Unit – 4

Association Analysis: Basic Concepts and Algorithms

- **Introduction**
- **Frequent Item Set Generation**
- **Rule Generation**
- **Compact Representation of Frequent Items**
- **FP-Growth Algorithm**

Introduction:

Frequent patterns are patterns (such as itemsets, subsequences, or substructures) that appear in a data set frequently. For example, a set of items, such as milk and bread that appear frequently together in a transaction data set is a *frequent itemset*. A subsequence, such as buying first a PC, then a digital camera, and then a memory card, if it occurs frequently in a shopping history database, is a (*frequent*) *sequential pattern*. A *substructure* can refer to different structural forms, such as subgraphs, subtrees, or sublattices, which may be combined with itemsets or subsequences. If a substructure occurs frequently, it is called a (*frequent*) *structured pattern*. Finding such frequent patterns plays an essential role in mining associations, correlations, and many other interesting relationships among data. Moreover, it helps in data classification, clustering, and other data mining tasks as well.

For example, huge amounts of customer purchase data are collected daily at the checkout counters of grocery stores. Table 1 illustrates an example of such data, commonly known as market basket transactions. Such data sets are called as Transactional Data Sets. Each row in this table corresponds to a transaction, which contains a unique identifier labeled TID and a set of items bought by a given customer. Retailers are interested in analyzing the data to learn about the purchasing behavior of their customers. Such valuable information can be used to support a variety of business-related applications such as marketing promotions, inventory management, and customer relationship management.

TID	Items
1	{Bread, Milk}
2	{Bread, Diapers, Beer, Eggs}
3	{Milk, Diapers, Beer, Cola}
4	{Bread, Milk, Diapers, Beer}
5	{Bread, Milk, Diapers, Cola}

Table 1: An example Transactional Data Set for market basket transactions

Association analysis is useful for discovering interesting relationships hidden in large data sets. The uncovered relationships can be represented in the form of association rules or sets of

frequent items. For example, the following rule is extracted using the data set shown in Table 1 is:

{Diapers} -> {Beer}

The rule suggests that a strong relationship exists between the sale of diapers and beer because many customers who buy diapers also buy beer.

Besides market basket data, association analysis is also applicable to other application domains such as bioinformatics, medical diagnosis, Web mining, and scientific data analysis.

Basic Terminology used in Association Analysis

Binary Representation

Market basket data can be represented in a binary format as shown in Table 2, where each row corresponds to a transaction and each column corresponds to an item. An item can be treated as a binary variable whose value is one if the item is present in a transaction and zero otherwise. Because the presence of an item in a transaction is often considered more important than its absence, an item is an asymmetric binary variable.

TID	Bread	Milk	Diapers	Beer	Eggs	Cola
1	1	1	0	0	0	0
2	1	0	1	1	1	0
3	0	1	1	1	0	1
4	1	1	1	1	0	0
5	1	1	1	0	0	1

Table 2: A binary 0/1 representation of market basket data

Itemset and Support Count

Let $I = \{ i_1, i_2, \ldots, i_d \}$ be the set of all items in a market basket data and $T = \{ t_1, t_2, \ldots, t_N \}$ be the set of all transactions. Each transaction t_i contains a subset of items chosen from I. In association analysis, a collection of zero or more items is termed an itemset. If an itemset contains k items, it is called a k-itemset. For instance, {Beer, Diapers, Milk} is an example of a 3-itemset. The null (or empty) set is an itemset that does not contain any items.

The transaction width is defined as the number of items present in a transaction. A transaction t_j is said to contain an itemset X if X is a subset of t_j. For example, the second transaction shown in Table 2 contains the itemset {Bread, Diapers} but not {Bread, Milk}. An important property of an itemset is its support count, which refers to the number of transactions that contain a particular itemset. Mathematically, the support count, σ(X), for an itemset X can be stated as follows:

$$\sigma(X) = \left|\{t_i | X \subseteq t_i,\ t_i \in T\}\right|$$

Where the symbol $|\cdot|$ denote the number of elements in a set. In the data set shown in Table 2, the support count for {Beer, Diapers, Milk} is equal to two because there are only two transactions that contain all three items.

Association Rule

An association rule is an implication expression of the form $X \longrightarrow Y$, where X and Y are disjoint itemsets, i.e., $X \cap Y = \emptyset$. The strength of an association rule can be measured in terms of its support and confidence. Support determines how often a rule is applicable to a given data set, while confidence determines how frequently items in Y appear in transactions that contain X. The formal definitions of these metrics are:

$$\text{Support, } s(X \longrightarrow Y) = \frac{\sigma(X \cup Y)}{N}$$

$$\text{Confidence, } c(X \longrightarrow Y) = \frac{\sigma(X \cup Y)}{\sigma(X)}$$

Example: Consider the rule {Milk, Diapers} −→ {Beer}. Since the support count for {Milk, Diapers, Beer} is 2 and the total number of transactions is 5, the rule's support is 2/5 = 0.4. The rule's confidence is obtained by dividing the support count for {Milk, Diapers, Beer} by the support count for {Milk, Diapers}. Since there are 3 transactions that contain milk and diapers, the confidence for this rule is 2/3 = 0.67.

■

Why use Support and Confidence?

- Support is an important measure because a rule that has very low support may occur simply by chance. A low support rule is also likely to be uninteresting from a business perspective because it may not be profitable to promote items that customers seldom buy. For these reasons, support is often used to eliminate uninteresting rules.
- Confidence, on the other hand, measures the reliability of the inference made by a rule. For a given rule $X \longrightarrow Y$, the higher the confidence, the more likely it is for Y to be present in transactions that contain X. Confidence also provides an estimate of the conditional probability of Y given X.

