Code:

```
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.api.java.function.*;
import org.apache.spark.broadcast.Broadcast;
import org.apache.spark.ml.feature.CountVectorizer;
import org.apache.spark.ml.feature.CountVectorizerModel;
import org.apache.spark.ml.linalg.Vector;
import org.apache.spark.sql.Dataset;
import org.apache.spark.sql.Row;
import org.apache.spark.sql.RowFactory;
import org.apache.spark.sql.SparkSession;
import org.apache.spark.sql.types.DataTypes;
import org.apache.spark.sql.types.StructField;
import org.apache.spark.sql.types.StructType;
import scala.Tuple2;

import java.util.*;

// this is the example in Chap 3, Example 3.6
// for the related homework question, see SparkBruteForceSimilarity
public class SparkMinHashLSH {

    private static final String FILE_URI = "file:///D:/Data/LSH_*.txt";
    private static final double sizeAdj = 1.0;

    /**
     * @param a double array
     * @param b double array
     * @return double     *
     * The purpose of this method is to calculate the jaccard similarity.
     */
    static private double jaccardSimilarity(String[] a, String[] b) {

        Set<String> s1 = new LinkedHashSet<String>();
        for (int i = 0; i < a.length; i++) {
            s1.add(a[i]);
        }
        Set<String> s2 = new LinkedHashSet<String>();
        for (int i = 0; i < b.length; i++) {
            s2.add(b[i]);
        }

        Set<String> intersection = new LinkedHashSet<>(s1);
        intersection.retainAll(s2);

        Set<String> union = new LinkedHashSet<>(s1);
        union.addAll(s2);
        return (double) intersection.size() / (double) union.size();
    }

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

```
        // initializing spark
        SparkSession spark = SparkSession.builder().config("spark.master",
"local[*]").getOrCreate();
        JavaSparkContext sc = new JavaSparkContext(spark.sparkContext());
        sc.setLogLevel("WARN");

        // create RDD by using text files
        JavaPairRDD<String, String> documents = sc.wholeTextFiles(FILE_URI);

        // convert original documents into shingle representation
        class ShinglesCreator implements Function<String, String[]> {
            @Override
            public String[] call(String text) throws Exception {
                return ShingleUtils.getTextShingles(text);
            }
        }
        JavaPairRDD<String, String[]> shinglesDocs = documents.mapValues(new
ShinglesCreator());
        shinglesDocs.values().foreach(new VoidFunction<String[]>() {
            public void call(String[] shingles) throws Exception {
                for (int i = 0; i < shingles.length; i++) {
                    System.out.print(shingles[i] + "|");
                }
                System.out.println();
            }
        });

        // create characteristic matrix representation of each document
        StructType schema = new StructType(
                new StructField[]{
                        DataTypes.createStructField("file_path",
DataTypes.StringType, false),
                        DataTypes.createStructField("file_content",
DataTypes.createArrayType(DataTypes.StringType, false), false)
                });
        Dataset<Row> df = spark.createDataFrame(
                shinglesDocs.map(new Function<Tuple2<String, String[]>, Row>() {
                    @Override
                    public Row call(Tuple2<String, String[]> record) {
                        return
RowFactory.create(record._1().substring(record._1().lastIndexOf("/") + 1),
record._2());
                    }
                }), schema);
        df.show(true);

        CountVectorizer vectorizer = new
CountVectorizer().setInputCol("file_content").setOutputCol("feature_vector").setBinar
y(true);
        CountVectorizerModel cvm = vectorizer.fit(df);
        Broadcast<Integer> vocabSize = sc.broadcast(cvm.vocabulary().length);

        System.out.println("vocab size = " + cvm.vocabulary().length);
        for (int i = 0; i < vocabSize.value(); i++) {
            System.out.print(cvm.vocabulary()[i] + "(" + i + ") ");
```

```
        }
        System.out.println();

        Dataset<Row> characteristicMatrix = cvm.transform(df);
        characteristicMatrix.show(false);



        /**
         * following is the Code to calculate cartesian product and filter out
repeated pairs.
         */



            //public JavaRDD<T> filter(Function<Tuple2<Tuple2<String, List<String>>,
Tuple2<String, List<String>>>, Boolean> f);

            JavaPairRDD<Tuple2<String, String[]>, Tuple2<String, String[]>> cart =
shinglesDocs.cartesian(shinglesDocs).filter(new Function<Tuple2<Tuple2<String,
String[]>, Tuple2<String, String[]>>, Boolean>() {

