Portfolio

AP – AA

9-1-2016

HAN

Remco van Alen (507276) en Bas van Summeren (479334)

# Inhoudsopgave

[Inhoudsopgave 1](#_Toc440122066)

[1 Inleiding 2](#_Toc440122067)

[2 Weekopdrachten 3](#_Toc440122068)

[2.1 Week 1 – Partition 3](#_Toc440122069)

[2.2 Week 2 – Balanced Tree 4](#_Toc440122070)

[2.3 Week 3 – Brute Force 5](#_Toc440122071)

[2.4 Week 4 – Graph algorithm 7](#_Toc440122072)

[2.5 Week 5 – Backtracking 8](#_Toc440122073)

[2.6 Week 6 – Approximation 10](#_Toc440122074)

[3 Eindopdrachten Remco 11](#_Toc440122075)

[3.1 Slotprobleem 2a – Geld Teruggeven 11](#_Toc440122076)

[3.2 Slotprobleem 10 - Anagrammen 11](#_Toc440122077)

[4 Eindopdrachten Bas 12](#_Toc440122078)

[4.1 Slotprobleem 12](#_Toc440122079)

[4.2 Slotprobleem 12](#_Toc440122080)

# Inleiding

# Weekopdrachten

## Week 1 – Partition

### De opdracht

### De uitvoering

### Het resultaat

## Week 2 – Balanced Tree

### De opdracht

Gekozen voor de AVL tree

### De uitvoering

### Het resultaat

## Week 3 – Brute Force

### De opdracht

De gekozen opgave is “Sudoku problem by brute force”. Dit houdt in dat een sudoku doormiddel van het brute force algorithme moet worden opgelost.

### De uitvoering

Voor het uitvoeren van deze opdracht is eerst het spelbord gerealiseerd. 9 rijen van 9 vakjes, 9 kolommen van 9 vakjes en 9 blokken van 3x3 vakjes. Door het gebruikt van pointers worden er in totaal maar 81 vakjes gemaakt. Dit geeft het onderstaande resultaat:



Daarna zijn de spelregels toegepast. Er is een functie geschreven die controleert of een bepaald getal al voorkomt in een bepaald rij. Dit is ook gedaan voor kolommen en blokken van 3x3.

Het algoritme bevat geen intelligentie. Er wordt eerst op het eerste vakje een ‘1’ geplaatst. Als dit vakje al vooraf is ingevuld zal deze worden overgeslagen. Anders wordt er gecontroleerd of deze ‘1’ aan de spelregels voldoet. Als dit een valide set is wordt de ‘1’ geplaatst en anders wordt het volgende getal geprobeerd. Net zo lang totdat het bord vol is. Get kan natuurlijk dat dit niet uitkomt, bijvoorbeeld als er geen valide mogelijkheid is voor een leeg vakje. Dan wordt er doormiddel van backtracking een stap terug gedaan, omdat alle volgende mogelijkheden kunnen worden uitgesloten van een goede oplossing.

Wanneer er een zet terug gedaan zal dat nummer worden opgehoogd. Als dit tot en met het getal ‘9’ ook niet meer kan wordt er nog een stap terug gezet totdat er een getal kan worden verandert. Daarna zal de richting weer vooruitgaan. Wanneer er bij het eerste vakje geen mogelijkheid meer is kan er worden geconcludeerd dat de sudoku niet kan worden opgelost.

### Het resultaat

Dit is de opgeloste sudoku:



Het algoritme:

Er is geen gebruik gemaakt van recursie, maar wanneer dit wel was toegepast had dit tot forse vermindering van code kunnen leiden.



**for** (**int** rowI = 0; rowI < size; rowI++) *// Per rij*  
{  
 **for** (**int** columnI = 0; columnI < size; columnI++) *// Per kolom*  
 {  
 **if** (! bord.isDefaultNumber(rowI, columnI)) *// Wanneer er geen default getal is ingevuld*  
 {  
 hasNewValue = **false**;  
 goBack = **false**;  
 **if** (bord.getValue(rowI, columnI) != **null**)  
 {  
 oldValue = bord.getValue(rowI, columnI);  
 } **else** {  
 oldValue = 0;  
 }  
 **while** (hasNewValue == **false**) *// Zolang er geen nieuwe waarde is*  
 {  
 counterSteps ++;  
 **if** (oldValue < 9) *// Als het huidige getal nog niet maximaal is*  
 {  
 newValue = oldValue + 1;  
 **if** (bord.isValidValue(rowI, columnI, newValue)) *// Controle van zet* {

bord.setValue(rowI, columnI, newValue, **false**);  
 hasNewValue = **true**;  
  
