Matej Vukosav
0036473765
5.6.2017

1. Zadatak

- 97 ulaznih datoteka
- Linija: 4 726 567
- Velicina izlazne datoteke 406.376 KB

```
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;

/**
 * Cilj zadatka je ucitati podatke iz nekoliko tekstualnih datoteka,
filtrirati ih i sortirati te zapisati na lokalni disk.
 * Ulazne teksualne datoteke sadrze senzorska ocitanja koja su
prikupljena projektom SensorScope.
 * Svaka datoteka sadrzi ocitanja s jedne mjerne postaje pa je cilj
zadatka dobiti jednu izlazno sortiranu datoteku
 * s ocitanjima sa svih senzorskih postaja.
 * <p>
 * Uspjesnim rjesenjem ovog zadatka steci cete sljedeca stanja:
 * - osnove rada s kolekcijskim tokovima iz Java 8 Streams APIja
 * - jednostavna predobrada ulaznih podataka
 */
public class SensorScope {


    public static void main(String[] args) {

        SensorScope sensorScope = new SensorScope();
        sensorScope.run("SensorScope-monitor");
    }

    private void run(String folderName) {
        parseInputData(folderName);
    }


    private void parseInputData(String folderName) {

        Stream<SensorscopeReading> sensorscopeReadingStream = null;
        try {
            sensorscopeReadingStream = Files.list(Paths.get(folderName))
                    .filter(Files::isRegularFile)
                    .flatMap(p -> {
                        try {
                            return Files.lines(p);
                        } catch (IOException e) {
                            e.printStackTrace();
                        }
                        return null;
                    })
                    .map(SensorscopeReading::new)
                    .sorted(Comparator.comparingInt(o ->
o.timeSinceTheEpoch));
//                      .forEach(o -> {
//                          System.out.println(o.energySource);
//                      });
        } catch (IOException e) {
            e.printStackTrace();
        }

        if (sensorscopeReadingStream == null) {
            System.out.println("Stream is null!");
            return;
        }

        try {
            FileWriter fileWriter = new FileWriter("result.txt");
            sensorscopeReadingStream.forEach(data -> {
                try {
                    fileWriter.append(data.toString());
                    fileWriter.append(System.lineSeparator());
                } catch (IOException e) {
                    e.printStackTrace();
                }
            });

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

//              try {
//                  streams.add(Files.lines(Paths.get(path)));
//              } catch (IOException e) {
//                  e.printStackTrace();
//              }
//          }


    }

    public static class SensorscopeReading {
        /**
         * Column definitions:
         * <p>
         * 1. Station ID
         * 2. Year
         * 3. Month
         * 4. Day
         * 5. Hour
         * 6. Minute
         * 7. Second
         * 8. Time since the epoch [s]
         * 9. Sequence Number
         * 10. Config Sampling Time [s]
         * 11. Data Sampling Time [s]
         * 12. Radio Duty Cycle [%]
         * 13. Radio Transmission Power [dBm]
```

```
 * 14. Radio Transmission Frequency [MHz]
 * 15. Primary Buffer Voltage [V]
 * 16. Secondary Buffer Voltage [V]
 * 17. Solar Panel Current [mA]
 * 18. Global Current [mA]
 * 19. Energy Source
 */
int stationId;
int year;
int month;
int day;
int hour;
int minute;
int second;
int timeSinceTheEpoch;
int sequenceNumber;
float configSamplingTime;
float dataSamplingTime;
float radioDutyCycle;
float radioTransmissionPower;
float radioTransmissionFrequency;
float primaryBufferVoltage;
float secondaryBufferVoltage;
float solarPanelCurrent;
float globalCurrent;
float energySource;

public SensorscopeReading(String line) {
    String[] data = line.split(" ");
    this.stationId = Integer.parseInt(data[0]);
    this.year = Integer.parseInt(data[1]);
    this.month = Integer.parseInt(data[2]);
    this.day = Integer.parseInt(data[3]);
    this.hour = Integer.parseInt(data[4]);
    this.minute = Integer.parseInt(data[5]);
    this.second = Integer.parseInt(data[6]);
    this.timeSinceTheEpoch = Integer.parseInt(data[7]);
    this.sequenceNumber = Integer.parseInt(data[8]);
    this.configSamplingTime = Float.parseFloat(data[9]);
    this.dataSamplingTime = Float.parseFloat(data[10]);
    this.radioDutyCycle = Float.parseFloat(data[11]);
    this.radioTransmissionPower = Float.parseFloat(data[12]);
    this.radioTransmissionFrequency = Float.parseFloat(data[13]);
    this.primaryBufferVoltage = Float.parseFloat(data[14]);
    this.secondaryBufferVoltage = Float.parseFloat(data[15]);
    this.solarPanelCurrent = Float.parseFloat(data[16]);
    this.globalCurrent = Float.parseFloat(data[17]);
    this.energySource = Float.parseFloat(data[18]);
}


@Override
public String toString() {
    String separator = ",";
    return
            stationId + separator
                    + year + separator
                    + month + separator
```

```
                                + day + separator
                                + hour + separator
                                + minute + separator
                                + second + separator
                                + timeSinceTheEpoch + separator
                                + sequenceNumber + separator
                                + configSamplingTime + separator
                                + dataSamplingTime + separator
                                + radioDutyCycle + separator
                                + radioTransmissionPower + separator
                                + radioTransmissionFrequency + separator
                                + primaryBufferVoltage + separator
                                + secondaryBufferVoltage + separator
                                + solarPanelCurrent + separator
                                + globalCurrent + separator
                                + energySource;
        }
    }
}
```

