DATA MINING AND DATA WAREHOUSING

(BCSDM515)

By,
Anusha K S
Assistant Professor, Dept. of CSE
VVCE, Mysuru.

MODULE 5

Association Analysis: Problem Definition, Frequent Itemset Generation, Rule Generation, Compact Representation of Frequent itemset, FP-Growth Algorithm.

SLT: Alternative Methods for Generating Frequent Itemsets

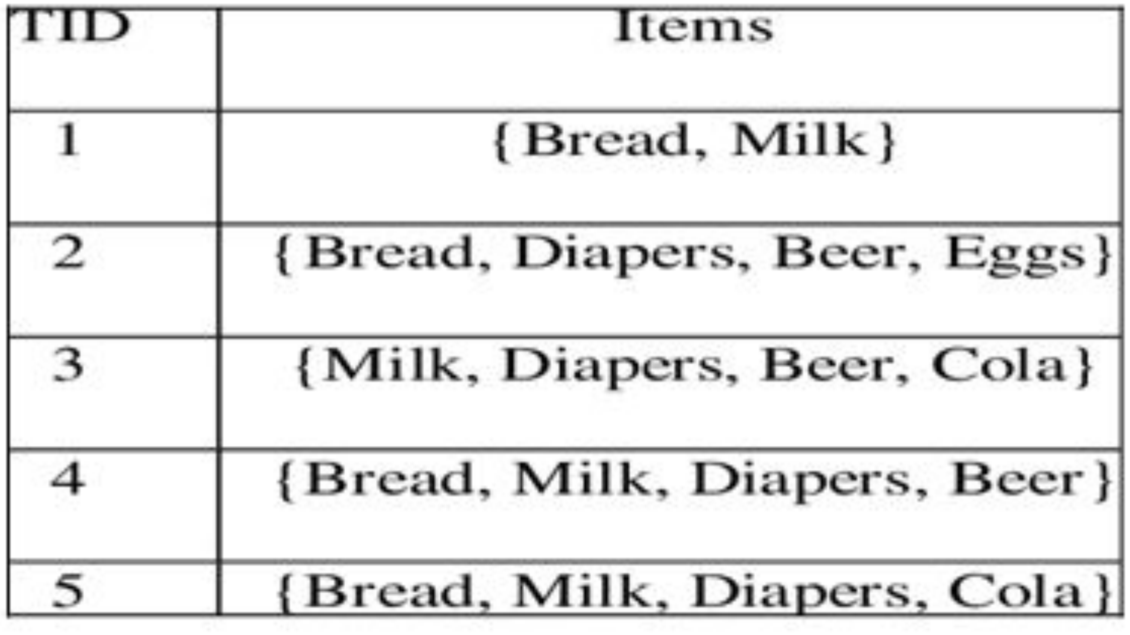
ASSOCIATION ANALYSIS

- Association discovers the association or connection among a set of items in large data sets and also identifies the relationship among objects.

Applications

- Advertising
- Direct Marketing
- Bio informatics
- Medical diagnosis
- Web mining
- Scientific data Analysis etc…

Example: Market Basket Transactions (MBT)



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}

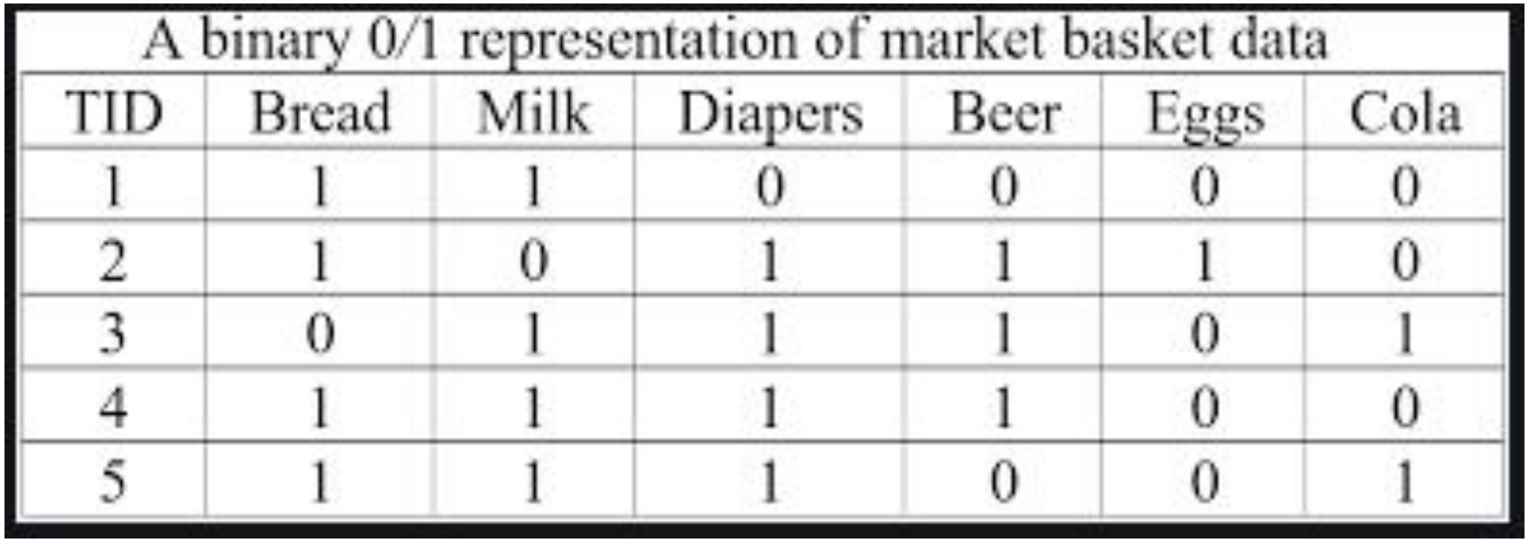
Figure: An Example of MBT

Set of frequent item:	**{Diapers, Beer}**
Association Rule:	**{Diapers} □ {Beer}**

6.1: PROBLEM DEFINITION

- This Section reviews the basic terminology used in association analysis and presents a formal description of the task.

1. Binary Representation



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

2. Itemset and Support Count

Let,

I= {i1, i2, ...,id} Set of all items in a market basket data
T= {t1, t2,...,tN} Set of all transactions.

- Each transaction *ti* contains a subset of items chosen from *I*
- In Association Analysis, a collection of Zero or more items is termed as **itemset**
- If the itemset contains *K* items, it is called *K*-itemset (ex: 3-itemset,null set)
- The transaction width is defined as the number of items present in a transaction.
- The **support count** , **σ (X)**, for an itemset X can be stated as follows:

$$(X) = |\{ti| X \subseteq ti, ti \in T\}|$$

Example

T = Bread | Milk | Diapers | Beer | Eggs | Cola

t_1 = {Bread, Milk}

t_2 = {Bread, Diapers, Beer, Eggs}

t_3 = {Milk, Diapers, Beer, Cola}

t_4 = {Bread, Milk, Diapers, Beer}

X = **{Beer, Diapers, Milk}**

- **Support Count, σ (X) = ?**

Answer: **2**

- **Support = 2 (support count)/4(total number of transaction) = 0.5**

ASSOCIATION RULE

- An association rule is an implication expression form **X ----> Y,** where X and Y are disjoint itemset. i.e., **X∩Y = Φ** (Condition that is satisfied by association rule)

- The strength of an association rule can be measured in terms of its
  - **Support**
  - **Confidence**

$$\text{SUPPORT} = \frac{\text{number of transactions containing X and Y}}{\text{total number of transactions}}$$

$$\text{CONFIDENCE} = \frac{\text{number of transactions containing X and Y}}{\text{number of transactions containing X}}$$

