

Damen-Problem

"gelöst mit Backtracking und Rekursion"

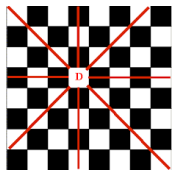


Einleitung:

Spielen Sie Schach?

Neben dem eigentlichen Spiel gibt es viele Knobel-Aufgaben, die sich dem Schachbrett und den Schachfiguren und deren Möglichkeiten bedienen.

So auch die Knobel-Aufgabe mit den acht Damen.



Aufgabe:

- Lesen und lösen Sie die Aufgabe *RekursionDameEinleitung*

Aufgabe:

- Erklären Sie, wie Sie die Aufgabe mit den acht Damen mit Denken gelöst haben.

Ein möglicher Ansatz:

Ich platziere in der obersten Reihe eine Dame und eine zweite Dame in der zweiten Reihe. Dabei beachte ich, dass die zweite Dame nicht in ein bedrohtes Feld gelegt wird.

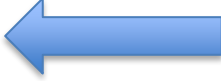
D							
X	X	D					

Dritte Dame und vierte Dame:

D							
1	1	D					
1	2	1	2	D			
1		2	1	2	3	D	

sechste Dame und siebte Dame:

D							
1	1	D					
1	2	1	2	D			
1		2	1	2	3	D	
1	D	2		1	2	3	4
1	3	2	D	3	1	2	3
1	5	2	4	3	D	1	2
1	5	2	6	3	7	4	1




Uuups wir haben in der 8ten Reihe kein Feld, auf das wir die Dame platzieren können.

Ich muss also eine bereits platzierte Dame anders platzieren und die Lösung weiter suchen.

In Zeile 7 und 6 habe ich keine Möglichkeit.

Erst in der der fünften Zeile sehe ich die Möglichkeit die Dame auf ein anderes Feld zu platzieren.

D							
1	1	D					
1	2	1	2	D			
1		2	1	2	3	D	
1	D	2		1	2	3	4



Nun probiere ich es wieder aus...

Und wenn das wiederum nicht geht?

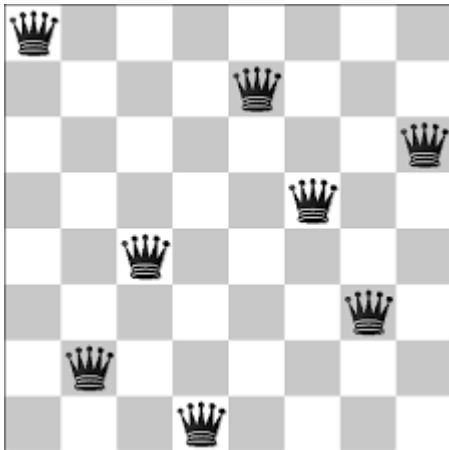
Dann muss ich wohl bis zur Zeile vier zurückgehen dort die Dame umplatzieren und es wieder versuchen.

- ⇒ Ich versuche von oben nach unten eine Lösung zu finden.
Gelange ich in eine Sackgasse, so gehe ich zurück, bis ich eine Variante offen habe.
Suche erneut...
Gehe falls nötig noch weiter zurück.

- ⇒ Das nennt man das **Backtracking** Verfahren.

Die Lösung:

Es gibt eine Lösung für das Rätsel.



Aufgabe:

- Kontrollieren Sie ob die obige Lösung stimmt.

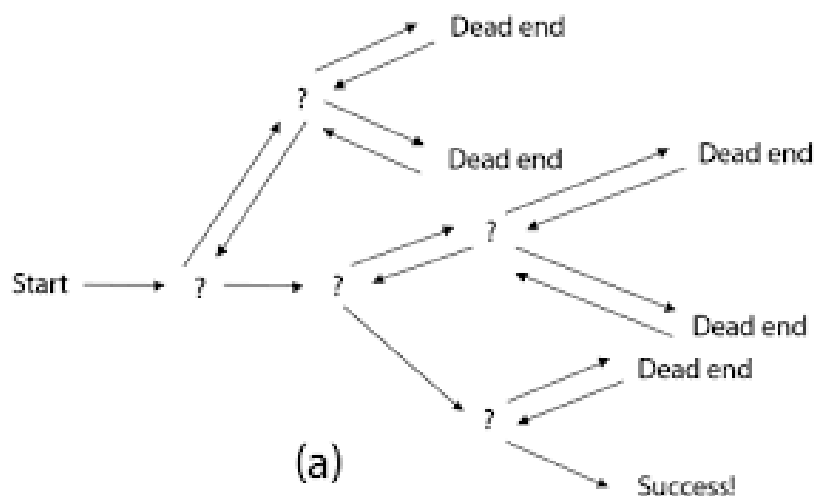
(Es gibt mehr als nur eine Lösung 😊)

