Lab Report

Exercise 08: Eight Queens

Autoren: Dennis Loska, Tony Dorfmeister, Bernhard Zaborniak 06.06.2017

1. Our goal is to write a program to determine if eight queens can be placed on an 8 x 8 chess board without them threatening each other! Start by deciding how to represent a chess board with a data structure. Don’t worry about the colors of the board yet. Write a Chessboard class! What methods will you need?

Um uns auf den später folgenden Alghorithmus zu konzentrieren, haben wir das Chessboard so einfach wie möglich gehalten. Wir nutzten ein zweidimensionales Array um es zu repräsentieren welches wir im Konstruktor der Chessboard-Klasse mit Nullen füllten, falls auf einem feld eine Queen stehen sollte, würden wir es mit einer 1 markieren.

Um dieses Schachfeld beobachten zu können, schrieben wir eine einfache displayBoard Methode, welche das Array mit zwei for-Schleifen durgeht und in einen String zusammenfasst.

Außerdem brauchten wir noch einige kleine Basis-Methoden für die weitere Aufgaben und zur übersichtlichkeit des Codes - wie putQueen(row,col), removeQueen(row,col) und boolean hasQueen(row,col).

```java
/*
 *  Exercise 1 zeigt uns wie das Schachfeld aussieht
 */
public void displayBoard(){
    String board="";
    for(int r=0;r<size;r++){
        for(int c=0;c<size;c++){
            if(hasQueen(r,c)) board+= "|1|";
            else board+= "|0|";
            if(c==size-1)board+= "\n";
        }
    }
    System.out.println(board);
}

public void putQueen(int row, int col){
    chessBoard[row][col]=1;
}
public void removeQueen(int row, int col){
    chessBoard[row][col]=0;
}
public boolean hasQueen(int row, int col){
    if(chessBoard[row][col]==1) return true;
    else return false;
}
```

Problems @ Javadoc Declaration Console

<terminated> ChessBoard [Java Application] C:\Program Files\Java\jre1.8.0_101\bin\javaw.exe (12.06

```
|1||0||0||0||0||0||0||0|
|0||0||0||0||1||0||0||0|
|0||0||0||0||0||0||0||1|
|0||0||0||0||0||0||1||0|
|0||0||1||0||0||0||0||0|
|0||0||0||0||0||0||1||0|
|0||1||0||0||0||0||0||0|
|0||0||0||1||0||0||0||0|
```

2. Write a method for determining if the current board state contains a queen that is threatening another one. What is the complexity of your method in terms of N, the number of rows on the board? If your algorithm is worse than linear, you might want to look for something better.

Hier nutzen wir die überlegungen aus dem Prelab - wir wussten wir wrauchen einen horizontalen, diagonalen und aufsteigend und abfallenden diagonalen-Check. Repräsendiert mit den Änderungen der x,y Koordinaten (oder col,row) wären das:

horizontale linie: Gefahr, wenn y1==y2;

vertikale linie: Gefahr, wenn x1==x2;

diagonal aufsteigend: Gefahr, wenn x1==x2+n && y1 == y2-n

diagonal absteigend: Gefahr, wenn x1==x2+n && y1 == y2+n

wobei n eine positive oder negative zahl ist, welche zusammen mit der Koordinate nicht über die größe Des gesamten Feldes hinausgeht

(y/x1 ist immer das Feld welches überprüft werden soll, y/x2 sind alle anderen verfügbaren felder mit einer Dame)

Diese Überlegung implementierten wir in der singleFieldThreatCheck Methode, welche so jedes einzelne Feld mit anderen Damenfeldern vergleicht. Diese methode wird von der wholeBoardCheck Methode für jedes Feld aufgerufen - daher steigt die Laufzeit dieses Alghorythmus bei vergrösertem Feld quadratisch. Dies haben wir aufgrund der schwierigkeit der folgenden Aufgabe nicht geschafft zu verbessern.

3. We speak of “backtracking” when we go back to a previous state and try a different branch. Use some coins on a paper chess board to figure out what to do when you reach a state in which one queen is threatened by another. There are iterative, recursive, and random solutions to this problem. Try and implement a recursive solution!

unser Alghorythmus ging das ganze Feld Reihe für Reihe durch

Die idee war, sich die vorher gesetzten Queens zu merken, und wenn es mit dem Alghorythmus nicht weiter geht, geht man eine Reihe zurück und setzt die Queen diesmal woanders hin, nur nicht da wo sie letztens stand. So implementierten wir bei uns auch später das backtracking, welches aufgrund des arrays, welches sich die zuletzt gesetzten Queens merkte - mehrere Schritte/Reihen zurück gehen konnte.

4. Now implement a search routine that looks for a state in which the queens don’t threaten each other. If there is a solution, print it to System.out. If there is more than one solution, print them as well. What is the complexity of your algorithm? Does your program work for 10 queens on a 10 x 10 board? 13 on a 13 x 13 board?

Dies war die wohl schwierigste Aufgabe, die großes Kopfzerbrechen bereitet hatte. Die grundlegende Idee war es eine Methode zu schreiben, welche eine Reihe als Parameter bekommt - in dieser Reihe setzt diese eine Dame, falls dies möglich ist, falls nicht, wird diselbe Methode für eine Reihe davor aufgerufen und die zuvor gesetzte Dame gelöscht um mit Backtracking einen anderen Weg zu finden - dies wird so oft wie nötig wiederholt. Das schwierigst war es sich die zuletzt gesetzten Damen zu merken und einen Grund zu finden das Rekursive Aufrufen zu stoppen - dazu nutzten wir anfangs einen Integer, danach einen Stack, weil das Backtracking ja manchmal auch mhrere Reihen zurück gehen musste.

Schileßlich entschieden wir uns einen Array zu benutzen, der unserem Schachfeld gleicht, jedoch mit ner 1 die Felder speichert wo schon eine Dame hingestellt wurde und es nicht passte - dies sorgt dafür, das das backtracking sich die davor falsch gewählten Felder merkt und sie nicht wieder benutzt. Die letzte schwierigkeit war es sicherzustellen, dass dieser Array mit den "badFields" beim Backtracking für die darunterliegenden Felder geleert wird, da es sonst bei Schachfeldern größer als 7x7 nicht funktioniert, weil anscheinend alleFelder dann als falsch gespeichert sein würden.

```
public void fillRowRecursive(int row){

        boolean finished = false;

        for(int i=0;i<size;i++){
            if(!singleFieldThreatCheck(row,i)&&!badFieldsCheck(row,i)){
                putQueen(row,i);
                badFields[row][i]=1;
                if(row<size-1)fillRowRecursive(row+1);
                finished = true;
                break;
            }
        }
        //when no place for the queen was found -> backtrack
        if(!finished){
            // den ganzen row oben säubern
            for(int i =0;i<size;i++){
                removeQueen((row-1),i);
            }
            //die bereits als falsch bewerteten Felder wieder leeren
            clearUpperbadFieldCheck(row);
            //rekursiver Aufruf - einen row drüber
            fillRowRecursive(row-1);
        }
        }

    }
```

Da das programm flexibel mit einem size-Parameter geschrieben wurde, reicht es aus eine Variable zu ändern um das Board beliebig von 1 bis 15x15 zu skalieren - über 15 gibt es einen StackOverflow error.