Definition : (Association Rule Discovery)

Given a set of transactions T, find all the rules having support ≥ *minsup* and confidence ≥ *minconf*, where *minsup* and *minconf* are the corresponding support and confidence thresholds.

A brute-force approach for mining association rules is to compute the sup- port and confidence for every possible rule. This approach is prohibitively expensive because there are exponentially many rules that can be extracted from a data set. More specifically, the total number of possible rules extracted from a data set that contains d items is

$$R = 3^d - 2^{d+1} + 1$$

Common strategy adopted by many association rule mining algorithms is to decompose the problem into two major subtasks:

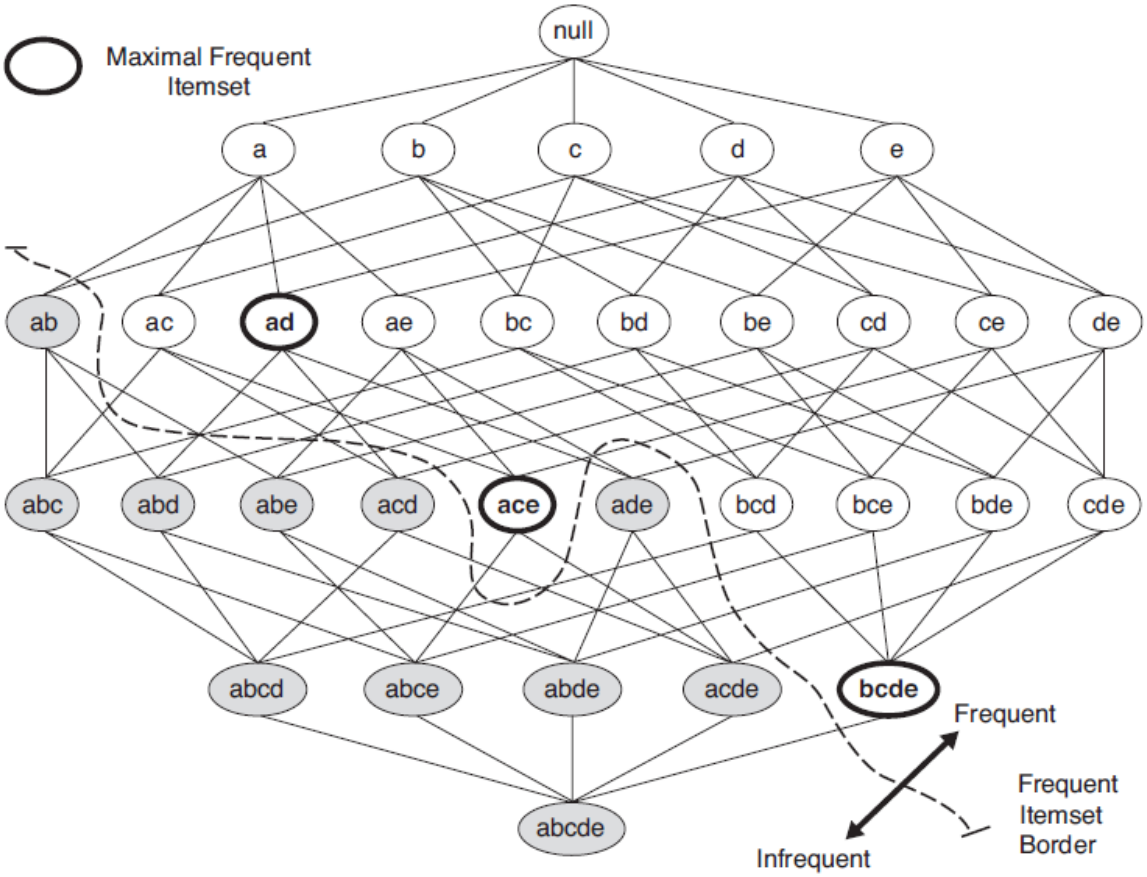
1. ***Frequent Itemset Generation***, whose objective is to find all the item- sets that satisfy the *minsup* threshold. These itemsets are called frequent itemsets.
2. ***Rule Generation***, whose objective is to extract all the high-confidence rules from

the frequent itemsets found in the previous step. These rules are called strong rules.

Maximal Frequent Itemset

Definition:

A maximal frequent item- set is defined as a frequent itemset for which none of its immediate supersets are frequent.

