UNIT-IV

Mining Frequent Patterns, Association, and Correlations

Objectives:

Students should be able to

- Discovers the interesting association relationships among huge amount of data.

Syllabus:

Basic Concepts, frequent item sets, closed item sets and association rules, frequent item set mining methods : Apriori Algorithm, generations, association rules form frequent item sets, A Pattern- Growth approach for mining frequent item sets.

Outcomes:

Students should be able to

- Apply the different methods for mining Association rules.
- Understand the role of support and confidence.

Learning Material

4.1. Basic Concepts

Association Rule Mining

- Association rule mining finding frequent patterns, associations, correlations among sets of items or objects in transactional databases, traditional databases, relational databases or other information repositories.
- The discovery of interesting association relationships among huge amounts of business transaction records can help in many business decision making processes, such as market basket analysis, catalog design, cross-marketing, and loss-leader analysis.

Market Basket Analysis:

Association rule mining searches for interesting relationships among items in a given data set.

This process analyzes customer buying habits by finding associations between the different items that customers place in their "shopping baskets". The discovery of such associations can help retailers develop marketing strategies by gaining insight into which **items are frequently purchased together by customers**.

For instance, if customers are buying milk, how likely are they to also buy bread on the same trip to the supermarket. Such information can lead to increased sales by helping retailers do selective marketing and plan their shelf space. For example, placing milk and bread within **close proximity** may further encourage the sale of these items together within single visits to the store.

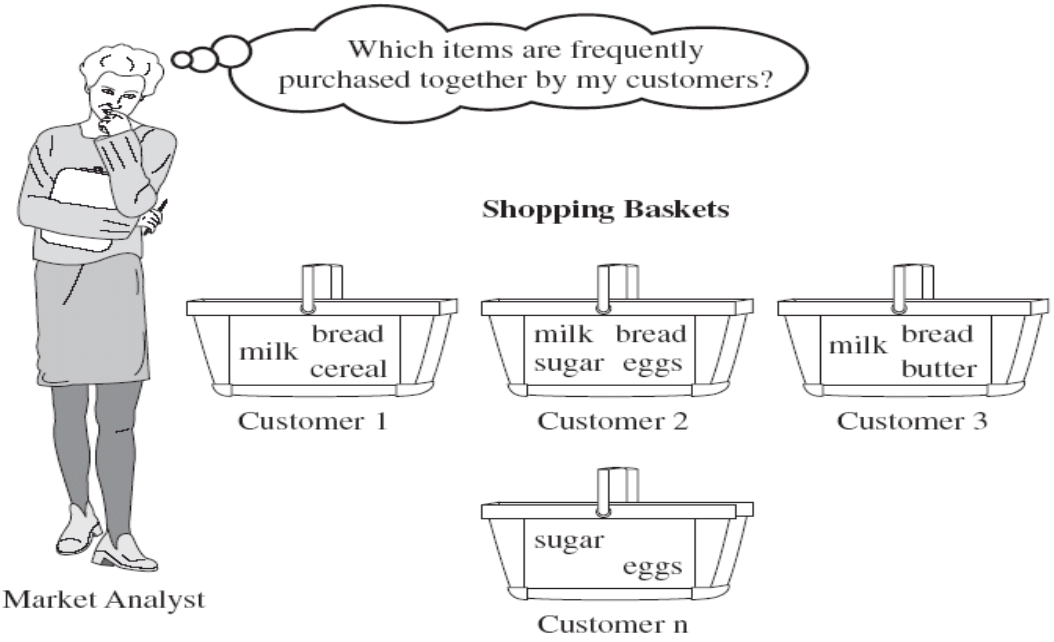


Figure 4.1 Market basket analysis.

Analyze the buying patterns that reflect items that are frequently Associated or purchased together. These patterns can be represented in the form of Association rules.

Rule form: "Body => Head [support, confidence]".

For example, the information that customers who purchase computers Also tend to buy financial management software at the same time is represented in Associated Rule below:

Computer=>financial_management_software[support=2%,confidence= 60%]

Rule **support and confidence** are two measures of rule interestingness, they respectively reflect the usefulness and certainty of discovered rules. Typically, association rules are considered interesting if they satisfy both a minimum support threshold and a minimum confidence threshold such thresholds can be set by the users or domain experts.

4.1.1. Frequent Itemsets, Closed Itemsets, and Association Rules

Let I be a set of items $\{I_1, I_2, I_3, \ldots, I_m\}$,Let D be a set of database transactions where each transaction T is a set of items such that $T \subseteq I$. *Each transaction is associated with an identifier , called TID.*

Let A be a set of items. A transaction T is said to contain A if and only if A $\subseteq$ T.

An **association rule** is an implication of the form $\boldsymbol{A \Rightarrow B}$ where $A \subseteq I$, $B \subseteq I$, and $A \cap B = \Phi$.

The rule A=> B holds in the transaction set D with **support s,** where **s** is **the percentage of transactions in D that contain A ∪ B** (i.e both A and B). This is taken to be the probability, P(A ∪ B).

