****

Due Wednesday 04/23/2014 by start of class --- WILL TAKE TIME…A LOT OF IT

To be done in groups already formed for projects

****

Implement the *Apriori* algorithm to find all frequent itemsets and association rules given a support and confidence thresholds (each as a percentage between 0 and 100) as user input.

PS: for both steps, YOU MAY IGNORE THE ANTI-MONOTONE PRUNING STEP THAT PRECEDES COUTING

a. Your program should take as input
(1) a data file which contains transactions each on a line containing items separated by spaces,
(2) support percentage threshold (***minsupp***) between 0 and 100,
(3) confidence percentage threshold (***minconf***) between 0 and 100, and
(4) an integer option (pass value 0 if only frequent itemsets need to generated and saved and 1 if, additionally, the rules also need to generated and saved).

b. Your program should output the frequent itemsets and association rule into two distinct output files called **`FIS.out`** and **`RULE.out`**, respectively, with the following formats:

`FIS.out`: e.g.
- 1 10 14 (0.89)
- 22 31 45 (0.09)
- …

`RULE.out`: e.g.
- 1 22 ==> 45 (s=0.89, c=0.12)
- 33 44 ==> 1 2 3 4 5 (s=0.10, c=0.92)
- …

Please pay close attention to the format of your input and output files as I will be using a program to test your code.

c. Document your code FULLY.

d. Folder **`Datasets`** contains a number of popular ARM datasets that you'll be using for this assignment. First, test your implementation on **`simpledataset.dat`**. Generate by hand all frequent items and rules for different thresholds (or use Weka instead) in order to make sure that your implementation is producing the correct results.

e. Experimental study: Once you're confident that your implementation is working properly, proceed to conduct an experimental study that shows how your frequent itemset mining step performs on real-life datasets. The table below shows four popular datasets (full descriptions can be found on http://fimi.cs.helsinki.fi/data/) available in folder **`Datasets`**.

Dataset name	Description	# of items	Avg. length	# of transactions
chess.dat (334.3KB)	a set of chess moves --- Game domain theory and inductive reasoning	75	37	3,196
retail.dat (4.0MB)	market basket data from an anonymous Belgian retail store	16,470	10.3	88,126
accidents.dat (33.9MB)	traffic accident data from a Belgian study on car accidents	468	33.8	340,183
kosarak.dat (30.5MB)	click-stream data for a Hungarian on-line news portal	41,270	8.1	990,002

Every dataset has its items numbered starting from 1 up to the total number of items (e.g., in **chess.dat**, items are named 1, 2, 3, … 75). However, items in **retail.dat** start at 0. If you need the total number of items or the total number of transactions, you will have to scan the dataset. The above datasets don't record this information directly. Please recall that I WILL RUN YOUR PROGRAM ON OTHER DATASETS THAT YOU HAVEN'T SEEN --- Your program should run smoothly ON ANY TRANSACTIONAL DATASET formatted as described above.

In ARM, experimental studies focus on showing the effect of varying the support threshold on the execution time needed. The two tables below show what support thresholds to use for testing your implementation on each dataset (shown below in percentage as well as actual numbers). Dense datasets contain longer transactions (e.g., containing a large number of items) and tend to produce large frequent itemsets, unlike sparse datasets. This explains why I've chosen high support thresholds for the dense datasets and very low support thresholds for sparse ones.

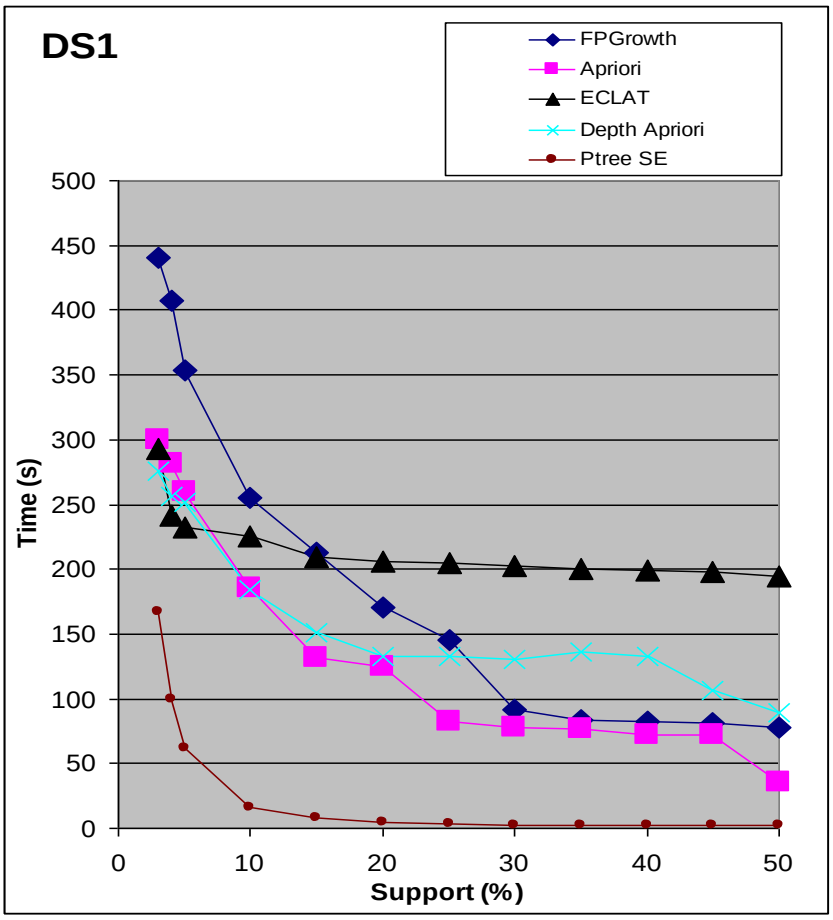
Dense Dataset name	95%	75%	55%	35%
chess.dat	3036	2397	1758	1119
accidents.dat	323174	255137	187101	119064

Sparse Dataset name	1%	0.75%	0.50%	0.25%
retail.dat	881	661	441	220
kosarak.dat	9900	7425	4950	2475

For each of the above datasets, run your implementation (ONLY THE FREQUENT ITEMSET MINING STEP) for the specified support thresholds and record

(1) the execution time it took to complete (in seconds) and
(2) the total number of frequent itemsets generated.

Find another team to compare your execution times against. If you prefer not to work with other teams, please compare your performance against any of the implementation shown on this page: http://fimi.ua.ac.be/src/ (**you'll to download the code and figure out how to run it on your machine**). Plot the recorded time information in descriptive graphs such as the one shown below which shows the effect of varying the support threshold on some dataset called DS1 for a number of implementations (in your case, you'll have only two implementations: yours and the one you're comparing against). You'll need one such graph per dataset. Also, include tables and similar graphs showing the effect of varying support on the total number of produced frequent itemsets.



****

- Please note that your program should work for ANY DATASET formatted as described above. I WILL RUN YOUR PROGRAM ON THESE DATA FILES AS WELL AS ON ONES THAT YOU HAVEN'T SEEN.
- Document your code properly AND INCLUDE INSTRUCTIONS FOR ME ON HOW TO RUN IT (A README FILE WILL DO)
- Please submit via email a zipped folder with your FULL name (include names of all team members) which contains the following files:
  (1) Self-contained source code for a program which performs K-means clustering along with instructions on how to run it (***PS: self-contained means that all needed external libraries must be included***), and
  (2) your report.

****