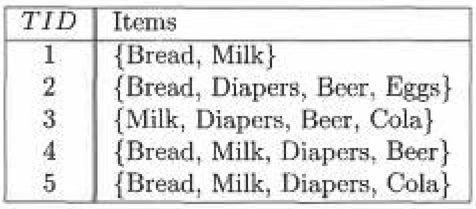
Association Analysis

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.** For example, the following rule can be extracted from the data set shown in Table 6.1:

{Diapers} --> {Beer}

Table 6.1. An example of market basket transactions.

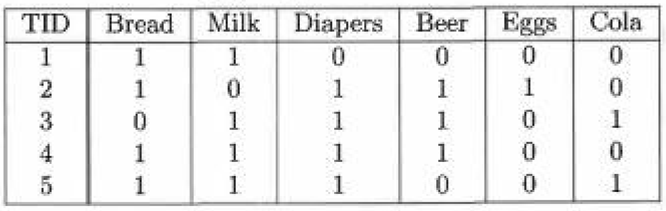


TID	Items
1	{Bread, Milk}
2	{Bread, Diapers, Beer, Eggs}
3	{Milk, Diapers, Beer, Cola}
4	{Bread, Milk, Diapers, Beer}
5	{Bread, Milk, Diapers, Cola}

The rule suggests that a strong relationship exists between the sale of diapers and beer because many customers who buy diapers also buy beer. Retailers can use this type of rules to help them identify new opportunities for cross-selling their products to the customers.

Binary Representation Market basket data can be represented in a binary format as shown in Table 6.2, 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.

Table 6.2. A binary 0/1 representation of market basket data.



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

Association **Rule** An association rule is an implication expression of the form X → *Y,* where X and Y are disjoint itemsets, i.e., $X \cap Y = 0$. 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.

Apriori* Principle:** ***If an itemset is frequent, then all of its subsets must also be frequent.

To illustrate the idea behind the *Apriori* principle, consider the itemset lattice shown in Figure 6.3. Suppose {c, *d,* e} is a frequent itemset. Clearly, any transaction that contains {c, *d,* e} must also contain its subsets, {c, *d},{c,e}, {d,e}, {c}, {d},* and *{e}*. As a result, if *{c,d,e}* is frequent, then all subsets of *{c, d,* e} (i.e., the shaded itemsets in this figure) must also be frequent.

Conversely, if an itemset such as *{a, b}* is infrequent, then all of its supersets must be infrequent too. As illustrated in Figure 6.4, the entire subgraph containing the supersets of *{a, b}* can be pruned immediately once *{a, b}* is found to be infrequent.

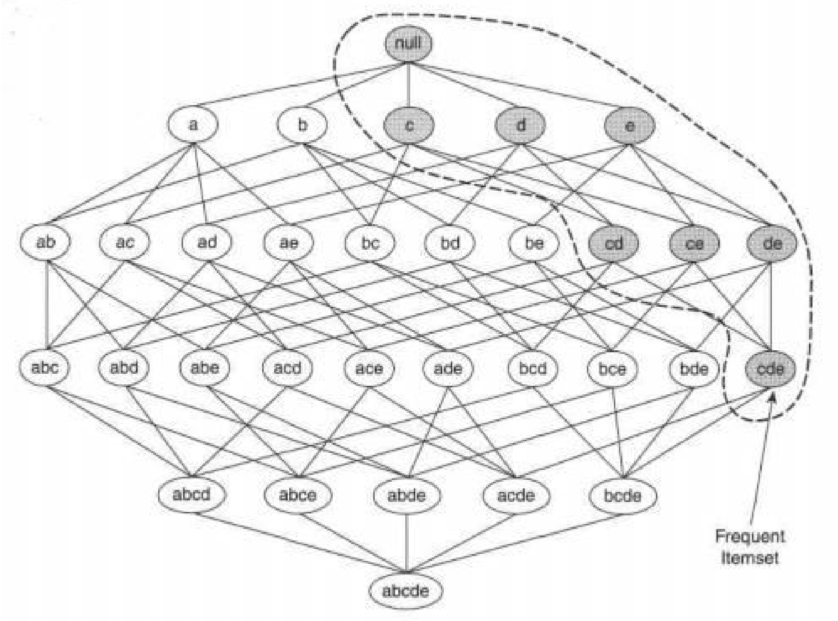


Figure 6.3. An illustration of the *Apriori* principle. If $\{c, d, e\}$ is frequent, then all subsets of this itemset are frequent.