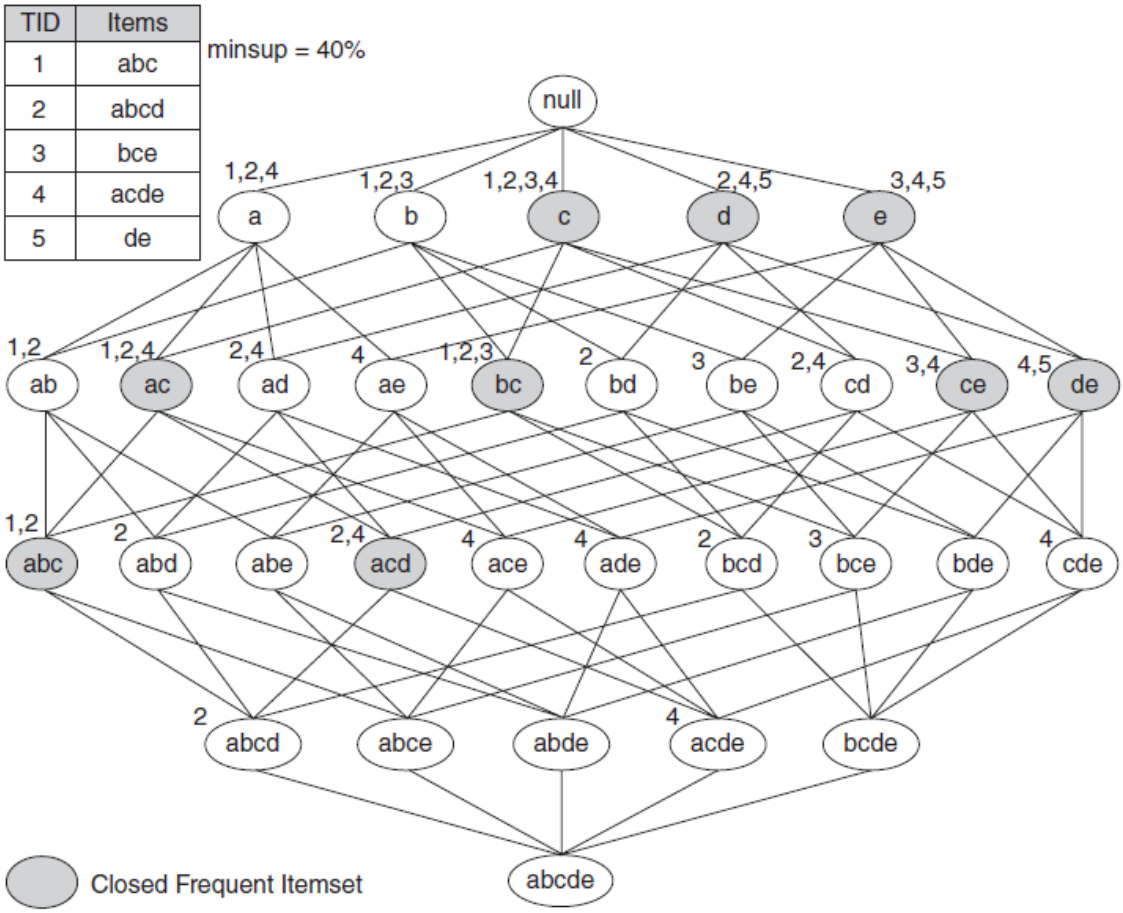


To illustrate this concept, consider the itemset lattice shown in the above Figure. The itemsets in the lattice are divided into two groups: those that are frequent and those that are infrequent. A frequent itemset border, which is represented by a dashed line, is also illustrated in the diagram. Every itemset located above the border is frequent; while those located below the border (the shaded nodes) are infrequent. Among the itemsets residing near the border, {*a, d*}, {*a, c, e*}, and {*b, c, d, e*} are considered to be maximal frequent itemsets because their immediate supersets are infrequent. An itemset such as {*a, d*} is maximal frequent because all of its immediate supersets, {*a,b, d*}, {*a,c,d*}, and {*a, d, e*}, are infrequent. In contrast, {*a, c*} is non-maximal because one of its immediate supersets, {*a,c,e*}, is frequent.

Closed Itemsets

Definition:

An itemset *X* is closed if none of its immediate supersets has exactly the same support count as *X*.



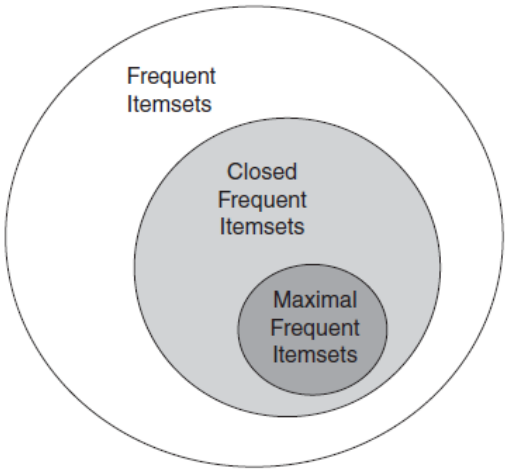
Put another way, *X* is not closed if at least one of its immediate supersets has the same support count as *X*. Examples of closed itemsets are shown in above Figure. To better illustrate the support count of each itemset, we have associated each node (itemset) in the lattice with a list of its corresponding transaction IDs. For example, since the node {*b, c*} is associated with transaction IDs 1, 2, and 3, its support count is equal to three. From the transactions given in this diagram, notice that every transaction that contains *b* also contains *c*. Consequently, the support for **{***b***}** is identical to **{***b, c***}** and **{***b***}** should not be considered a closed itemset. Similarly, since *c* occurs in every transaction that contains both *a* and *d*, the itemset {*a, d*} is not closed. On the other hand, {*b, c*} is a closed itemset because it does not have the same support count as any of its supersets.

Closed Frequent Itemset

Definition

An itemset is a closed frequent itemset if it is closed and its support is greater than or equal to *minsup*.

In the previous example, assuming that the support threshold is 40%, {b,c} is a closed frequent itemset because its support is 60%. The rest of the closed frequent itemsets are indicated by the shaded nodes.



Relationships among frequent, maximal frequent, and closed frequent itemsets.

APRIORI Algorithm

Finding Frequent Itemsets Using Candidate Generation

Apriori is a seminal algorithm proposed by R. Agrawal and R. Srikant in 1994 for mining frequent itemsets for Boolean association rules. The name of the algorithm is based on the fact that the algorithm uses *prior knowledge* of frequent itemset properties. Apriori employs an iterative approach known as a *level-wise* search, where *k*-itemsets are used to explore (*k*+1)-itemsets. First, the set of frequent 1-itemsets is found by scanning the database to accumulate the count for each item, and collecting those items that satisfy minimum support. The resulting set is denoted L_1.Next, L_1 is used to find L_2, the set of frequent 2-itemsets, which is used to find L_3, and so on, until no more frequent *k*-itemsets can be found. The finding of each L_k requires one full scan of the database.

