CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

Computing@CIT

BIG DATA & ANALYTICS

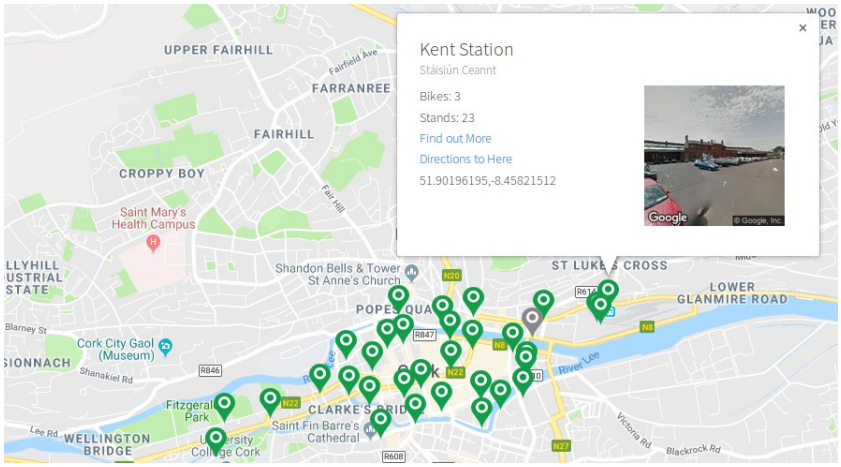
ASSIGNMENT 1: MAP-REDUCE AND SPARK

BACKGROUND.

Cork Smart Gateway is the home of open data for Cork: http://data.corkcity.ie/
At the moment it contains 8 datasets in open format. One of them is the "Coca-Cola Zero Bikes" (http://data.corkcity.ie/dataset/coca-cola-zero-bikes) which provides data related to the bike-sharing rental service supported by the city council.

While historic information on the state of the 31 bike stations in Cork city is not available, real-time information can be obtained via:

- The Coca-Cola Zero Bikes website: Visual Format
  https://www.bikeshare.ie/cork.html



- The Open Data API Endpoint: JSON Format.
  https://data.bikeshare.ie/dataapi/resources/station/data/list

{"schemeId":2, "schemeShortName":"Cork", "stationId":2032,
"name":"Kent Station", "nameIrish":"Stáisiún Ceannt", "docksCount":30,
"bikesAvailable":3, "docksAvailable":23, "status":0,
"latitude":51.902, "longitude": -8.458, "dateStatus":"15-10-2018 15:32:30" }

MY_DATASET FOLDER

My former colleague Michael O'Keefe collected data from mid January 2017 to late September 2017 by creating a service quering the API every 5 minutes from 6am to midnight and gathering all entries of a day into a file.

I have selected the files for the period 01/02/2017 – 31/08/2017 and provided the entries in the following csv format:
status ; name ; longitude ; latitude ; dateStatus ; bikesAvailable ; docksAvailable
For example, the aforementioned entry for Kent Station would be represented as follows:
0;Kent Station;-8.45821512;51.90196195;15-10-2018 15:32:30;3;23

In total, the dataset for the selected period contains 1,339,200 entries for 43,200 API requests over 200 days.

MY_CODE FOLDER

The assignment is divided into 4 hints:

- Hint1: Introductory Data Analysis with MapReduce.
- Hint2: Advanced Data Analysis with MapReduce.
- Hint3: Introductory Data Analysis with Spark Core.
- Hint4: Advanced Data Analysis with Spark Core.

The four hints are provided in the folder "my_code".
Each hint provides a Python file (A01_Hint_Number.py) with the exercises to be completed.

MY_RESULT FOLDER

The correct results to be obtained from each exercise are provided in the folder "my_result".

MARK BREAKDOWN.

The assignment is worth 25 marks. It consists on 4 hints, each of them worth 6.25 marks.

Marks will be given to each submitted exercise in the following basis:

- The function is correct and the code is efficient: 100% of marks.
- The function is correct but the code has some inefficiencies: 75% of marks.
- The function is not correct due to a single error in the code: 50% of marks.
- The function is not correct due to multiple errors in the code: 25% of marks.
- The function has not been attempted or has been barely attempted: 0% of marks.

SUBMISSION DETAILS.

Submission deadline: Sunday 17th of March, 11:59pm

I know, better to have it done at least 24 hours before :)

Please submit to Canvas (folder 6. Assignment Submission Links) the following files:

- Hint 1 -> my_mapper.py and my_reducer.py
- Hint 2 -> my_mapper.py and my_reducer.py (both for Round 1 and 2)
- Hint 3 -> A01_Hint3.py
- Hint 4 -> A01_Hint4.py

Assignment Demo.

