Mining Association Rules

Road map

- Basic concepts
- Apriori algorithm
- FP-Growth Algorithm
- Summary

Association rule mining

- Proposed by Agrawal et al in 1993.
- It is an important data mining model studied extensively by the database and data mining community.
- Assume all data are *categorical*.
- No good algorithm for numerical data.
- Initially used for Market Basket Analysis to find how items purchased by customers are related.

Bread $\rightarrow$ Milk [sup = 5%, conf = 100%]

The model: data

- $I = \{i_1, i_2, \ldots, i_m\}$: a set of *items*.
- Transaction t :
  - t a set of items, and $t \subseteq I$.
- Transaction Database T: a set of transactions $T = \{t_1, t_2, \ldots, t_n\}$.

Transaction data: supermarket data

- Market basket transactions:

  t1: {bread, cheese, milk}

  t2: {apple, eggs, salt, yogurt}

  … …

  tn: {biscuit, eggs, milk}

- Concepts:
  - An *item*: an item/article in a basket
  - *I*: the set of all items sold in the store
  - A *transaction*: items purchased in a basket; it may have TID (transaction ID)
  - A *transactional dataset*: A set of transactions

Transaction data: a set of documents

- **A text document data set. Each document is treated as a "bag" of keywords**

doc1: Student, Teach, School

doc2: Student, School

doc3: Teach, School, City, Game

doc4: Baseball, Basketball

doc5: Basketball, Player, Spectator

doc6: Baseball, Coach, Game, Team

doc7: Basketball, Team, City, Game

The model: rules

- A transaction *t* contains *X*, a set of items (itemset) in *I*, if $X \subseteq t$.
- An association rule is an implication of the form:

  $X \rightarrow Y$, where $X, Y \subset I$, *and* $X \cap Y = \varnothing$

- An itemset is a set of items.
  - E.g., X = {milk, bread, cereal} is an itemset.
- A *k*-itemset is an itemset with *k* items.
  - E.g., {milk, bread, cereal} is a 3-itemset

Rule strength measures

- Support: The rule holds with support *sup* in *T* (the transaction data set) if sup% of transactions contain $X \cup Y$. [$X \cup Y$ happens sup% times.]
  - *sup* = Pr($X \cup Y$).
- Confidence: The rule holds in *T* with confidence *conf* if *conf*% of transactions that contain *X* also contain *Y.*
  - *conf* = Pr($Y \mid X$)
- An association rule is a pattern that states when *X* occurs, *Y* occurs with certain probability.

Support and Confidence

- Support count: The support count of an itemset *X*, denoted by *X.count*, in a data set *T* is the number of transactions in *T* that contain *X*. Assume *T* has *n* transactions.
- Then,

$$support = \frac{(X \cup Y).count}{n}$$

$$confidence = \frac{(X \cup Y).count}{X.count}$$

Goal and key features

- **Goal:** Find all rules that satisfy the user-specified *minimum support* (minsup) and *minimum confidence* (minconf).
- **Key Features**
  - Completeness: find all rules.
  - Mining with data on hard disk (not in memory)

An example

t1:	Beef, Chicken, Milk
t2:	Beef, Cheese
t3:	Cheese, Boots
t4:	Beef, Chicken, Cheese
t5:	Beef, Chicken, Clothes, Cheese, Milk
t6:	Chicken, Clothes, Milk
t7:	Chicken, Milk, Clothes

- Transaction data
- Assume:

  minsup = 30%

  minconf = 80%
- An example frequent *itemset*:

{Chicken, Clothes, Milk} [sup = 3/7]

- Association rules from the itemset:

Clothes → Milk, Chicken [sup = 3/7, conf = 3/3]

… …

Clothes, Chicken → Milk, [sup = 3/7, conf = 3/3]

Many mining algorithms

- There are a large number of them!!
- They use different strategies and data structures.
- Their resulting sets of rules are all the same.
  - Given a transaction data set *T*, and a minimum support and a minimum confident, the set of association rules existing in *T* is uniquely determined.
- Any algorithm should find the same set of rules although their computational efficiencies and memory requirements may be different.
- We study here: the Apriori Algorithm

Road map

- Basic concepts
- Apriori algorithm
- FP-Growth Algorithm
- Summary

The Apriori algorithm

- **Probably the best known algorithm**
- **Two steps**:
  - Find all itemsets that have minimum support (*frequent itemsets*, also called large itemsets).
  - Use frequent itemsets to generate rules.