2. Zadatak

1. Most unpopular girl name is: Anaissa
2. Most popular male names are:

```
James 4938965
John 4829733
Robert 4710600
Michael 4295779
William 3829026
David 3554102
Richard 2529952
Joseph 2479602
Charles 2244617
Thomas 2216356
```

3.

Most childrens born in 1946 are from NY.

4. Female born childrens through years

```
1910 352089
1911 372371
1912 504283
1913 566950
1914 696886
1915 908543
1916 965971
1917 1000429
1918 1073241
1919 1047125
1920 1111874
1921 1145019
1922 1114028
1923 1119997
1924 1161238
1925 1130217
1926 1099788
```

1927	1106858
1928	1069872
1929	1035735
1930	1044440
1931	986844
1932	987891
1933	931551
1934	966643
1935	971717
1936	963165
1937	985346
1938	1023657
1939	1017138
1940	1062825
1941	1125424
1942	1262701
1943	1305423
1944	1239319
1945	1219589
1946	1475510
1947	1669192
1948	1596392
1949	1608637
1950	1610127
1951	1694666
1952	1745640
1953	1769233
1954	1827890
1955	1839033
1956	1888744
1957	1924204
1958	1889908
1959	1899916
1960	1894794
1961	1887645
1962	1835888
1963	1794939
1964	1760119
1965	1635216
1966	1561276
1967	1517486
1968	1501476
1969	1540546
1970	1591419
1971	1502628
1972	1362036
1973	1296444
1974	1299746
1975	1284988
1976	1288847
1977	1347825
1978	1344396
1979	1409915
1980	1461972
1981	1469706
1982	1494101
1983	1475425
1984	1487244

```
1985 1517484
1986 1509113
1987 1525768
1988 1558020
1989 1613574
1990 1657185
1991 1629792
1992 1596003
1993 1558806
1994 1534448
1995 1506897
1996 1497012
1997 1478655
1998 1496360
1999 1497106
2000 1527133
2001 1506192
2002 1498084
2003 1520794
2004 1523496
2005 1528165
2006 1564886
2007 1576648
2008 1544411
2009 1493172
2010 1439346
2011 1422539
2012 1422020
2013 1419351
2014 1446259
```