Backtracking:

Backtracking geht nach dem Versuch-und-Irrtum-Prinzip (trial and error) vor, das heisst, es wird versucht, eine erreichte Teillösung zu einer Gesamtlösung auszubauen. Wenn absehbar ist, dass eine Teillösung nicht zu einer endgültigen Lösung führen kann, wird der letzte Schritt beziehungsweise werden die letzten Schritte zurückgenommen, und es werden stattdessen alternative Wege probiert. Auf diese Weise ist sichergestellt, dass alle in Frage kommenden Lösungswege ausprobiert werden können

Nach diesem Prinzip lassen sich viele Knobelaufgaben lösen.

Allerdings ist das wohl nicht immer der effizienteste Weg.



Backtracking für das Dame Problem umsetzen:

Nun sollen Sie das Problem mit einem rekursiven Algorithmus mit Backtracking implementieren.

Das Resultat kann so aussehen:

```
Damen Problem
Start mit Dame in Spalte: 0

Q * * * * *
* * * * Q * *
* * * * * * Q
* * * * Q * *
* * Q * * * *
* * * * * * Q
* Q * * * * *
* * * Q * * *
```

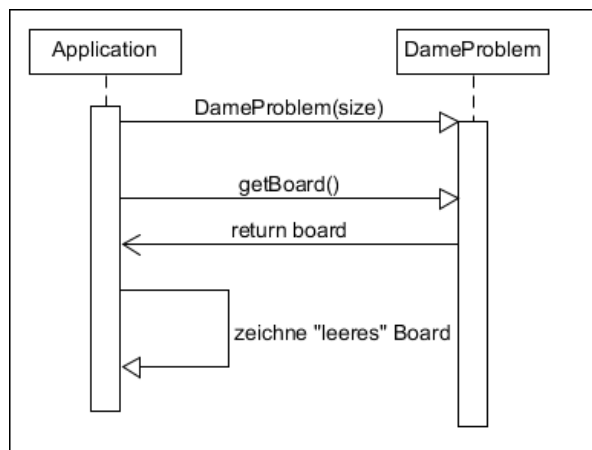
```
ch.bbw.pr.dame
├── Application.java
│   └── Application
│       └── main(String[]): void
├── DameProblem.java
│   ├── DameProblem
│   │   ├── FIELD_FREE
│   │   ├── FIELD_OCCUPIED
│   │   ├── board
│   │   ├── size
│   │   ├── DameProblem(int)
│   │   ├── getBoard(): int[][]
│   │   ├── isValid(int, int): boolean
│   │   └── setQueen(int): boolean
```

Ein paar Tipps zur Implementierung

Schritt 1:

- Übernehmen Sie das vorbereitete Projekt 10_2_10_DameVorgabe.zip.
Das board ist das Attribut in der Klasse DameProblem: `private int[][] board`
→ zweidimensionalen Array von int
- Die Elemente des Arrays können die Werte FIELD_FREE und FIELD_OCCUPIED haben.
`private static final int FIELD_FREE = 0; //Platz frei`
`private static final int FIELD_OCCUPIED = 1; //Platz durch Dame belegt`

Im Konstruktor werden alle Felder des Boards mit **FIELD_FREE** initialisiert.



