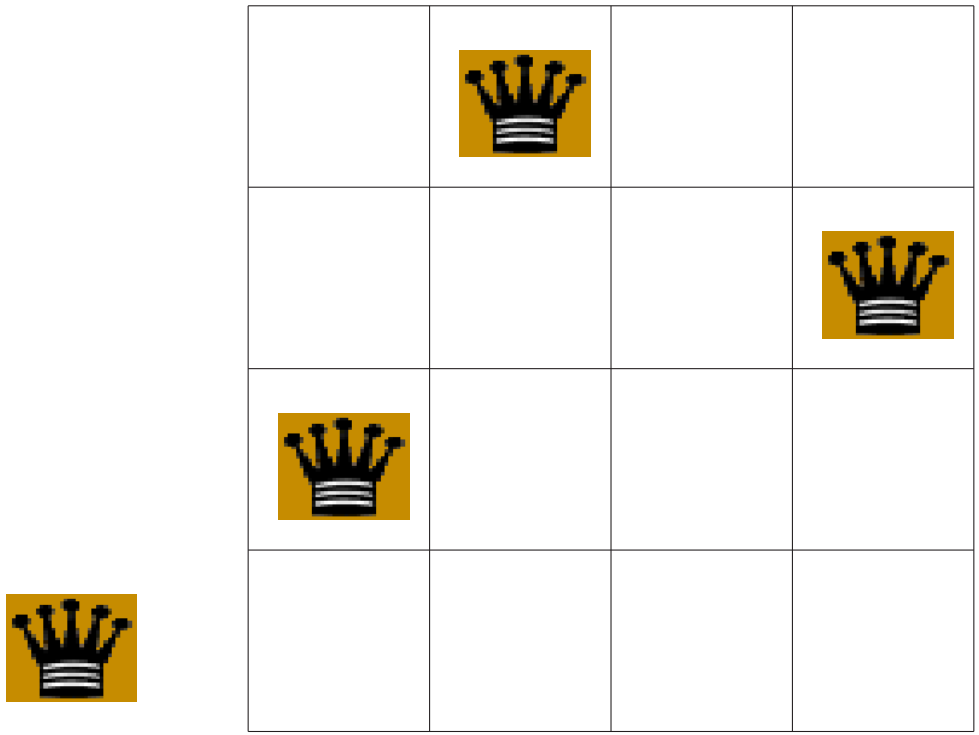
blijkbaar zijn er geen oplossingen met de eerste dame in de eerste kolom; herzie dus je keuze



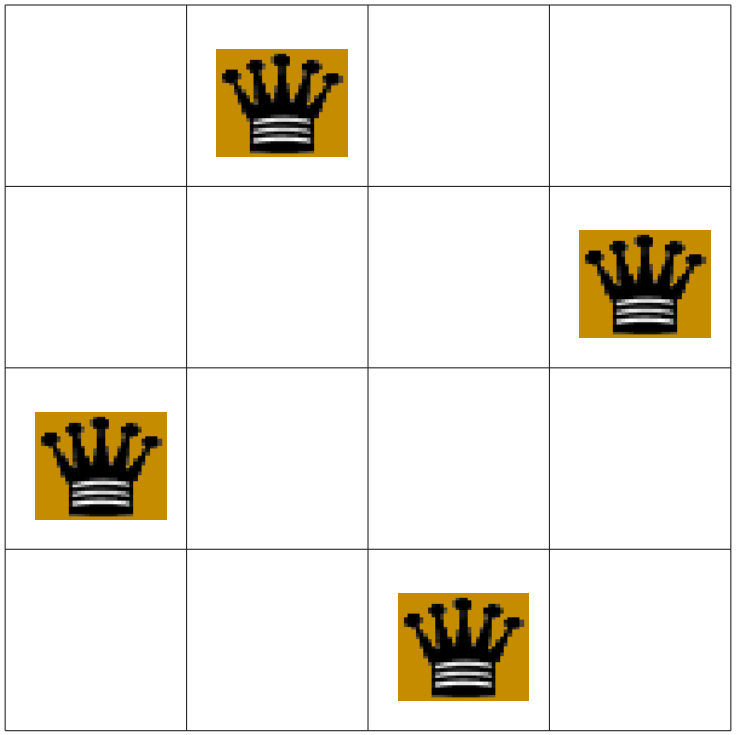
probeer de tweede dame in kolom 1 (nee), 2 (nee), 3 (nee), 4 (ja)



probeer de derde dame in kolom 1 (ja)