5. Female name Mary through years:

```
1910,6.4892683384030745
1911,6.549919300912261
1912,6.405728529417014
1913,6.4628274098244995
1914,6.506659625821152
1915,6.404429949930823
1916,6.360128823743155
1917,6.425243570508251
1918,6.2772480738249845
1919,6.287692491345351
1920,6.383457118342546
1921,6.461464831587947
1922,6.478382949082069
1923,6.395909989044614
1924,6.331776948394731
1925,6.2466765231809465
1926,6.166915805591623
1927,6.381306364502041
1928,6.249906530874721
1929,6.131684262866466
1930,6.140515491555283
1931,6.109678936083109
1932,6.0604864301830865
1933,5.957054417847225
1934,5.888213125217893
```

```
1935,5.6669791719193965
1936,5.644204264066905
1937,5.646747436940933
1938,5.491194804509714
1939,5.397497684679955
1940,5.287323877402207
1941,5.156367733405365
1942,5.008153157398308
1943,5.068931679616492
1944,5.040106703762308
1945,4.860981855362749
1946,4.572181821878537
1947,4.294532923713989
1948,4.296876957539251
1949,4.156438027970263
1950,4.065517813191134
1951,3.8762210370657106
1952,3.762574184826196
1953,3.6364345453651388
1954,3.7201910399422284
1955,3.4344136293367225
1956,3.269421372086424
1957,3.175131119153687
1958,2.9551703045862547
1959,2.867126757182949
1960,2.716865263453441
1961,2.525209983868789
1962,2.369262177213425
1963,2.314730472734728
1964,2.328592555389721
1965,2.0957475954247022
1966,1.8500252357686917
1967,1.6681537753890314
1968,1.4467097709187493
1969,1.288569117702425
1970,1.2065332888447353
1971,1.1112530845957882
1972,1.01039913776141
1973,0.9503688551144516
1974,0.9040227859904935
1975,0.8532375399614626
1976,0.8011812108031442
1977,0.7909780572403687
1978,0.7473244490462632
1979,0.7485557639999575
1980,0.784761951665285
1981,0.7509665198345792
1982,0.7261891933677844
1983,0.67051866411373
1984,0.6245108401849326
1985,0.6088367323807038
1986,0.5634435592298257
1987,0.5502147115419906
1988,0.5460777140216428
1989,0.5355812624645662
1990,0.5228142904986468
1991,0.5372464707152815
1992,0.5295729393992368
```

```
1993,0.5200133948676102
1994,0.5045462602838284
1995,0.4932652994862953
1996,0.4637237376854695
1997,0.4478394216365548
1998,0.429642599374482
1999,0.42475282311339346
2000,0.40481084489694086
2001,0.38003123107810954
2002,0.3635977688834538
2003,0.32897289179205075
2004,0.31506482458765894
2005,0.2906099799432653
2006,0.26027455035063257
2007,0.2324551834017485
2008,0.22565236844337422
2009,0.21082634820368987
2010,0.19835397465237684
2011,0.18846583468010367
2012,0.17953333989676656
2013,0.18452095359076084
2014,0.18032731343417743
```

