DAT470/DIT066
Computational techniques for large-scale data

Assignment 3
Deadline: 2025-05-04 23:59

Problem 1 (6 pts)

Star Control II: The Ur-Quan Masters, released in 1992, is one of the greatest video games of all time. Today, the game is available as free software under the name *The Ur-Quan Masters*[1].

A key aspect of the game is travelling between the stars and collecting *minerals* that can be then sold for *Resource Units (RUs)*, the currency of the game. The files `/data/courses/2025_dat470_dit066/sc2/stars.csv` and `/data/courses/2025_dat470_dit066/sc2/planets.csv` contain information about individual stars and planets, respectively. For the purposes of this assignment, we are interested in the content of `planets.csv`.

The stars form *constellations* and individual stars within a constellation are identified by a Greek letter. For example, *Alpha Centauri* is the star *Alpha* in the *Centauri* constellation, close to *Sol*, the star that Earth orbits. Some particularly noteworthy stars may simply have a name, like *Betelgeuse*. In the dataset, the *Star* column denotes this by having *Prime* instead of the Greek letter, and the name of the star is listed under the Constellation column.

There can be multiple planets orbiting a star. These are denoted by Roman numerals. For example, *Sol III* would be the designation of Earth. Alpha Centauri has 9 planets orbiting it, from *Alpha Centauri I* to *Alpha Centauri IX*. The numbers are given in increasing order of orbit length, so the farther the planet is from the star, the higher the number. There can also be multiple moons orbiting a planet. These are denoted by a lowercase latin letter. For example, Earth's moon (Luna) has the designation *Sol IIIa*. Likewise, the moons are ordered in an increasing order of orbit length, so *Alpha Centauri IIb* is farther from its planet than *Alpha Centauri IIa*.

We want to apply MapReduce programming to determine the *total mineral value* of a star. That is, the sum of all mineral values from each planet and moon orbiting the star. The *Mineral value (RU)* column contains this information for a particular planet or a moon, the total number of RUs collected if all minerals on that planet were harvested.

The output of your program should be key-value pairs such that the name of the star system (for example, *Alpha Centauri* or *Betelgeuse*) is the key and the total mineral value is the value, for all star systems in the dataset. The order of results is unimportant. Table 1 shows some example values that you can use to verify the correctness of your implementation.

(a) Describe the operations that the programmer needs to define, using pseudocode. Include a clear description of the input and the output of each operation. (2 pts)

[1] `https://sc2.sourceforge.net/`

Table 1: Example total mineral values for some star systems.

Star system	Total value (RU)
Fomalhaut	4158
Theta Giclas	1548
Alpha Sagittarii	3417
Alpha Lacertae	2019
Zeta Corvi	1520

(b) Draw a diagram that shows the data flow of the program, across the different operations. You may assume that initially the dataset is spread across different nodes and the results potentially end on different nodes. The diagram should clearly show when data is transferred between nodes. (1 pt)

(c) Implement the program using `MRJob`. Use the file `assignment3_problem1_skeleton.py` as your starting point. The program should be runnable using the local runner and the results should be written to *stdout*.

For example, running

```
$ python3 assignment3_problem1.py -r local --num-cores 64 \
    /data/courses/2025_dat470_dit066/sc2/planets.csv
```

should output (among the other star systems) output such as the following:

```
"Fomalhaut" 4158
"Alpha Sagittarii" 3417
"Zeta Corvi" 1520
"Theta Giclas" 1548
"Alpha Lacertae" 2019
```

In your report, include a table that contains the total mineral values for the following systems: Capella, Alpha Cancri, Gamma Sagittae, Beta Lyrae, and Alpha Geminorum. (3 pts)

Problem 2 (4 pts)

We shall modify the solution to Problem 1 as follows. Instead of simply listing all star systems' values in an undefined order, we shall add a new command line parameter `-k` such that only the top k most valuable systems are reported.

(a) Create a diagram that shows the dataflow in the new program. As before, assume the dataset is distributed across the nodes. However, in this case, *the final end result*, an array of k elements, must end up on a single node. (1 pt)

(b) Implement the new program using MRJob. You probably need more than one *step*. The program shall work such that if one runs

```
$ python3 assignment3_problem2.py -r local --num-cores 64 \
    -k 10 /data/courses/2025_dat470_dit066/sc2/planets.csv
```

the output will contain the names and total mineral values of the 10 most valuable systems. Include a table that contains the names and mineral values of the 10 most valuable systems in your report. (3 pts)

Problem 3 (7 pts)

File `/data/courses/2025_dat470_dit066/twitter/twitter-2010_full.txt` contains a snapshot of the *social network* of Twitter from 2010 [1]. That is, it contains the description of a directed graph of *follows* relations: which users follow which other users. The original dataset is available at `https://snap.stanford.edu/data/twitter-2010.html`.

The data is formatted as follows, one line per user:

$$id_0\texttt{:}\quad id_1\texttt{,}\ id_2\texttt{,}\ \ldots$$

The *id*s are the Twitter IDs of users, unique integers that identify a user account (the user may change their handle but they cannot change their ID). The interpretation is that the user id_0 *follows* the users id_1, id_2, ... The *follows* relation is not reflexive, symmetric, or transitive, so users do not follow themselves, and users need not follow back their followers. Not all users have followers, and not all users follow anyone.

