BIG DATA PROCESSING ASSIGNMENT

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.

1. Mapper

Input format: (Object, Text)
Output format: (Text, Intwritable)

Here I am taking the raw input data of tweets in the text format, and giving output in (Text, Intwritable) format, where I have assigned the bin range for each tweet length in Text format to the key, for e.g. (for tweet of length 7, we will assign it to '6-10' bin) and assign constant 'one' value for each key.

Logic:

1. As per the given information, data is not sanitised, so the first thing to include in the map is to filter the records which do not comply with the expected format.
   For filtering the data, I am counting the number of semicolons (data separator) present in each record using **replaceAll("[^;]","").length()**. If number of semicolons are **not equal to three** than I am discarding that record.
2. After data filtration, I am taking the third field from the record(tweet) and computing the length of the tweet. To again filter the tweets written in foreign languages, which contain characters with non-standard encoding, I am removing the tweets which are of length more than 140.
3. To calculate the bins in map reduce, I am using Java's for and if conditions in map, so that input to the reducer will be directly the bin range as a key using below mentioned logic:

```
for (int i=5; i<=140; i=i+5) {
        if (i >= fields[2].length()) {
                binRange=(i-4)+"-"+(i);
                context.write(new Text(outKey), one);
                break;
        }
}.
```

4. Finally, output is written with bin-range as a key and one as a value. Once the bin-range is found for a tweet length, I am breaking the loop using **break** statement.

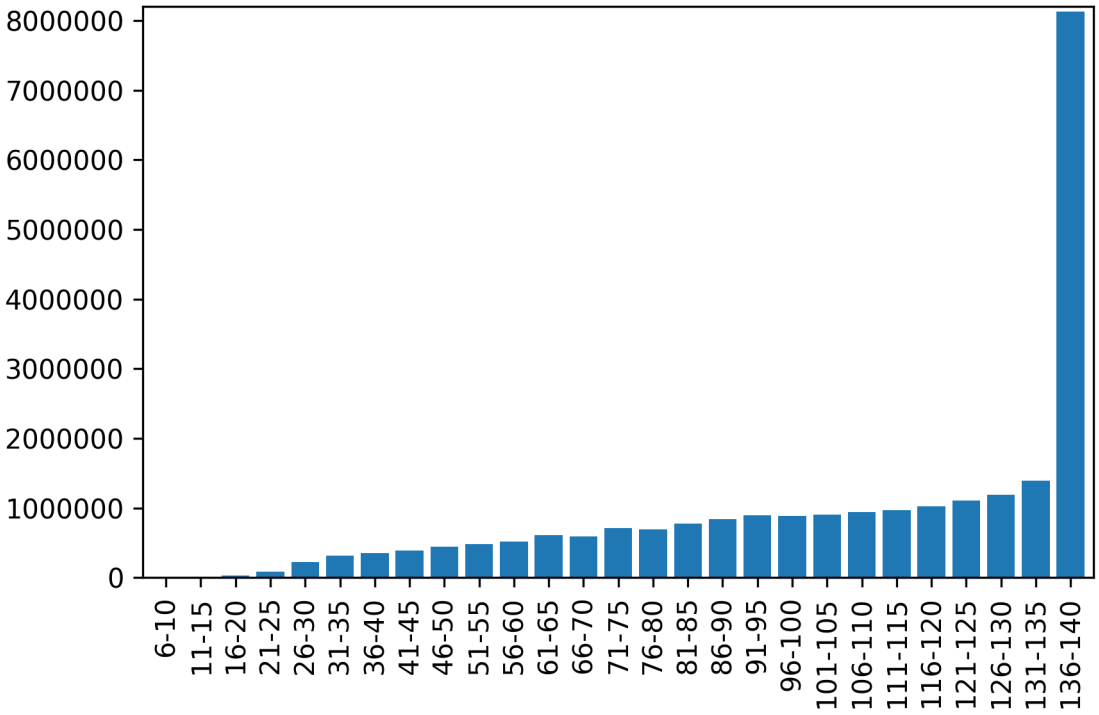
2. Reducer

Input format: (Text, Intwritable)
Output format: (Text, Intwritable)
Logic:

Reducer's input will be the (bin-range, Iterable(one)). That is for each bin-range as a key, it will have list of ones which I have aggregated by using below mentioned logic:

```
 for (IntWritable value : values) {
        sum+=value.get();
}
```

3. Histogram Plot



I have used python's matplotlib library to plot. Highest tweet lengths were observed in the last bin (136-140).