Apriori property: *All nonempty subsets of a frequent itemset must also be frequent.*

The Apriori property is based on the following observation. By definition, if an itemset *I* does not satisfy the minimum support threshold, *min sup*, then *I* is not frequent; that is, $P(I) < min\ sup$. If an item *A* is added to the itemset *I*, then the resulting itemset (i.e., $I \cup A$) cannot occur more frequently than *I*. Therefore, $I \cup A$ is not frequent either; that is, $P(I \cup A) < min\ sup$.
This property belongs to a special category of properties called antimonotone in the sense that *if a set cannot pass a test, all of its supersets will fail the same test as well*. It is called *antimonotone* because the property is monotonic in the context of failing a test.
"How is the Apriori property used in the algorithm?" To understand this, let us look at how L_{k-1} is used to find L_k for $k \geq 2$. A two-step process is followed, consisting of join and prune actions.

1. **The Join Step:**
   To find L_k, a set of candidate *k*-itemsets is generated by joining L_{k-1} with itself. This set of candidates is denoted C_k. Let l_1 and l_2 be itemsets in L_{k-1}. The notation $l_i[j]$ refers to the j^{th} item in l_i (e.g., $l_1[k-2]$ refers to the second to the last item in l_1). By convention, Apriori assumes that items within a transaction or itemset are sorted in lexicographic order. For the (*k*-1)-itemset, l_i, this means that the items are sorted such that $l_i[1] < l_i[2] < : : : < l_i[k-$

1]. The join, L_{k-1} on L_{k-1}, is performed, where members of L_{k-1} are joinable if their first (k-2) items are in common. That is, members l_1 and l_2 of L_{k-1} are joined if ($l_1[1] = l_2[1]$) ^ ($l_1[2] = l_2[2]$) ^: : :^($l_1[k-2] = l_2[k-2]$) ^($l_1[k-1] < l_2[k-1]$). The condition $l_1[k-1] < l_2[k-1]$ simply ensures that no duplicates are generated. The resulting itemset formed by joining l_1 and l_2 is $l_1[1]$, $l_1[2]$, : : : , $l_1[k-2]$, $l_1[k-1]$, $l_2[k-1]$.

2. **The Prune Step:**

C_k is a superset of L_k, that is, its members may or may not be frequent, but all of the frequent k-itemsets are included in C_k. A scan of the database to determine the count of each candidate in C_k would result in the determination of L_k (i.e., all candidates having a count no less than the minimum support count are frequent by definition, and therefore belong to L_k). C_k, however, can be huge, and so this could involve heavy computation. To reduce the size of C_k, the Apriori property is used as follows. Any (k-1)-itemset that is not frequent cannot be a subset of a frequent k-itemset. Hence, if any (k-1)-subset of a candidate k-itemset is not in L_{k-1}, then the candidate cannot be frequent either and so can be removed from C_k. This subset testing can be done quickly by maintaining a hash tree of all frequent itemsets.

Example:

Consider the following Transactional Data Set with 9 transactions.

TID	*List of item_IDs*
T100	I1, I2, I5
T200	I2, I4
T300	I2, I3
T400	I1, I2, I4
T500	I1, I3
T600	I2, I3
T700	I1, I3
T800	I1, I2, I3, I5
T900	I1, I2, I3

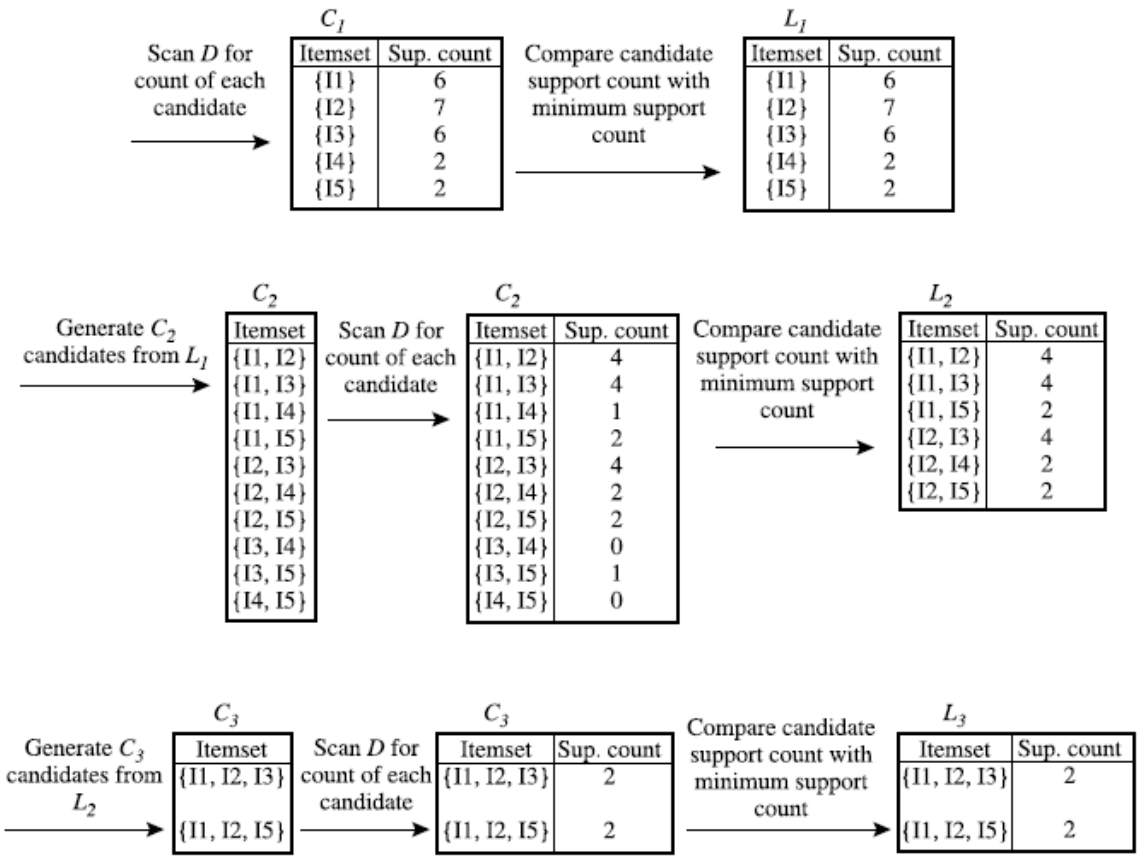
There are nine transactions in this database, that is, $|D| = 9$. The following illustrates about the working out of Apriori algorithm with the transaction data set D.

1 *In the first iteration of the algorithm, each item is a member of the set of candidate 1-itemsets, C_1. The algorithm simply scans all of the transactions in order to count the number of occurrences of each item.*

2 *Suppose that the minimum support count required is 2, that is, min sup = 2. (Here, we are referring to absolute support because we are using a support count. The corresponding relative support is 2/9 = 22%). The set of frequent 1-itemsets, L_1, can then be determined. It consists of the candidate 1-itemsets satisfying minimum support. In our example, all of the candidates in C_1 satisfy minimum support.*

3 *To discover the set of frequent 2-itemsets, L_2, the algorithm uses the **Join**(L_1, L_1) to generate a candidate set of 2-itemsets, C_2.*

4 *Next, the transactions in D are scanned and the support count of each candidate itemset inC_2*



5 *The set of frequent 2-itemsets, L_2, is then determined, consisting of those candidate 2-itemsets in C_2 having minimum support.*

6 *The generation of the set of candidate 3-itemsets, C_3, is detailed in Figure. From the join step, we first get C_3 =**Join**(L_2, L_2) = {{I_1, I_2, I_3}, {I_1, I_2, I_5}, {I_1, I_3, I_5}, {I_2, I_3, I_4}, {I_2, I_3, I_5}, {I_2, I_4, I_5}}. Based on the Apriori property that all subsets of a frequent itemset must also be frequent, we can determine that the four latter candidates cannot possibly be frequent. We therefore remove them fromC_3, thereby saving the effort of unnecessarily obtaining their counts during the subsequent scan of D to determine L_3. Note that when given a candidate k-itemset, we only need to check if its (k-1)-subsets are frequent since the Apriori algorithm uses a level-wise search strategy. The resulting pruned version of C_3 is shown.*

7 *The transactions in D are scanned in order to determine L_3, consisting of those candidate 3-itemsets in C_3 having minimum support*

8 *The algorithm uses **Join(**L_3, L_3) to generate a candidate set of 4-itemsets, C_4. Although the join results in {{I_1, I_2, I_3, I_5}}, this itemset is pruned because its subset {{I_2, I_3, I_5}} is not frequent. Thus, C_4 = Θ, and the algorithm terminates, having found all of the frequent itemsets.*

Algorithm: Apriori. Find frequent itemsets using an iterative level-wise approach based on candidate generation.

Input:

- D, a database of transactions;
- min_sup, the minimum support count threshold.

Output: L, frequent itemsets in D.

Method:

```
(1)   L1 = find_frequent_1-itemsets(D);
(2)   for (k = 2; Lk−1 ≠ ϕ; k++) {
(3)       Ck = apriori_gen(Lk−1);
(4)       for each transaction t ∈ D { // scan D for counts
(5)           Ct = subset(Ck, t); // get the subsets of t that are candidates
(6)           for each candidate c ∈ Ct
(7)               c.count++;
(8)       }
(9)       Lk = {c ∈ Ck | c.count ≥ min_sup}
(10)  }
(11)  return L = ∪k Lk;
```

```
procedure apriori_gen(Lk−1:frequent (k − 1)-itemsets)
(1)   for each itemset l1 ∈ Lk−1
(2)       for each itemset l2 ∈ Lk−1
(3)           if (l1[1] = l2[1]) ∧ (l1[2] = l2[2]) ∧ ... ∧ (l1[k − 2] = l2[k − 2]) ∧ (l1[k − 1] < l2[k − 1]) then {
(4)               c = l1 ⋈ l2; // join step: generate candidates
(5)               if has_infrequent_subset(c, Lk−1) then
(6)                   delete c; // prune step: remove unfruitful candidate
(7)               else add c to Ck;
(8)           }
(9)   return Ck;
```

```
procedure has_infrequent_subset(c: candidate k-itemset;
          Lk−1: frequent (k − 1)-itemsets); // use prior knowledge
(1)   for each (k − 1)-subset s of c
(2)       if s ∉ Lk−1 then
(3)           return TRUE;
(4)   return FALSE;
```

Generating Association Rules from Frequent Itemsets

Once the frequent itemsets from transactions in a database *D* have been found, it is straightforward to generate strong association rules from them (where *strong association* rules satisfy both minimum support and minimum confidence). This can be done using confidence, which is shown below:

$$confidence(A \Rightarrow B) = P(B|A) = \frac{support\_count(A \cup B)}{support\_count(A)}$$

Suppose the data contain the frequent itemset $l = \{I_1, I_2, I_5\}$. What are the association rules that can be generated from l? The nonempty subsets of l are $\{I_1, I_2\}$, $\{I_1, I_5\}$, $\{I_2, I_5\}$, $\{I_1\}$, $\{I_2\}$, and $\{I_5\}$. The resulting association rules are as shown below, each listed with its confidence:

$$I1 \wedge I2 \Rightarrow I5, \quad confidence = 2/4 = 50\%$$
$$I1 \wedge I5 \Rightarrow I2, \quad confidence = 2/2 = 100\%$$
$$I2 \wedge I5 \Rightarrow I1, \quad confidence = 2/2 = 100\%$$
$$I1 \Rightarrow I2 \wedge I5, \quad confidence = 2/6 = 33\%$$
$$I2 \Rightarrow I1 \wedge I5, \quad confidence = 2/7 = 29\%$$
$$I5 \Rightarrow I1 \wedge I2, \quad confidence = 2/2 = 100\%$$

If the minimum confidence threshold is, say, 70%, then only the second, third, and last rules above are output, because these are the only ones generated that are strong. Note that, unlike conventional classification rules, association rules can contain more than one conjunct in the right-hand side of the rule.