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

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

Example : {Milk, diapers} □ {beer}

Support = ? Answer : 2/5 = 0.4

Confidence = ? Answer : 2/3 = 0.67

FORMULATION OF ASSOCIATION RULE MINING PROBLEM

- The association rule mining problem can be formally stated as follows

Association Rule Discovery:

- Given a set transactions T, the goal of association rule mining is to find all rules having
  - ❖ Support ≥ Minsup Threshold
  - ❖ Confidence ≥ Minconf Threshold
- "Minsup" and "Minconf" are the corresponding support and confidence thresholds.
- A common strategy adopted by many association rule mining algorithms is to decompose the problem into two major subtasks

1. **Frequent Itemset Generation :** is a generation of all itemsets whose support > minsup
2. **Rule Generation :** is a generation of high confidence rules from each frequent itemset

6.2: FREQUENT ITEMSET GENERATION (APRIORI ALGORITHM)

- Apriori Algorithm is the first association Rule Mining Algorithm that pioneered the use of support-based pruning to systematically control the exponential growth of candidate itemsets.
- Apriori uses a "Bottom-up" approach, where frequent subsets are extended one item at a time.
- Apriori is designed to operate on database containing transaction.
- Ex: Details of a website frequentation, collections of items bought by customers.

STEPS TO PERFORM APRIORI ALGORITHM

STEP 1: Scan the transaction database to get the support of search 1-itemset, compare with “MinSup”.

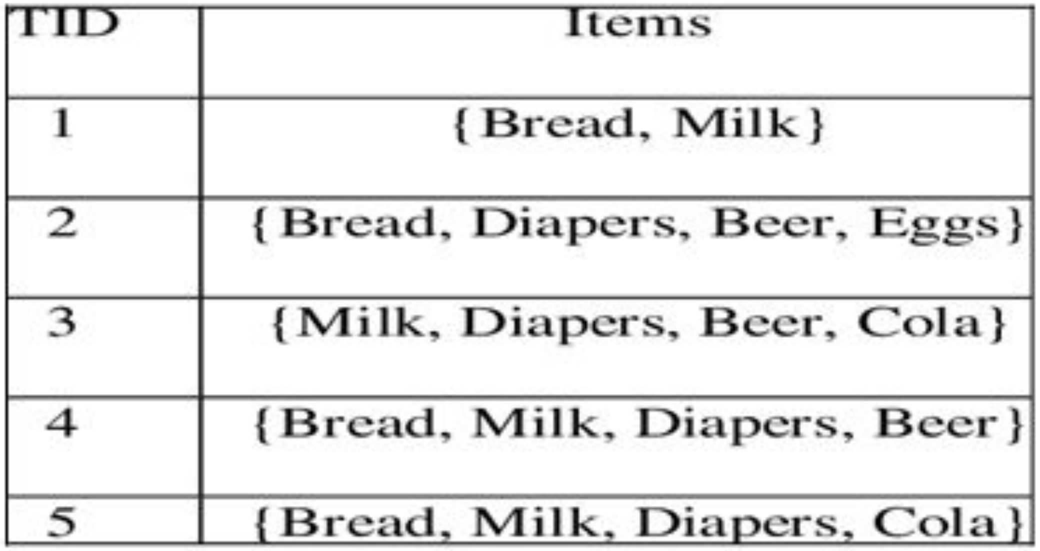
STEP 2: Use Apriori property to prune the unfrequented k-itemsets from the set.

STEP 3: Count the support of each candidate by scanning the DB.

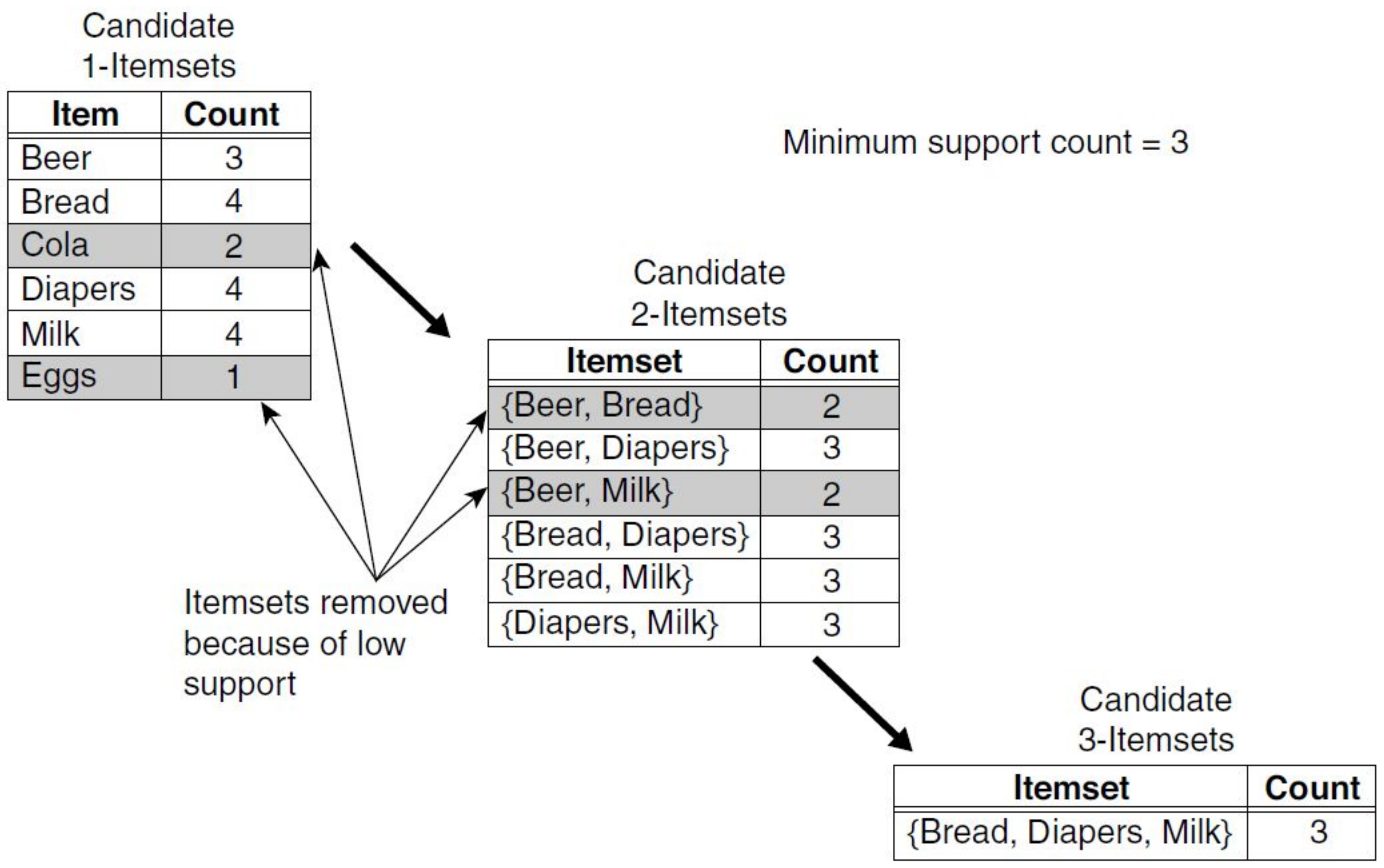
STEP 4: Eliminate candidates that are infrequent, leaving only those that are frequent.

Finally, Repeat the whole procedure until no new frequent itemsets are identified.