6.
Sum of born children of all times is: 298883326

7. Distinct names sum is: 30274

```
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 scala.Tuple2;

import java.io.FileWriter;
import java.io.IOException;
import java.io.Serializable;
import java.util.NoSuchElementException;

/**
 * Cilj zadatka je obaviti analizu podataka o ucestalosti imena 
novorodencadi u Sjedinjenim Americkim Drzavama po
 * godinama i drzavama.
 * <p>
 * <p>
 * Potreban apache-spark
 * <p>
 * Created by Vuki on 4.6.2017..
 */
public class ChildNames {


    public static void main(String[] args) {


        System.setProperty("hadoop.home.dir", "C:\\Program 
Files\\WinUtils");
```

```
        //ChildNames childNames = new ChildNames();

        //JavaSparkContext context = childNames.getContext();
        JavaSparkContext context = getContext();
        //childNames.parseInput(context, args);
        parseInput(context, args);
    }

    private static JavaSparkContext getContext() {
        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 static void parseInput(JavaSparkContext context, String[]
args) {

        String path = "StateNames.csv";

        //create an RDD from text file lines. Resilient Distributed
Dataset
        JavaRDD<String> lines = context.textFile(path);

        JavaRDD<USBabyNameRecord> dataLineJavaRDD =
                lines.filter(USBabyNameRecord::canParse)
                        .map(USBabyNameRecord::new);


        //getMostUnPopularGirlsName(dataLineJavaRDD);
        //getTop10MaleNames(dataLineJavaRDD);
        //getStateWithMaxBornChilds(dataLineJavaRDD, 1946);
        //getFemaleBirthsCountThroughYears(dataLineJavaRDD);
        //getFemaleNameThroughYears(dataLineJavaRDD, "Mary");
        //getSumOfBornChildern(dataLineJavaRDD);
        getSumOfDistinctNames(dataLineJavaRDD);
    }

    private static void getSumOfDistinctNames(JavaRDD<USBabyNameRecord>
dataLineJavaRDD) {
        long count = dataLineJavaRDD
                .map(usBabyNameRecord -> usBabyNameRecord.name)
                .distinct()
                .count();
        System.out.println("Distinct names sum is: " + count);
    }

    private static void
getFemaleNameThroughYears(JavaRDD<USBabyNameRecord> dataLineJavaRDD,
String name) {
        JavaPairRDD<Integer, Long> femaleBirthsCountThroughYears =
getFemaleBirthsCountThroughYears(dataLineJavaRDD);

        System.out.println("Female name " + name + " through years:");
        JavaPairRDD<Integer, Long> maryBirthsCountThroughYears =
dataLineJavaRDD.filter(usBabyNameRecord -> usBabyNameRecord.gender ==
'F')
                .filter(usBabyNameRecord ->
usBabyNameRecord.name.equals(name))
                .mapToPair(usBabyNameRecord -> new
Tuple2<>(usBabyNameRecord.year, usBabyNameRecord.count))
                //reduciraj po kljucu i zbroji vrijednosti
                .reduceByKey((x, y) -> x + y)
                .sortByKey();


        JavaPairRDD<Integer, Double> result =
maryBirthsCountThroughYears.join(femaleBirthsCountThroughYears)
                .mapToPair(record -> new Tuple2<>(record._1, ((double)
record._2._1 / record._2._2) * 100))
                .sortByKey();
        result
                .foreach(record -> {
                    System.out.println(record._1 + " " + record._2);
                });

        try {
            FileWriter fw = new FileWriter("./out5.txt");
            result.collect().forEach(record -> {
                String line = record._1 + "," + record._2 + "\n";
                try {
                    fw.write(line);
                } catch (IOException ex) {
                    ex.printStackTrace();
                }
            });
            fw.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }


    private static JavaPairRDD<Integer, Long>
getFemaleBirthsCountThroughYears(JavaRDD<USBabyNameRecord>
dataLineJavaRDD) {

        System.out.println("Female born childrens through years:");
        JavaPairRDD<Integer, Long> pairs =
dataLineJavaRDD.filter(usBabyNameRecord -> usBabyNameRecord.gender ==
'F')
                .mapToPair(usBabyNameRecord -> new
Tuple2<>(usBabyNameRecord.year, usBabyNameRecord.count))
                //reduciraj po kljucu i zbroji vrijednosti
                .reduceByKey((x, y) -> x + y)
                .sortByKey();
        pairs.foreach(record -> {
            System.out.println(record._1 + " " + record._2);
        });

        return pairs;

    }

    private static void getSumOfBornChildern(JavaRDD<USBabyNameRecord>
dataLineJavaRDD) {

        Tuple2<Integer, Long> result = dataLineJavaRDD
                .mapToPair(usBabyNameRecord -> new Tuple2<>(0,
usBabyNameRecord.count))
                .reduceByKey((x, y) -> x + y)
                .first();


        long count = result._2;

        System.out.println("Sum of born children of all times is: " +
count);
    }

    private static void
getStateWithMaxBornChilds(JavaRDD<USBabyNameRecord> dataLineJavaRDD, int
year) {


        Tuple2<Long, String> first = dataLineJavaRDD
                .filter(usBabyNameRecord -> usBabyNameRecord.year ==
year)
                .mapToPair(record -> new Tuple2<>(record.state,
record.count))
                //reduciraj po drzavi
                .reduceByKey((x, y) -> x + y)
                .mapToPair(Tuple2::swap)
                .sortByKey(false)
                .first();


        System.out.println("Most childrens born in " + year + " are from
" + first._2 + ".");
    }

    private static void getTop10MaleNames(JavaRDD<USBabyNameRecord>
dataLineJavaRDD) {

        System.out.println("Most popular male names are: ");

        dataLineJavaRDD.filter(usBabyNameRecord ->
usBabyNameRecord.gender == 'M')
                .mapToPair(record -> new Tuple2<>(record.name,
record.count))
                .reduceByKey((x, y) -> x + y)
                .mapToPair(Tuple2::swap)
                .sortByKey(false)
                .take(10)
                .forEach(record -> {
                    System.out.println(record._2 + " " + record._1);
                });
    }

    private static void
getMostUnPopularGirlsName(JavaRDD<USBabyNameRecord> dataLineJavaRDD) {

        JavaPairRDD<Long, String> stringLongJavaPairRDD = dataLineJavaRDD
                .filter(USBabyNameRecord -> USBabyNameRecord.gender ==
'F')
                .mapToPair(record -> new Tuple2<>(record.name,
record.count))
                .reduceByKey((integer, integer2) -> integer + integer2)
                .mapToPair(Tuple2::swap)
                .sortByKey(true);

        System.out.println("Most unpopular girl name is: " +
stringLongJavaPairRDD.first()._2);
    }


    static class USBabyNameRecord implements Serializable {
        long id;
        String name;
        int year;
        char gender; //F, M
        String state; //two letters
        long count;

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

            this.id = Long.parseLong(data[0]);
            this.name = data[1];
            this.year = Integer.parseInt(data[2]);
            this.gender = data[3].charAt(0);
            this.state = data[4];
            this.count = Long.parseLong(data[5]);
        }

        public static boolean canParse(String line) {
            String[] data = line.split(",");

            try {
                long id = Long.parseLong(data[0]);
                String name = data[1];
                int year = Integer.parseInt(data[2]);
                char gender = data[3].charAt(0);
                String state = data[4];
                long count = Long.parseLong(data[5]);
            } catch (Exception e) {
                return false;
            }

            return true;
        }
    }
```

```
}
```

