Programming Project 2022-23

Updated version: 10th January 2023

1 Introduction

The computation of frequent itemsets is a fundamental primitive for data analysis. Its definition was inspired by *market-basket analysis*, an application domain whose target is the discovery of significant purchase behaviours in a population of buyers. Consider a dataset $T = \{t_1, t_2, \ldots, t_N\}$ of N *transactions* (i.e., baskets) over a set I of d *items*, with $t_i \subseteq I$, for $1 \leq i \leq N$. An *itemset* is a set of items $X \subseteq I$, and its *support w.r.t.* T, is defined as

$$\sigma_T(X) = |\{t_i \in T : X \subseteq t_i\}|,$$

that is, the number of transactions of T that contain X. It is immediate to see that for two itemsets X, Y with $X \subseteq Y$, their supports are such that $\sigma_T(X) \geq \sigma_T(Y)$, a property referred to as **antimonotonicity of support**. As an example, consider the following dataset defined over the set $I = \{\text{A, B, C, D, E, F}\}$ of items (with $d = 6$):

Dataset T	
i	t_i
1	ABEF
2	ACF
3	AD
4	BEF

With respect to this dataset, the itemset $X = \text{AF}$ has support $\sigma_T(X) = 2$, while the itemset $Y = \text{ABF}$ has support $\sigma_T(Y) = 1$. The classical frequent itemset problem requires that given T and a support threshold $\bar{\sigma}$ all itemsets X of support $\sigma_T(X) \geq \bar{\sigma}$, referred to as *frequent itemsets*, be determined. In the project, you will explore a variant of the problem, defined below.

For a dataset T of transactions over a set I of d items, let $X_1, X_2, \ldots, X_{2^d-1}$ be an enumeration of all non-empty itemsets by non-increasing support, so that $\sigma_T(X_i) \geq \sigma_T(X_j)$ for every $i \leq j$. Given T and a positive integer $K < 2^d$, the set of **Top-K frequent itemsets w.r.t.** T is defined as

$$\text{TopK-FI}(T, K) = \{Y \subseteq I : \sigma_T(Y) \geq \sigma_T(X_K)\}.$$

Some important observations:

- TopK-FI(T, K) contains *at least* K itemsets: namely, X_i, with $1 \leq i \leq K$, and any other itemset X_j, with $j \geq i$, such that $\sigma_T(X_j) = \sigma_T(X_K)$.

- While in general the enumeration of the itemsets $X_1, X_2, \ldots$ by non-increasing support is not unique, since itemsets with the same support may occur in any relative order among themselves, the set TopK-FI(T, K) is unique.

The following table lists all non-empty itemsets of support ≥ 0 by non-increasing support, for the dataset T of the above example.

i	X_i	$\sigma_T(X_i)$	i	X_i	$\sigma_T(X_i)$
1	A	3	12	AB	1
2	F	3	13	AC	1
3	B	2	14	AD	1
4	E	2	15	AE	1
5	AF	2	16	CF	1
6	BE	2	17	ABE	1
7	BF	2	18	ABF	1
8	EF	2	19	ACF	1
9	BEF	2	20	AEF	1
10	C	1	21	ABEF	1
11	D	1			

From the table, it can be seen that TopK-FI$(T, 2)$ = {A, F}, while TopK-FI$(T, 5)$ = {A, F, B, E, AF, BE, BF, EF, BEF}.

2 Algorithm for computing TopK-FI(T, K)

In the project you will implement the following algorithm to compute TopK-FI(T, K) for a given T and K, and for a positive integer parameter M provided in input. The algorithm is described at a high level as a sequence of steps. Let I be the set of items occurring in the transactions of T, and assume that a total ordering among the items is defined.

1. Compute the supports of all singleton itemsets $\{a\}$, with $a \in I$.
2. Initialize a Priority Queue Q with the set of entries $\{(\sigma_T(\{a\}), \{a\}) : a \in I\}$. Note that the support is the key of the entry, and the larger the support, the larger the priority.
3. Initialize an empty list S of entries.
4. For K iterations, do
   (a) If $Q = \emptyset$ then return. Else extract from Q the entry $(\sigma_T(X), X)$ with largest key.
   (b) Add $(\sigma_T(X), X)$ to S.
   (c) Let a be the largest item in X. For every item $b \in I$ with $b > a$, compute the support of $Y = X \cup \{b\}$ and, if $\sigma_T(Y) > 0$, insert $(\sigma_T(Y), Y)$ in Q.

5. Let $(\sigma_T(X), X)$ be the K-th entry extracted from Q (i.e., the entry extracted in the last iteration), and define $\bar{\sigma}_K = \sigma_T(X)$. Repeat Steps 4a÷4c until Q contains no more itemsets of support $\bar{\sigma}_K$. In this fashion, all itemsets of support $\bar{\sigma}_K$ will be generated, extracted, and added to S.
6. Output $|S|$.
7. If $|S| \leq M$, output the list S.

It can be easily shown that at the end of the algorithm, S contains the itemsets of TopK-FI(T, K) and their supports. As an example, let us see how Q and S evolve in the first 3 iterations of the algorithm, when applied to the dataset shown before to compute TopK-FI$(T, 5)$. For simplicity we depict Q as a unsorted list. Note that the usual alphabetical ordering for the items is assumed.

