Hashing

Practicum 3

Bart Endhoven 500761169 & Peter Sabel (Zabel) 500759473

2017-2018

Hashing

Practicum 3

AUTEUR

Bart Endhoven 500761169 & Peter Sabel (Zabel) 500759473

Datum

6 oktober 2017

VAK

Datastructure

Versie

2.0

© 2016 Copyright Hogeschool Amsterdam

Alle rechten voorbehouden. Niets uit deze uitgave mag worden verveelvoudigd, opgeslagen in een geautomatiseerd gegevensbestand, of openbaar gemaakt, in enige vorm of op enige wijze, hetzij elektronisch, mechanisch, door print-outs, kopieën, of op welke manier dan ook, zonder voorafgaande schriftelijke toestemming van de Hogeschool Amsterdam.

Inhoudsopgave

[Inleiding. 4](#_Toc496298376)

[1. Opdracht 1. 5](#_Toc496298377)

[2. Opdracht 1.1 7](#_Toc496298378)

[2.1 1.1 Output. 8](#_Toc496298379)

[3. Opdracht 1.2 11](#_Toc496298380)

[4. Opdracht 1.3 12](#_Toc496298381)

[Samenvatting. 13](#_Toc496298382)

[Bronnen 14](#_Toc496298383)

[5. Gehele code. 15](#_Toc496298384)

[5.1 Main: 15](#_Toc496298385)

[5.2 Student class: 16](#_Toc496298386)

[5.3 Queue: 17](#_Toc496298387)

[5.4 Bucket: 18](#_Toc496298388)

[5.5 Bucket sort: 19](#_Toc496298389)

[5.6 Klas Generator: 20](#_Toc496298390)

Inleiding.

In opdracht 2 gaan we aan de slag met arrays. Het sorteren van deze arrays en het tellen van het aantal dezelfde elementen. Ook itereren wij door de lijst met klassen heen zodat alle 10.000 leerlingen worden verdeeld in de 5 soorten klassen.

# Opdracht 1.1

Er werd gevraagd een fuctie te schrijven om de Ldap van een student om te zetten in een hashcode. Wij hebben gekozen voor een Long omdat onze verzonnen hash niet in een Int past.

/\*\*

**\*** creer een hashcode van 9 cijfers waarbij het eerste cijfer de lengte is

**\*** de string van de letters worden omgezet in een cijfer afhankelijk van de

**\*** plek in het alfabet. a = 01**,** b = 02 etc.

**\*** **@param** ldap

**\*** **@return**

     \*/

**public** **static** long hashing(String ldap) {

long hash;

int lengte = ldap.length();

**List**<**String**> t = **new** ArrayList<>();

int cijfer1 = 0;

int cijfer2 = 0;

int cijfer3 = 0;

int cijfer4 = 0;

**for** (int i = 0; i < ldap.length(); ++i) {

int n;

**try** {

char ch = ldap.charAt(i);

n = (int) ch - (int) 'a' + 1;

} **catch** (**Exception** e) {

n = 0;

}

t.add(**String**.valueOf(n));

}

**String**[] cijfers = **new** **String**[t.size()];

t.toArray(cijfers);

**try** {

cijfer1 = **Integer**.parseInt(cijfers[0]);

cijfer2 = **Integer**.parseInt(cijfers[1]);

cijfer3 = **Integer**.parseInt(cijfers[2]);

} **catch** (**Exception** e) {

**System**.out.println("somthing went wrong");

}

**try** {

cijfer4 = **Integer**.parseInt(cijfers[3]);

} **catch** (**Exception** e) {

}

hash = (lengte \* 100000000) + (cijfer1 \* 1000000) + (cijfer2 \* 10000) + (cijfer3 \* 100) + (cijfer4);

**return** hash;

}

Daarna werd er gevraagt om alle Ldap’s van de studenten te hashen met onze geschreven functie. De modulo 97 hiervan te nemen en deze om te zetten in een grafiek. Dit is gedaan via Excel.

Hier zijn we dat de modulo 9 maar 6 keer voorkomt en de modulo 64 zelfs 23 keer voorkomt.

