1. 6.1 Set C of all frequent closed itemsets on a data set D, as well as the support count for each frequent closed itemset. Describe an algorithm to determine whether a given itemset X is frequent or not, and the support of X if it is frequent.

   - Because C $\{C_i\}$is all frequent closed itemsets, which means it contains complete information regarding its corresponding frequent itemsets.(Page 248) The complete information here can represent the actual support count for all subset Y belonging to C. If X is one of Y belonging to C_i, then X is frequent with the same support of C_i. Otherwise, X is not frequent.

     When interpreting with math notation: if X $\subseteq C_i$, then support(X)=support(C_i)

2. 6.3 Apriori, prior knowledge.

   (a) Prove that all nonempty subsets s of a frequent itemset l must also be frequent.

   - if l appear n times in a certain database D, then its subset s must also appeared n times at least. $supcount(s) \geq supcount(l) = n \geq minsupport$, thus all nonempty subsets s of a frequent itemset l must also be frequent. (*suppcount* is short for absolute support count here)

   (b) Prove that the support of any nonempty subset s' of itemset s must be at least as great as the support of s.

   - From the prove of (a), we know $supcount(s') \geq supcount(s)$, when the two sides dividing the same total count of the whole dataset, we get $P_{sup}(s') \geq P_{sup}(s)$, meaning that the support of any nonempty subset s' of itemset s must be at least as great as the support of s. (P_{sup} stands for relative support here)

   (c) Given frequent itemset l and subset s of l, prove that the confidence of the rule " $s' \Rightarrow (l - s')$ " cannot be more than the confidence of "$s \Rightarrow (l - s)$ ", where s' is a subset of s.

   - $s' \subseteq s \subseteq l$, accordint to (b), $P_{sup}(s') \geq P_{sup}(s) \geq P_{sup}(l)$.
     Hence $P_{conf}(s') = \frac{P_{sup}(s' \cup l - s')}{P_{sup}(s')} = \frac{P_{sup}(l)}{P_{sup}(s')} \leq \frac{P_{sup}(l)}{P_{sup}(s)} = \frac{P_{sup}(l-s)}{P_{sup}(s)} = P_{conf}(s)$. ($P_{conf}$ means confidence here)

   (d) A partitioning variation of Apriopri subdivides the transactions of a database D into n noneoverlapping partitions. Prove that any itemset that is frequent in D must be frequent in at least one partition of D.

   - Assume itemset s is frequent in D, D has N transactions being subdivided into n noneoverlapping partitions. Asume that one itemset A is frequent in D but not frequent in any partition of D, we can test if this A exist. If the support count of A is n(A) in D, and $n_1, n_2, n_3 \cdots n_n$in n partition seprately. $\frac{n(A)}{N} \geq s$, according to our assumption, $\frac{n_i(A)}{N/n} \leq s$, then $\sum_i \frac{n_i(A)}{N/n} = \frac{n(A)}{N} < \frac{n(A)}{N}$, which is not true. Therefore, any itemset that is frequent in D must be frequent in at least one partition of D.

3. 6.4 Let c be a candidate itemset in C_k generated by the Apriori algrithm. How many length-(k-1) subsets do we need to check in the prune step? Per your previous answer, can you give an improved version of procedure has_infrequent_subset in Figure 6.4?

(a) If any (k-1) subset of c is not in the L_{k-1}, then the candidate cannot be frequent either and so can be removed from C_k. (Page 250)

(b) After scanning, all the removed itemsets in $L_1, L_{2,}L_3 \cdots L_{k-1}$, can be used as ones to remove itensets of C_k to make L_k.

4. 6.5 Section 6.2.2 describes a method for generating association rules from frequent itemsets. Propose a more efficient method. Explain why it is more efficient than the one proposed there. (Hint: Consider incorporating the properties of Exercises 6.3(b), (c) into your design.

