Exercises sheet n° 1: algorithms and programming in MapReduce

Warm up exercise on data processing. Design and implement in Python an algorithm for n-way merge-sort of n lists of integers. We assume that each of the n lists is sorted in ascendent fashion.

The algorithm is encapsulated into a function which we name n-way-ms taking as input the list of sorted integer lists.

For instance, if L = [[1, 2, 3, 4, 5, 6, 7], [2, 3, 6, 9], [3, 7, 10]]

then n-way-ms(L) returns

[1, 2, 2, 3, 3, 3, 4, 5, 6, 6 , 7, 7, 9, 10]

Consider that you are not allowed to use any sort operation, the n-way-ms algorithm is required to produce its output by means of a **linear** scan of all the n lists inside the input list L.

In coding the algorithm it is worth using the Python pop(i) function for lists. For instance l.pop(0) returns the first value of the list l and discard it from the list itself.

Question: can you estimate the cost of the algorithm? Do you believe more efficient versions would exist?

Remark: besides being interesting in per se, n-way merge-sort is important to know because this kind of data manipulation is behind one of the main steps of shuffle-and-sort in MapReduce Hadoop.

Exercise 2. Perform execution simulation of the WordCount MapReduce job. You can find here the code for the basic version of WordCount. Use the following command for the download

```
> wget https://www.dropbox.com/s/5i0xbsd1hrldily/mapper.py
```

```
> wget https://www.dropbox.com/s/iuu4y243466gvhl/reducer.py
```

Concerning the input text file:

```
>wget https://www.dropbox.com/s/7ae58iydjloajvt/shake.txt
```

Do not forget to give the x right to the Python programs so that they can be self-executable:

```
> chmod +x mapper.py
```

```
> chmod +x reducer.py
```

Exercise 3. Design in *pseudo-code* a variant of the WordCount algorithm where the Map function performs local aggregation. The idea is to have a Map that emits couples of the kind (w, c), rather than couples of the form (w, 1), where c>=1 is the number of w occurrences in the value v of the input couple (k,v).

Do you see any problem with this approach?

Is the combiner still needed?

Can Reduce be considered as a combiner as it is?

Observe the result of the Job, do you see any problems due to data cleaning issues?

Exercise 4. Design in *pseudo-code* a MapReduce job without Combine that takes as input a file where each line contains a string indicating a Web URL and a natural number indicating the amount of time (in seconds) a user has spent on that Web page during a visit session. Of course you can have multiple lines for the same URL. The job is expected to return the list of unique URLs together with the average visit time.

Can the Reduce be considered as a Combiner?

If not, indicate the necessary modifications at the Map and Reduce level in order to have a Combine, and provide a definition of this last one.

Exercise 5. Provide a Python encoding of the following algorithms dealt with in classes: WordCount+ (Exercise 3), WordCount solving data cleaning issues, and Average calculation (Exercice 4).

Exercise 6. Given two *multi-sets* A and B, design first in pseudo-code then in Python, a MapReduce job computing their set-intersection: elements in common have to be resulted *only once;* recall that since A and B are multi-sets those elements may appear more than once in each of them.

For instance assume multi-sets A and B consisting of string values in the following two files

https://www.dropbox.com/s/bv5nn2tusy4fizi/MultiSetA.txt
https://www.dropbox.com/s/7b12qge58mu6xm2/MultiSetB.txt

The first problem we have to tackle is how to pass multiple files to a MapReduce job. This is easy as you actually need to indicate an input directory when launching a MR job, and put all the files in that directory before launching the job. Then every line in all the files in the input directory will be processed by the Map.

The second problem we have is that the MapReduce job for the intersection needs to be able to distinguish A lines from B lines, in order to detect that an element has been met in both A and B. Note that since A and B are multi-sets it is not sufficient to check in the Reduce that an element has been met twice, as that element could occur twice in only one multi-set.