 } **else** {  
 oldValue = newValue;  
 }  
 }**else if** (columnI >= 1) *Wanneer het niet de eerste kolom is*  
 {  
 bord.setValue(rowI, columnI, 0, **false**);  
 columnI = columnI - 2;  
 goBack = **true**;  
 **break**;  
 } **else if** (rowI >= 1) *Wanneer het niet de eerste rij is*  
 {  
 bord.setValue(rowI, columnI, 0, **false**);  
 rowI = rowI - 1;  
 columnI = size - 2;  
 goBack = **true**;  
 **break**;  
 }  
 }  
 } **else if** (goBack == **true**) *// Als er een zet achteruit wordt gemaakt*  
 {  
 **if** (columnI >= 1) *// Wanneer het niet de eerste kolom is*  
 {  
 columnI = columnI - 2; *// -2 omdat de for-loop dit weer met 1 ophoogt*  
 goBack = **true**;  
 } **else if** (rowI >= 1) *// Wanneer het niet de eerste rij is*  
 {  
 rowI = rowI - 1;  
 columnI = size - 2; *-2 omdat de for-loop dit weer met 1 ophoogt*  
 goBack = **true**;  
 } **else**{ *// Als het de eerste kolom van de eerste rij is*  
 rowI = size;  
 columnI = size;  
 System.***out***.println(**"This puzzle cannot be solved!!"**);  
 **break**;  
 }

## Week 4 – Graph algorithm

### De opdracht

Gekozen voor greedy

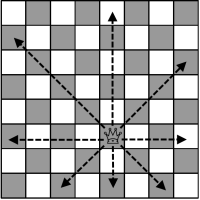
### De uitvoering

### Het resultaat

## Week 5 – Backtracking

### De opdracht

De gekozen opgave is “n-Queens-problem”. Voor deze puzzel worden er N queens om een schaakbord geplaatst van N\*N groot waarbij de queens elkaar niet kunnen slaan. Op de onderstaande afbeelding staat een queen met de mogelijk staprichtingen weergegeven.