(a) Method: From Exercise 6.3(b), (c), we know that $s' \subseteq s \subseteq l$, then $P_{sup}(s') \geq P_{sup}(s) \geq P_{sup}(l)$, and $P_{conf}(s') \leq P_{conf}(s) \leq P_{conf}(l)$. Therefore, to satisfy a certain (s, c), which are minimum support threshold and minimum confidence threshold, we can first test the largest frequent k-itemsets, (the largest here means k is the largest). If any of the frequent k-itemset , lets assume it is A, don't satisfy confidence rule, then any of its subset won't satisfy minimum confidence threshold. we generate k-1, k-2 ... 1 subset of A. Then eluminate these subsets from the frequent k-1, k-2, ..1-term itemset. And if any frequent k-itemset , let's say B, satisfy minimum support threshold, then all its k-1, k-2, ... 1-item subsets satisfy the support rule. We keep these subsets of B.

(b) Reason for efficiency: this methods avoid generating too many unnecessary itemsets as shown in the book, because eluminating and keeping subsets according to $P_{sup}(s') \geq P_{sup}(s) \geq P_{sup}(l)$, and $P_{conf}(s') \leq P_{conf}(s) \leq P_{conf}(l)$ will save lots of time.

5. 6.6 A database has 5 transactions. Let $minsup = 60\%$, $minconf = 80\%$.

TID	items_bought
T100	$\{M, O, N, K, E, Y\}$
T200	$\{D, O, N, K, E, Y\}$
T300	$\{M, A, K, E\}$
T400	$\{M, U, C, K, Y\}$
T500	$\{C, O, O, K, I, E\}$

After sorting⇒

TID	items_bought
T100	$\{E, K, M, N, O, Y\}$
T200	$\{D, E, K, N, O, Y\}$
T300	$\{A, E, K, M\}$
T400	$\{C, K, M, U, Y\}$
T500	$\{C, E, I, K, O, O\}$

(a) Finding all frequent itemsets using Apriori and FP-growth, respectively. Compare the efficiency of the two mining processes.

i. Apriori(minimum support count = 5*0.6=3, and **NOTE** here I count O as 2 times for the last T500 transaction. The reason I generate like this not 1 time is that the possiblity of choosing O increases by twice, when I choosing one specific itemset including O; The result of like is that for Apriori L1, L2 and L3, itemset containing O is 4, 4, 4 rather than 3, 3, 3):

TID	items_bought
T100	$\{E, K, M, N, O, Y\}$
T200	$\{D, E, K, N, O, Y\}$
T300	$\{A, E, K, M\}$
T400	$\{C, K, M, U, Y\}$
T500	$\{C, E, I, K, O, O\}$

$\rightarrow C_1 \rightarrow$

itemset	Sup.count
{A}	**1**
{C}	**2**
{D}	**1**
{E}	4
{I}	**1**
{K}	5
{M}	3
{O}	4
{N}	**2**
{O}	4
{U}	**1**
{Y}	3

$\rightarrow L_1 \rightarrow$

itemset	Sup.count
{E}	4
{K}	5
{M}	3
{O}	4
{Y}	3

$\rightarrow L_1 \bowtie L_1 \rightarrow$

itemset
{E,K}
{E,M}
{E,O}
{E,Y}
{K,M}
{K,O}
{K,Y}
{M,O}
{M,Y}
{O,Y}

$\rightarrow C_2 \rightarrow$

itemset	Sup.count
{E,K}	4
{E,M}	**2**
{E,O}	4
{E,Y}	**2**
{K,M}	3
{K,O}	4
{K,Y}	3
{M,O}	**1**
{M,Y}	**2**
{O,Y}	**2**

$\rightarrow L_2 \rightarrow$

itemset	Sup.count
{E,K}	4
{E,O}	4
{K,M}	3
{K,O}	4
{K,Y}	3

$\rightarrow L_2 \bowtie L_2 \rightarrow$

itemset
{E,K,O}
{K,M,O}
{K,M,Y}
{K,O,Y}

$\rightarrow C_3 \rightarrow$

itemset	Sup.count
{E,K,O}	4

ii. FP-growth (here I count O as 4 times, if O is counted as 3 times, the hash tree and result may be a little different.)