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. [30 marks]

1. Mapper

Input format: (Object, Text)
Output format: (Intwritable, Intwritable)

Here I am taking the raw input data of tweets in the text format, and giving output in (Intwritable, Intwritable) format, where I have assigned an Hour (Irrespective of the day) of the tweet posted to the key and constant 'one' value for each key.

Logic:

1. As per the given information, data is not sanitised, so the first thing to include in the map is to filter the records which do not comply with the expected format.
   For filtering the data, I am counting the number of semicolons (data separator) present in each record using **replaceAll("[^;]","").length()**. If number of semicolons **are not equal to three** than I am discarding that record.

2. After data filtration, I am taking the first field from the record(epoch time) and computing the hour using Java's Date class by converting the epoch time into Long using **Long.parseLong(fields[0])** and then using it to create Date object (**new Date(timestamp)**).
   NOTE: I have not converted epoch time into any particular time zone and used it in default UTC format.
   Conversion of first field into long was giving run time exception, hence I have used the Java's **try catch** block, to handle the exception at run time.
   To fetch the hour from the Date object, I have used **getHours()** method.
3. Finally, output is written with hour as a key and one as a value if exception is not encountered while conversion.

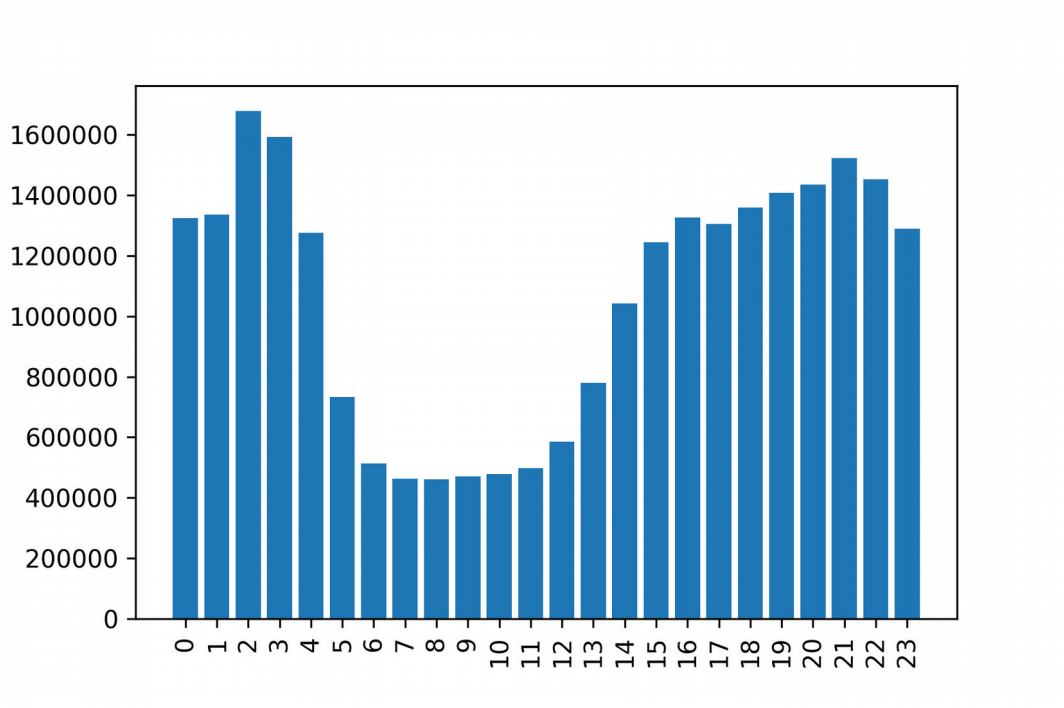
2. Reducer

Input format: (Intwritable, Intwritable)
Output format: (Intwritable, Intwritable)
Logic:

Reducers input will be the (Hour, Iterable(one)). That is for each hour as a key, it will have list of ones(number of tweets posted in that hour) which I have aggregated using below mentioned logic:

```
for (IntWritable value : values) {
        sum+=value.get();
}
```

3. Histogram Plot:



I have used python's matplotlib library to plot. Highest number of tweets were posted at 2 AM UTC.

