LINGI2364: Mining Patterns in Data
Project 1: Implementing Apriori

1 Context

This project is focused on the Apriori algorithm which aims to find the frequent itemsets in a dataset given a fixed minimum support. The Apriori algorithm is the most basic *join-based* algorithm for frequent itemset mining. It exploits the *anti-monotonicity* property and uses a *level-wise* approach. Many optimisations have been proposed to improve the performances of the basic implementation. In this project, you will have to implement variations of the algorithm and compare their performances.

2 Directives

The project will be done by groups of two students and graded using INGInious. Please join the course on INGInious (`https://inginious.info.ucl.ac.be/course/LINGI2364`) and register in a (INGInious) group as soon as possible.

For this project, you will have to implement **at least two** different frequent itemset miner algorithms in Python. One of them **must** be a version of the Aprioi algorithm (you can choose the candidate generation method, how to compute the frequency, etc). The other must be either a different version of the Apriori algorithm with additional optimisations or an implementation of a Depth First Search algorithm such as ECLAT. You will then have to compare your frequent itemset miners on several datasets with different support values in order to determine the impact of your implementation choices and optimisation(s) on the performance of the algorithms. Additionally, you will have to write a short report explaining the key aspects of your algorithms and containing your performance analysis.

The project will be done in Python and a template (`frequent_itemset_miner.py`) is provided. You are free to modify this template as long as you respect the following directives

Input The methods calling your frequent itemset miners should be called `apriori` for your implementation of the apriori algorithm and `alternative_miner` for your second implementation (Depth First or Apriori variation). These methods must have the following signatures: Each method takes two parameters: i) The path to the data set file ii)) the minimum frequency threshold. We must be able to make the followings call (in a python script): `apriori("toy.dat", 0.125)` and `alternative_miner("toy.dat", 0.1)`.

Output Your methods must print **on the standard output** (i.e. use the `print` method from Python) each pattern on the following form:

$$[item_1,\ \ldots,\ item_n]\ (freq)$$

where $item_1,\ \ldots,\ item_n$ are the n items in the itemset and $freq$ is the frequency of the itemset

You will be graded based on several criteria:

- The correctness and performances of your implementations (8/20).
- The quality and relevance of your report, justifications and performance analysis. Your source code will also be checked and graded here (e.g. hard-coding the solutions will give a grade of 0) (12/20).

Your submission should be uploaded following the modalities given on INGInious.

3 Report

Your report must not exceed 4 pages. It has to contain a **very short** description of the optimisations you implemented. In addition to that, you must also discuss the **implementation choices** that you made and explain the impact they have on the execution of your program. The report must also contain an experimental comparison of the performance in terms of time (and optionally memory) on several datasets with different support values. You can also (this is not mandatory) briefly discuss the difficulties that you encountered during the project.

Your report must contain the number of your group as well as the names and NOMAs of each member. It should be written in correct English. Be precise and concise. If you do not have the content to fill four pages, it is ok to submit less pages. Do not hesitate to use tables or graphics to depict the results of your comparison of the variations of the algorithm.

4 Datasets

Different data sets are available for your experiments. For each data set we provide the solution for some thresholds. We also provide a python script that can be used to compare two output files. We **strongly encourage** you to **test locally** your code before submitting on INGInious. We keep some hidden instances on INGInious, but the available instances are part of the grading process on INGInious (except the Toy data set).

5 Important tips

- The algorithms might take a lot of time on some of the datasets for low frequency values. When testing your implementations, always start with a high frequency and decrease it until your algorithm takes too much time.
- One of the files used in the tests has a large number of different items. Depending on how you implemented your algorithms, this can have a huge impact on your performances. Think about what you can do to deal with this problem.
- If you have other performance issues, try to identify which are the particularities in the datasets that make your algorithm inefficient. You can also find the parts of your code that cause this issue. Also, be careful with the structures and operations that are used in your code.
- The performance analysis of your algorithms is the part of the report that is worth the most points. It will require some time to perform an appropriate number of experiments. Do not underestimate this part of the project.
- Test your code locally before submitting on INGInious. This tool is used by several courses and its resources limited. INGInious is a tool to grade your code, not debug it!
- Follow carefully the directives on INGInious when submitting your code.
- Make full use of the time allocated. Do not start the project just before the deadline!
- Do not forget to comment your code. Your code source will be read during the evaluation process.
- Plagiarism is forbidden and will be checked against! Do not share code between groups. If you use online resources, cite them.
- Do not include the Toy data sets in your experiments, it is only provided for debugging purposes.

- If you have surprising results, discuss them. The goal of the report is to discuss and analyze what you find, not to have predefined results.