$C_1 \rightarrow$

itemset	Sup.count
{A}	**1**
{C}	**2**
{D}	**1**
{E}	4
{I}	**1**
{K}	5
{M}	3
{N}	**2**
{O}	4
{U}	**1**
{Y}	3

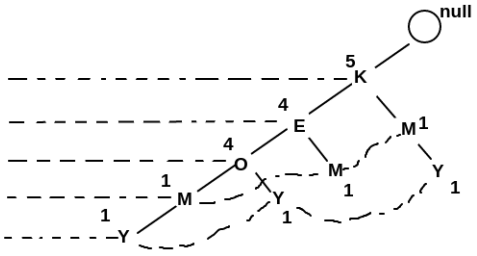
$\rightarrow L_1 \rightarrow$

itemset	Sup.count
{E}	4
{K}	5
{M}	3
{O}	4
{Y}	3

$\rightarrow sort \rightarrow$

$L \rightarrow$

itemset	Sup.count	Node-link
{K}	5	
{E}	4	
{O}	4	
{M}	3	
{Y}	3	



Item	Conditional Pattern Base	Conditional FP-tree	Frequent Pattern Generated
Y	{{K,E,M,O:1}, {K,E,O:1},{K,M:1}}	⟨K:3⟩	{K,Y:3}
M	{{K:1}, {K,E:1}{K,E,O:1}}	⟨K:3⟩	{K,M:3}
O	{K,E:4}	⟨K:4,E:4⟩	{K,E,O:4}
E	{K:4}	⟨K:4⟩	{K,E:4}

Therefore, FP-growth is more efficient than Apriori: Apriori scanned the database at least 6 times while FP-growth only scanned one time to build the tree. Apriori also needs to prune and eluminate itemsets from C_k, to make L_k,while FP-growth don't need to do this work. FP-growth only need to generate the most frequent visited path.

(b) List all the strong association rules (with support s and confidence c) matching the following metarule, where X is a variable representing customers, and item$_i$ denotes variables representing items (e.g., "A,""B,"):

$$\forall x \in transaction,\ \text{buys(X, } item_1) \wedge buys(X,\ item_2 \Rightarrow buys(X,\ item_3)[s, c]$$

i. K,O, then E, $s = \frac{3}{5} = 60\%$, $c = \frac{3}{3} = 1$;
ii. E, O, then K, $s = \frac{3}{5} = 60\%, c = \frac{3}{3} = 1$;

6. 6.11 Most frequent pattern mining algorithms consider only distinct items in a transaction. However, multiple occurrences of an item in the same shopping basket, such as four cakes and three jugs of milk, can be important in transactional data analysis. How can one mine frequent items efficiently consider multiple occurrences of items? Propose modifications to the well-known algorithms, such as Apriori and FP-growth, to adapt to such a situation.

   (a) Consider multiple occurrences of items, for example, if we want to know transcations containing more than four cakes and three jugs of milk: we can first mine items, then put in the specific number count of that item and mine for the count. After all these, mining for multiple occurrences of items can be done.

   (b) Modifications: Apriori: 1st step as usual, mine for specific interesting frequent k-itemset. 2nd step, adding item counts such as (cake, 4)(cake, 5) ..., and find the maximum count. 3rd step, mining sets of (cake, 4) ...(cake, i)... $i \leq maxcount$. For each multiple occurrences of items (cake, i), we can get the count for (cake, i) as a whole. Then use usual Apriori to mine it again.

   (c) Modifactions: FP-growth, using (cake, i) (milk,j) as a node in the tree, however, this may generate a much bigger tree than only using (cake)(milk) etc. nodes.