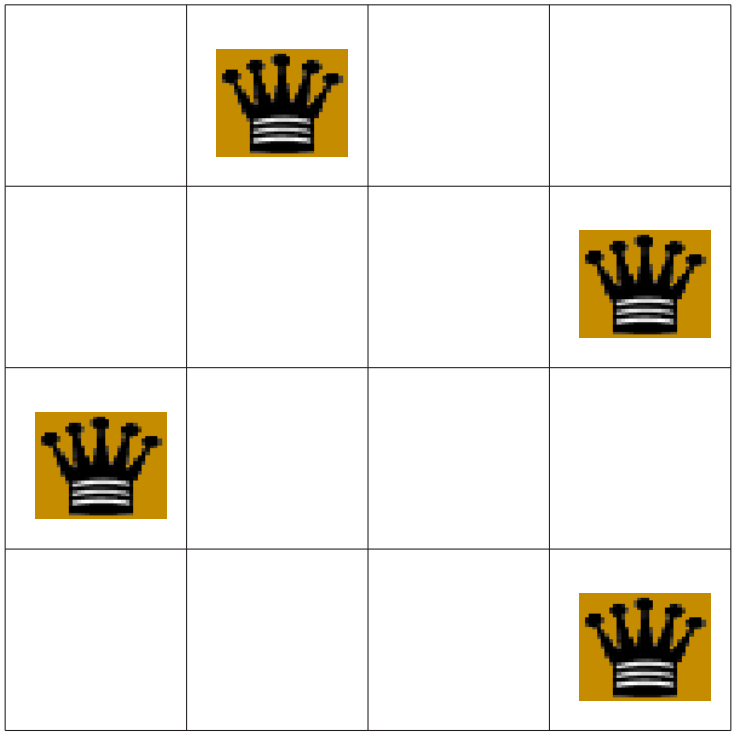


probeer de vierde dame in kolom 1 (nee), 2 (nee), 3 (ja)

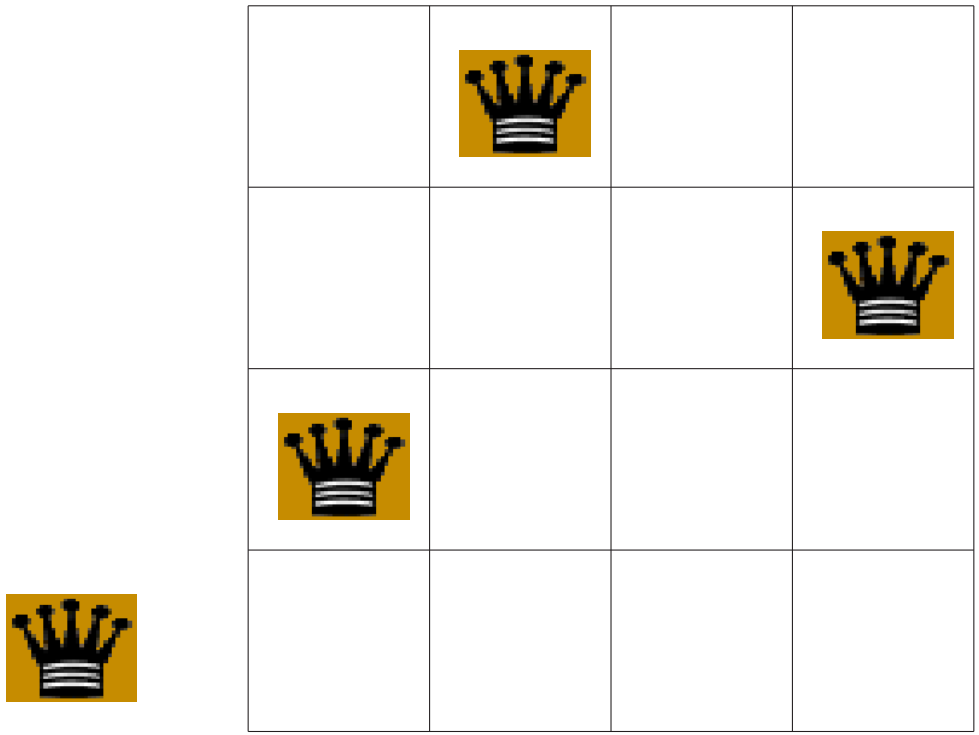


oplossing gevonden !!