- E.g., a frequent itemset

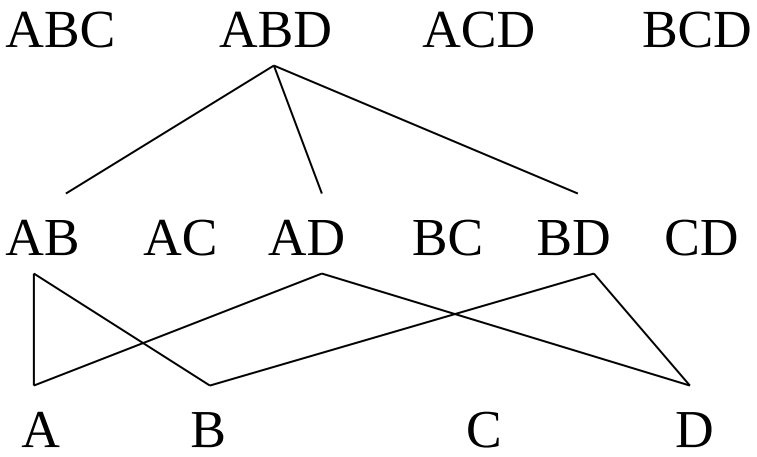
  {Chicken, Clothes, Milk} [sup = 3/7]

  and one rule from the frequent itemset

  Clothes $\rightarrow$ Milk, Chicken [sup = 3/7, conf = 3/3]

Step 1: Mining all frequent itemsets

- A frequent *itemset* is an itemset whose support is ≥ minsup.
- Key idea: The apriori property (downward closure property): any subsets of a frequent itemset are also frequent itemsets



The Algorithm

- Iterative algo. (also called level-wise search): Find all 1-item frequent itemsets; then all 2-item frequent itemsets, and so on.
  - In each iteration *k*, only consider itemsets that contain some *k*-1 frequent itemset.

- Find frequent itemsets of size 1: F_1
- From $k = 2$
  - C_k = candidates of size *k*: those itemsets of size *k* that could be frequent, given F_{k-1}
  - F_k = those itemsets that are actually frequent, $F_k \subseteq C_k$ (need to scan the database once).

Details: the algorithm

Algorithm Apriori(*T*)

$C_1 \leftarrow$ init-pass(T);
$F_1 \leftarrow \{f \mid f \in C_1, f.\text{count}/n \geq minsup\}$; // n: no. of transactions in T
for (k = 2; $F_{k-1} \neq \emptyset$; k++) **do**
  $C_k \leftarrow$ candidate-gen(F_{k-1});
  for each transaction $t \in T$ **do**
    for each candidate $c \in C_k$ **do**
      if c is contained in t **then**
        $c.count$++;
    end
  end
  $F_k \leftarrow \{c \in C_k \mid c.count/n \geq minsup\}$
end
return $F \leftarrow \bigcup_k F_k$;

Apriori candidate generation

- The candidate-gen function takes F_{k-1} and returns a superset (called the candidates) of the set of all frequent k-itemsets. It has two steps
  - *join* step: Generate all possible candidate itemsets C_k of length k
  - *prune* step: Remove those candidates in C_k that cannot be frequent.

Candidate-gen function

Function candidate-gen(F_{k-1})

$C_k \leftarrow \emptyset$;

forall $f_1, f_2 \in F_{k-1}$

with $f_1 = \{i_1, \ldots, i_{k-2}, i_{k-1}\}$

and $f_2 = \{i_1, \ldots, i_{k-2}, i'_{k-1}\}$

and $i_{k-1} < i'_{k-1}$ **do**

$c \leftarrow \{i_1, \ldots, i_{k-1}, i'_{k-1}\}$; // join f_1 and f_2

$C_k \leftarrow C_k \cup \{c\}$;

for each (k-1)-subset s of c **do**

if ($s \notin F_{k-1}$) **then**

delete c from C_k; // prune

end

end

return C_k;