Resultat:

```
Damen Problem

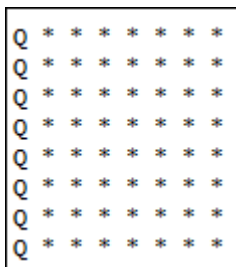
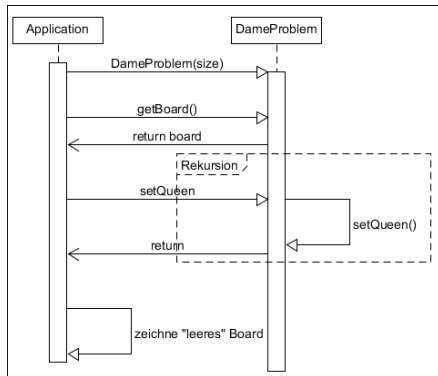
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
```

Schritt 2:

- Ergänzen Sie in der Klasse *DameProblem.java* die vorbereitete Methode, so dass rekursiv 8 Damen ganz links gezeichnet werden und das noch *ohne Prüfung auf Kollision* mit anderen Damen.

"Für alle *rows* setze Feld 0 auf *FIELD_OCCUPIED* → *board[row][0] = FIELD_OCCUPIED;*"

- `public boolean setQueen(int row)`

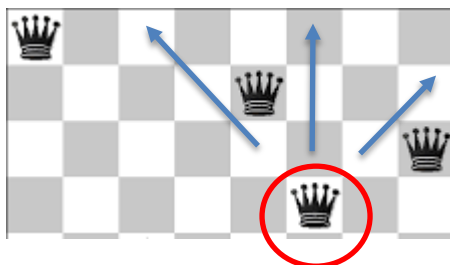


Schritt 3:

Sie sehen oben, dass die Dame auf Zeile 2 von der Dame auf Zeile 1 bedroht wird.

Die Methode `boolean isValid(int row, int column)` **prüft**, ob die Queen an der Stelle *row* und *column* von einer anderen Queen bedroht wird. Das muss gerade nach oben und diagonal nach oben geschehen.

Beispiel:



Die Pfeile geben an, welche Felder überprüft werden müssen, ob dort eine andere Queen steht, die unsere Queen bedroht.

Das kann man mit **drei for loops** machen, die man nacheinander aufruft, und die jeweils in eine Richtung suchen.

Hmm, da müssen Sie etwas überlegen...!!!

Waagrecht müssen Sie nicht suchen, denn in der gleichen Zeile kann keine Queen stehen.

Nach unten müssen Sie nicht suchen, da dort noch keine Queen stehen kann.

Wenn ich nun auf meine bisherige Anwendung *isValid* aufrufe und mit einer Ausgabe auf die Console verknüpfe, dann könnte das das Ergebnis sein.

6

Und wo genau ist nun da Backtracking?

Der Begriff Backtracking bezieht sich auf diesen Schritt:

*Dann ruft man die Methode setQueen wieder auf, und zwar für die nächste Zeile.
Falls dieser Aufruf true zurück gibt, gibt man auch return true zurück
Falls dieser Aufruf false zurück gibt, muss man ein **backtracking** machen.
Das heisst, der aktuelle Platz muss wieder freigegeben werden
(und der for loop oben muss weitersuchen, die Suche geht weiter in der aktuellen
Zeile, mit der nächste Spalte.*

Man ruft rekursiv die nächste Zeile (n+1) auf und dort wird versucht eine Queen zu setzen.
Findet man dort keine Lösung, so kommt man zurück und muss die Queen auf der Zeile n
wegnehmen und weitersuchen.

Zusatzaufgabe

- Probieren Sie aus, ob das Programm auch eine Lösung findet, wenn die erste Dame nicht ganz links (Spalte = 0) gesetzt wird.
Finden Sie damit Lösungen für alle 8 Positionen der Dame in der ersten Zeile.
- Kann das Programm auch ein grösseres Schachbrett berechnen.
Probieren Sie zuerst mal 10x10 und steigern Sie sich dann.
Irgendwann wird der Rechenaufwand sehr gross!! (>30)
- Ergänzen Sie das Programm, so dass am Ende ausgegeben wird, wie oft das Backtracking gemacht wurde.