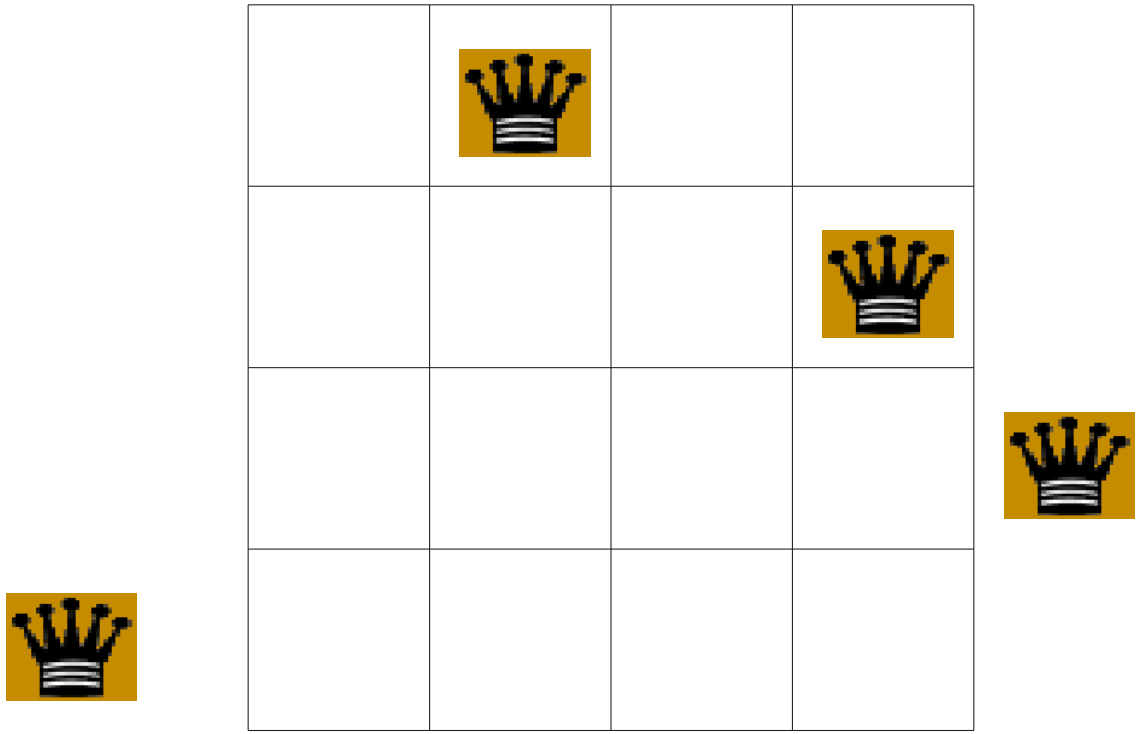
op dezelfde manier doorgaan om de andere oplossing(en) te vinden