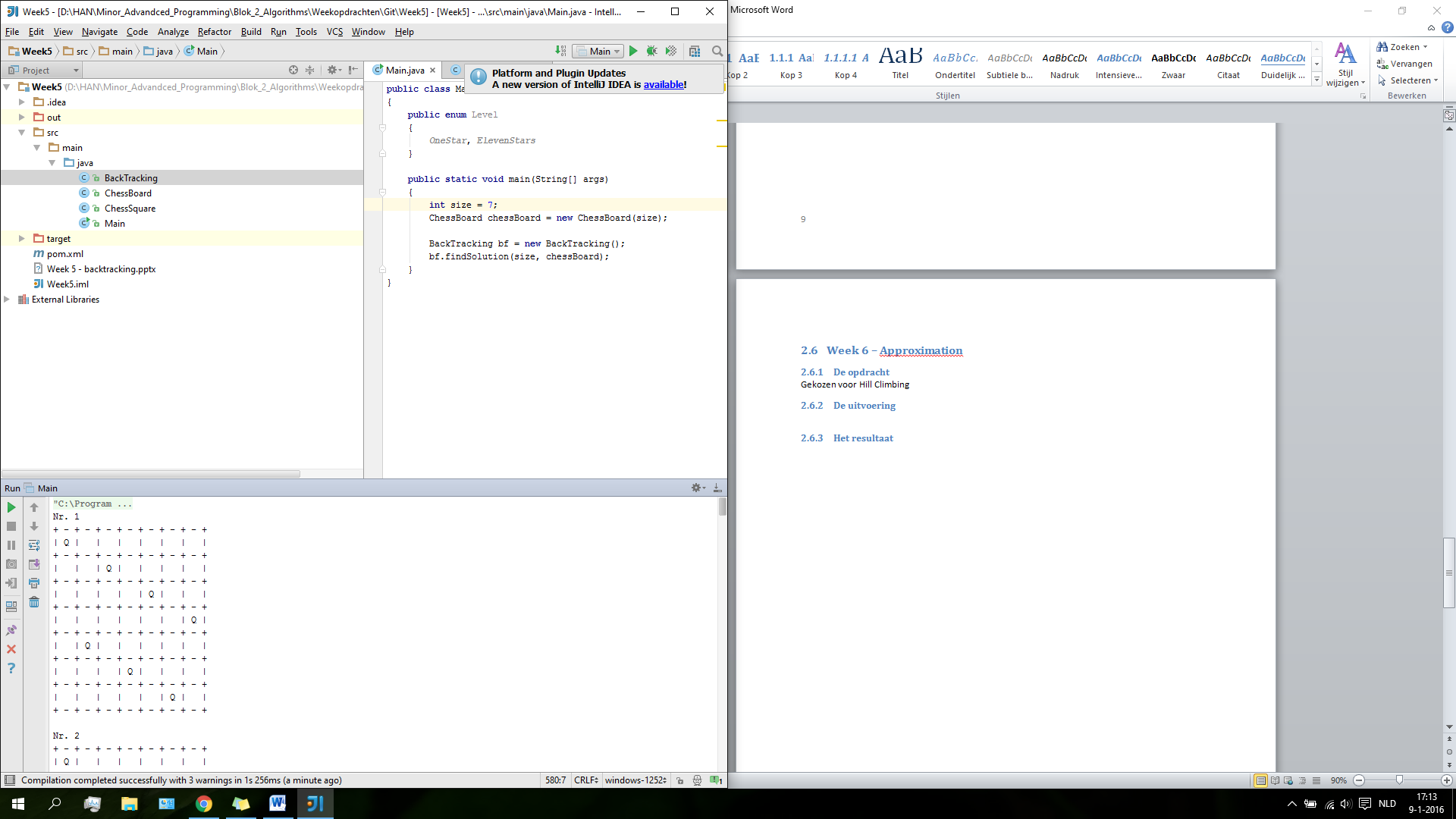
### De uitvoering

Er is gebruik gemaakt van de sudoku opdracht. Daardoor was het spelbord al af. De spelregels zijn wel aangepast. Er moet nog steeds een horizontale en verticale controle plaatsen vinden, maar niet een blok van 3 x 3. Daarentegen komt er wel een extra controle voor de diagonalen.