Apriori Candidate Generation — Example

Given, minimum support = 50% -> (50/100) *(n=4) => 2
minimum confidence =70%

n=number of transactions

Database D

TID	Items
100	1 3 4
200	2 3 5
300	1 2 3 5
400	2 5

Scan D

C_1

itemset	sup.
{1}	2
{2}	3
{3}	3
{4}	1
{5}	3

F_1

itemset	sup.
{1}	2
{2}	3
{3}	3
{5}	3

C_2

itemset
{1 2}
{1 3}
{1 5}
{2 3}
{2 5}
{3 5}

Scan D

C_2

itemset	sup
{1 2}	1
{1 3}	2
{1 5}	1
{2 3}	2
{2 5}	3
{3 5}	2

F_2

itemset	sup
{1 3}	2
{2 3}	2
{2 5}	3
{3 5}	2

C_3

itemset
{2 3 5}

Scan D

F_3

itemset	sup
{2 3 5}	2

:2

{1, 2,3}:0
{1, 3,5}:1

Step 2: Generating rules from frequent itemsets

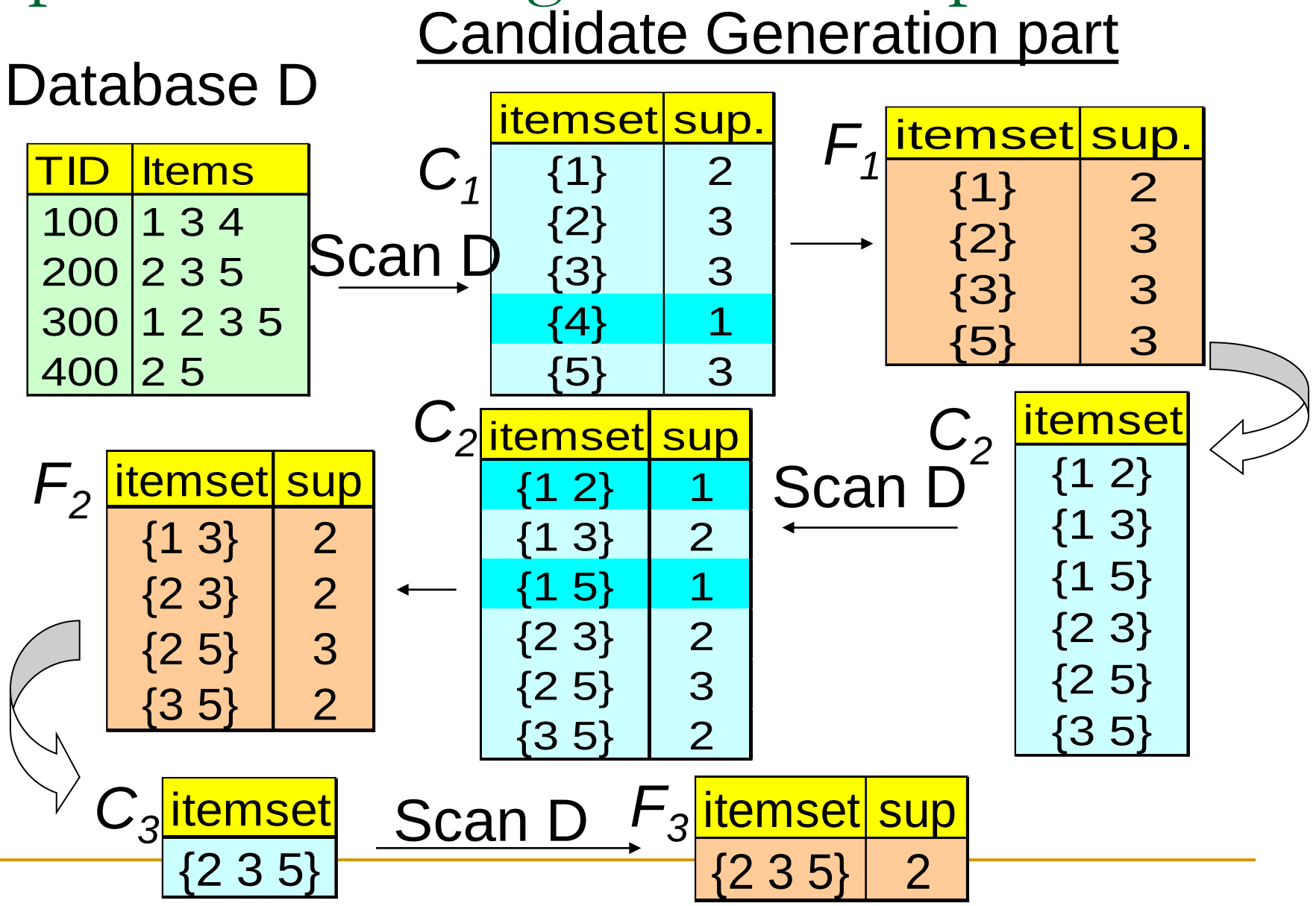
- Frequent itemsets ≠ association rules
- One more step is needed to generate association rules
- For each frequent itemset *X*,