Example: MinSup Count = 3



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}


Candidate 1-Itemsets
Minimum support count = 3
Candidate 2-Itemsets
Itemsets removed because of low support
Candidate 3-Itemsets

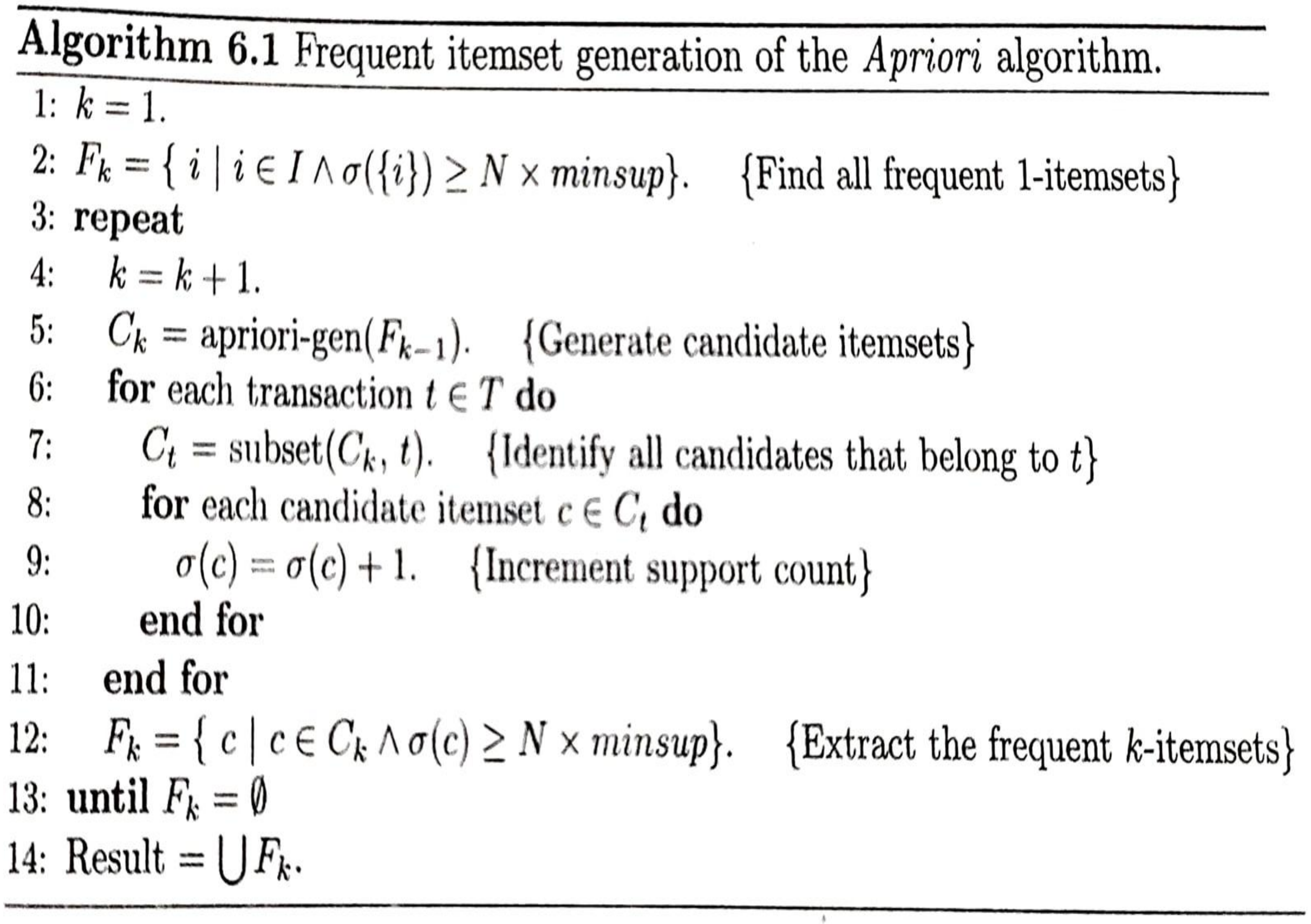
Item	Count
Beer	3
Bread	4
Cola	2
Diapers	4
Milk	4
Eggs	1

Itemset	Count
{Beer, Bread}	2
{Beer, Diapers}	3
{Beer, Milk}	2
{Bread, Diapers}	3
{Bread, Milk}	3
{Diapers, Milk}	3

Itemset	Count
{Bread, Diapers, Milk}	3

Figure 6.5. Illustration of frequent itemset generation using the *Apriori* algorithm.

- With the *Apriori* principle (If an itemset is frequent, then all of its subsets must also be frequent.), we only need to keep candidate 3-itemsets whose subsets are frequent.
- The only candidate that has this property is *{*Bread, Diapers, Milk*}*. However, even though the subsets of *{*Bread, Diapers, Milk*}* are frequent, the itemset itself is not.