Files `/data/courses/2025_dat470_dit066/twitter/twitter_`x`.txt`, where x can be `1k`, `10k`, `100k`, `1M`, or `10M`, represent downsampled versions of the dataset (a subgraph induced on a random subset of users). It will make sense to use these for debugging and perhaps using the `10M` variant for making the running time measurements, as the main dataset is rather large (approximately 13 GiB).

(a) Describe a MapReduce algorithm determines the Twitter ID of the person who follows the largest amount of users, the number of users this person follows, the average number of users followed, and the number of accounts that do not follow anyone.

Describe the user-defined operations in pseudocode, and also provide a diagram that shows the flow of data. (2 pts)

(b) The file `mrjob_twitter_follows.py` contains the interface for a MRJob implementation that where you will need to fill in the blanks by implementing your algorithm. Implement the algorithm. (3 pts)

(c) Measure the scalability of your algorithm on 1, 2, 4, ..., 32 cores. Plot the empirical speedup as the function of cores. Use the 10M dataset for scalability experiments. In addition to the plot, report the single-core runtime on the dataset. (1 points)

(d) Run your implementation (with a large number of cores) on the full dataset. Report the values (ID and number of accounts followed by the account that has follows most accounts, average number of accounts followed, and the number of accounts that follow no-one. (1 point)

Note. Remember that you *cannot run any other commands* in a MRJob file. The file must only contain the class. This is because of limitations of the Hadoop architecture. Your measurements must be carried out in a *separate Python file*. You are provided a skeleton file for this purpose (`mrjob_twitter_measure.py`); you needn't return this file. You only need to return the MRJob class files.

Table 2 contains example output for the three smallest datasets. You can use this information to verify the correctness of your output.

Table 2: Correct example output for the small datasets in Problem 3.

Size	Max id	Max count	# not following	Avg. count
1k	53679303	1	999	0.00100
10k	12926542	19	9902	0.01630
100k	15839517	68	95288	0.08054

Table 3: Correct example output for the small datasets in Problem 4.

Size	Max id	Max count	# no followers	Avg. count
1k	26493932	1	999	0.00100
10k	12926542	20	9904	0.01630
100k	24083587	754	95958	0.08054

Problem 4 (7 pts)

We will now invert the relation from *follows* to *is followed by*.

(a) Describe a MapReduce algorithm determines the Twitter ID of the person who has the largest amount of followers, the number of followers this person has, the average number of followers, and the number of accounts that do not have any followers.

Describe the user-defined operations in pseudocode, and also provide a diagram that shows the flow of data. (2 pts)

(b) The file `mrjob_twitter_followers.py` contains the interface for a MRJob implementation that where you will need to fill in the blanks by implementing your algorithm. Implement the algorithm. (3 points)

(c) Measure the scalability of your algorithm on 1, 2, 4, ..., 32 cores. Plot the empirical speedup as the function of cores. Use the 10M dataset for scalability experiments. In addition to the plot, report the single-core runtime on the dataset. (1 points)

(d) Run your implementation (with a large number of cores) on the full dataset. Report the values (ID and number of followers by the account that has most followers, average number of followers, and the number of accounts that have no followers. (1 point)

Table 3 contains example output for the three smallest datasets. You can use this information to verify the correctness of your output.

Hints

- Remember that the code for measuring the execution time must be in a different file from your job class.
- Look up the documentation of MRJob, it contains useful examples.
- You may need more than one job step in Problems 2 and 4.

- You will need to use combiners to achieve reasonable performance.
- Do not hog resources: if you only measure the running time with one core, do not request 32.
- Remember that you need to do two things: request an appropriate number of cores through SLURM and also specify the correct `--num-cores` in your MRJob program.
- You must update the measurement file to use the correct class for Problem 4.
- Start experimenting with the small files first. You may even do this locally. Also, you can compute the baseline values yourself, e.g., with NumPy for the small files (it doesn't take that long).
- In order to create an appropriate environment for MRJob, use the following invocation:
  `$ conda create -n mrjob python==3.12 mrjob`
  The reason is that Python 3.13 has compatibility issues with MRJob.

Returning your assignment

Return your assignment on Canvas. Your submission should consist of a report that answers all questions as PDF file (preferably typeset in LaTeX) called `assignment3.pdf`. In addition, you should provide the code you used in Problems 1, 2, 3, and 4 as `assignment3_problem1.py`, `assignment3_problem2.py`, `assignment3_problem3.py`, and `assignment3_problem4.py`, respectively. The code must match the interfaces of the skeleton files provided; the command line parameters must not be changed, and output must be correct. Use a separate file for making the runtime measurements; you do not need to return this file. Do *not* deviate from the requested filenames and do *not* produce the plots in these files; these files will be used for evaluating the quality of your implementations automatically.

References

[1] Haewoon Kwak et al. "What is Twitter, a Social Network or a News Media?" In: *Proceedings of the 19th international conference on World wide web (WWW '10)*. 2010. DOI: https://doi.org/10.1145/1772690.1772751.