  For each proper nonempty subset *A* of *X*,
  - Let *B* = X - *A*
  - ***A* → *B* is an association rule if**
    - Confidence(A → B) ≥ minconf,

      support(A → B) = support(A∪B)

      confidence(A → B) = support(A ∪ B) / support(A)

Apriori Generating Rule- Example



In C3, other itemsets are not shown due to their supp<minsup.

Apriori Generating Rule- Example

Given,
Minimum support = 50%
Minimum confidence = 70%

itemset	sup
{2 3 5}	2

$$confidence = \frac{(X \cup Y).count}{X.count}$$

Rule	Support	Confidence	Confidence(%)
{2,3} -> {5}	2	2/2 = 1	100% >=70%
{2,5} -> {3}	2	2/3 = 0.67	67%
{3,5} -> {2}	2	2/2 = 1	100% >=70%
{5} -> {2,3}	2	2/3 = 0.67	67%
{3} -> {2,5}	2	2/3 = 0.67	67%
{2} -> {3,5}	2	2/3 = 0.67	67%

We found the following rules:

{2,3} -> {5}
{3,5} -> {2}

Interpretation of rule: {2,3} -> {5}:

If we sell 2 and 3 together, probability of selling 5 is 100%

Generating rules: summary

- To recap, in order to obtain $A \rightarrow B$, we need to have support($A \cup B$) and support(A)
- All the required information for confidence computation has already been recorded in itemset generation. No need to see the data *T* any more.
- This step is not as time-consuming as frequent itemsets generation.

On Apriori Algorithm

Seems to be very expensive

- Level-wise search
- K = the size of the largest itemset
- It makes at most K passes over data
- In practice, K is bounded (10).
- The algorithm is very fast. Under some conditions, all rules can be found in linear time.

Apriori Limitations

- Using Apriori needs a generation of candidate itemsets. These itemsets may be large in number if the itemset in the database is huge.

- Apriori needs multiple scans of the database to check the support of each itemset generated and this leads to high costs.

Road map

- Basic concepts
- Apriori algorithm
- FP-Growth Algorithm
- Summary

FP-Growth Algorithm Introduction

- This algorithm is an improvement to the Apriori method. A frequent pattern is generated without the need for candidate generation. FP growth algorithm represents the database in the form of a tree called a frequent pattern tree or FP tree.
- For so much, it uses a divide-and-conquer strategy.
- The core of this method is the usage of a special data structure named frequent-pattern tree (FP-tree), which retains the item set association information.

FP-Growth Algorithm Introduction (Contd.)

