COURSEWORK 1 - TWITTER ANALYSIS

The goal of this coursework is to apply the techniques covered in the first half of Big Data Processing to analyse a dataset of Tweets collected during the 2016 Rio Olympic Games. You will create several Map/Reduce programs to perform multiple types of computation. You will submit a report containing your results alongside an explanation of how they were obtained.

DATASET

The folder **/data/olympictweets2016rio** contains a large collection of Twitter messages captured during the Rio 2016 Olympics period. The messages were collected by connecting to Twitter Streaming API, and filtering only messages directly related to the Olympic Games (by requesting they include a related hashtag such as #Rio2016 or #rioolympics ).

A MapReduce job that points to that folder as input will receive as Text value the whole contents for a single entry (tweet), in the format:

**epoch\_time;tweetId;tweet(including #hashtags);device**

**epoch\_time**provides the time the message was published, expressed in miliseconds since 01-01-1970.

**tweetId** is a unique identifier per message.

**tweet**includes the message itself. **hashtags** are in the tweets, identified by the hash symbol.

**device**provides additional meta-information, including the type of device/app used to submit the message, and a shortened url to access the message.

An example entry for the dataset is:

**1469453965000;757570957502394369;Over 30 million women footballers in the world. Most of us would trade places with this lot for #Rio2016  https://t.co/Mu5miVJAWx;<a href="http://twitter.com/download/iphone" rel="nofollow">Twitter for iPhone</a>**

We provide as part of the coursework the **IntIntPair** object, a **Writable**class that can store a pair of int values in a single Writable. We do not enforce the use of that class as part of some of your solutions, but it might become useful at some point during the coursework.

**DATA CLEANING**

The provided dataset is the direct output from a Twitter crawler, and it has NOT been sanitised. As a consequence of that, there will be some entries from the dataset that do not comply with the expected format. This might cause MapReduce jobs fail (if, for example, the code assumes that every single entry will have x number of elements separated by a ';'.

You can avoid that by ***filtering***your Map/Reduce programs, making sure you only process lines that conform to the expected format. (that is, a String that has N parts separated by a ';'.

**Assignment**

 Write a set of Map/Reduce jobs that process the given input and generate the data required to answer the following questions:

PART A. MESSAGE LENGTH ANALYSIS(35%)

1. Create a **Histogram plot** that depicts the distribution of tweet sizes (measured in number of characters) among the Twitter dataset. To make the data more readable, the histogram must aggregate bars in groups of 5 (that is, first bar counts tweets of length 1-5, second bar counts tweets 6-10, and so on) as part of your MapReduce job. Your MapReduce program must compute the histogram bins for a correct solution. Aggregating bins outside MapReduce will deduct marks from the complete grade.

***Note 1:*** For considering the size of a message you should simply refer to the **length()** of the String provided as input.

***Note 2:***There are numerous tweets written in foreign languages, which contain characters with non-standard encoding that might cause some unexpected (i.e. too high) values. This is a common occurrence when dealing with real data. We recommend you filter out all the messages with a length longer than 140 characters. You can also handle them differently if you prefer so and provide an appropriate explanation.  The report has to explain the approach you took, and the reasoning behind it

PART B. TIME ANALYSIS (45%)

1. Create a **bar plot**showing the number of Tweets that were posted **each hour of the event**. You should aggregate together all the messages emitted at the same hour, regardless of the day the messages were sent (hence, you will have 24 different groups). When checking the correctness of your results, keep in mind the timezone of the 2016 Olympic games, as that should give you base expectations about the prime time when the main activities occurred. [25 marks]
2. For the **most popular hour**of the games, compute the **top 10 hashtags**that were emitted during that hour. Hashtags are words contained inside the tweet, starting with the hashcode (#) character. Does that information provide you any hint on the main events/activities that took place at peak time? [20 marks]

***Note 3:*** We recommend the usage of the new Java 8's time API for parsing the epoch time, and obtaining information The Date class can easily parse data from the ;epoch\_time; field. Check the [Javadoc documentation of java.util.Date](https://docs.oracle.com/javase/7/docs/api/java/util/Date.html) to find about the methods you can invoke of this object. In particular, the following method from LocalDateTime can obtain a Date object with convenient methods for accessing hour, and other time-related fields from an epoch.

