CS 412: Spring '24
Introduction To Data Mining

Assignment 5

(Due Monday, April 29, 23:59)

- The homework is due on Monday, April 29, 2024, at 23:59. Note that this is a hard deadline. We are using Gradescope for all homework assignments. In case you haven't already, make sure to join this course on Gradescope using the code shared on Canvas. Contact the TAs if you face any technical difficulties while submitting the assignment. Please do NOT email a copy of your solution. We will NOT accept late submissions (without a reasonable justification).
- Please use Campuswire if you have questions about the homework. Make sure to appropriately tag your post. Also, scroll through previous posts to make sure that your query was not answered previously. In case you are sending us an email regarding this Assignment, start the subject with "CS 412 Spring '24 HW5:" and include **all** TAs and the Instructor (Jeffrey, Xinyu, Kowshika, Sayar, Ruby).
- Please write your code entirely by yourself. All programming needs to be in Python 3.
- The homework will be graded using Gradescope. You will be able to submit your code as many times as you want.
- The grade generated by the autograder upon submission will be your final grade for this assignment. There are no post deadline tests.
- Do NOT add any third-party libraries in your code. Built-in Python libraries are allowed.
- For submitting on Gradescope, you would need to upload a Python file named `homework5.py`. A python file named `homework5.py` containing starter code is available on Canvas.
- You are provided two sample test cases on Canvas, you can try debugging your code with minsup values of 2 or 3 with the given sample inputs. On Gradescope, your code will be evaluated on these sample test cases as well as additional test cases. You will get autograder feedback for the sample test cases but not for the other hidden test cases.
- Late submission policy: there will be a 24-hour grace period without any grade reduction, i.e., Gradescope will accept late submissions until **Tuesday, April 30, 2024, at 23:59.**. Unfortunately, we will NOT accept late submissions past the grace period (without a reasonable justification).

Problem Description

The focus of the programming assignment is to implement a frequent itemset mining algorithm based on **Apriori method with pruning**. Given a transacion database TDB and a minimum support threshold *minsup*, the algorithm should simulate the Aprirori method with pruning - returning all the candidate itemsets and the frequent itemsets at each scan of the algorithm.

We will test your code on relatively small transaction databases (maximum 15 transactions of length 10). Please make sure the runtime of your code does not exceed 10 seconds for such small databases.
You will not get any credit if your code does not work.

Input Format: The input will be a plain text file with a transaction database, with each line corresponding to a transaction composed of a string of letters. Each letter in a transaction corresponds to an item. For example, the transaction database *Test-1.txt* is as following:

$$\begin{array}{|c|}\hline ACD \\ BCE \\ ABCE \\ BE \\ \hline \end{array}$$

Your code will take two inputs:

1. Path to a plain text file pointing to the transaction database; and
2. An integer, the minimum support.

Output Format: Your code will implement a function called apriori based on Apriori algorithm with pruning. It will return a 3-level nested dictionary.