# Opdracht 1.2

Er moest een lijst komen waarin je kon aantonen dat er niet vaak hetzelfde cijfer werd getoond. Zodat alle cijfers tussen 1.1 en 10.0 eerlijk verdeeld waren.

Hiervoor hebben wij een treemap gebruikt. Deze sorteert automatisch de key input en daarachter hebben wij de count bijgehouden in de value van de map.

Wij hebben stiekem het afgeronde double getal overgenomen uit uw eigen voorbeeld.

**package** nl.hva.dmci.ict.se.datastructures;

**import** java.util.ArrayList;

**import** java.util.List;

**import** java.util.Map;

**import** java.util.Random;

**import** java.util.TreeMap;

/\*\*

**\***

**\*** **@author** **Bart & Peter**

 \*/

**public** **class** Main {

/\*\*

**\*** **@param** args the command line arguments

     \*/

**public** **static** void main(**String**[] args) {

**final** int AANTAL\_STUDENTEN = 10000;

Student[] studenten = **new** Student[AANTAL\_STUDENTEN];

**for** (int i = 0; i < AANTAL\_STUDENTEN; i++) {

**Random** rand = **new** **Random**();

double afronden = (rand.nextInt(91) + 10)/10.0;

studenten[i] = **new** Student(i, afronden);

}

studenten = Student.sorterenStudenten(studenten);

**for** (Student student : studenten) {

**System**.out.println(student);

}

**List** lijstje = **new** ArrayList();

**for** (Student student : studenten) {

lijstje.add(student);

}

//System.out.println(new Rijtjes().isStijgend(lijstje));

Map<**Double**, **Integer**> map = **new** TreeMap<**Double**, **Integer**>();

**for** (Student temp : studenten) {

**Integer** count = map.get(temp.getToetsCijfer());

map.put(temp.getToetsCijfer(), (count == **null**) ? 1 : count + 1);

}

printCijfers(map);

}

**public** **static** void printCijfers(Map<**Double**, **Integer**> map) {

**for** (Map.Entry<**Double**, **Integer**> entry : map.entrySet()) {

**System**.out.println("Cijfer : " + entry.getKey() + " Aantal : "

+ entry.getValue());