Initialization	
Q	(3,A)(2,B)(1,C)(1,D)(2,E)(3,F)
S	$\emptyset$
After Iteration 1	
Q	(2,B)(1,C)(1,D)(2,E)(3,F)(1,AB)(1,AC)(1,AD)(1,AE)(2,AF)
S	(3,A)
After Iteration 2	
Q	(2,B)(1,C)(1,D)(2,E)(1,AB)(1,AC)(1,AD)(1,AE)(2,AF)
S	(3,A)(3,F)
After Iteration 3	
Q	(1,C)(1,D)(2,E)(1,AB)(1,AC)(1,AD)(1,AE)(2,AF)(2,BE)(2,BF)
S	(3,A)(3,F)(2,B)

3 Assignment

You must develop a Java program `TopKFI.java` which receives in input, as command-line arguments:

1. a path to a text file containing a dataset T of transactions (format described below)
2. an integer $K \geq 0$
3. an integer $M \geq 0$

Input format. The items of the transactions are represented as integers (`int`), and each transaction of T is stored in a separate line of the file, with its items separated by a space and

sorted in increasing order. You can assume that the items are unique in every transaction (there are no repeated items in the same line).

The program must do the following:

1. Read the input parameters from the command line, and read the transactions of T from the dataset file.
2. Compute the list S of entries $(\sigma_T(X), X)$, containing all itemsets $X \in \text{TopK-FI}(T, K)$, using the algorithm described in the previous section.
3. Print the size $|S|$ of S
4. If $|S| \leq M$, print all the entries in S in the same order as they have been generated.

Output format. Print $|S|$ in the first line. Then, if the program outputs the entries of S, for each entry $(\sigma_T(X), X)$ use a separate line and print X (with items separated by spaces) and $\sigma_T(X)$ (separated by a space and between parenthesis), in this order, with no other characters. Note that the relative order of itemsets with the same support is not important.

Example of the I/O format. Consider the example dataset given in Section 1. Assume that $A = 1, B = 2, C = 3, D = 4, E = 5$, and $F = 6$. The correct input for the dataset T is

```
1 2 5 6
1 3 6
1 4
2 5 6
```

A correct output for the elements in the set TopK-FI$(T, 5)$ (if $M \geq 9$) is

```
9
1 (3)
6 (3)
2 (2)
5 (2)
1 6 (2)
2 5 (2)
2 6 (2)
5 6 (2)
2 5 6 (2)
```

Note that, when $M < 9$, the code should only print a single line with the size of S:

```
9
```

4 Deliverables

Submit the `TopKFI.java` to the dedicated section in the Moodle Exam site. We will provide you a template for the `TopKFI.java` file, with some code to parse the input commands and read the transactions from the dataset file, and some example datasets to test your program. While you are free to modify the code of the template, note that your program cannot call library functions other than those in java.lang, java.io, and java.util.

IMPORTANT: do not change the name of the java file, and make sure that:

- your program compiles using the command `javac TopKFI.java`
- your program runs using the command `java TopKFI dataset K M`, where `dataset` is the relative paths to the dataset file, and `K` and `M` are positive integers
- the formats of the input and the output are **exactly** the same of the ones described in previous sections

Note that, otherwise, we may not be able to evaluate your submission (and you will get 0 points for the project)!

5 Grading

The grading scheme will be the following:

- **1 points:** if your implementation is correct (it will be tested on a number if instances unknown to you);
- **2 points:** if your implementation is correct and is in the top-30 of all returned algorithms in terms of running time;
- **3 points:** if your implementation is correct and is in the top-20 of all returned algorithms in terms of running time.

The bonus points apply to all 4 exams of 22/23 academic year.

6 Hints for the implementation

- To compute the supports of all singleton itemsets $\{a\}$, with $a \in I$, you may use a Hash Table, as implemented by the class `HashMap` in `java.util`. The same data structure might turn out useful also in Step 4c of the algorithm.

- To implement the Priority Queue Q you may adapt the class `PriorityQueue` in `java.util` or develop your own implementation of a priority queue (Chapter 9 of the textbook may help to this purpose).
- You might consider enriching the Priority Queue Q so that it contains entries $(\sigma_T(X), (X, L_X))$, where the key is the support $\sigma_T(X)$ of the itemset X, and the value is the pair (X, L_X), where L_X is the list of indices of transactions containing X: $L_X = \{i : X \subseteq t_i\}$. This might allow for a faster computation of supports in Step 4c of the algorithm.
- In case the space used by the Priority Queue Q grows too large, you might consider removing entries with low frequency, as follows. Suppose that, at a given point of the execution of the algorithm, it holds that $|S| = i$, for some $0 \leq i < K$, and define $j = K - i$. Let $\tilde{\sigma}_j$ be the key of the j-th entry of Q in non-increasing order, or $\tilde{\sigma}_j = 1$ if Q contains less than j entries. Then, you can safely remove from Q all entries with key $< \tilde{\sigma}_j$. Instead, if $|S| = K$ you can safely remove from Q all entries whose key is smaller than the key of the last entry inserted in S.