                    /**
                     *
                     */
                    private static final long serialVersionUID = 1L;

                    @Override
                    public Boolean call(Tuple2<Tuple2<String, String[]>,
Tuple2<String, String[]>> bucketDocument) throws Exception {

      //System.out.println(bucketDocument._1.toString().substring(56, 57));
                        return
(Integer.parseInt((bucketDocument._1.toString().substring(19,20)))<Integer.parseInt((
bucketDocument._2.toString().substring(19,20))));

                    }
                });

        System.out.printf("Cartesian Products : ");
        System.out.println(cart.take((int) cart.count()).toString());

        /**
         * following is the Code to iterate through the cartesian product elements
and invoke jaccard similarity for each of the cartesian results.
         */
        Iterator<Tuple2<Tuple2<String, String[]>, Tuple2<String, String[]>>> iterator
= cart.collect().iterator();
        while (iterator.hasNext()) {
            Tuple2<Tuple2<String, String[]>, Tuple2<String, String[]>> temp =
iterator.next();
            if (!temp._1._1.equals(temp._2._1)) {
                String key = temp._1._1.substring(15, 18) + "-" +
temp._2._1.substring(15, 18);
                int index1 = 0;
```

```
                List<String> list1 = Arrays.asList(temp._1._2);
                String[] array1 = new String[list1.size()];
                for (String num1 : temp._1._2) {
                    array1[index1] = num1;
                    index1++;
                }

                int index2 = 0;
                List<String> list2 = Arrays.asList(temp._2._2);
                String[] array2 = new String[list2.size()];
                for (String num1 : temp._2._2) {
                    array2[index2] = num1;
                    index2++;
                }
                double similarity = jaccardSimilarity(array1, array2);
                System.out.println("Similarity for : " + key + " is : " +
similarity);

            }
        }
        vocabSize.unpersist();
        vocabSize.destroy();

        sc.close();
    }
```

Jaccard Similarity:

```
static private double jaccardSimilarity(String[] a, String[] b) {


    Set<String> s1 = new LinkedHashSet<String>();

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

      s1.add(a[i]);

    }

    Set<String> s2 = new LinkedHashSet<String>();

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

      s2.add(b[i]);

    }


    Set<String> intersection = new LinkedHashSet<>(s1);

    intersection.retainAll(s2);
```

```
    Set<String> union = new LinkedHashSet<>(s1);
    union.addAll(s2);
    return (double) intersection.size() / (double) union.size();
  }
```

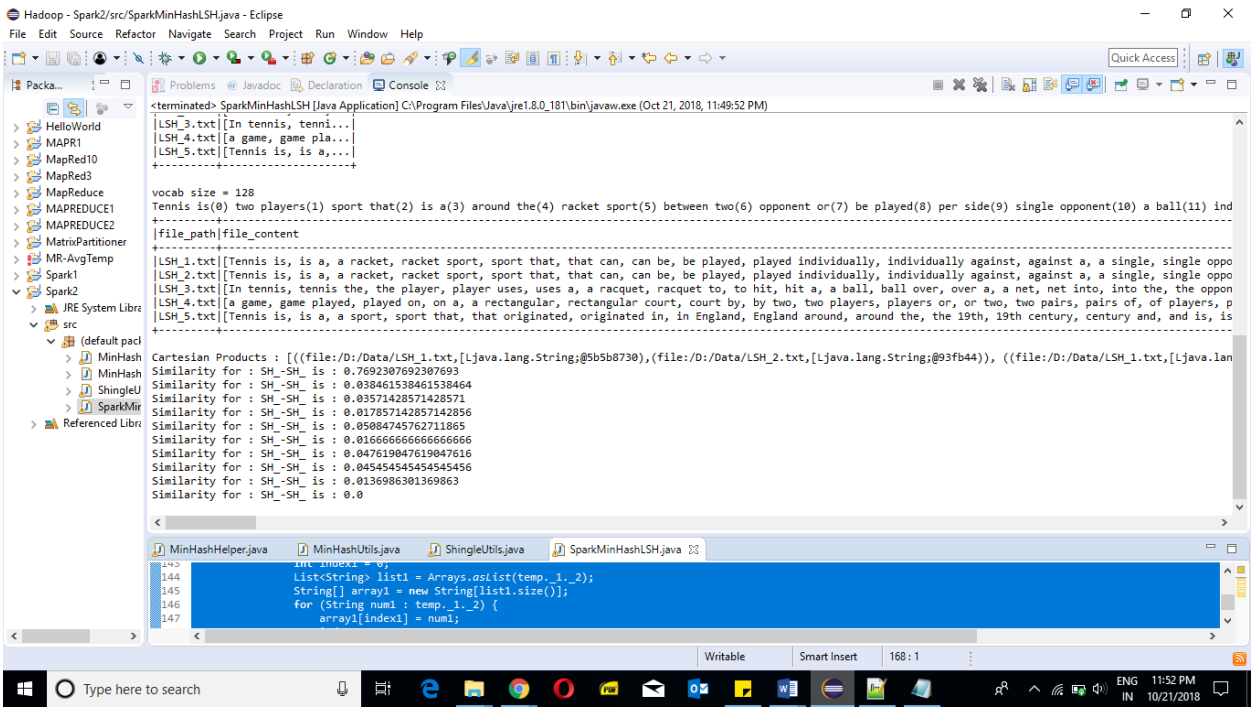
Filter Function:

```
 public Boolean call(Tuple2<Tuple2<String, String[]>, Tuple2<String, String[]>> bucketDocument) throws Exception {
                                        //System.out.println(bucketDocument._1.toString().substring(56, 57));
            return (Integer.parseInt((bucketDocument._1.toString()).substring(56, 57))<Integer.parseInt((bucketDocument._2.toString()).substring(56, 57)));

          }
        });

    System.out.printf("Cartesian Products : ");
    System.out.println(cart.take((int) cart.count()).toString());
```

Output:


Hadoop - Spark2/src/SparkMinHashLSH.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Problems Javadoc Declaration Console
<terminated> SparkMinHashLSH [Java Application] C:\Program Files\Java\jre1.8.0_181\bin\javaw.exe (Oct 21, 2018, 11:49:52 PM)
LSH_3.txt	[In tennis, tenni...
LSH_4.txt	[a game, game pla...
LSH_5.txt	[Tennis is, is a,...
vocab size = 128
Tennis is(0) two players(1) sport that(2) is a(3) around the(4) racket sport(5) between two(6) opponent or(7) be played(8) per side(9) single opponent(10) a ball(11) ind
file_path	file_content
LSH_1.txt	[Tennis is, is a, a racket, racket sport, sport that, that can, can be, be played, played individually, individually against, against a, a single, single oppo
LSH_2.txt	[Tennis is, is a, a racket, racket sport, sport that, that can, can be, be played, played individually, individually against, against a, a single, single oppo
LSH_3.txt	[In tennis, tennis the, the player, player uses, uses a, a racquet, racquet to, to hit, hit a, a ball, ball over, over a, a net, net into, into the, the oppon
LSH_4.txt	[a game, game played, played on, on a, a rectangular, rectangular court, court by, by two, two players, players or, or two, two pairs, pairs of, of players, p
LSH_5.txt	[Tennis is, is a, a sport, sport that, that originated, originated in, in England, England around, around the, the 19th, 19th century, century and, and is, is
Cartesian Products : [((file:/D:/Data/LSH_1.txt,[Ljava.lang.String;@5b5b8730),(file:/D:/Data/LSH_2.txt,[Ljava.lang.String;@93fb44)), ((file:/D:/Data/LSH_1.txt,[Ljava.lan
Similarity for : SH_-SH_ is : 0.7692307692307693
Similarity for : SH_-SH_ is : 0.0384615384615384 64
Similarity for : SH_-SH_ is : 0.03571428571428571
Similarity for : SH_-SH_ is : 0.017857142857142856
Similarity for : SH_-SH_ is : 0.05084745762711865
Similarity for : SH_-SH_ is : 0.016666666666666666
Similarity for : SH_-SH_ is : 0.047619047619047616
Similarity for : SH_-SH_ is : 0.045454545454545456
Similarity for : SH_-SH_ is : 0.013986301369863
Similarity for : SH_-SH_ is : 0.0
MinHashHelper.java MinHashUtils.java ShingleUtils.java SparkMinHashLSH.java
144 List<String> list1 = Arrays.asList(temp._1._2);
145 String[] array1 = new String[list1.size()];
146 for (String num1 : temp._1._2) {
147 array1[index1] = num1;
Writable Smart Insert 168 : 1
Type here to search
ENG IN 11:52 PM 10/21/2018