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:    $C_k = \text{apriori-gen}(F_{k-1})$.    {Generate candidate itemsets}
6:    for each transaction $t \in T$ do
7:       $C_t = \text{subset}(C_k, t)$.    {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 $= \bigcup F_k$.
```

Two important characteristics of Apriori algorithm is

1. It is a level-wise algorithm
2. It employs a generate-and-test strategy

Limitations

1. This algorithm can be very slow and the bottleneck is candiadate generation.
2. To compare those with support more than MinSup, the database need to be scanned at every level. Hence it consumes “Huge Memory”.

COMPUTATIONAL COMPLEXITY OF APROIORI ALGORITHM

- Support Threshold
- Number of Items
- Average Transaction width
- Generation of frequent 1-itemsets
- Candidate Generation
- Support Counting (two Approach's)

1. Compare each transaction against every candidate itemset and update the support counts of candidates contained in the transaction. (this approach is computationally expensive).
2. Enumerate the itemsets contained in each transaction and use them to update the support counts of their respective candidate itemset.

Support counting : Enumerating subsets and Using Hash Tree

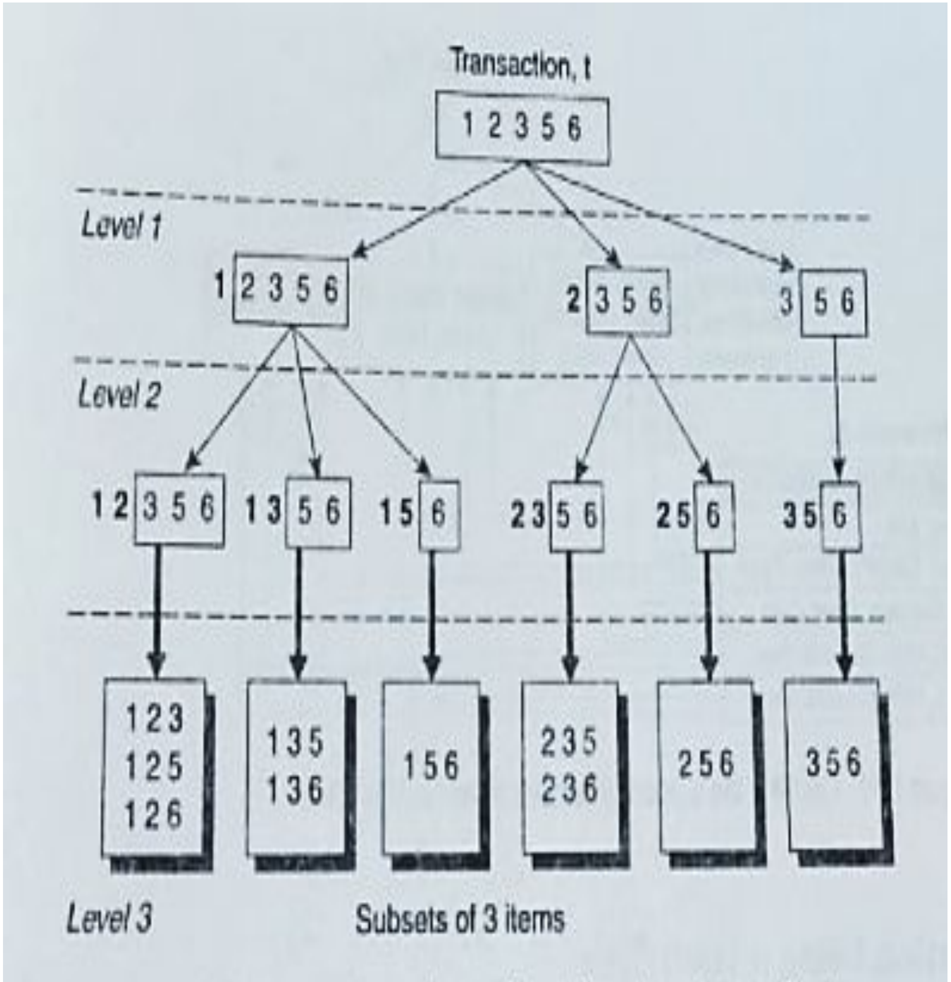


Figure 6.9. Enumerating subsets of three items from a transaction t.

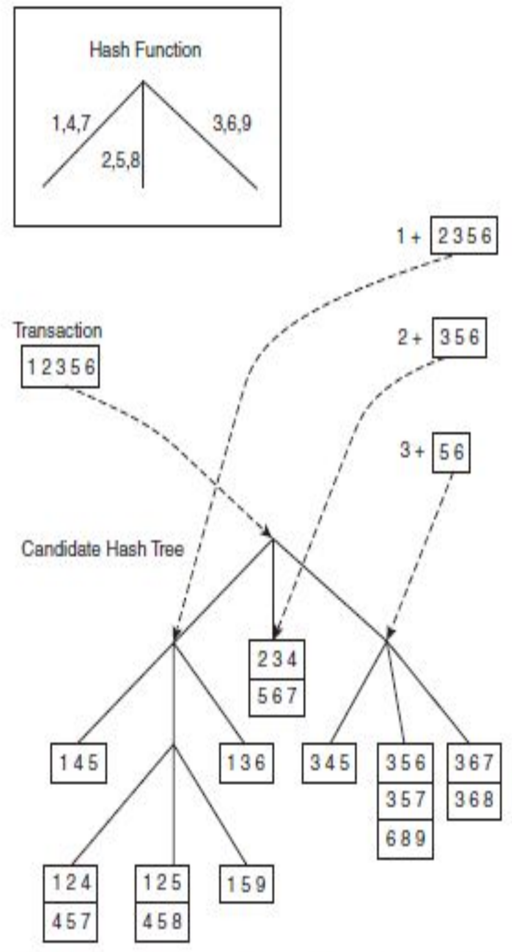


Figure 5.11. Hashing a transaction at the root node of a hash tree.

6.3: RULE GENERATION

The Association rule is an implication expression of the form X □ Y, where X and Y are itemsets.

Algorithm 6.2 Rule generation of the *Apriori* algorithm.

```
1: for each frequent $k$-itemset $f_k$, $k \geq 2$ do
2:     $H_1 = \{i \mid i \in f_k\}$        {1-item consequents of the rule.}
3:     call ap-genrules($f_k, H_1$.)
4: end for
```

Algorithm 6.3 Procedure ap-genrules(f_k, H_m).

```
1: $k = |f_k|$     {size of frequent itemset.}
2: $m = |H_m|$     {size of rule consequent.}
3: if $k > m + 1$ then
4:     $H_{m+1} = \text{apriori-gen}(H_m)$.
5:     for each $h_{m+1} \in H_{m+1}$ do
6:         $conf = \sigma(f_k)/\sigma(f_k - h_{m+1})$.
7:         if $conf \geq minconf$ then
8:             output the rule $(f_k - h_{m+1}) \longrightarrow h_{m+1}$.
9:         else
10:            delete $h_{m+1}$ from $H_{m+1}$.
11:        end if
12:    end for
13:    call ap-genrules($f_k, H_{m+1}$.)
14: end if
```

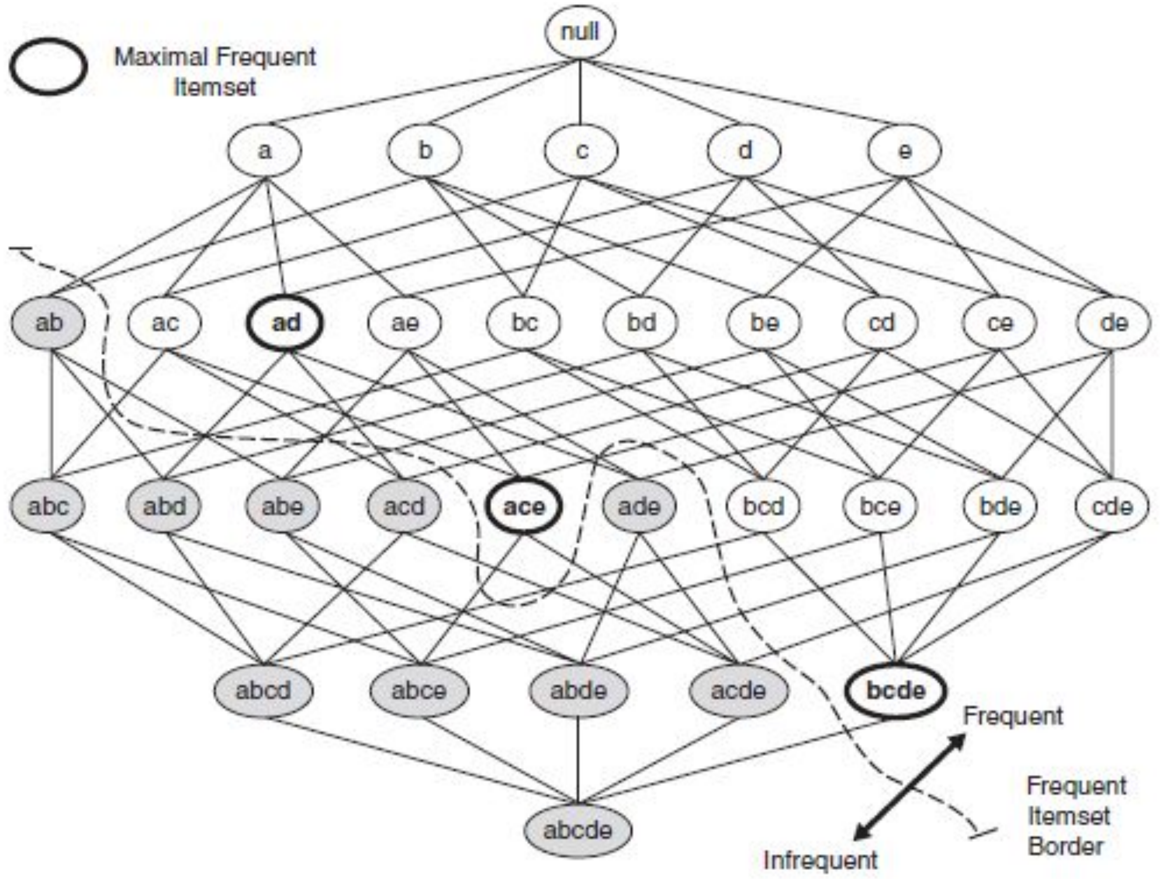
$$\text{Support}, \; s(X \longrightarrow Y) = \frac{\sigma(X \cup Y)}{N};$$

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

6.4: Compact Representation of Frequent Itemsets

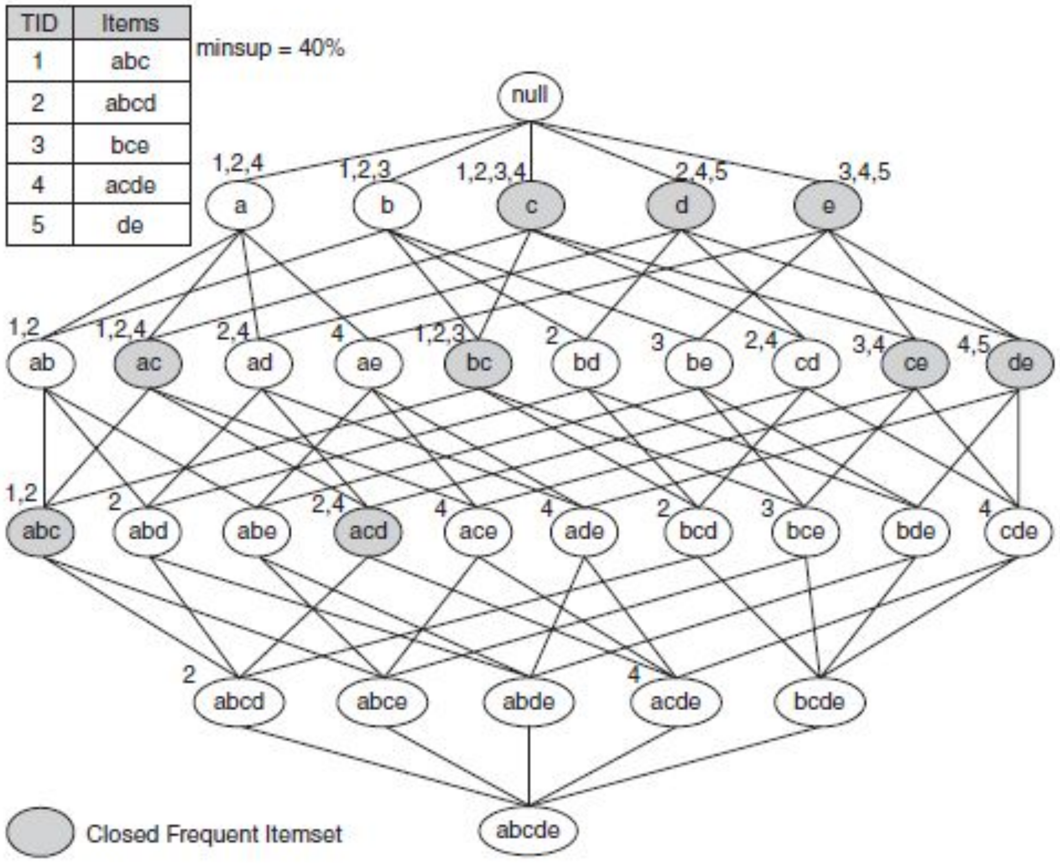
1. Maximal Frequent Itemsets

A frequent itemset is maximal if none of its immediate supersets are frequent.



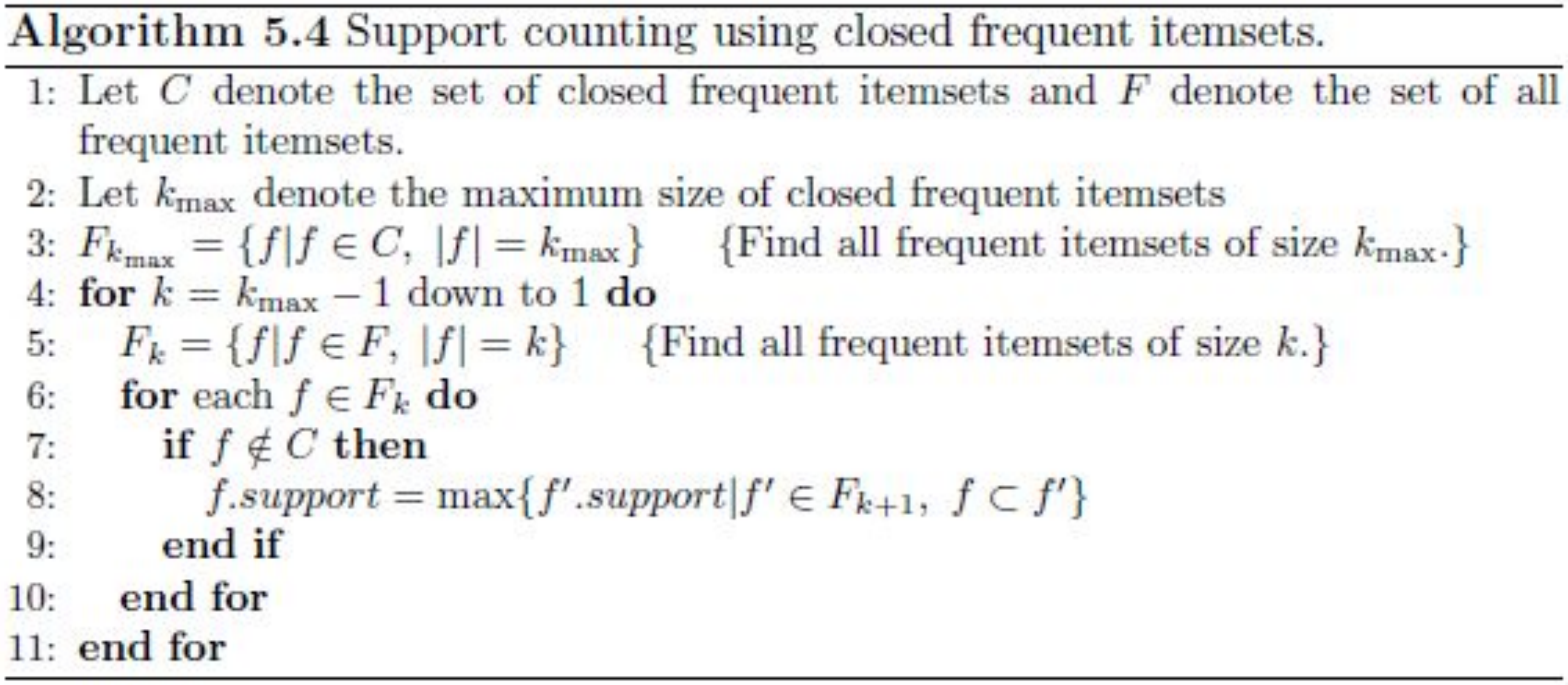
2. Closed Frequent Itemsets

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



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

Closed Frequent Itemset: An itemset is a closed frequent itemset if it is closed and its support is greater than or equal to minsup.



Algorithm 5.4 Support counting using closed frequent itemsets.