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? [5 marks)]

1. Mapper

Input format: (Object, Text)
Output format: (Text, Intwritable)

Here I am taking the raw input data of tweets in the text format, and giving output in (Text, Intwritable) format, where I have assigned the Hash tags of the tweets posted at 2 AM UTC(Found using above program) to the key and constant 'one' value for each key.

Logic:

1. Data filtration was done using the same method as described in part 1.
2. I have computed hour with same logic as described in part 1.
3. After creating Date class object, tweets with the time of 2 AM UTC is considered to find the top hash tags during that hour.
   Syntax: **if (dt.getHours() == 2)**
4. To find the hash tags from the tweet, I have used Pattern and Matcher class of Java using the below mentioned logic:
   Pattern MY_PATTERN = Pattern.compile("#(\\S+)")
   Matcher mat = MY_PATTERN.matcher(fields[2])
   Pattern is used to compile the regular expression(to find hash tags#) and Matcher to find the all the occurrences of hash tags in the tweets(fields[2]).
5. To iterate over all the hash tags found, I have used **find()** method on matcher object.
   Syntax: **while (mat.find())**
6. Finally, hash tags were extracted using group(1) method of matcher object and output is written with hash tag as a key and one as a value.
   Syntax: **tweet.set(mat.group(1).toLowerCase())**
   here I have converted all the hashtags to the lower case, to fetch more information from the hashtags, for eg: To consider rio2016/Rio2016 as similar.

2. Reducer

Input format: (Text, Intwritable)
Output format: (Text, Intwritable)
Logic:

Reducers input will be the (hashtags, Iterable(one)). That is for each hash tag mentioned in the tweets at 2 AM UTC as a key, it will have list of ones(Number of times it has been mentioned) as values. I have aggregated all the values for each key by using below mentioned logic:

```
for (IntWritable value : values) {
        sum+=value.get();
}
```

3. Top 10 Hashtags:

Rank	hashtags	count
1	rio2016	1362380
2	olympics	86888
3	gold	61283
4	futebol	48690
5	bra	47055
6	usa	36909
7	oro	36319
8	cerimoniadeabertura	36026
9	swimming	35584
10	openingceremony	34104

1. Top two hashtags were rio2016 and Olympics, which are the events of our dataset.
2. Oro and gold are the same words in different languages, so it tells the gold medals won during that hour of the game.
3. Football and swimming are the two competitions which must have taken place during 2AM UTC.
4. Bra and USA are the name of the country which were mentioned in the hash tags.
5. Opening ceremony and cerimoniadeadeabertua means same. Which are the event happened during that hour.

Part C. Support Analysis (20%)

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]

1. Mapper

Input format: (Object, Text)
Output format: (Text, Intwritable)

- In this part, I have joined medalistrio dataset to the tweeter dataset. To do so I have used map-side replication join, where Hadoop's Distributed Cache is used for loading the smaller dataset. In the job file I have configure the cache using the following command: **job.addCacheFile(new Path("/data/medalistsrio.csv").toUri()).**
- This medalistrio dataset will be shared across all the mappers.
- Here I am taking the raw input data of tweets in the text format, and giving output in (Text, Intwritable) format, where we will assign the name of the athlete (from the medalistrio dataset) mentioned in the tweet to the key and constant 'one' value for each key.

Logic:

1. Setup method defined in the mapper will be invoked once during the initialisation of the job, which will load all the athlete names into the **arrayList** of strings using arrayList add() method.
   Syntax: **athleteList.add(fields[1]) –** second field contains athlete name
2. In map method, I have cleaned the data by filtering the records which do not comply with the expected format.For filtering the data, I am counting the number of semicolons (data separator) present in each record using **replaceAll("[^;]","").length()**. If number of semicolons are **not equal to three** than I am discarding that record.

3. Now, I am iterating over the arrayList(athlete names) which I have stored in the cache, and checking whether athlete name is present in the tweet or not using the **contains()** method.
   For iteration on arrayList, I have used: **for (String athlete : athleteList)**
   Syntax: **if (tweet[2].toLowerCase().contains(athlete.toLowerCase()))**
   Here I have converted both athlete name and tweet to the lower case before matching them, because name in the lower or upper case refers to the same athlete.
4. Finally, if the athlete name is found in the tweet then I am writing the output with athlete name as key and value as one.

