INF-617 - Big Data - 2018 - 1st semester
Prof: Edson Borin
TA: Antonio Carlos Guimarães Junior

Before you start

1. Turn on the Virtual Machine and login.
2. Install the OpenCV library and python bindings using the following command:
    a. sudo apt install python-opencv
    b. Type Y (yes) when prompted and press Enter
3. Download the auxiliary files (lab1.tar.gz) and transfer it to the VM. (In case you don't know how to transfer files to/from the VM see the SCP/FileZilla Tutorial available on Moodle)
4. Unpack the files using the following command:
    a. tar xvf lab1.tar.gz

Activity 1

In this activity you will implement a classical MapReduce example: the Word Counting.

We provide a template file *WordCount.py* which already reads the input file (argv[1]) and calls the functions *map* and *reduce*. We are using Python built-in function *map*. Its syntax is `map(function, iterable)`. It applies the *function* to each element of *iterable* and puts the result in a new iterable. Although Python has a *reduce* function in the Functools module, it has a different behavior from the expected for a MapReduce application. Therefore, we defined our own reduce function, which emulates the behavior of a MapReduce reduce. Its syntax is `reduce(function, iterable)`. It groups the tuples in `iterable` by their keys (first field) and applies the `function` to each group.

Your job is to implement the functions *myMap* and *myReduce*. You can test your code using a raw text version of Moby Dick (mobydick.txt) using the following command:

```
$ python wordCount.py mobydick.txt
```

myMap:

- Input: A *line* of text.
- Output: One tuple for each word in the input *line*. Each tuple should contain the word and the number of occurrences.

Example:
Input: Big Data is like teenager sex.
Output: (Big, 1), (Data, 1), (is, 1), (like, 1), (teenager, 1), (sex, 1)
Hint: *To return multiple tuples in Python you may use the keyword yield. For our purposes, it will work in a similar way as the keywork **emit**, discussed during the MapReduce class, works.*

Hint: *You are not required to account for replicated words. Just generate a tuple for each occurence of the word.*

myReduce:

- Input: A tuple *(key, values)*, where *key* is the word and *values* is a list containing the number of occurrences.
- Output: A tuple containing the word and the total number of occurrences.

Example:
Input: (Better, [1, 1, 1, 1, 1, 1, 1, 1])
Output: (Better, 8)

Activity 2

In this activity you are going to find Wally. Wally is the main character in the series of puzzle books "Where's Wally?", in which the objective is to find him, among other characters, in a usually overcrowded scenario.

Similarly to Activity 1, we provide a template file *whereIsWally.py* which already reads the input, splits the image and calls the functions *map* and *reduce*. Your job is to implement the functions *myMap* and *myReduce* to find not only Wally but also his friends, Wizard, Wilma, and his nemesis, Odlaw. Our template also presents a function called *sequentialSearch*, which executes the search sequentially and can be used as an example.

myMap:

- Input: A chunk of the image.
- Output: One tuple with two fields. The first field must contain the name of the character whose the probability of being found in that chunk is the greatest, and the second field, must contain another tuple with two fields. The first field must contain the probability of finding the character and the second field must contain the chunk itself.

myReduce:

- Input: A tuple *(key, values)*, where *key* is the name of the character and *values* is a list of tuples, each one containing two fields, the first one being the probability of finding the character in the chunk and the second one being the chunk itself.
- Output: A tuple containing two fields. The first field must contain the name of the character and the second field must contain another tuple with two fields. The first field must contain the greatest probability and the second field must contain the associated chunk.

Example:
Input: (wally, [(0.1, chunk0), (0.7, chunk4), (0.9, chunk7)]
Output: (wally, (0.9, chunk7))

To help with your implementation, we provide the function `whoIsHere(image)`. It receives an image and returns the tuple *(who, probability)*, where *who* is the name of the character with the greatest probability of being found in that image; and *probability* is the probability of finding him.

You can test your code with the following commands. The image *output.png* will be created in the directory.

```
$ python whereIsWally.py robin_hood.jpg
$ python whereIsWally.py rome.jpg
```