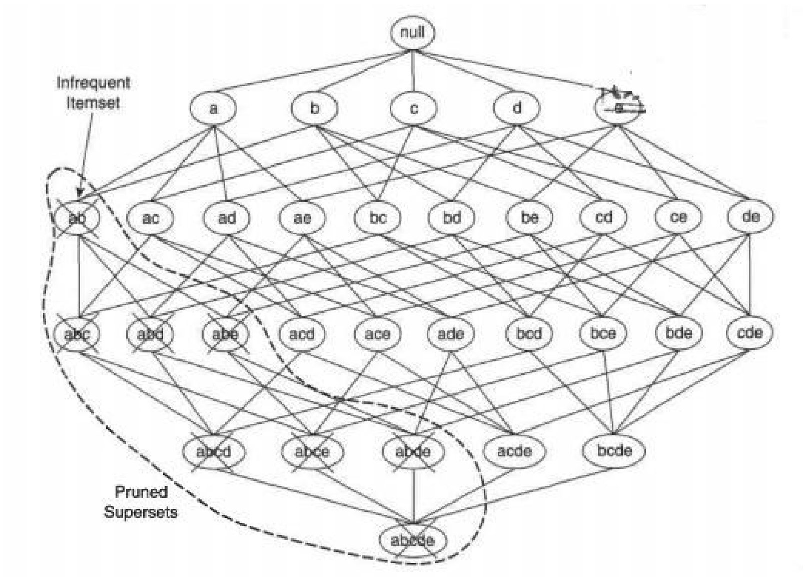


Figure 6.4. An illustration of support-based pruning. If $\{a, b\}$ is infrequent, then all supersets of $\{a, b\}$ are infrequent.

Apriori algorithm:

Algorithm 6.1 Frequent itemset generation of the *Apriori* algorithm.

```
1: $k = 1$.
2: $F_k = \{ i \mid i \in I \wedge \sigma(\{i\}) \geq N \times minsup\}$.   {Find all frequent 1-itemsets}
3: repeat
4:    $k = k + 1$.
5:    $C_k$ = apriori-gen($F_{k-1}$).   {Generate candidate itemsets}
6:    for each transaction $t \in T$ do
7:       $C_t$ = subset($C_k$, $t$).   {Identify all candidates that belong to $t$}
8:       for each candidate itemset $c \in C_t$ do
9:          $\sigma(c) = \sigma(c) + 1$.   {Increment support count}
10:      end for
11:   end for
12:   $F_k = \{ c \mid c \in C_k \wedge \sigma(c) \geq N \times minsup\}$.   {Extract the frequent $k$-itemsets}
13: until $F_k = \emptyset$
14: Result = $\bigcup F_k$.
```

The pseudocode for the frequent itemset generation part of the *Apriori* algorithm is shown in Algorithm 6.1. Let *Ck* denote the set of candidate k-itemsets and *Fk* denote the set of frequent k-itemsets:

• The algorithm initially makes a single pass over the data set to determine the support of each item. Upon completion of this step, the set of all frequent 1-itemsets, H, will be known (steps 1 and 2).

• Next, the algorithm will iteratively generate new candidate k-itemsets using the frequent *(k* - 1)- itemsets found in the previous iteration (step 5). Candidate generation is implemented using a function called apriori- gen.

To count the support of the candidates, the algorithm needs to make an additional pass over the data set (steps 6-10). The subset function is used to determine all the candidate itemsets in C_k that are contained in each transaction *t*.

• After counting their supports, the algorithm eliminates all candidate itemsets whose support counts are less than *minsup* (step 12).

• The algorithm terminates when there are no new frequent itemsets generated, i.e., $F_k = 0$ (step 13).

*****

Note: Refer class note book for examples on frequent itemset generation and association rules generation by using Apriori algorithm.

Compact Representation of Frequent Itemsets

The number of frequent itemsets produced from a transaction data set can be very large. it is useful to identify a small set of itemsets from which all other frequent itemsets can be derived.

Two such representations are maximal frequent item sets and closed frequent itemsets.

Maximal Frequent Itemset: A maximal frequent item set is defined as a frequent itemset for which none of its immediate supersets are frequent.

For example, consider the itemset lattice shown in Figure 6.16. The itemsets in the lattice are divided into two groups: They are frequent and infrequent. A frequent itemset border is represented by a dashed line. 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.