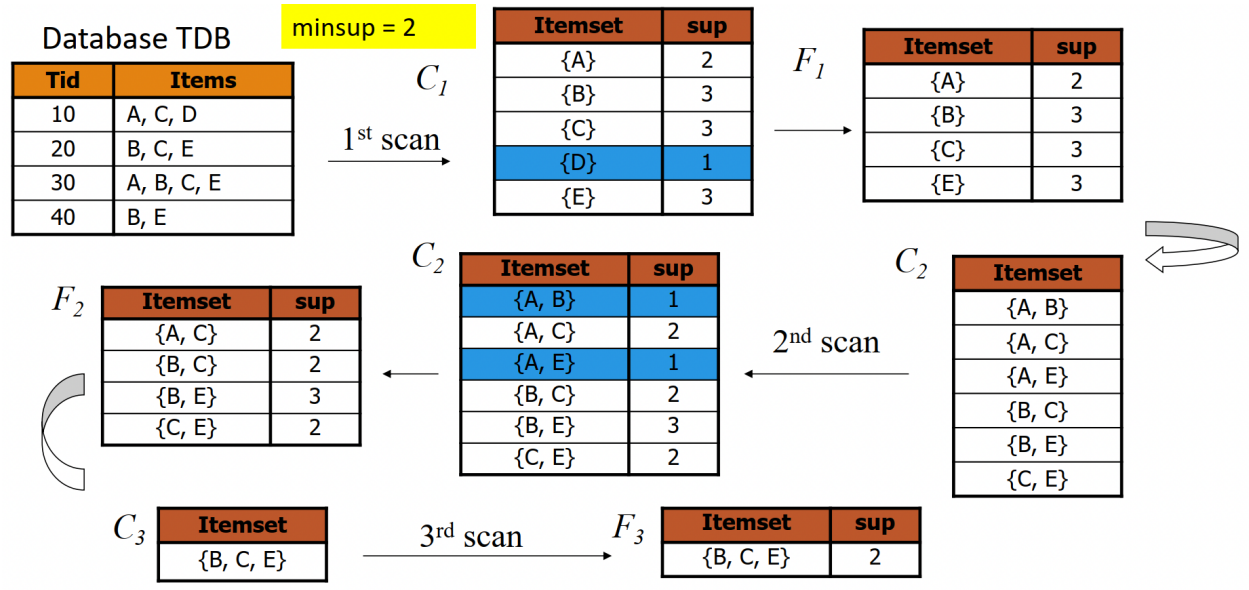

Figure 1: Simulation of *Test-1.txt*

Figure 1 shows the simulation of the Apriori algorithm with pruning for an example. The expected output (3-level nested dictionary to be returned from the *apriori* function of your code) is shown in Figure 2.

Output dictionary structure

Let's consider the 3 levels of the dictionary as *outer*, *middle*, and *inner* levels. The keys of the outer level will denote the scans (or iterations) of the algorithm. For example, in Figure 1, the algorithm terminates after 3 scans and so in the dictionary of Figure 2, we have 3 elements in the *outer* dictionary, where the keys of these 3 elements are integers 1, 2, and 3 denoting the first, second and third scans of the algorithm, respectively. The scan numbers must start from 1 and should of integer data type.

Value of each scan no.(i.e., each key in the *outer* layer) is a dictionary, which are the *middle* layer dictionaries. In Figure 1, the algorithm generates the candidate itemsets and the frequent itemsets in each scan. So each *middle* dictionary will have two elements - the key c denoting the candidate itemsets and the key f denoting the frequent itemsets. The data type of keys c and f should be string.

Value for the keys c and f will be dictionaries - denoting the candidate itemsets and the frequent itemsets of the corresponding scan. The keys of these dictionaries will be of string data type denoting the itemsets. The values will be of integer data type denoting the support of the associated itemset.