```
Let $C$ denote the set of closed frequent itemsets and $F$ denote the set of all frequent itemsets.
Let $k_{\max}$ denote the maximum size of closed frequent itemsets
$F_{k_{\max}} = \{f | f \in C,\ |f| = k_{\max}\}$    {Find all frequent itemsets of size $k_{\max}$.}
for $k = k_{\max} - 1$ down to 1 do
  $F_k = \{f | f \in F,\ |f| = k\}$    {Find all frequent itemsets of size $k$.}
  for each $f \in F_k$ do
    if $f \notin C$ then
      $f.support = \max\{f'.support | f' \in F_{k+1},\ f \subset f'\}$
    end if
  end for
end for
```

- The pseudocode for this algorithm is shown in Algorithm 5.4. The algorithm proceeds in a **specific-to-general** fashion, i.e., from the largest to the smallest frequent itemsets

- This is because, in order to find the support for a non-closed frequent itemset, the support for all of its supersets must be known. Note that the set of all frequent itemsets can be easily computed by taking the union of all subsets of frequent closed itemsets

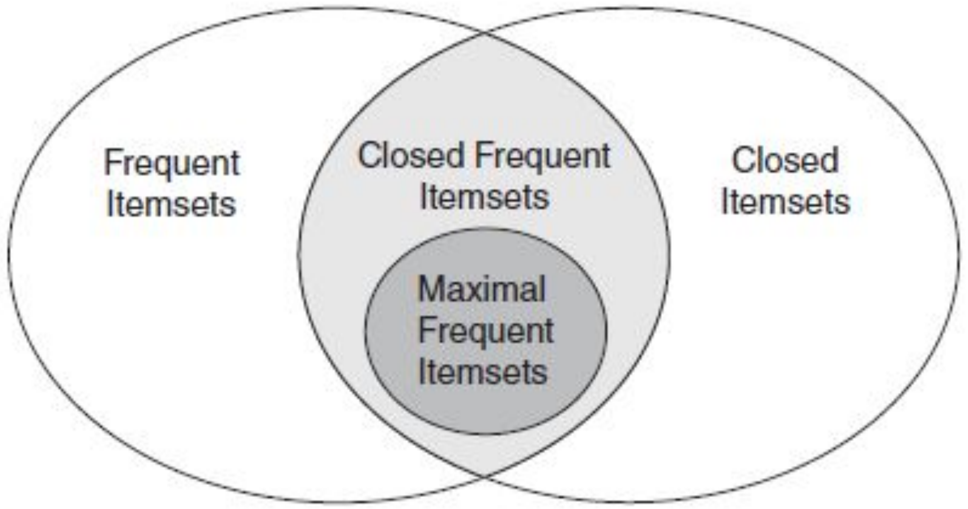
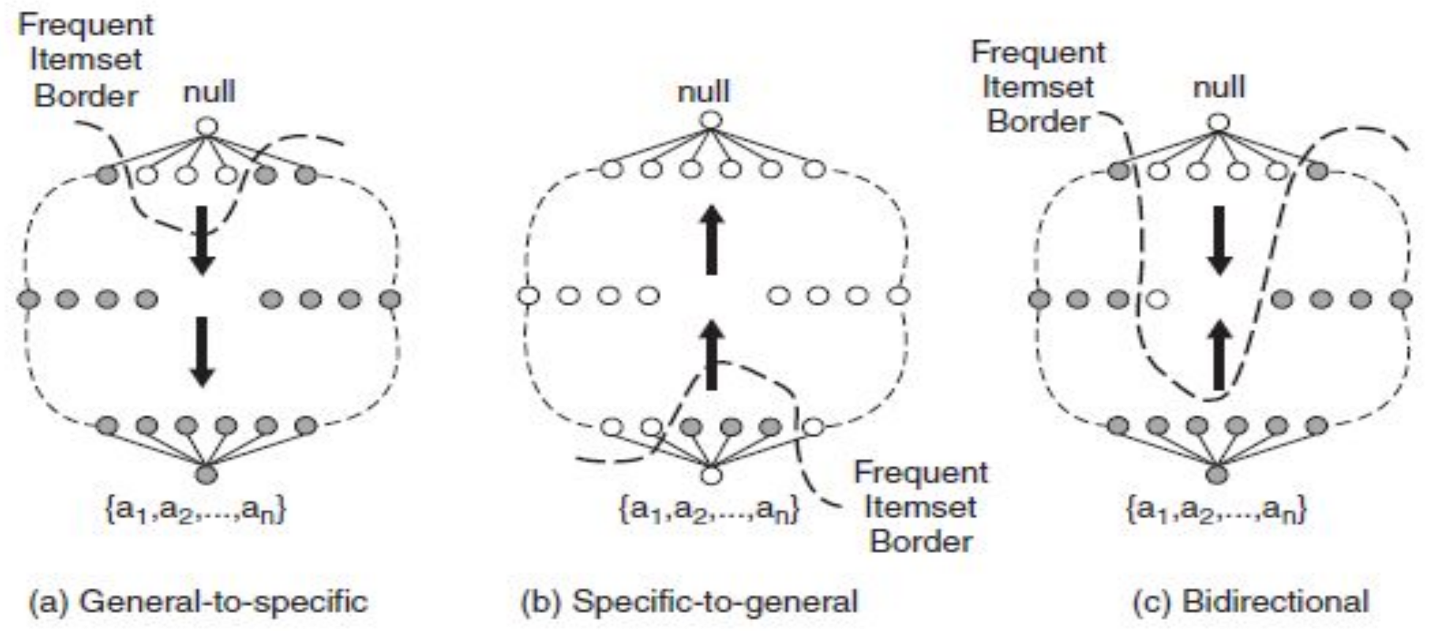


Figure 5.18. Relationships among frequent, closed, closed frequent, and maximal frequent itemsets.

- Note that all maximal frequent itemsets are closed because none of the maximal frequent itemsets can have the same support count as their immediate supersets.
- The relationships among frequent, closed, closed frequent and maximal frequent itemsets are shown in Figure 5.18.

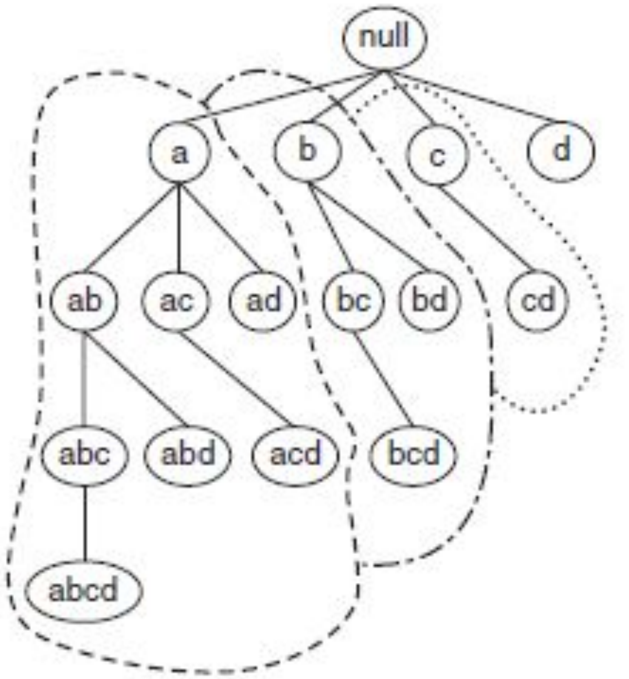
6.5: Alternative Methods for Generating Frequent Itemsets

1. General-to-Specific vs Specific-to-General

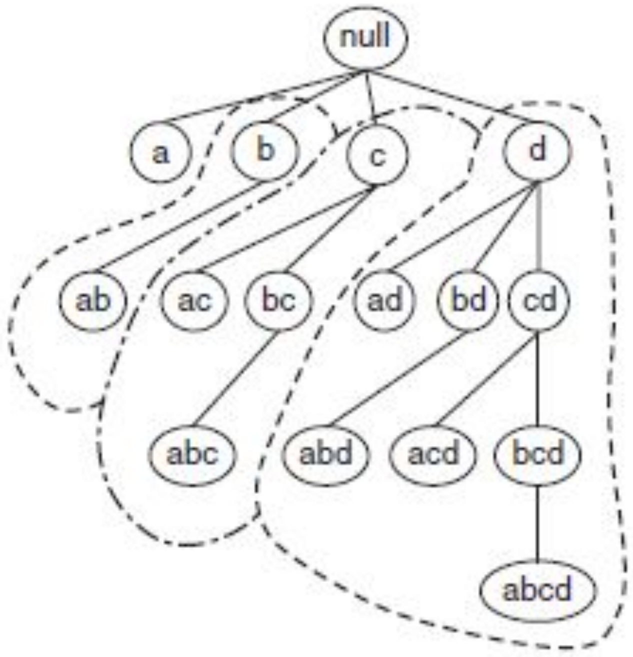


- **"General-to-Specific"** – where pairs of frequent(k-1) itemsets are merged to obtain k-itemsets (used by Apriori algorithm)