}

}

}

## 1.1 Output.

Cijfer : 1.0 Aantal : 120

Cijfer : 1.1 Aantal : 98

Cijfer : 1.2 Aantal : 99

Cijfer : 1.3 Aantal : 122

Cijfer : 1.4 Aantal : 120

Cijfer : 1.5 Aantal : 128

Cijfer : 1.6 Aantal : 108

Cijfer : 1.7 Aantal : 108

Cijfer : 1.8 Aantal : 98

Cijfer : 1.9 Aantal : 112

Cijfer : 2.0 Aantal : 117

Cijfer : 2.1 Aantal : 102

Cijfer : 2.2 Aantal : 112

Cijfer : 2.3 Aantal : 115

Cijfer : 2.4 Aantal : 126

Cijfer : 2.5 Aantal : 99

Cijfer : 2.6 Aantal : 107

Cijfer : 2.7 Aantal : 96

Cijfer : 2.8 Aantal : 103

Cijfer : 2.9 Aantal : 87

Cijfer : 3.0 Aantal : 110

Cijfer : 3.1 Aantal : 116

Cijfer : 3.2 Aantal : 105

Cijfer : 3.3 Aantal : 111

Cijfer : 3.4 Aantal : 116

Cijfer : 3.5 Aantal : 106

Cijfer : 3.6 Aantal : 123

Cijfer : 3.7 Aantal : 101

Cijfer : 3.8 Aantal : 125

Cijfer : 3.9 Aantal : 95

Cijfer : 4.0 Aantal : 100

Cijfer : 4.1 Aantal : 107

Cijfer : 4.2 Aantal : 95

Cijfer : 4.3 Aantal : 110

Cijfer : 4.4 Aantal : 101

Cijfer : 4.5 Aantal : 111

Cijfer : 4.6 Aantal : 102

Cijfer : 4.7 Aantal : 109

Cijfer : 4.8 Aantal : 98

Cijfer : 4.9 Aantal : 107

Cijfer : 5.0 Aantal : 96

Cijfer : 5.1 Aantal : 124

Cijfer : 5.2 Aantal : 120

Cijfer : 5.3 Aantal : 113

Cijfer : 5.4 Aantal : 105

Cijfer : 5.5 Aantal : 110

Cijfer : 5.6 Aantal : 108

Cijfer : 5.7 Aantal : 105

Cijfer : 5.8 Aantal : 119

Cijfer : 5.9 Aantal : 106

Cijfer : 6.0 Aantal : 118

Cijfer : 6.1 Aantal : 123

Cijfer : 6.2 Aantal : 103

Cijfer : 6.3 Aantal : 126

Cijfer : 6.4 Aantal : 104

Cijfer : 6.5 Aantal : 114

Cijfer : 6.6 Aantal : 101

Cijfer : 6.7 Aantal : 118

Cijfer : 6.8 Aantal : 113

Cijfer : 6.9 Aantal : 110

Cijfer : 7.0 Aantal : 112

Cijfer : 7.1 Aantal : 106

Cijfer : 7.2 Aantal : 102

Cijfer : 7.3 Aantal : 101

Cijfer : 7.4 Aantal : 136

Cijfer : 7.5 Aantal : 97

Cijfer : 7.6 Aantal : 129

Cijfer : 7.7 Aantal : 108

Cijfer : 7.8 Aantal : 102

Cijfer : 7.9 Aantal : 101

Cijfer : 8.0 Aantal : 110

Cijfer : 8.1 Aantal : 111

Cijfer : 8.2 Aantal : 112

Cijfer : 8.3 Aantal : 91

Cijfer : 8.4 Aantal : 102

Cijfer : 8.5 Aantal : 112

Cijfer : 8.6 Aantal : 113

Cijfer : 8.7 Aantal : 118

Cijfer : 8.8 Aantal : 99

Cijfer : 8.9 Aantal : 104

Cijfer : 9.0 Aantal : 102

Cijfer : 9.1 Aantal : 125

Cijfer : 9.2 Aantal : 124

Cijfer : 9.3 Aantal : 122

Cijfer : 9.4 Aantal : 119

Cijfer : 9.5 Aantal : 107

Cijfer : 9.6 Aantal : 120

Cijfer : 9.7 Aantal : 112

Cijfer : 9.8 Aantal : 113

Cijfer : 9.9 Aantal : 114

Cijfer : 10.0 Aantal : 115

# Opdracht 1.2

in deze opdracht werd er gevraagd om de array vol met studenten en cijfers te sorteren op de cijfers. Als studenten hetzelfde cijfer hebben worden ze gesorteerd op studentnummer.

De Class

@Override

**public** int compareTo(Student s) {

**if** (**this**.getToetsCijfer() == s.getToetsCijfer()) {

**return** **this**.getStudentnummer() - s.getStudentnummer();

} **else** **if** (**this**.getToetsCijfer() < s.getToetsCijfer()) {

**return** -1;

} **else** {

**return** 1;

}

}

**public** **static** Student[] sorterenStudenten(Student[] studentenlijst) {

Student temp;

**for** (int i = 1; i < studentenlijst.length; i++) {

**for** (int j = i; j > 0; j--) {

**if** (studentenlijst[j] != **null**) {

**if** (studentenlijst[j].compareTo(studentenlijst[j - 1]) < 0) {

temp = studentenlijst[j];

studentenlijst[j] = studentenlijst[j - 1];

studentenlijst[j - 1] = temp;

}

}

}

}

**return** studentenlijst;

}

De Main

studenten = Student.sorterenStudenten(studenten);

for (Student student : studenten) {

**System**.out.println(student);

}

List lijstje = new ArrayList();

for (Student student : studenten) {

lijstje.add(student);

}

System.out.println(new Rijtjes().isStijgend(lijstje));

# Opdracht 1.3

Samenvatting.

Helaas hebben wij niet meer kunnen bereiken dan deze opdracht. Al aangegeven door ziekte van Bart hebben wij niet samen kunnen zitten om tot een oplossing te komen.

Het maken van de opdrachten vonden wij erg moeilijk maar was zeer interessant.

Bronnen

(mkyoung, 2012)

# Gehele code.

## Main:

## Student class:

## Queue:

## Bucket:

## Bucket sort:

## Klas Generator: