Matej Vukosav

0036473765

ROVKP 4.Laboratorijska vjezba

1. Zadatak

```
import java.io.FileWriter;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.Comparator;
import java.util.stream.Stream;

/**
 * Created by Vuki on 5.6.2017..
 */
public class AirPollution {

    private static final String OUTPUT_DATA =
"ApacheSparkLab/pollutionData-all.csv";
    private static final String CSV_DELIMITER = ",";

    /**
     * Cilj zadatka je ucitati sve datoteke s mjerenjima senzorskih
postaja i kao izlaz dobiti jednu izlaznu datoteku
     * sa svim ocitanjima.
     * Ova datoteka ce biti koristena u 3. zadatku kao ulaz generatora
toka senzorskih podataka.
     *
     * @param args
     */
    public static void main(String[] args) {

        AirPollution airPollution = new AirPollution();
        airPollution.parseInput("ApacheSparkLab/pollutionData");

    }

    private void parseInput(String folderName) {
        Stream<PollutionReading> data = null;
        try {
            data = Files.list(Paths.get(folderName))
                    .filter(Files::isRegularFile)
                    .flatMap(p -> {
                        try {
                            return Files.lines(p);
                        } catch (IOException e) {
                            e.printStackTrace();
                        }
                        return null;
                    }).filter(PollutionReading::canParse)
                    .map(PollutionReading::new)
                    .sorted(Comparator.comparing(pollutionReading ->
pollutionReading.timestamp));
```

```
        } catch (IOException e) {
            e.printStackTrace();
        }

        writeToFile(data);


    }

    @SuppressWarnings("Duplicates")
    private void writeToFile(Stream<PollutionReading>
pollutionReadingStream) {

        if (pollutionReadingStream == null) {
            return;
        }
        try {
            FileWriter fileWriter = new FileWriter(OUTPUT_DATA);
            pollutionReadingStream.forEach(data -> {
                try {
                    fileWriter.append(data.toString());
                    fileWriter.append(System.lineSeparator());
                } catch (IOException e) {
                    e.printStackTrace();
                }
            });

        } catch (IOException e) {
            e.printStackTrace();
        }
    }


    /**
     *
ozone,particullate_matter,carbon_monoxide,sulfure_dioxide,nitrogen_dioxi
de,longitude,latitude,timestamp
     * 101,94,49,44,87,10.104986076057457,56.23172069428216,2014-08-01
00:05:00
     */
    static class PollutionReading {

        int ozone;
        int particullateMatter;
        int carbonMonoxide;
        int sulfure_dioxide;
        int nitrogen_dioxide;
        double longitude;
        double latitude;
        String timestamp;

        public PollutionReading(String line) {
            String[] data = line.split(",");
            this.ozone = Integer.parseInt(data[0]);
            this.particullateMatter = Integer.parseInt(data[1]);
            this.carbonMonoxide = Integer.parseInt(data[2]);
            this.sulfure_dioxide = Integer.parseInt(data[3]);
            this.nitrogen_dioxide = Integer.parseInt(data[4]);
            this.longitude = Double.parseDouble(data[5]);
            this.latitude = Double.parseDouble(data[6]);
            this.timestamp = data[7];
```

```
        }

        public static boolean canParse(String line) {
            try {
                String[] data = line.split(",");
                int ozone = Integer.parseInt(data[0]);
                int particullateMatter = Integer.parseInt(data[1]);
                int carbonMonoxide = Integer.parseInt(data[2]);
                int sulfure_dioxide = Integer.parseInt(data[3]);
                int nitrogen_dioxide = Integer.parseInt(data[4]);
                double longitude = Double.parseDouble(data[5]);
                double latitude = Double.parseDouble(data[6]);
                String timestamp = data[7];
            } catch (Exception e) {
                return false;
            }
            return true;
        }

        @Override
        public String toString() {
            return ozone + CSV_DELIMITER
                    + particullateMatter + CSV_DELIMITER
                    + carbonMonoxide + CSV_DELIMITER
                    + sulfure_dioxide + CSV_DELIMITER
                    + nitrogen_dioxide + CSV_DELIMITER
                    + longitude + CSV_DELIMITER
                    + latitude + CSV_DELIMITER
                    + timestamp;
        }
    }


}
```

2.Zadatak

```
import org.apache.log4j.Level;
import org.apache.log4j.Logger;
import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaPairRDD;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;
import org.apache.spark.storage.StorageLevel;
import scala.Tuple2;

import java.util.List;
import java.util.NoSuchElementException;

/**
 * 1. Koliko je zenskih osoba umrlo u lipnju kroz citav period?