- **"Specific-to-General"** – it looks more specific frequent itemsets first, before finding the more general frequent itemsets (used by Maximal frequent itemsets)

2. Equivalence Classes



(a) Prefix tree.

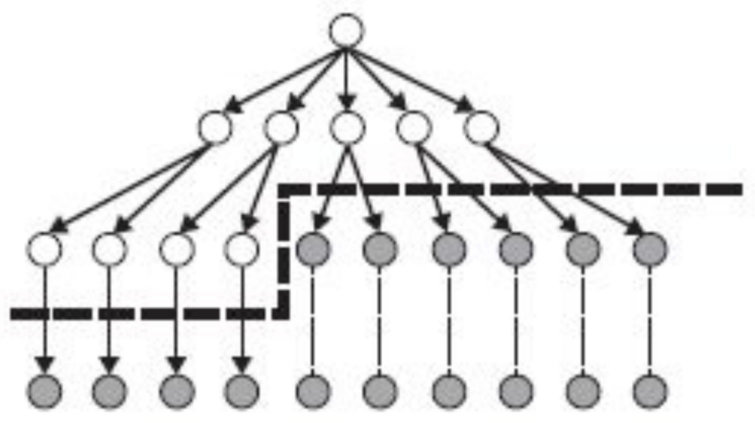


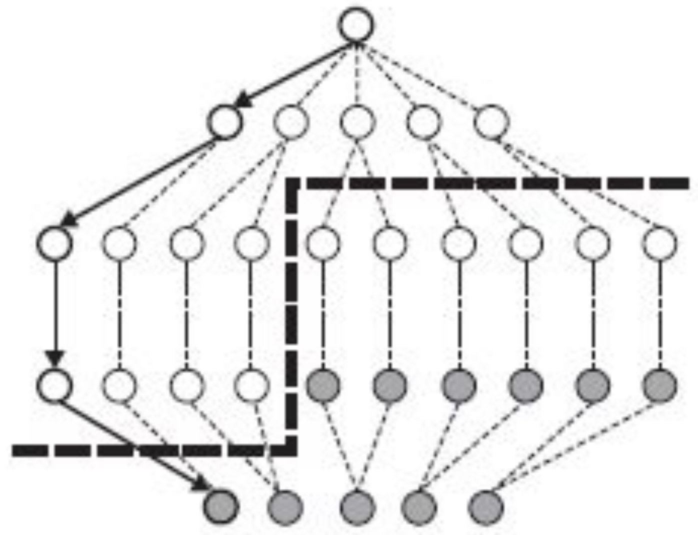
(b) Suffix tree.

- A frequent itemset generation algorithm searches for frequent itemsets within a particular equivalence class first before moving to another equivalence class.

- Equivalence classes can also be defined according to the prefix or suffix labels of an itemset. In this case, two itemsets belong to the same equivalence class if they share a common prefix or suffix of length k.

3. Breadth-First vs Depth-First


(a) Breadth first


(b) Depth first

- The Apriori algorithm traverses the lattice in a breadth-first manner. It first discovers all the frequent 1-itemsets, followed by the frequent 2-itemsets, and so on, until no new frequent itemsets are generated
- The depth-first approach is often used by algorithms designed to find maximal frequent itemsets. This approach allows the frequent itemset border to be detected more quickly than using a breadth-first approach.

6.6: FP – Growth Algorithm

The FP growth Algorithm is divided into 5 simple steps as shown

STEP 1: The first step is to count all the items in whole transactions to get support count.

STEP 2: Next, we apply the threshold (MinSup) we have set previously and remove item which posses support count less than "MinSup".

STEP 3: Now, we sort the list of items with support count greater than "MinSup" in descending order.

STEP 4: Now, we build the FP-tree, we go through each of the transactions and add all the items in the order they appear in our sorted list.

STEP 5: In order to get the associations now we go through every branch of the tree and only include in the association all the nodes whose count passed the threshold (MinSup).