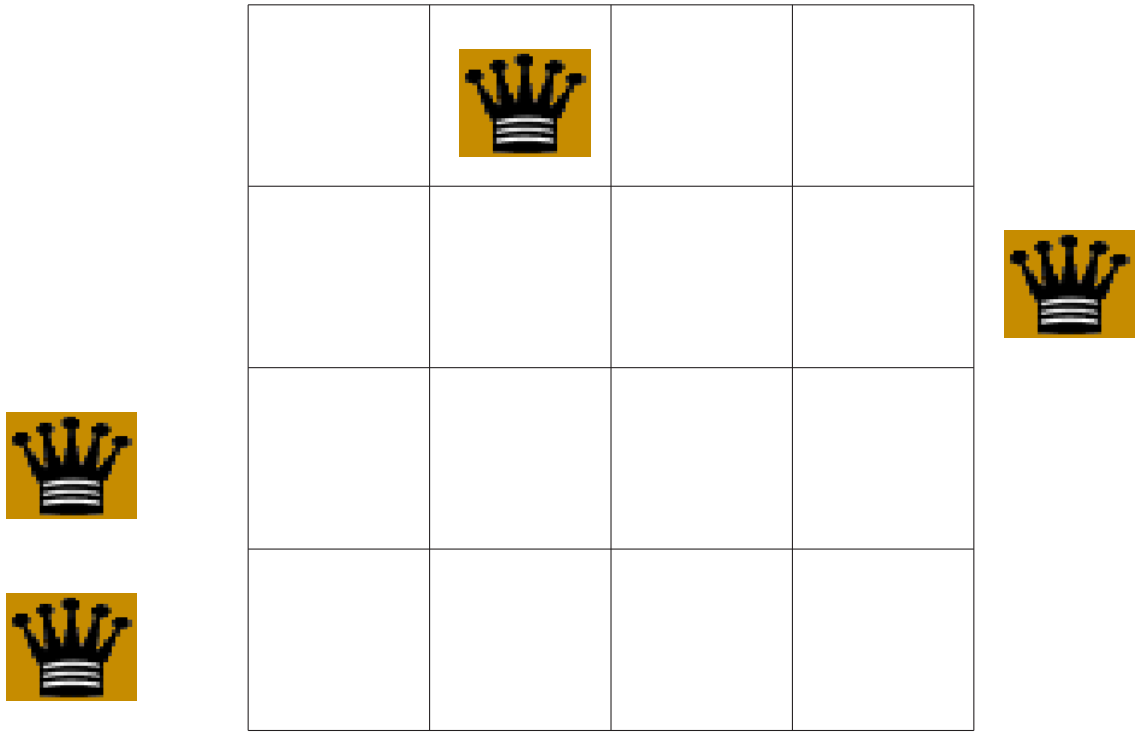
dit kan niet, dus vorige dame verschuiven



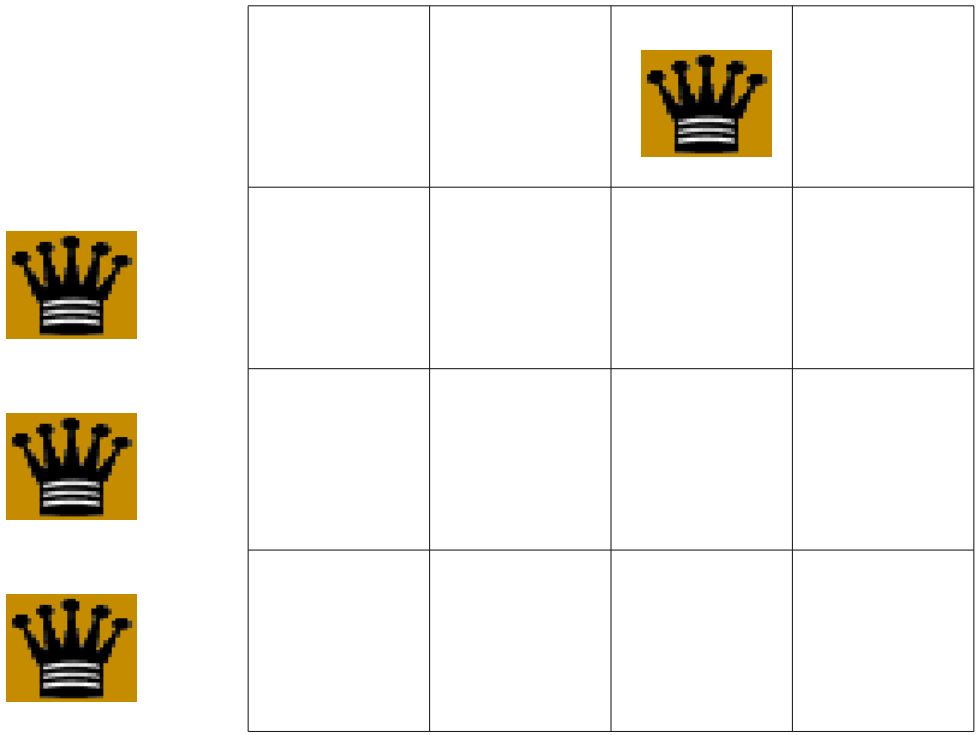
probeer de derde dame in kolom 2 (nee), 3 (nee), 4 (nee)