Zusatzaufgabe "Das Damen mit einem Bauer"

Das Neun-Damen-Problem verlangt, auf einem Brett neun Damen und einen Bauern derart unterzubringen, dass die Damen einander nicht bedrohen können, also keine direkte waagerechte, senkrechte oder diagonale Sichtlinie zueinander haben.

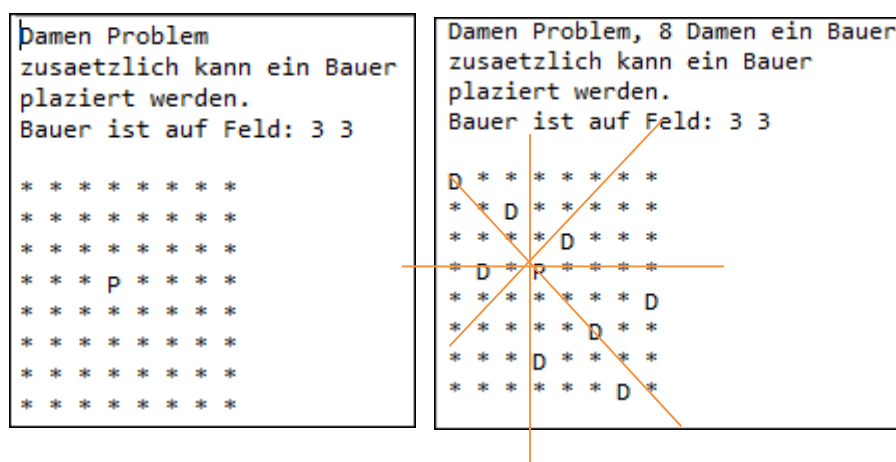
Der Bauer wirkt als Sichtschutz.

- Ausgangslage: Bauer wird im Spielfeld platziert
- Bauer versperrt Sicht auf andere Damen

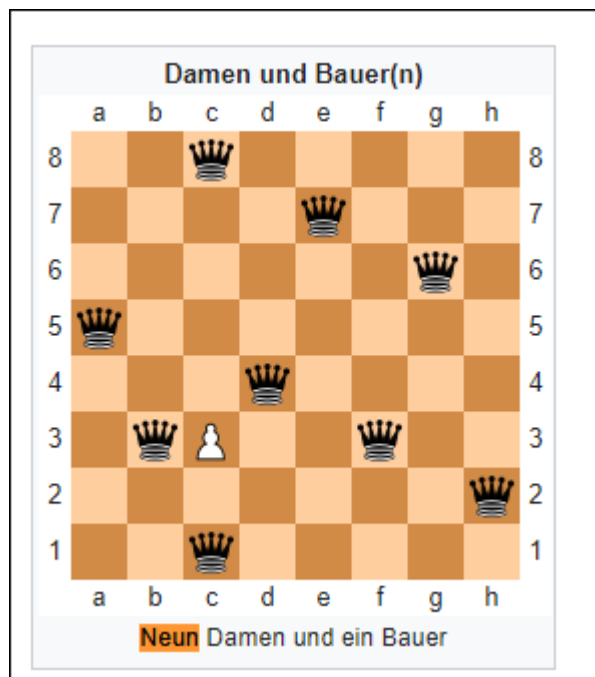
Ich habe **zuerst** das Problem mal **mit einem Bauer und nur 8 Damen gelöst**.

Das ist vorerst einfacher. Denn ich muss nur den Bauern bei isValid berücksichtigen.

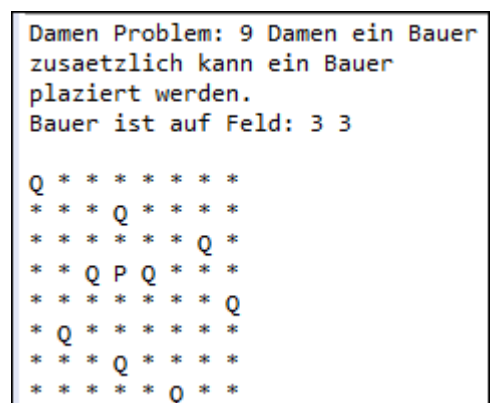
⇒ "Suche nach oben, links und rechts endet auch, wenn ich auf einen Bauern treffe."