Horizontal
Data Layout

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

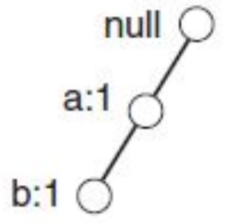
Vertical Data Layout

a	b	c	d	e
1	1	2	2	1
4	2	3	4	3
5	5	4	5	6
6	7	8	9	
7	8	9		
8	10			
9				

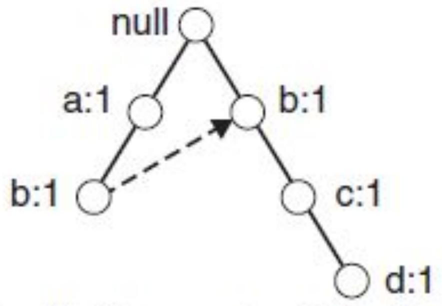
Figure 6.23. Horizontal and vertical data format.

Transaction
Data Set

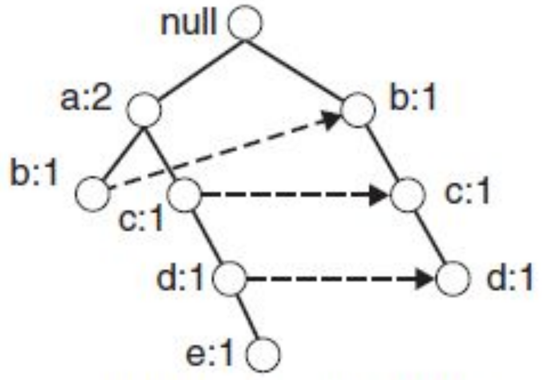
TID	Items
1	{a,b}
2	{b,c,d}
3	{a,c,d,e}
4	{a,d,e}
5	{a,b,c}
6	{a,b,c,d}
7	{a}
8	{a,b,c}
9	{a,b,d}
10	{b,c,e}



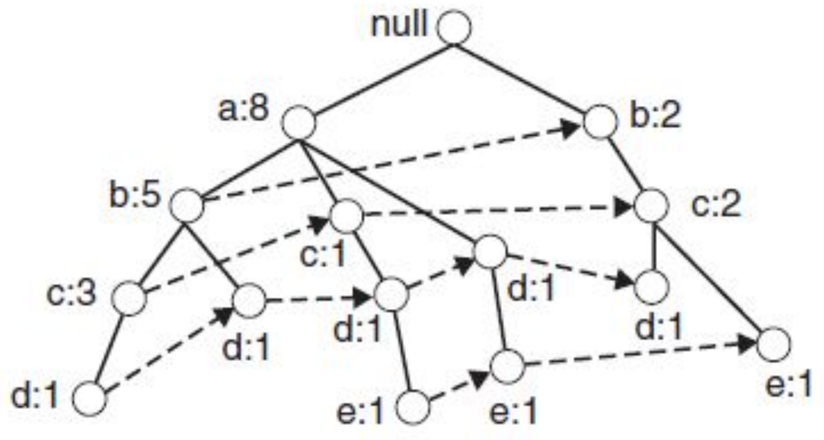
(i) After reading TID=1



(ii) After reading TID=2



(iii) After reading TID=3



(iv) After reading TID=10

Frequent Itemset Generation in FP-Growth Algorithm

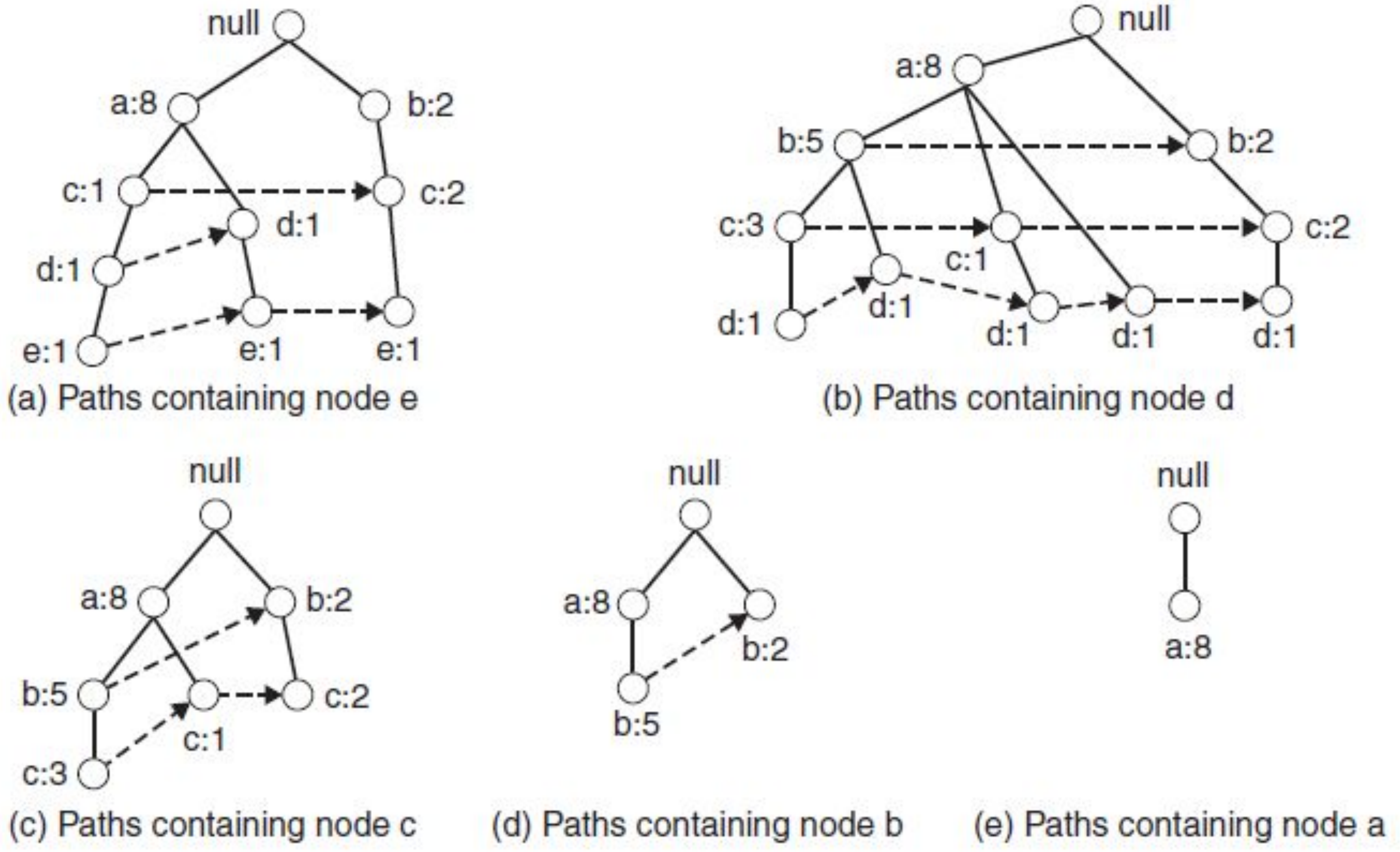
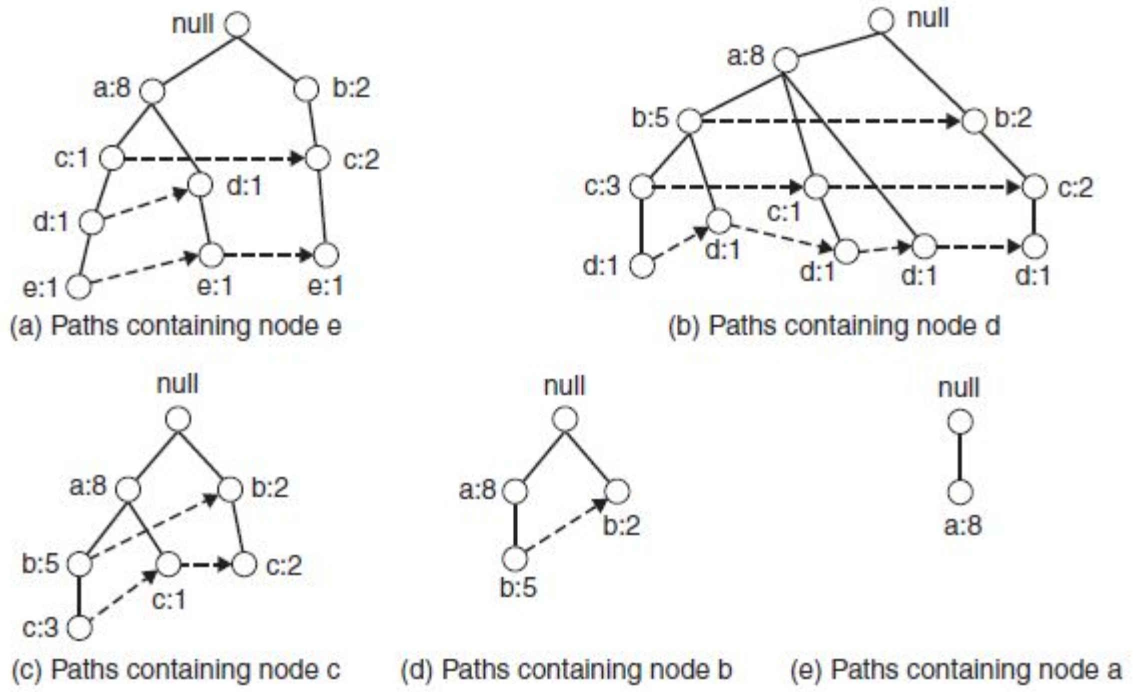

null
a:8
b:2
c:1
c:2
d:1
d:1
e:1
e:1
e:1
(a) Paths containing node e
null
a:8
b:5
b:2
c:3
c:1
c:2
d:1
d:1
d:1
d:1
d:1
(b) Paths containing node d
null
a:8
b:2
b:5
c:1
c:2
c:3
(c) Paths containing node c
null
a:8
b:2
b:5
(d) Paths containing node b
null
a:8
(e) Paths containing node a

Figure 5.26. Decomposing the frequent itemset generation problem into multiple subproblems, where each subproblem involves finding frequent itemsets ending in e, d, c, b, and a.



null
a:8
b:2
c:1
c:2
d:1
d:1
e:1
e:1
e:1
(a) Paths containing node e
null
a:8
b:5
b:2
c:3
c:1
c:2
d:1
d:1
d:1
d:1
d:1
(b) Paths containing node d
null
a:8
b:2
b:5
c:1
c:2
c:3
(c) Paths containing node c
null
a:8
b:2
b:5
(d) Paths containing node b
null
a:8
(e) Paths containing node a

Figure 5.26. Decomposing the frequent itemset generation problem into multiple subproblems, where each subproblem involves finding frequent itemsets ending in e, d, c, b, and a.

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}

Thank you