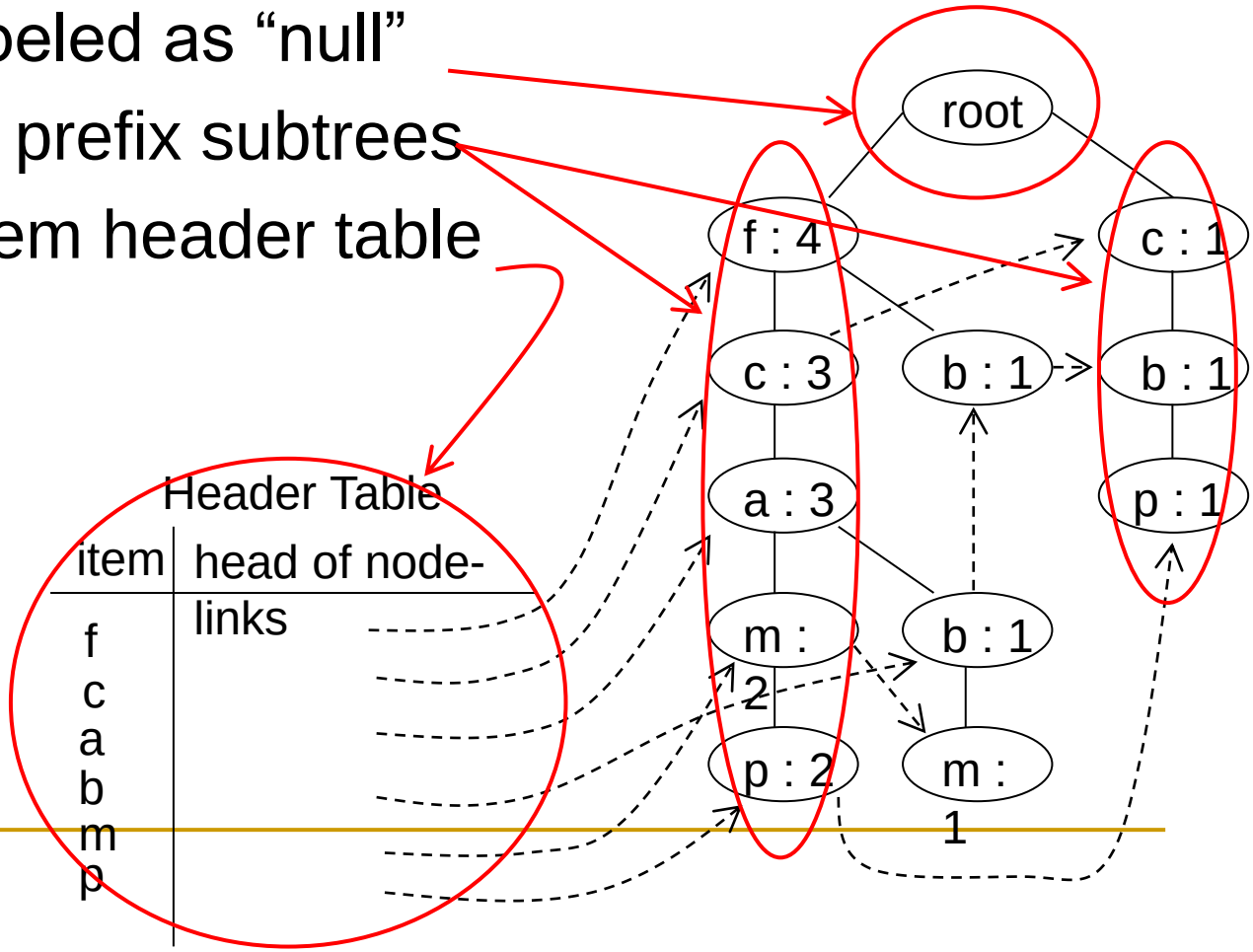
This algorithm works as follows:

- First, it compresses the input database creating an FP-tree instance to represent frequent items.
- After this first step, it divides the compressed database into a set of conditional databases, each associated with one frequent pattern.
- Finally, each such database is mined separately.

Using this strategy, the FP-Growth reduces the search costs by recursively looking for short patterns and then concatenating them into the long frequent patterns.

FP-Tree

- Three components:
  - One root: labeled as “null”
  - A set of item prefix subtrees
  - A frequent-item header table



An Example

The given data is a hypothetical dataset of transactions with each letter representing an item. **The minimum support given is 3.**

TID	Items Bought
100	*f, a, c, d, g, i, m, p*
200	*a, b, c, f, l, m, o*
300	*b, f, h, j, o*
400	*b, c, k, s, p*
500	*a, f, c, e, l, p, m, n*

- In the frequent pattern growth algorithm, first, we find the frequency of each item. The following table gives the frequency of each item in the given data.