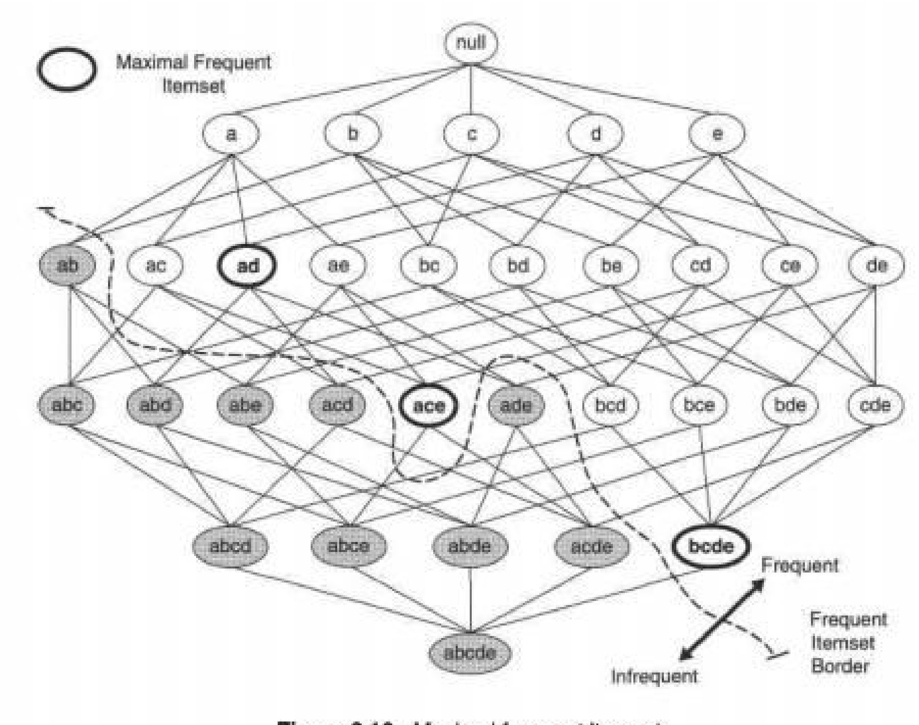


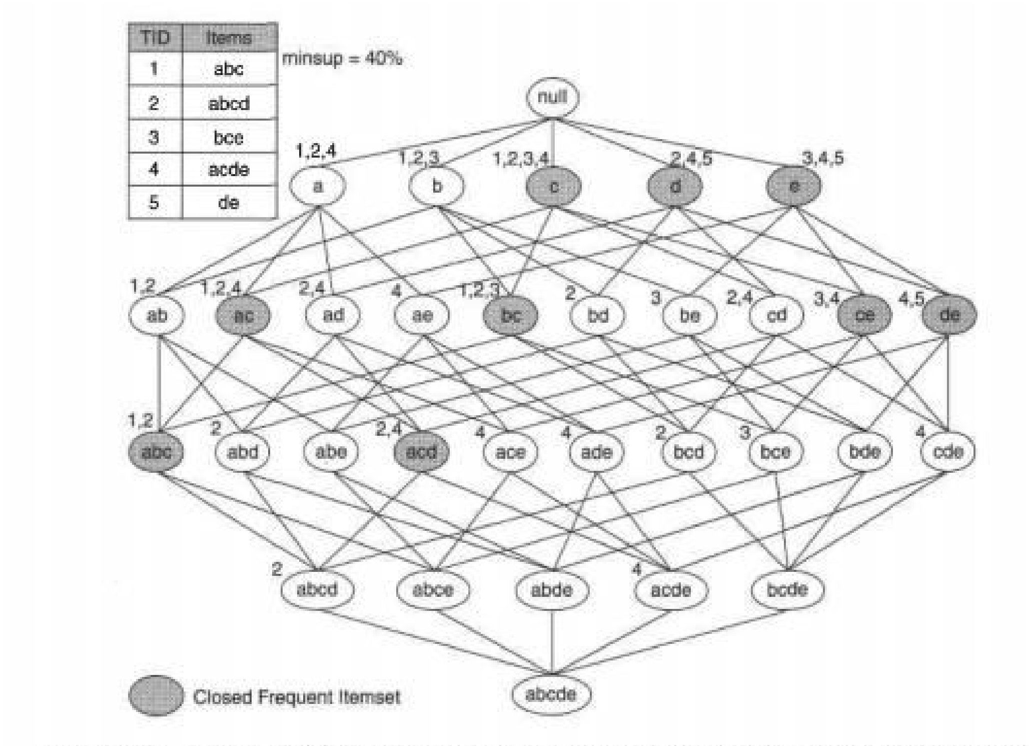
Figure 6.16. Maximal frequent itemset.

Closed Frequent Itemsets:

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

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



TID	Items
1	abc
2	abcd
3	bce
4	acde
5	de

Figure 6.17. An example of the closed frequent itemsets (with minimum support count equal to 40%).

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.