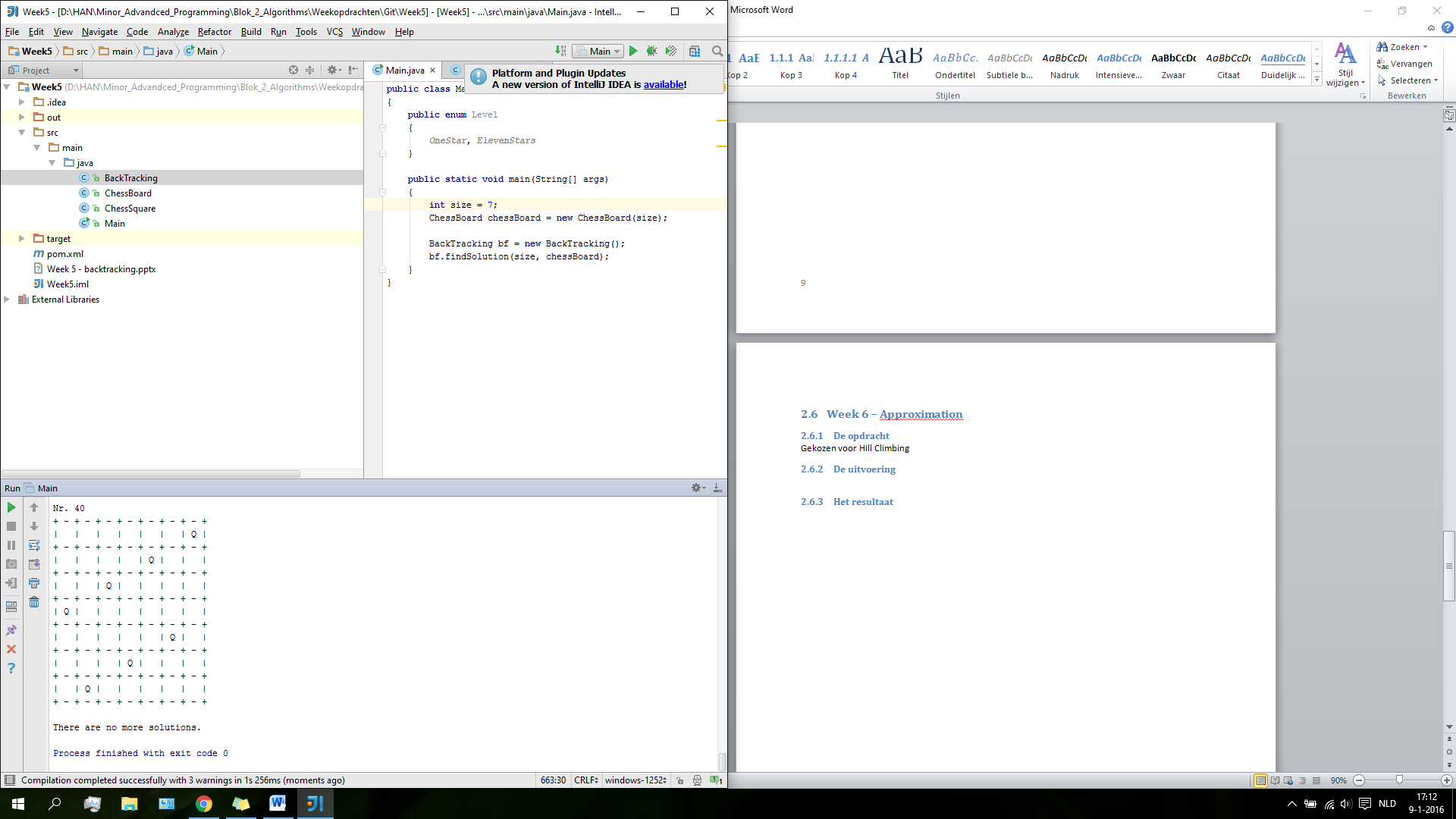
Wanneer er een queen wordt geplaatst wordt de rij niet afgemaakt, maar wordt er gelijk naar de volgende rij gekeken. Als bij het laatst vakje blijkt dat de rij geen queen bevat zal backtracking worden toegepast. Dit houdt in dat er terug wordt gekeken en dat de laatst geplaatste queen wordt weggehaald. Daarna zal er ergens anders op die rij alsnog een queen worden geplaatst. Als er nergens op die rij een nieuwe queen kan worden geplaatst zal er nog een stap terug worden gezet. Het kan voorkomen dat er bij het eerste vakje van de eerste kolom een stap terug met worden gezet en dan kan geconcludeerd worden dat de puzzel niet op telossen is. Dit kan alleen voorkomen bij een spelbord van 2x2 en 3x3 groot.

### Het resultaat

Dit algoritme gaat door totdat er geen andere oplossingen meer zijn. Bij een veld van 7 x 7 worden er 40 oplossingen gevonden.



…



Het algoritme:

Ook hier is er geen gebruik gemaakt van recursie en dit had wel tot forse vermindering van code kunnen leiden.

**for** (**int** rowI = 0; rowI < size; rowI++)  
{  
 **for** (**int** columnI = 0; columnI < size; columnI++)  
 {  
 **if** (!goBack && bord.isValidValue(rowI, columnI, **true**))  
 {  
 bord.setQueenValue(rowI, columnI, **true**);  
 goBack = **false**;   
 }  
 **else if**(goBack && bord.isQueen(rowI, columnI))  
 {  
 bord.setQueenValue(rowI, columnI, **false**);  
 goBack = **false**;  
 }  
 **if**(columnI >= size - 1)  
 {  
 **if**(!bord.rowContainsQueen(rowI))  
 {

goBack = **true**;  
 }  
 }  
 **if**(goBack)  
 {  
 ….

}  
 }  
}

## Week 6 – Approximation

### De opdracht

Gekozen voor Hill Climbing

### De uitvoering

### Het resultaat

# Eindopdrachten Remco

## Slotprobleem 2a – Geld Teruggeven

## Slotprobleem 10 - Anagrammen

# Eindopdrachten Bas

## Slotprobleem

## Slotprobleem