 * - In june was born 100654 females.
 * 2. Koji dan u tjednu je umrlo najvise muskih osoba starijih od 50
godina?
 * Most males older than 50 dies at 4 day of week.
 * 3.Koliko osoba je bilo podvrgnuto obdukciji nakon smrti?
 * 203681 peoples past authopsy after death.
 * 4. Kakvo je kretanje broja umrlih muskaraca u dobi izmedu 45 i 65 godina
po mjesecima?
```

```
 * (Rezultat je sortirana lista tipa Pair2(kljuc je redni broj mjeseca, a
vrijednost je broj umrlih muskaraca))
 * Male deaths between 46 and 65 age:
 * Month : NumOfDeaths
 * 1  29881
 * 2  25510
 * 3  26891
 * 4  25516
 * 5  25679
 * 6  24989
 * 7  25713
 * 8  25384
 * 9  24584
 * 10  26083
 * 11  25821
 * 12  27531
 * 5. Kakvo je kretanje postotka umrlih ozenjenih muskaraca u dobi izmedu
45 i 65 godina po mjesecima?
 * Male deaths between 45 and 65 age per month percentage:
 * Month : Percentage
 * 1  42.88678424416854
 * 2  43.669149353194825
 * 3  43.802759287493956
 * 4  44.14485029001411
 * 5  43.91915573036333
 * 6  43.81527872263796
 * 7  43.386613775133206
 * 8  43.681058934762056
 * 9  43.46729580214774
 * 10  43.737300157190504
 * 11  44.08427249138298
 * 12  42.92615596963423
 * 6.Koji je ukupni broj umrlih u nesreci (kod 1) u cjelokupnom periodu?
 * 132684 peoples died from accident
 * 7. Koliki je broj razlicitih godina starosti umrlih osoba koji se
pojavljuju u zapisima?
 * There are 117 different death ages.
 * <p>
 * Created by Vuki on 6.6.2017..
 */
public class DeathAnalisysUSA {

    private static final String INPUT_FILE =
"ApacheSparkLab/DeathRecords.csv";

    public static void main(String[] args) {

        Logger.getLogger("org").setLevel(Level.ERROR);
        Logger.getLogger("akka").setLevel(Level.ERROR);

        DeathAnalisysUSA deathAnalisysUSA = new DeathAnalisysUSA();

        JavaSparkContext context = deathAnalisysUSA.getJavaSparkContext();
        JavaRDD<USDeathRecord> records =
deathAnalisysUSA.parseInput(context, INPUT_FILE);

        records.persist(StorageLevel.MEMORY_AND_DISK());

        //deathAnalisysUSA.calculateFemaleDeathInJune(records);

//deathAnalisysUSA.calculateDayInWeekWhereMaleOlderThan50Died(records);
```

```
        //deathAnalisysUSA.calculateObductionDeath(records);


//deathAnalisysUSA.calculateMaleDeathBetween45And65ByMonth(records);

deathAnalisysUSA.calculateMaleDeathBetween45And65ByMonthPercentage(records)
;
        //deathAnalisysUSA.calculateAccidentDeaths(records);
        //deathAnalisysUSA.calculateDifferentDeathAges(records);

    }

    private JavaRDD<USDeathRecord> parseInput(JavaSparkContext context,
String inputFile) {
        JavaRDD<String> lines = context.textFile(inputFile);
        return lines.filter(USDeathRecord::canParse)
                .map(USDeathRecord::new);
    }


    private JavaSparkContext getJavaSparkContext() {
        SparkConf conf = new SparkConf().setAppName("ChildNames");
        //set the master if not already set through the command line
        try {
            conf.get("spark.master");
        } catch (NoSuchElementException e) {
            conf.setMaster("local");
        }
        return new JavaSparkContext(conf);
    }


    private void calculateFemaleDeathInJune(JavaRDD<USDeathRecord> records)
{

        long count = records.filter(usDeathRecord -> usDeathRecord.Sex ==
'F')
                .filter(usDeathRecord -> usDeathRecord.MonthOfDeath == 6)
                .count();

        System.out.println("In june was born " + count + " females.");

    }

    private void
calculateDayInWeekWhereMaleOlderThan50Died(JavaRDD<USDeathRecord> records)
{


        Tuple2<Integer, Integer> day = records.filter(usDeathRecord ->
usDeathRecord.Sex == 'M')