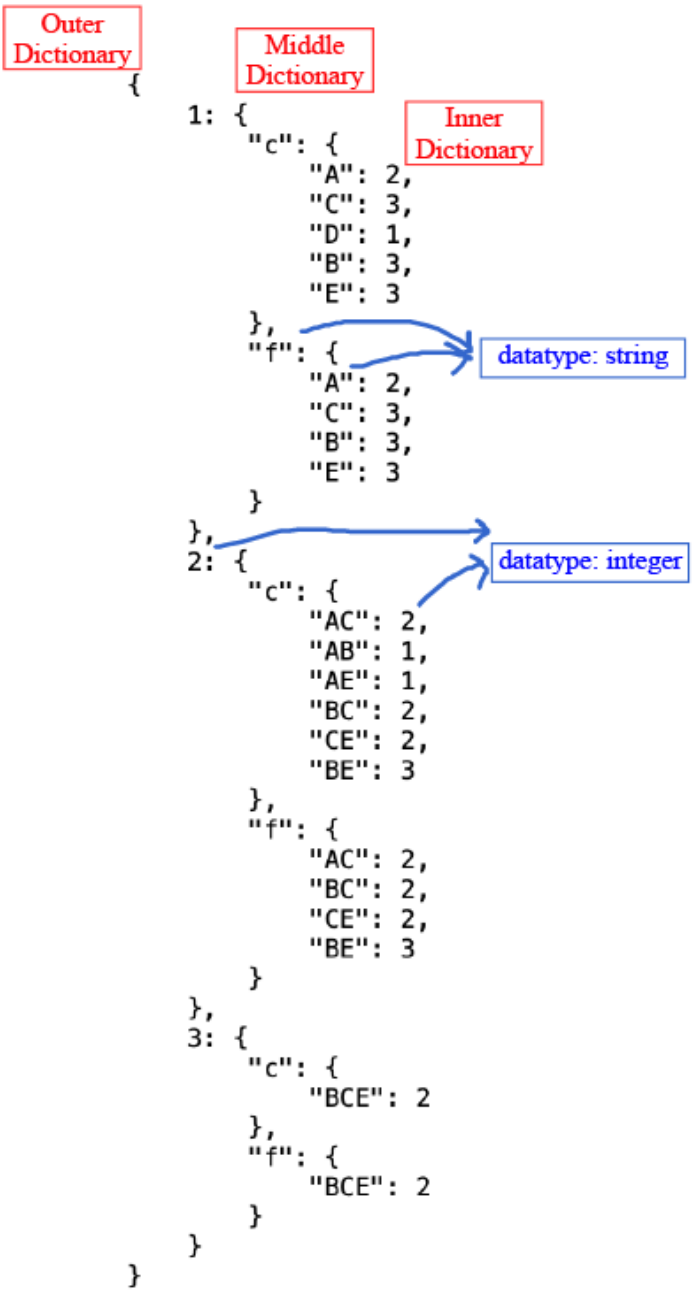

Figure 2: Expected output for *Test-1.txt*

Notes

1. **Pruning:** While creating the candidate itemsets at every scan, you are supposed to apply pruning. For example, in Figure 1, at the 2nd scan, merging AC and BC can generate the candidate ABC for the 3rd scan, but as a subset AB of ABC is absent in the frequent set F_2, ABC is pruned and not included in the candidate set C_3. Similarly, the ABC is absent in the corresponding *inner* dictionary of Figure 2.

2. **Sorting:** The alphabets in the strings of the keys of the *inner* dictionaries should be alphabetically sorted. For example, BCE should not be any of BEC, CBE, CEB, ECB, EBC.

3. **Filename:** The submitted file should be named homework5.py, otherwise Gradescope will generate an error.

4. **Terminating:** If the frequent itemsets of a scan has only one itemset, the algorithm will terminate and no further scan will be done. For example, in Figure 1, F_3 has only one itemset BCE, so the 4^{th} scan was not performed.

   Also, if the candidate itemsets of a scan is empty, that scan will be discarded and won't be included in the output. For example, let's assume for some input, the frequent itemsets F_2 obtained at 2^{nd} scan are AC, BC. So the candidate itemsets C_3 for the 3^{rd} scan will be empty (ABC won't be in C_3 as AB is absent in F_2 and so ABC will be pruned). In this case, the output will not include the 3^{rd} scan as both C_3 and F_3 are empty.

5. **Error:** If you get an error from the autograder that says the code could not be executed properly and suggests contacting the course staff, please first check carefully if your code is running into an infinite loop. An infinite loop is the most likely cause of this error.

What you have to submit

You need to submit a Python file named homework5.py. A starter code is posted on Canvas. Implement the code to compute the required output. You can add as many functions in your code as you need. Your code should be implemented in Python 3 and do NOT add any third-party packages in your code; you can use Python's built-in packages.

Your code must include a function named apriori which takes following two inputs:

1. Transaction database (*filename* in the starter code): path to a plain text file with the sequence database as shown in the example above. Each line will have a transaction. Note that there will be an empty line at the end of the file.

2. Minimum support (*minsup* in the starter code): an integer indicating the minimum support for the frequent itemset mining.

A call to the function will be like:

```
apriori("hw5_sample_input_1.txt", 2)
```

Additional Guidelines

The assignment needs you to both understand algorithms for frequent itemset mining, in particular Apriori with pruning, as well as being able to implement the algorithm in Python. Here are some guidelines to consider for the homework:

- Please start early. It is less likely you will be able to do a satisfactory job if you start late.

- It is a good idea to make early progress on the assignment, so you can assess how long it will take: (a) start working on the assignment as soon as it is posted. Within the first week, you should have a sense of the parts that will be easier and parts that will need extra effort from you; (b) Solve an example

(partly) by hand as a warm-up to get comfortable with the steps that you will have to code. For the warm-up, you can use the two sample test cases provided on Canvas named hw5_sample_input_1.txt and hw5_sample_input_2.txt.