Zadatak 3.
-Koliko često (u sekundama) nastaje novi direktorij na disku?
 Svakih 10 sekundi
- Koliko često (u sekundama) se pokreće izračun?
 Svakih 10 sekundi
- Može li vrijednost parametra solarPanelCurrent neke stanice biti manja u nekom direktoriju nego u
njegovom neposrednom prethodniku. Zašto?

Moze. To se moze dogoditi u situaciji ako je maksimalna vrijednost upravo izasla iz prozora. (rolling max)

- Kako se kreću vrijednosti parametra solarPanelCurrent neke postaje u prva 3 direktorija koji su nastali?
Zašto?
Vrijednosti iz prva 3 direktorija koja su nastala

```
(100,98.126)
(100,98.126)
(100,98.126)
...
(100,99.713)
```

Prvih par vrijednosti je isto jer je prozor veci od slide durationa.

```
import org.apache.spark.SparkConf;
import org.apache.spark.streaming.Duration;
import org.apache.spark.streaming.Durations;
import org.apache.spark.streaming.api.java.JavaDStream;
import org.apache.spark.streaming.api.java.JavaStreamingContext;
import scala.Tuple2;

import java.util.NoSuchElementException;

/**
 * Koristi spark-streaming
 * <p>
 * <p>
 * Created by Vuki on 5.6.2017..
 */
public class SensorDataStreaming {


    public static void main(String[] args) {

        SparkConf conf = new
SparkConf().setAppName(SensorDataStreaming.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[*]");
        }
        JavaStreamingContext context = new JavaStreamingContext(conf,
Durations.seconds(5));
        context.checkpoint("./checkpoint");

        JavaDStream<String> lines = context.socketTextStream("localhost",
SensorStreamGenerator.PORT);

        final int WINDOW_DURATION = 60000;
        final int SLIDE_DURATION = 10000;
        JavaDStream<String> result = lines
                .filter(SensorScope.SensorscopeReading::isParseable)
                .map(SensorScope.SensorscopeReading::new)
                .mapToPair(record -> new Tuple2<>(record.stationId,
record.solarPanelCurrent))
                .reduceByKeyAndWindow(Math::max, (x, y) -> x,
                        new Duration(WINDOW_DURATION), new
Duration(SLIDE_DURATION))
                .map(record -> record._1() + " , " + record._2());


        result.dstream().print();

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

    }
}


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 void main(String[] args) throws Exception {

        String fileName = "Sensorscope-monitor";

        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));
            Stream<String> lines = Files.list(Paths.get(fileName))
                    .filter(Files::isRegularFile)
                    .flatMap(p -> {
                        try {
                            return Files.lines(p);
                        } catch (IOException e) {
                            e.printStackTrace();
                        }
                        return null;
                    });

            lines.forEach(line -> {
                out.println(line);
                try {
                    Thread.sleep(WAIT_PERIOD_IN_MILLISECONDS);
                } catch (InterruptedException ex) {
                    ex.printStackTrace();
                }
            });

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