FP Growth Algorithm

This section presents an alternative algorithm called FP-growth that takes a radically different approach for discovering frequent itemsets. The algorithm does not subscribe to the generate-and-test paradigm of Apriori. Instead, it encodes the data set using a compact data structure called an FP-tree and extracts frequent itemsets directly from this structure. The details of this approach are as follows:

FP-Tree Representation

An FP-tree is a compressed representation of the input data. It is constructed by reading the data set one transaction at a time and mapping each transaction onto a path in the FP-tree. As different transactions can have several items in common, their paths may overlap. The more the paths overlap with one another, the more compression we can achieve using the FP-tree structure.

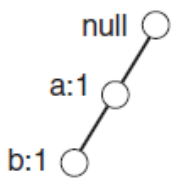
Consider the following transactional data set:

TID	Items
1	{a,b}
2	{b,c,d}
3	{a,c,d,e}
4	{a,d,e}
5	{a,b,c}
6	{a,b,c,d}
7	{a}
8	{a,b,c}
9	{a,b,d}
10	{b,c,e}

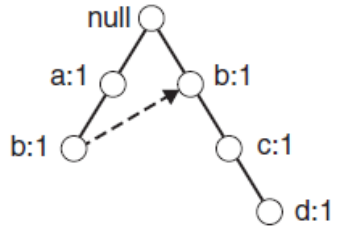
The above transactional data set contains 10 transactions and five items. FP Tree after reading 10 transactions one by one is as follows:

(Note: each node is labeled with item name and its support count after reading a transaction)