                .filter(usDeathRecord -> usDeathRecord.Age > 50)
                .mapToPair(usDeathRecord -> new
Tuple2<>(usDeathRecord.DayOfWeekOfDeath, 1))
                .reduceByKey((x1, x2) -> x1 + x2)
                .mapToPair(Tuple2::swap)
                //ascending
                .sortByKey(false)
                .first();

        System.out.println("Most males older than 50 dies at " + day._2 + "
```

```
day of week.");
    }

    private void calculateObductionDeath(JavaRDD<USDeathRecord> records) {
        long count = records.filter(usDeathRecord -> usDeathRecord.Autopsy
== 'Y')
                .count();

        System.out.println(count + " peoples past authopsy after death.");
    }

    private void
calculateMaleDeathBetween45And65ByMonth(JavaRDD<USDeathRecord> records) {

        //key is month, value is num of male deaths
        JavaPairRDD<Integer, Integer> deathPerMonth =
records.filter(usDeathRecord -> usDeathRecord.Sex == 'M')
                .filter(usDeathRecord -> usDeathRecord.Age > 45 &&
usDeathRecord.Age < 65)
                .mapToPair(usDeathRecord -> new
Tuple2<>(usDeathRecord.MonthOfDeath, 1))
                .reduceByKey((integer, integer2) -> integer + integer2)
                .sortByKey();

        System.out.println("Male deaths between 46 and 65 age per month:");
        System.out.println("Month : NumOfDeaths");
        deathPerMonth.foreach(day -> {
            System.out.println(day._1 + "  " + day._2);
        });
    }

    private void
calculateMaleDeathBetween45And65ByMonthPercentage(JavaRDD<USDeathRecord>
records) {
        //key is month, value is num of male deaths
        JavaPairRDD<Integer, Integer> deathPerMonth =
records.filter(usDeathRecord -> usDeathRecord.Sex == 'M')
                .filter(usDeathRecord -> usDeathRecord.Age > 45 &&
usDeathRecord.Age < 65)
                .filter(usDeathRecord -> usDeathRecord.MaritalStatus ==
'M')
                .mapToPair(usDeathRecord -> new
Tuple2<>(usDeathRecord.MonthOfDeath, 1))
                .reduceByKey((integer, integer2) -> integer + integer2)
                .sortByKey();
        //mjesec, vrijednost
        List<Tuple2<Integer, Integer>> collect =
records.filter(usDeathRecord -> usDeathRecord.Sex == 'M')
                .filter(usDeathRecord -> usDeathRecord.Age > 45 &&
usDeathRecord.Age < 65)
                .mapToPair(usDeathRecord -> new
Tuple2<>(usDeathRecord.MonthOfDeath, 1))
                .reduceByKey((integer, integer2) -> integer + integer2)
                // .mapToPair(day -> new Tuple2<>(0, day._2))
                // .reduceByKey((integer, integer2) -> integer + integer2)
                .sortByKey()
                .collect();


        System.out.println("Male deaths between 46 and 65 age per month
percentage:");
```

```
        System.out.println("Month : Percentage");
        deathPerMonth.foreach(month -> {
            System.out.println(month._1 + "  " + ((double) month._2 /
collect.get(month._1 - 1)._2) * 100);
        });
    }

    private void calculateAccidentDeaths(JavaRDD<USDeathRecord> records) {

        long count = records.filter(usDeathRecord ->
usDeathRecord.MannerOfDeath == 1)
                .count();

        System.out.println(count + " peoples died from accident");
    }


    private void calculateDifferentDeathAges(JavaRDD<USDeathRecord>
records) {
        long count = records.map(usDeathRecord -> usDeathRecord.Age)
                .distinct()
                .count();

        System.out.println("There are " + count + " different death
ages.");
    }

    static class USDeathRecord {
        /**
         * Id,
         * ResidentStatus,
         * Education1989Revision,
         * Education2003Revision,