LocalDateTime.ofEpochSecond(epochTime/1000, ...)

***Note 4***: For part 2 (hashtags during the most popular hour, you can 'hardcode' the hour value, based on what you found in part 1. There is no need to programmatically compute the maximum hour as part of your MapReduce set of programs to obtain full marks.

PART C. SUPPORT ANALYSIS (20%)

The goal of the final part of the coursework is to compute the 30 athletes that received the highest support, according to the Twitter messages of our dataset. Please note that there might be a bias towards English speaking countries/athletes, because of the methodology used when collecting the dataset.

In this final section, you will have to use an additional dataset, containing the list of Rio 2016 athletes that obtained a medal, and their discipline. The dataset can be found at:

**/data/medalistsrio.csv**

The data is saved in Comma(,)-separated-values format, with the first row providing header names. Dataset has been downloaded from <https://www.kaggle.com/rio2016/olympic-games/data> . The dataset has been cropped to only include medalists in an effort to reduce the computation time it will impose to the cluster.

You need to match full names, exactly as they are written in the athletes dataset. Do not try to split names and surnames, or look for additional information sources, such as Twitter handles for athletes.

1. **Draw a table** with the top 30 athletes in number of mentions across the dataset. For each athlete, include the number of mentions retrieved.  For this question you can sort results and compute the top X outside your MapReduce code. [10 marks]
2. **Draw a table**with the top 20 sports according to the mentions of olympic athletes captured. For resolving athletes into sports use the medalistsrio secondary dataset. For this question you can sort results and compute the top X outside your MapReduce code. [10 marks]

***Note 5:***Section C is designed to be more challenging than the other parts of the course, but the overall weighting is lower than the other two.

SUBMISSION

You must upload a short report in PDF format detailing your answer to each of the questions, including the plots requested. Make sure to also explain how you used Map/Reduce programs to obtain the data you needed in each case.

Submit also the compressed source code for all the Map/Reduce jobs you have implemented in this exercise.

For generating the plots you can use any visualisation toolkit; Python's matplotlib, R, gnuplot, or excel.

ANNEX I: INTINTWRITABLE SOURCE CODE

Source code for**IntIntPair.java**

**import java.io.\*;  
import org.apache.hadoop.io.\*;  
  
public class IntIntPair implements WritableComparable<IntIntPair> {  
  
 private IntWritable first;  
 private IntWritable second;  
  
 public IntIntPair() {  
 set(new IntWritable(), new IntWritable());  
 }  
  
 public IntIntPair(int first, int second) {  
 set(new IntWritable(first), new IntWritable(second));  
 }  
  
 public void set(IntWritable first, IntWritable second) {  
 this.first = first;  
 this.second = second;  
 }  
  
 public IntWritable getFirst() {  
 return first;  
 }  
  
 public IntWritable getSecond() {  
 return second;  
 }  
  
 @Override  
 public void write(DataOutput out) throws IOException {  
 first.write(out);  
 second.write(out);  
 }  
  
 @Override  
 public void readFields(DataInput in) throws IOException {  
 first.readFields(in);  
 second.readFields(in);  
 }  
  
 @Override  
 public int hashCode() {  
 return first.hashCode() \* 163 + second.hashCode();  
 }  
  
 @Override  
 public boolean equals(Object o) {  
 if (o instanceof IntIntPair) {  
 IntIntPair tp = (IntIntPair) o;  
 return first.equals(tp.first) && second.equals(tp.second);  
 }  
 return false;  
 }  
  
 @Override  
 public String toString() {  
 return first + "\t" + second;  
 }  
  
 @Override  
 public int compareTo(IntIntPair tp) {  
 int cmp = first.compareTo(tp.first);  
 if (cmp != 0) {  
 return cmp;  
 }  
 return second.compareTo(tp.second);  
 }  
}**