TID	Items Bought
100	*f, a, c, d, g, i, m, p*
200	*a, b, c, f, l, m, o*
300	*b, f, h, j, o*
400	*b, c, k, s, p*
500	*a, f, c, e, l, p, m, n*

Item	Frequency	Item	Frequency
a	3	j	1
b	3	k	1
c	4	l	2
d	1	m	3
e	1	n	1
f	4	o	2
g	1	p	3
h	1	s	1
i	1		

A **Frequent Pattern set (L)** is built, which will contain all the elements whose frequency is greater than or equal to the minimum support.

These elements are stored in descending order of their respective frequencies.

As minimum support is 3,after insertion of the relevant items, the set L looks like this:-

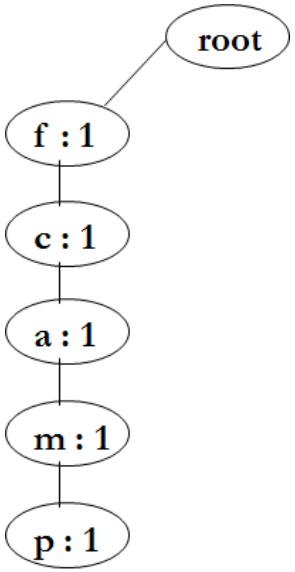
L = { (f:4), (c:4), (a:3), (b:3), (m:3), (p:3) }

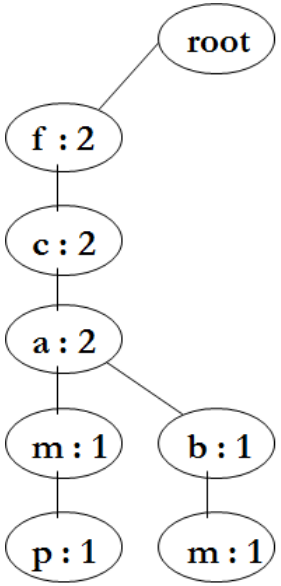
Now, for each transaction, the respective **Ordered-Item set** is built.

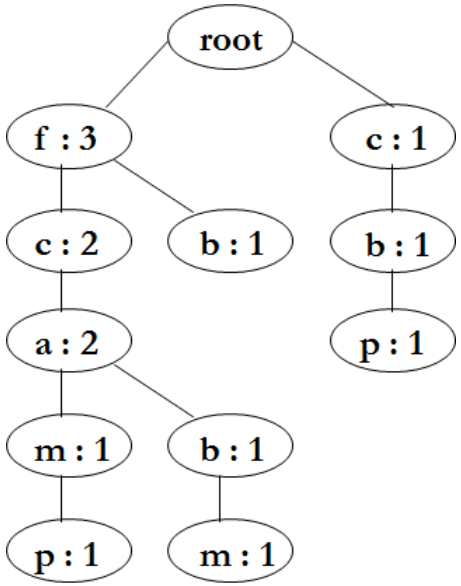
Frequent Pattern set L = { (f:4), (c:4), (a:3), (b:3), (m:3), (p:3) }

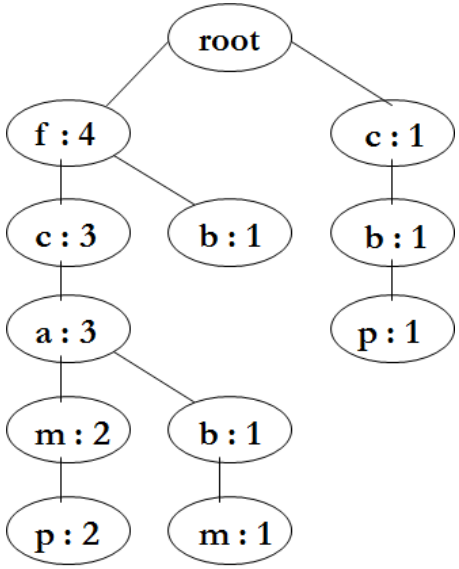
TID	Items Bought	(Ordered) Frequent Items
100	*f, a, c, d, g, i, m, p*	*f, c, a, m, p*
200	*a, b, c, f, l, m, o*	*f, c, a, b, m*
300	*b, f, h, j, o*	*f, b*
400	*b, c, k, s, p*	*c, b, p*
500	*a, f, c, e, l, p, m, n*	*f, c, a, m, p*

Now, all the Ordered-Item sets are inserted into a Tree Data Structure (**frequent pattern tree**).