Auf Wikipedia ist das richtige Bild mit 9 Damen auf einem 8x8 Feld



meine Lösung:



Es folgen meine Überlegungen.

Eigentlich ist alles ganz normal, bis auf die Zeile wo der Bauer steht.

Dort muss ich in der gleichen Zeile in einer anderen Spalte eine zweite Dame hinstellen.

Das muss ich in diese Zeile nur einmal machen. → Das ist also ein einmaliger Spezialfall.

- Ich merke mir die Zeile wo der Bauer steht.
Damit weiss ich, wann ich den Spezialfall machen muss.
- Wenn ich in dieser Zeile bin
 - o Setze ich in `setQueen()` die erste Dame.
 - o Doch danach rufe ich den Spezialfall auf, den ich in der Methode `setSecondQueen()` implementiert habe.
Als Parameter übergeben ich die *momentane Zeile* und *momentane spalte + 1*.

```

setQueen(int, int) : boolean
setQueen(int) : boolean
setSecondQueen(int, int) : boolean
    
```

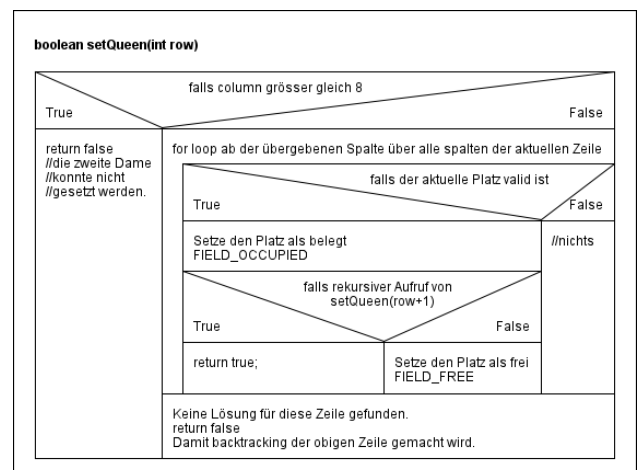
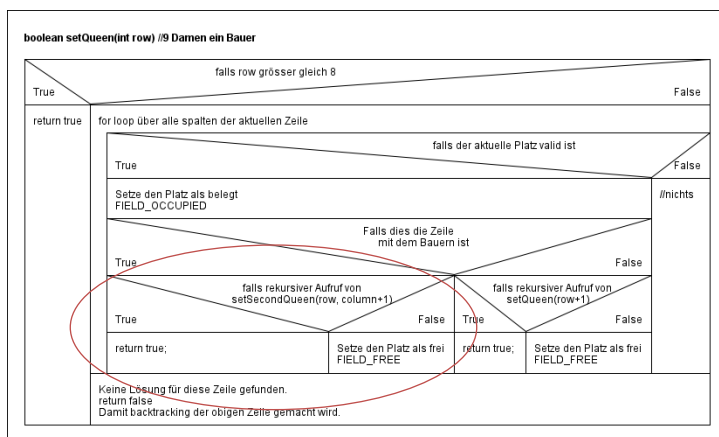
In `setSecondQueen()` hat es einen Loop, der ab der übergeben Spalte bis zur maximalen Spalte eine Dame zu setzen versucht.
(Das ist fast wie in `setQueen()` mit den Zeilen).

`setSecondQueen()` benützt auch `isValid()` für den Test, ob die Dame gesetzt werden kann.

Und ruft danach seinerseits `setQueen()` auf. Um weitere Damen in unteren Zeilen zu setzen.

Wenn das nicht gelingt, macht `setSecondQueen()` ein Backtracking und versucht die nächste Spalte...

- `IsValid()` musste ich ergänzen, so dass geprüft wird, ob in der Zeile links nicht schon eine Dame steht. Damit `setSecondQueen()` nicht die zweite Dame neben die erste stellt.



Zusatzaufgabe "Das Springerproblem"

Ein ähnliches Rätsel ist das "Springerproblem"

- Lesen Sie dazu das Dokument `DasSpringerProblem_RW_Trier_2013_p1-6.pdf`
- Lösen Sie das Problem mit Backtracking