2. Reducer

Input format: (Text, Intwritable)
Output format: (Text, Intwritable)
Logic:

Reducers input will be the (athlete name, Iterable(one)). That is for each athlete name mentioned in the tweet as a key, it will have list of ones(Number of times it has been mentioned) as values. I have aggregated all the values for each key by using below mentioned logic:

```
for (IntWritable value : values) {
        sum+=value.get();
}
```

3. Top 30 athletes:

Rank	athlete_name	count
1	Michael Phelps	190894
2	Neymar	178648
3	Usain Bolt	176413
4	Simone Biles	81995
5	William	65139
6	Ryan Lochte	41688
7	Katie Ledecky	39893
8	Yulimar Rojas	35638
9	Simone Manuel	28415
10	Joseph Schooling	26992
11	Rafaela Silva	26066
12	Sakshi Malik	25210
13	Wayde van Niekerk	23055
14	Andy Murray	22326
15	Kevin Durant	21509
16	Tontowi Ahmad	21153
17	Liliyana Natsir	20498
18	Monica Puig	18710
19	Andre de Grasse	18213
20	Penny Oleksiak	18109
21	Rafael Nadal	16805
22	Laura Trott	16364
23	Ruth Beitia	15939
24	Luan	15724
25	Teddy Riner	14730
26	Lilly King	14423
27	Jason Kenny	12564
28	Shaunae Miller	12399
29	Elaine Thompson	12283
30	Caster Semenya	12031

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]

1. Mapper

Input format: (Object, Text)
Output format: (Text, Intwritable)

- In this part, I have joined the medalistrio dataset to the tweeter dataset. To do so I have used map-side replication join, where Hadoop's Distributed Cache is used for loading the smaller dataset. In the job file I have configure the cache using the following command: **job.addCacheFile(new Path("/data/medalistsrio.csv").toUri()).**
- This medalistrio dataset will be shared across all the mappers.
- Here I am taking the raw input data of tweets in the text format, and giving output in (Text, Intwritable) format, where we will assign name of the corresponding sport of athlete (from the medalistrio dataset) mentioned in the tweet to the key and constant 'one' value for each key.

Logic:

1. Setup method defined in the mapper will be invoked once during the initialisation of the job, which will load all the athlete names, sport into the Hashtable of <string, string> using Hashtable's put() method.
   Syntax: **athleteInfo.put(fields[1], fields[7])**
   second field – athlete name, eighth field - sport
2. In map method, I have cleaned the data by filtering the records which do not comply with the expected format.
   For filtering the data, I am counting the number of semicolons (data separator) present in each record using **replaceAll("[^;]","").length()**. If number of semicolons **are not equal to three** than I am discarding that record.
3. Now, I am iterating over the keys of Hashtable(key:athlete names, value: Sport) using **keySet()** method of Hashtable. During iteration, I am checking whether athlete name is present in the tweet or not using the **contains()** method.
   For iterating over keys of Hashtable: **for (String athlete : athleteInfo.keySet())**
   Syntax: **if (tweet[2].toLowerCase().contains(athlete.toLowerCase()))**
   Here I have converted both athlete name and tweet to the lower case before matching them, because name in the lower or upper case refers to the same athlete.
4. If the athlete name is found in the tweet then, I am fetching the corresponding value(sport) from the Hashtable by passing name of the athlete(key) to **get()** method of Hashtable.
   Syntax: **athleteInfo.get(athlete)**
5. Finally, I am writing name of the sport as key and value as one.

2. Reducer

Input format: (Text, Intwritable)
Output format: (Text, Intwritable)
Logic:

Reducers input will be the (Sport, Iterable(one)). That is for each corresponding sport of athlete name mentioned in the tweet as a key, it will have list of ones(Number of times that sport has been mentioned) as values. I have aggregated all the values for each key by using below mentioned logic:

```
for (IntWritable value : values) {
        sum+=value.get();
}
```

3. Top 20 Sports:

Rank	sport	count
1	athletics	496335
2	aquatics	471467
3	football	312301
4	gymnastics	137955
5	judo	105185
6	tennis	89120
7	basketball	76390
8	cycling	71497
9	badminton	63806
10	wrestling	35779
11	weightlifting	26027
12	sailing	25045
13	canoe	24578
14	shooting	24345
15	equestrian	24047
16	boxing	23782
17	volleyball	18071
18	rowing	17936
19	taekwondo	16335
20	fencing	13771