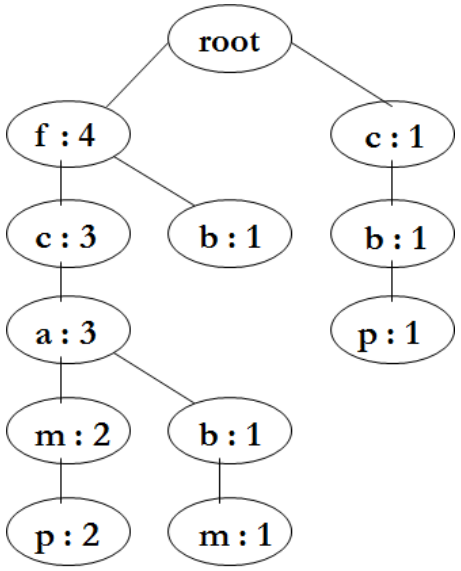






TID	(Ordered) Frequent Items
100	*f, c, a, m, p*
200	*f, c, a, b, m*
300	*f, b*
400	*c, b, p*
500	*f, c, a, m, p*

- Now, for each item, the **Conditional Pattern Base** is computed which is the path labels of all the paths which lead to any node of the given item in the frequent-pattern tree.



Item	Conditional Pattern Base
p	{{f, c, a, m : 2}, {c, b : 1}}
m	{{f, c, a : 2}, {f, c, a, b : 1}}
b	{{f, c, a : 1}, {f : 1}, {c : 1}}
a	{{f, c : 3}}
c	{{f : 3}}
f	Φ

- Now for each item, the **Conditional Frequent Pattern Tree** is built. It is done by taking the set of elements that is common in all the paths in the Conditional Pattern Base of that item and calculating its support count by summing the support counts of all the paths in the Conditional Pattern Base.

Item	Conditional Pattern Base	Conditional FP-Tree
p	{{f, c, a, m : 2}, {c, b : 1}}	{c : 3}
m	{{f, c, a : 2}, {f, c, a, b : 1}}	{f, c, a :3}
b	{{f, c, a : 1}, {f : 1}, {c : 1}}	Φ
a	{{f, c : 3}}	{f, c : 3}
c	{{f : 3}}	{f : 3}
f	Φ	Φ

- From the Conditional Frequent Pattern tree, the **Frequent Pattern rules** are generated by pairing the items of the Conditional Frequent Pattern Tree set to the corresponding item.

Item	Conditional Pattern Base	Conditional FP-Tree	Frequent Patterns Generated
p	{{f, c, a, m : 2}, {c, b : 1}}	{c : 3}	{<c, p : 3>}
m	{{f, c, a : 2}, {f, c, a, b : 1}}	{f, c, a :3}	{ <f, m : 3>, <c, m : 3> <a, m : 3>, <f, c, m : 3> <f, a, m : 3>, <c, a, m :3>}
b	{{f, c, a : 1}, {f : 1}, {c : 1}}	Φ	{}
a	{{f, c : 3}}	{f, c : 3}	{<f, a : 3>, <c, a : 3>, <f, c, a:3>}
c	{{f : 3}}	{f : 3}	{ <f, c : 3>}
f	Φ	Φ	{}

Generating Association Rules

- From the frequent itemsets, we will follow the same procedure of “Rule Generation” as shown in Apriori Algorithm.
  - Here, for each pattern we will generate all possible rules.
  - For all rules, we select the valid rules those pass the minconf threshold.

Differences between Apriori and FP-Growth

Apriori	FP Growth
Apriori generates frequent patterns by making the itemsets using pairings such as single item set, double itemset, and triple itemset.	FP Growth generates an FP-Tree for making frequent patterns.
Apriori uses candidate generation where frequent subsets are extended one item at a time.	FP-growth generates a conditional FP-Tree for every item in the data.
Since apriori scans the database in each step, it becomes time-consuming for data where the number of items is larger.	FP-tree requires only one database scan in its beginning steps, so it consumes less time.
A converted version of the database is saved in the memory	A set of conditional FP-tree for every item is saved in the memory
It uses a breadth-first search	It uses a depth-first search.