         * EducationReportingFlag,
         * MonthOfDeath,
         * Sex,
         * AgeType,
         * Age,
         * AgeSubstitutionFlag,
         * AgeRecode52,
         * AgeRecode27,
         * AgeRecode12,
         * InfantAgeRecode22,
         * PlaceOfDeathAndDecedentsStatus,
         * MaritalStatus,
         * DayOfWeekOfDeath,
         * CurrentDataYear,
         * InjuryAtWork,
         * MannerOfDeath,
         * MethodOfDisposition,
         * Autopsy,
         * ActivityCode,
         * PlaceOfInjury,
         * Icd10Code,
         * CauseRecode358,
         * CauseRecode113,
         * InfantCauseRecode130,
         * CauseRecode39,
         * NumberOfEntityAxisConditions,
         * NumberOfRecordAxisConditions,
         * Race,
```

```
         * BridgedRaceFlag,
         * RaceImputationFlag,
         * RaceRecode3,
         * RaceRecode5,
         * HispanicOrigin,
         * HispanicOriginRaceRecode
         *
1,1,0,2,1,1,M,1,87,0,43,23,11,0,4,M,4,2014,U,7,C,N,99,99,I64,238,70,0,24,1,
1,1,0,0,1,1,100,6
         */

        int Id;
        int ResidentStatus;
        int Education1989Revision;
        int Education2003Revision;
        int EducationReportingFlag;
        int MonthOfDeath;
        char Sex;
        int AgeType;
        int Age; //age in moment of death
        int AgeSubstitutionFlag;
        int AgeRecode52;
        int AgeRecode27;
        int AgeRecode12;
        int InfantAgeRecode22;
        int PlaceOfDeathAndDecedentsStatus;
        char MaritalStatus; //M married, D divorced, W widow
        int DayOfWeekOfDeath;
        int CurrentDataYear;
        char InjuryAtWork;
        int MannerOfDeath;
        char MethodOfDisposition;
        char Autopsy; //Y yes, N no, U unknown
        int ActivityCode;
        int PlaceOfInjury;
        String Icd10Code;
        int CauseRecode358;
        int CauseRecode113;
        int InfantCauseRecode130;
        int CauseRecode39;
        int NumberOfEntityAxisConditions;
        int NumberOfRecordAxisConditions;
        int Race;
        int BridgedRaceFlag;
        int RaceImputationFlag;
        int RaceRecode3;
        int RaceRecode5;
        int HispanicOrigin;


        public USDeathRecord(String line) {
            String[] data = line.split(",");

            Id = Integer.parseInt(data[0]);
            ResidentStatus = Integer.parseInt(data[1]);
            Education1989Revision = Integer.parseInt(data[2]);
            Education2003Revision = Integer.parseInt(data[3]);
            EducationReportingFlag = Integer.parseInt(data[4]);
            MonthOfDeath = Integer.parseInt(data[5]);
            Sex = data[6].charAt(0);
            AgeType = Integer.parseInt(data[7]);
```

```
            Age = Integer.parseInt(data[8]);
            AgeSubstitutionFlag = Integer.parseInt(data[9]);
            AgeRecode52 = Integer.parseInt(data[10]);
            AgeRecode27 = Integer.parseInt(data[11]);
            AgeRecode12 = Integer.parseInt(data[12]);
            InfantAgeRecode22 = Integer.parseInt(data[13]);
            PlaceOfDeathAndDecedentsStatus = Integer.parseInt(data[14]);
            MaritalStatus = data[15].charAt(0);
            DayOfWeekOfDeath = Integer.parseInt(data[16]);
            CurrentDataYear = Integer.parseInt(data[17]);
            InjuryAtWork = data[18].charAt(0);
            MannerOfDeath = Integer.parseInt(data[19]);
            MethodOfDisposition = data[20].charAt(0);
            Autopsy = data[21].charAt(0);
            ActivityCode = Integer.parseInt(data[22]);
            PlaceOfInjury = Integer.parseInt(data[23]);
            Icd10Code = data[24];
            CauseRecode358 = Integer.parseInt(data[25]);
            CauseRecode113 = Integer.parseInt(data[26]);