A demo for the assignment will take place in Week 8.
I will organise a brief individual interviews with each student to run the code and discuss about it (student is expected to explain the approach followed when tackling each exercise and explain the code being submitted).

The demo is mandatory for the assignment to be evaluated.

ASSIGNMENT 1 – HINT 1 **(Week 4)**

GOAL.

Given the Coca-Cola bikes dataset provided:

- Create a MapReduce job to compute the number of times each Coca-Cola bike station ran out of bikes. Sort the stations by decreasing number of ran outs.
    - Note: A bike station is ran out of bikes if:

        status == 0 and *bikes_available == 0*

EXERCISE.

To perform the MapReduce job, complete the following code:

- **my_mapper.py** ✉ Program the function **my_map**.
- **my_reducer.py** ✉ Program the function **my_reduce**.

Note: You can use as many auxiliary functions as you need.

ASSIGNMENT 1 – HINT 2 **(Week 5)**

GOAL.

Given the Coca-Cola bikes dataset provided:

- Create a MapReduce job to compute the amount of times per day and hour that the Fitzgerald Park station was run out of bikes. Present this result as the total amount and as the percentage of total ran outs the station had across the dataset.
    - Note: A bike station is ran out of bikes if:
        status == 0 and *bikes_available == 0*

EXERCISE.

Please note that, given the percentage requirement, the MapReduce job requires two stages:

- Stage 1: Resolve the global dependency ✉ Compute the total amount of ran outs.
- Stage 2: Perform the actual MapReduce job, given the total amount of ran outs from Stage 1.

To perform the MapReduce job, complete the following code:

First_Round_MapReduce

- **my_mapper.py** ✉ Program the function **my_map**.
- **my_reducer.py** ✉ Program the function **my_reduce**.

Second_Round_MapReduce

- **my_mapper.py** ✉ Program the function **my_map**.
- **my_reducer.py** ✉ Program the function **my_reduce**.

Note: You can use as many auxiliary functions as you need.

ASSIGNMENT 1 – HINT 3 **(Week 6)**

GOAL.

Given the Coca-Cola bikes dataset provided:

- Create a MapReduce job to compute the actual ran-out times for the Fitzgerald Park station. Sort the ran-out times by increasing time in the calendar.
    - We define an "actual" ran-out time as the first moment in which our station is measured to be out of bikes. We define further measurements where the station still has no bikes to be "continuations" of the actual ran-out time previously measured.

      For example: given our 5 minutes measurements, if we notify Fitzgerald Park to be ran-out of bikes at the following times:
        - 10:06:00
        - 10:11:00
        - 15:31:00
        - 15:36:00
        - 15:41:00
        - 19:56:00
    - Then the "actual" ran-out-times are highlighed in red, with "continuations" for them highlighted in blue. In this example, 15:31:00 represents a measurement in which we realised that Fitzgerald Park was ran out of bikes. We can ensure this as 15:26:00 is not in the list. Thus, some Coca-Cola user(s) must took the last bike(s) available at the station between 15:26:00 – 15:31:00.
    - At 15:36:00 we notify that Fitzgerald park has actually ran-out of bikes, and we report it.
    - At 15:41:00 and 15:46:00 we just confirm that the bike station is still with no bikes.
    - Finally, provided that 15:46:00 is not in the list, we can ensure a Coca-Cola user(s) have brought bike(s) back to the station between 15:41:00 – 15:46:00.

      All in all, the goal of the assignment is to print by the screen the actual ran-out times, together with the length (amount of measurements) registered for each of them. In the example before we should have printed:
      - Date (10:06:00, 2)
      - Date (15:31:00, 3)
      - Date (19:56:00, 1)

EXERCISE.

To perform the MapReduce job, complete the following code:

- **my_mapper.py** ✉ Program the function **my_map**.
- **my_reducer.py** ✉ Program the function **my_reduce**.

Note: You can use as many auxiliary functions as you need.

ASSIGNMENT 1 – HINT 4 **(Week 7)**

EXERCISE.

Given the Coca-Cola bikes dataset provided:

- Complete the lines 113, 124, 135, 146, 157 and 168 of the function my_main.
    - Each line has to be completed with a call to the higher-order functions my_map, my_filter and my_reduce, as required.
    - For doing such these calls you can create your own functions or use lambda abstractions, as you consider more relevant.