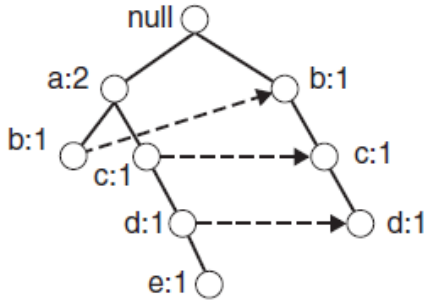
1. After reading TID = 1



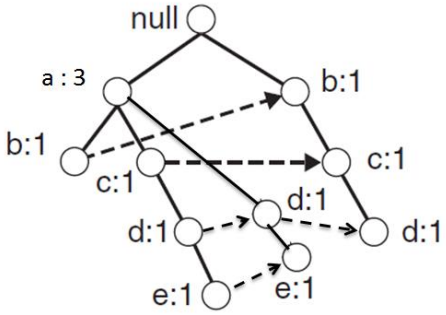
2. After reading TID = 2



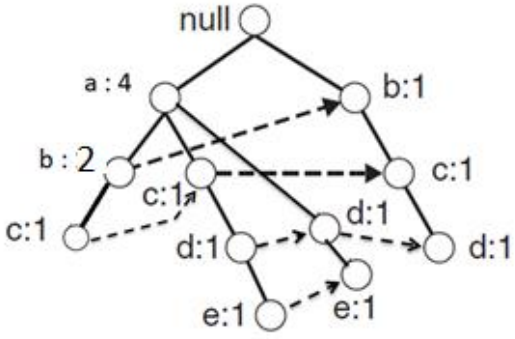
3. After reading TID = 3



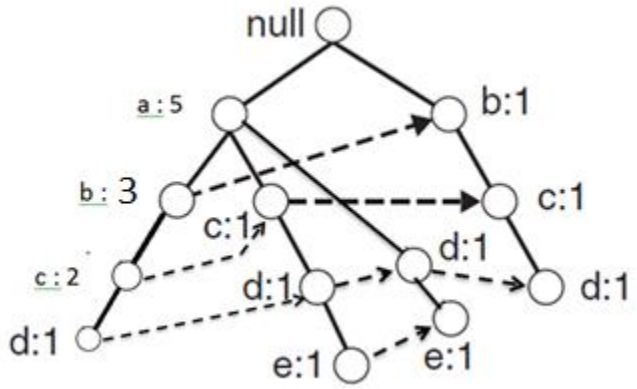
4. After reading TID = 4



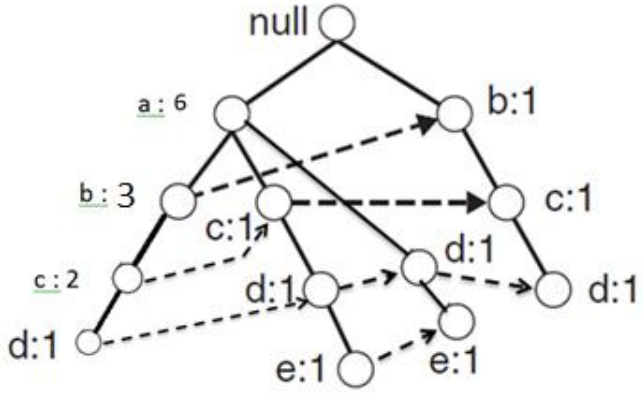
5. After reading TID = 5



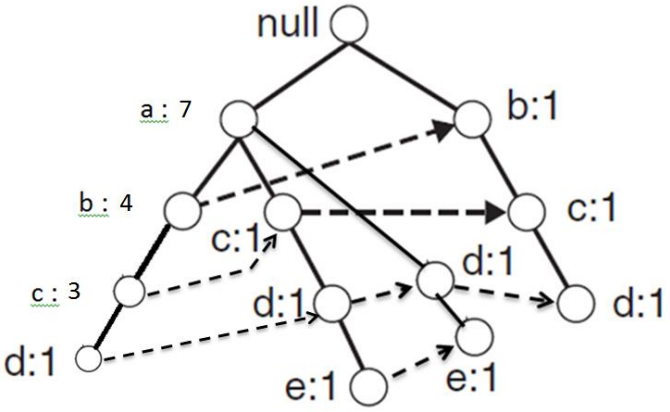
6. After reading TID = 6



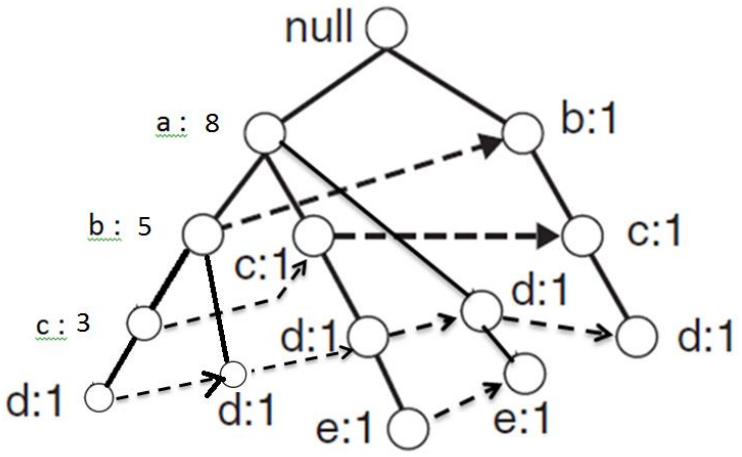
7. After reading TID = 7



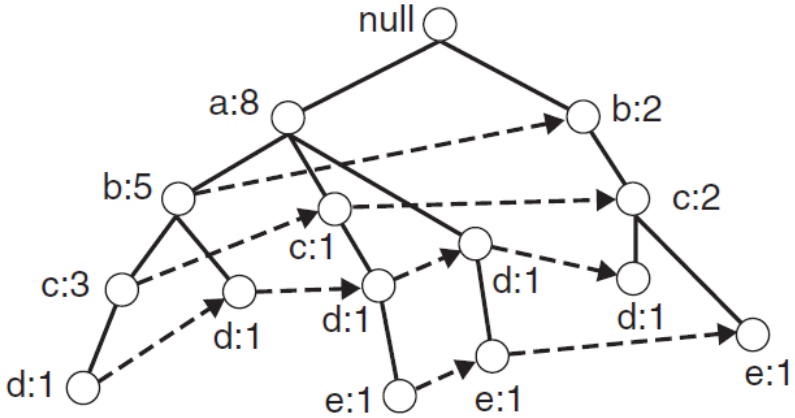
8. After reading TID = 8



9. After reading TID = 9



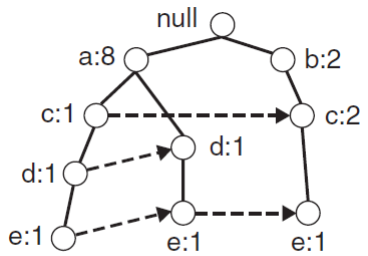
10. After reading TID = 10



Frequent Itemset Generation in FP-Growth Algorithm

Consider the task of finding frequent itemsets ending with e.

1. The first step is to gather all the paths containing node **e**. These initial paths are called **prefix paths** and are shown:

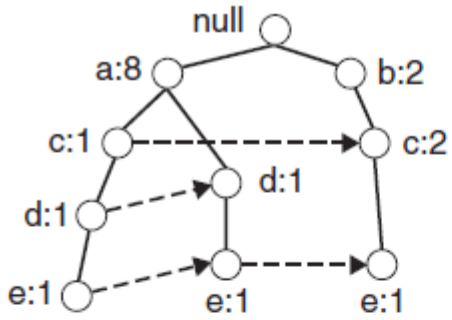


Prefix paths ending in e

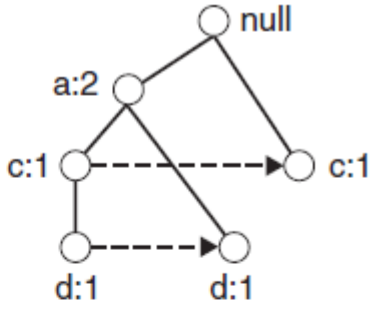
2. From the prefix paths shown in above Figure, the support count for e is obtained by adding the support counts associated with node **e**. Assuming that the minimum support count is 2, **{e}** is declared a frequent itemset because its support count is 3.

3. Because **{e}** is frequent, the algorithm has to solve the sub problems of finding frequent itemsets ending in **de, ce, be**, and **ae**. Before solving these sub-problems, it must first convert the prefix paths into a **conditional FP-tree**, which is structurally similar to an FP-tree, except it is used to find frequent itemsets ending with a particular suffix. A conditional FP-tree is obtained in the following way:
   a) First, the support counts along the prefix paths must be updated because some of the counts include transactions that do not contain item **e**. For example, the rightmost path shown in Figure**, null –→ b:2 –→ c:2 –→ e:1**, includes a transaction **{b, c}** that does not contain item **e**. The counts along the prefix path must therefore be adjusted to 1 to reflect the actual number of transactions containing **{b, c, e}.**
   b) The prefix paths are truncated by removing the nodes for **e.** These nodes can be removed because the support counts along the prefix paths have been updated to reflect only transactions that contain e and the subproblems of finding frequent itemsets ending in **de, ce, be,** and **ae** no longer need information about node **e.**
   c) After updating the support counts along the prefix paths, some of the items may no longer be frequent. For example, the node b appears only once and has a support count equal to 1, which means that there is only one transaction that contains both **b** and **e**. Item **b** can be safely ignored from subsequent analysis because all itemsets in **be** must be infrequent.