***

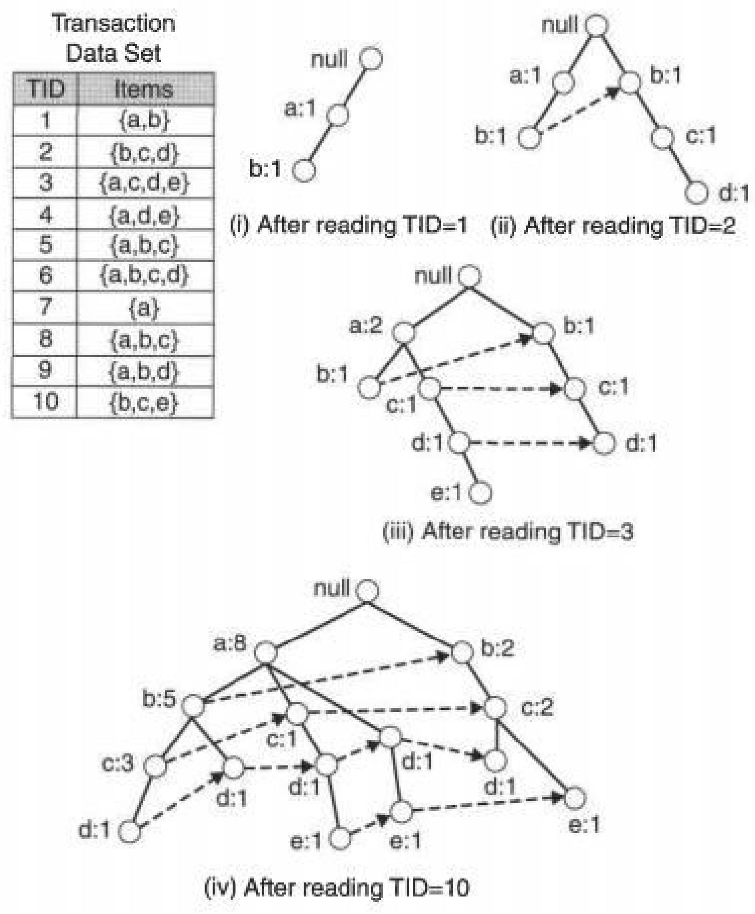
FP-Growth Algorithm:

FP-Growth Algorithm encodes the data set using a compact data structure called an FP-tree and extracts frequent itemsets directly from this structure

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. If the size of the FP-tree is small enough to fit into main memory, this will allow us to extract frequent itemsets directly from the structure in memory instead of making repeated passes over the data stored on disk.



Transaction 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}

Figure 6.24. Construction of an FP-tree.

Figure 6.24 shows a data set that contains ten transactions and five items. The structures of the FP-tree after reading the first three transactions are also depicted in the diagram. Each node in the tree contains the label of an item along with a counter that shows the number of transactions mapped onto the given path. Initially, the FP-tree contains only the root node represented by the null symbol. The FP-tree is subsequently extended in the following way

The data set is scanned once to determine the support count of each item. Infrequent items are discarded, while the frequent items are sorted in decreasing support counts. For the data set shown in Figure 6.24, *a* is the most frequent item, followed by *b,* c, *d,* and e.

The algorithm makes a second pass over the data to construct the FP- tree. After reading the first transaction, *{a, b},* the nodes labeled as *a* and *b* are created. A path is then formed from null $\rightarrow$ *a* $\rightarrow$ *b* to encode the transaction. Every node along the path has a frequency count of 1

After reading the second transaction, *{b,c,d},* a new set of nodes is created for items *b,* c, and *d.* A path is then formed to represent the transaction by connecting the nodes null $\rightarrow$ *b* $\rightarrow$ C $\rightarrow$ *d.* Every node along this path also has a frequency count equal to one. Although the first two transactions have an item in common, which is *b,* their paths are disjoint because the transactions do not share a common prefix.

The third transaction, *{a,c,d,e},* shares a common prefix item (which is *a)* with the first transaction. As a result, the path for the third transaction, null $\rightarrow$ *a* $\rightarrow$ c $\rightarrow$ *d* $\rightarrow$ e, overlaps with the path for the first transaction, null $\rightarrow$ *a* $\rightarrow$ *b.* Because of their overlapping path, the frequency count for node *a* is incremented to two, while the frequency counts for the newly created nodes, c, *d,* and e, are equal to one.

This process continues until every transaction has been mapped onto one of the paths given in the FP-tree. The resulting FP-tree after reading all the transactions is shown at the bottom of Figure 6.24.

Note: Refer class note book for examples on frequent itemset generation and association rules generation by using FP-growth algorithm.

*****