            InfantCauseRecode130 = Integer.parseInt(data[27]);
            CauseRecode39 = Integer.parseInt(data[28]);
            NumberOfEntityAxisConditions = Integer.parseInt(data[29]);
            NumberOfRecordAxisConditions = Integer.parseInt(data[30]);
            Race = Integer.parseInt(data[31]);
            BridgedRaceFlag = Integer.parseInt(data[32]);
            RaceImputationFlag = Integer.parseInt(data[33]);
            RaceRecode3 = Integer.parseInt(data[34]);
            RaceRecode5 = Integer.parseInt(data[35]);
            HispanicOrigin = Integer.parseInt(data[36]);
        }

        public static boolean canParse(String line) {
            try {

                String[] data = line.split(",");

                int Id = Integer.parseInt(data[0]);
                int ResidentStatus = Integer.parseInt(data[1]);
                int Education1989Revision = Integer.parseInt(data[2]);
                int Education2003Revision = Integer.parseInt(data[3]);
                int EducationReportingFlag = Integer.parseInt(data[4]);
                int MonthOfDeath = Integer.parseInt(data[5]);
                char Sex = data[6].charAt(0);
                int AgeType = Integer.parseInt(data[7]);
                int Age = Integer.parseInt(data[8]);
                int AgeSubstitutionFlag = Integer.parseInt(data[9]);
                int AgeRecode52 = Integer.parseInt(data[10]);
                int AgeRecode27 = Integer.parseInt(data[11]);
                int AgeRecode12 = Integer.parseInt(data[12]);
                int InfantAgeRecode22 = Integer.parseInt(data[13]);
                int PlaceOfDeathAndDecedentsStatus =
Integer.parseInt(data[14]);
                char MaritalStatus = data[15].charAt(0);
                int DayOfWeekOfDeath = Integer.parseInt(data[16]);
                int CurrentDataYear = Integer.parseInt(data[17]);
                char InjuryAtWork = data[18].charAt(0);
                int MannerOfDeath = Integer.parseInt(data[19]);
                char MethodOfDisposition = data[20].charAt(0);
                char Autopsy = data[21].charAt(0);
                int ActivityCode = Integer.parseInt(data[22]);
                int PlaceOfInjury = Integer.parseInt(data[23]);
```

```
                String Icd10Code = data[24];
                int CauseRecode358 = Integer.parseInt(data[25]);
                int CauseRecode113 = Integer.parseInt(data[26]);
                int InfantCauseRecode130 = Integer.parseInt(data[27]);
                int CauseRecode39 = Integer.parseInt(data[28]);
                int NumberOfEntityAxisConditions =
Integer.parseInt(data[29]);
                int NumberOfRecordAxisConditions =
Integer.parseInt(data[30]);
                int Race = Integer.parseInt(data[31]);
                int BridgedRaceFlag = Integer.parseInt(data[32]);
                int RaceImputationFlag = Integer.parseInt(data[33]);
                int RaceRecode3 = Integer.parseInt(data[34]);
                int RaceRecode5 = Integer.parseInt(data[35]);
                int HispanicOrigin = Integer.parseInt(data[36]);


            } catch (Exception e) {
                return false;
            }
            return true;
        }
    }

}
```

3.Zadatak

```
import org.apache.log4j.Level;
import org.apache.log4j.Logger;
import org.apache.spark.SparkConf;
import org.apache.spark.streaming.Durations;
import org.apache.spark.streaming.api.java.JavaDStream;
import org.apache.spark.streaming.api.java.JavaPairDStream;
import org.apache.spark.streaming.api.java.JavaStreamingContext;
import scala.Tuple2;

import java.util.NoSuchElementException;

/**
 * Prvo pokrenuti SensorStreamGenerator pa se zatim zakacit na njega.