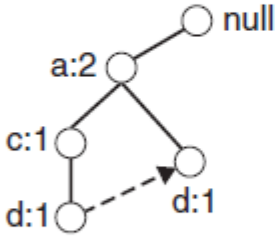
      The tree looks different than the original prefix paths because the frequency counts have been updated and the nodes **b** and **e** have been eliminated.
4. FP-growth uses the conditional FP-tree for **e** to solve the sub-problems of finding frequent itemsets ending in **de, ce,** and **ae.** To find the frequent itemsets ending in de, the prefix paths for **d** are gathered from the conditional FP-tree for **e**. By adding the frequency counts associated with node d, we obtain the support count for **{d, e}.** Since the support count is equal to 2, **{d, e}** is declared a frequent itemset. Next, the algorithm constructs the conditional FP-tree for de using the approach described in step 3. After updating the support counts and removing the infrequent item **c**, the conditional FP-tree for **de** is shown in Figure. Since the conditional FP-tree contains only one item, a, whose support is equal to *minsup*, the algorithm extracts the frequent itemset **{a, d, e}** and moves on to the next sub-problem, which is to generate frequent itemsets ending in **ce**. After processing the prefix paths for **c**, only **{c, e}** is found to be frequent. The algorithm proceeds to solve the next subprogram and found **{a, e}** to be the only frequent itemset remaining.



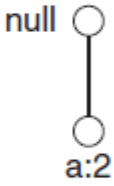
(a) Prefix paths ending in e



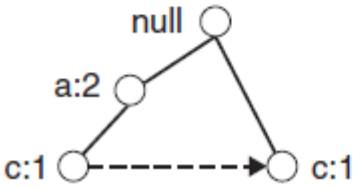
(b) Conditional FP-tree for e



(c) Prefix paths ending in de



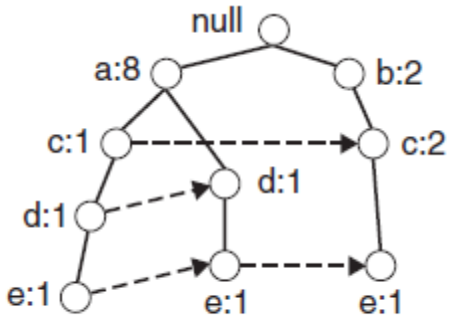
(d) Conditional FP-tree for de



(e) Prefix paths ending in ce

(f) Prefix paths ending in ae

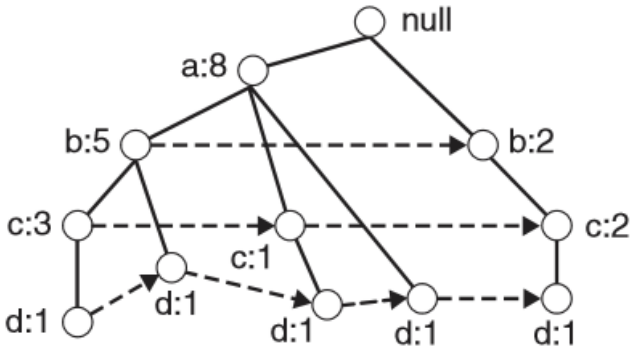
Prefix paths ending in 'e'



(a) Prefix paths ending in e

From the above prefix tree frequent itemsets including item 'e' are:
{e}, {a, e}, {c, e}, {d, e}, {a, d, e}

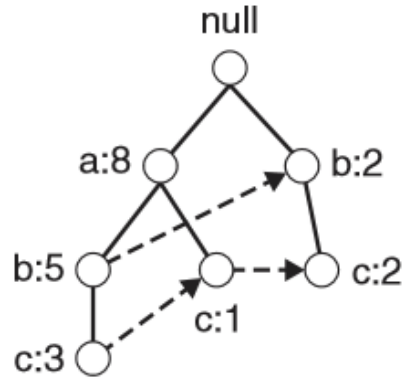
Prefix paths ending with 'd'



(b) Paths containing node d

From the above prefix tree frequent itemsets including item 'd' are:
{d}, {a, d}, {b, d}, {c, d}, {a, b, d}, {a, c, d}, {b, c, d}

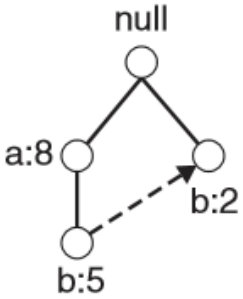
Prefix paths ending with 'c'



(c) Paths containing node c

From the above prefix tree frequent itemsets including item 'c' are:
{c}, {a, c}, {b, c}, {a, b, c}

Prefix paths ending with 'b':



(d) Paths containing node b

From the above prefix tree frequent itemsets including item 'd' are:
{b}, {a, b}

Prefix paths ending with 'a':

(e) Paths containing node a

From the above prefix tree frequent itemsets including item 'a' are:
{a}

The list of frequent itemsets ordered by their corresponding suffixes.

Suffix	Frequent Itemsets
e	{e}, {d,e}, {a,d,e}, {c,e},{a,e}
d	{d}, {c,d}, {b,c,d}, {a,c,d}, {b,d}, {a,b,d}, {a,d}
c	{c}, {b,c}, {a,b,c}, {a,c}
b	{b}, {a,b}
a	{a}