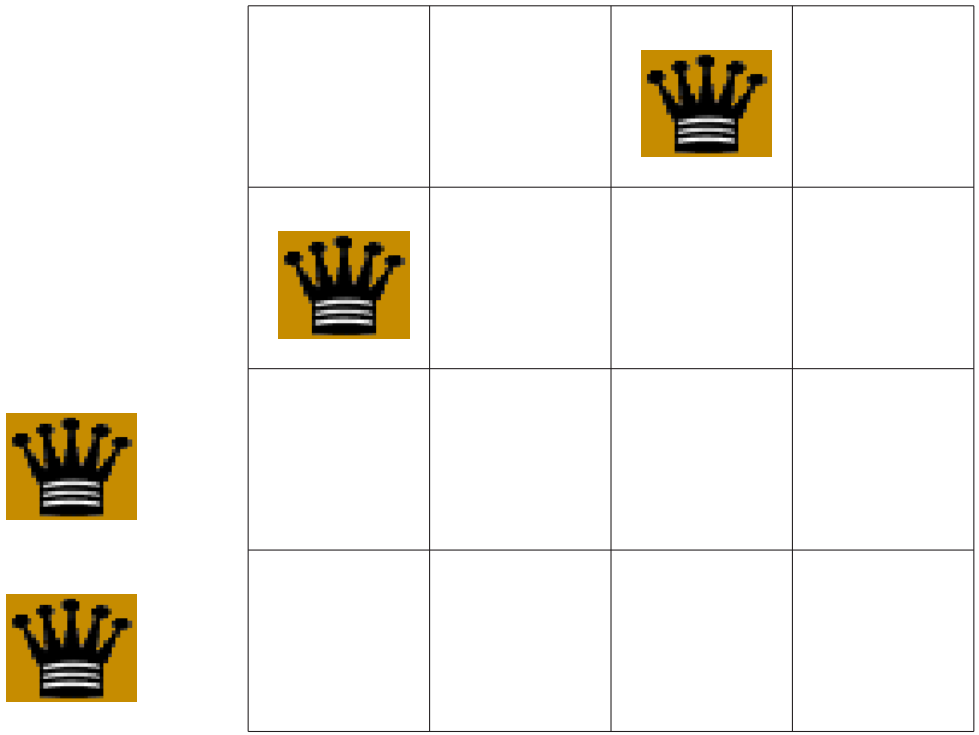
kan niet, dus tweede dame herzien



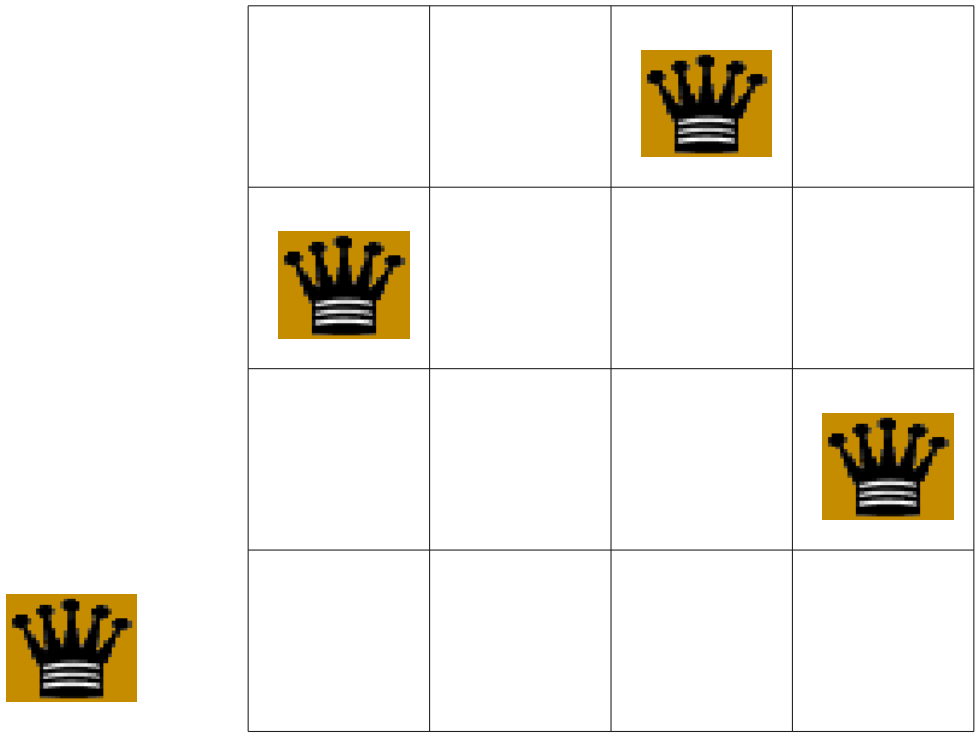
kan niet, dus eerste dame herzien



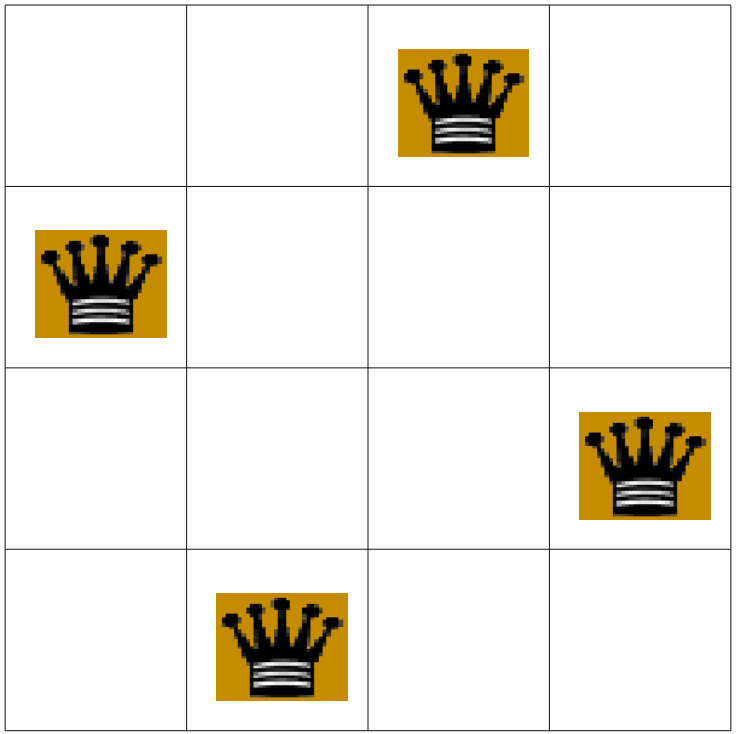
probeer dame 2 in kolom 1 (ja)



probeer dame 3 in kolom 1 (nee), 2 (nee), 3 (nee), 4 (ja)



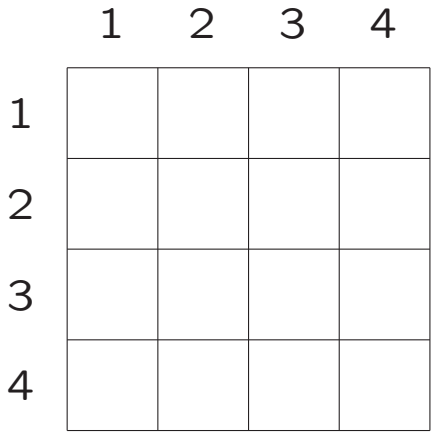
probeer dame 4 in kolom 1 (nee), 2 (ja)



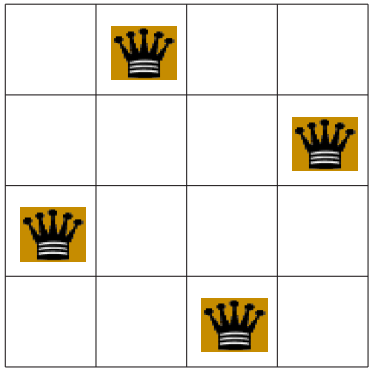
oplossing gevonden !!

zo doorgaand vinden we verder geen andere oplossingen meer

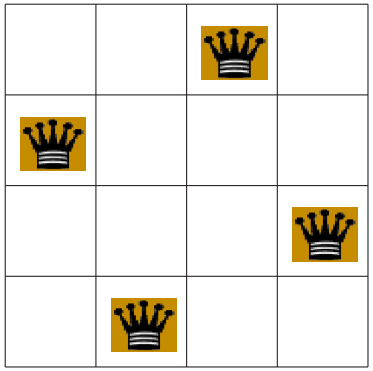
Oplossingen dames: $n = 4$



oplossing 1



oplossing 2



- **Lezen/leren bij dit college:**
  Paragraaf 3.4, 3.5, slides

- **Werkcollege:** brute force en exhaustive search
  vrijdag, 11.00–12.45, zalen BW.0.07, BW.0.08

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

- **Practicumbijeenkomst programmeeropdracht 1:**
  dinsdag, 09.00-10.45, DM.0.09, DM.0.13

- **Volgend college:**
  woensdag 12 maart 2025, 15.15–17.00