 * Created by Vuki on 6.6.2017..
 */
public class PollutionReadStreaming {

    private static final int WINDOW_DURATION = 45;
    private static final int SLIDE_DURATION = 15;


    public static void main(String[] args) {

        Logger.getLogger("org").setLevel(Level.ERROR);
        Logger.getLogger("akka").setLevel(Level.ERROR);

        PollutionReadStreaming pollutionReadStreaming = new
PollutionReadStreaming();

        JavaStreamingContext context = pollutionReadStreaming.getContext();
```

```
        JavaDStream<String> lines =
pollutionReadStreaming.parseInput(context);
        pollutionReadStreaming.calculateOzonPerStationId(lines);
        pollutionReadStreaming.run(context);
    }

    private void run(JavaStreamingContext context) {
        context.start();

        try {
            context.awaitTermination();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    private void calculateOzonPerStationId(JavaDStream<String> lines) {
        JavaPairDStream<String, Integer> ozonePerStation =
lines.filter(AirPollution.PollutionReading::canParse)
                .map(AirPollution.PollutionReading::new)
                .mapToPair(pollutionReading -> new
Tuple2<>(pollutionReading.latitude + "" + pollutionReading.longitude,
pollutionReading.ozone))
                .reduceByKeyAndWindow(Math::min,
Durations.seconds(WINDOW_DURATION), Durations.seconds(SLIDE_DURATION));


ozonePerStation.dstream().saveAsTextFiles("ApacheSparkLab/ozonePerStation",
"");
    }

    private JavaStreamingContext getContext() {
        SparkConf conf = new
SparkConf().setAppName(PollutionReadStreaming.class.getName());
        //set the master if not already set through the command line
        try {
            conf.get("spark.master");
        } catch (NoSuchElementException e) {
            //spark streaming application requires at least 2 threads
            conf.setMaster("local[2]");
        }

        return new JavaStreamingContext(conf, Durations.seconds(3));
    }

    private JavaDStream<String> parseInput(JavaStreamingContext context) {
        return context.socketTextStream("localhost",
SensorStreamGenerator.PORT);
    }
}
```

```
import java.io.IOException;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.stream.Stream;
```

```
public class SensorStreamGenerator {

    private static final int WAIT_PERIOD_IN_MILLISECONDS = 1;
    public static final int PORT = 10002;

    public static final int STATION_ID = 158324;


    @SuppressWarnings("Duplicates")
    public static void main(String[] args) throws Exception {

        String fileName = "ApacheSparkLab/pollutionData/pollutionData" +
STATION_ID + ".csv";

        if (args.length != 1) {
            //         System.err.println("Usage: SensorStreamGenerator
<input file>");
            //           System.exit(-1);
        } else {
            fileName = args[0];
        }


        System.out.println("Waiting for client connection");

        try (ServerSocket serverSocket = new ServerSocket(PORT);
             Socket clientSocket = serverSocket.accept()) {

            System.out.println("Connection successful");

            PrintWriter out = new
PrintWriter(clientSocket.getOutputStream(),
                    true);


            Stream<String> lines = Files.lines(Paths.get(fileName));
            lines.forEach(line -> {
                out.println(line);
                try {
                    Thread.sleep(WAIT_PERIOD_IN_MILLISECONDS);
                } catch (InterruptedException ex) {
                    ex.printStackTrace();
                }
            });

        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
}
```