The rule A=>B has **confidence c** in the transaction set D if c is the **percentage of transactions in D containing A that also contain B**. This is taken to be the **conditional probability P(B/A)**. That is

Support (A=>B) = P(A ∪ B).

Confidence (A=>B) = P(B/A).

Strong Association Rules : Rules that satisfy both a minimum support threshold (min_sup) and a minimum confidence threshold (min_conf) are called **strong.** By convention, we write support and confidence value so as to occur between 0% and 100% rather than 0 to 1.0.

Item set : A set of items is referred to as an itemset.An itemset that contains k items is a k-itemset. Ex : The set{computer, financial_management_software} is a 2-itemset.

Support count : The occurrence frequency of an itemset is the number of transactions that contain the itemset. This is also known as the **frequency**, **support count** or **count** of the itemset.

An itemset satisfies minimum support if the occurrence frequency of the itemset is greater than or equal to the product of min_sup and the total number of transactions in D. The number of transactions required for the itemset to satisfy minimum support is therefore referred to as the minimum support count.

Frequent itemset: if an itemset satisfies minimum support, then it is a frequent itemset **.**

The set of frequent K-itemsets is commonly denoted by L_K.

Association rule mining is a two-step process:

step 1: Find all frequent itemsets: By definition, each of these itemsets will occur at least as frequently as a pre-determined minimum support count.

step 2 : Generate strong association rules from the frequent itemsets: By definition ,these rules must satisfy minimum support and minimum confidence. The overall performance is determined by the First step.

Support & confidence:

Support(s) of an association rule is defined as the percentage/fraction of records that contain A U B to the total number of records in the database.

Support (A->B)=P (A U B) = support_count(A U B)/count(all_transactions).

Confidence of an association rule is defined as the percentage/fraction of the number of transactions that contain A U B to the total number of records that contain A.

Confidence(A-> B) =P(B/A) =support(A U B)/support (A).
= support_count(A U B)/support_count(A).

Example :

Data set *D*

TID	Itemsets
T100	1 3 4
T200	2 3 5
T300	1 2 3 5
T400	2 5

Support count, Support and Confidence:

Support count(1,3)=2

|D| = 4

Support(1->3)=2/4 = 0.5

Support(3→2)=2/4 =0.5

Confidence(3→2)= count(2 U 3) / count(3)

= 2/3

= 0.67

Association Rules

Association rules can be **classified** in various ways, based on the following criteria:

- **Based on the types of values handled in the rule**:

  if a rule concerns associations between the presence or absence of items ,it is a **Boolean association rule.**

  Computer=>financial_management_software[support=2%,confidence= 60%]

  For example, the rule above is a Boolean association rule obtained from market basket analysis.

  If a rule describes associations between quantitative items or attributes, then it is a **quantitative association rule**

  In these rules , quantitative values for items or attributes are partitioned into intervals. The following rule is an example of a quantitative association rule, where X is a variable representing a customer:

  age(X,"30.....39")^income(X,"42K.....48")=>buys(X,highresolutionTV).

  The quantitative attributes age and income ,have been discretized,

- **Based on dimensions of data involved in the rule:**

  If the items or attributes in an association rule reference only one dimension, then it is a **single-dimensional** association rule.

  buys(X,"computer") => buys(X,"financial_management_software")

  The above rules a single dimensional association rule since it refers to only one dimension, buys.

If a rule references two or more dimensions, such as the dimensions buys, time_of_transaction ,and customer_category, then it is a **multidimensional association rule**.

age(X,"30.....39")^income(X,"42K.....48")=>buys(X, high resolution TV).

- **Based on the levels of abstraction in the rule set:**

A method for association rule mining can find rules at differing levels of abstraction. For example, suppose that a set of association rule mined includes the following rules:

age(x,"30...39") buys(x,"laptop computer").

age(x,"30...39") buys{x,"computer").

In above rules the items bought are referenced at different levels of abstraction.

e.g.," computer" is a higher-level abstraction of "laptop computer".

- **Based on various extensions to association mining:**

Association mining can be extended to correlation analysis, where the absence or presence of correlated items can be identified. It can also be extended to mining max patterns (i.e., maximal frequent patterns) and frequent closed itemsets.

Closed and max pattern frequent itemsets

Closed and max pattern frequent itemsets is a frequent pattern,p,such that any proper sub pattern of p is not frequent. A frequent **closed itemset** is a frequent closed itemset where an itemset c is closed if there exists no proper superset of c , c ' such that every transaction containing c also contains c'.

Maxpatterns and frequent closed itemset can be used to substantially reduce the number of frequent itemsets generated in mining.

4.2 Frequent Item set mining Methods:

It is a two step process:

Step 1: Generation of frequent itemsets.

- Apriori algorithm.
- *Frequent-pattern growth (FP-Growth).*

Step 2: Generation of Association rules for the above frequent itemsets.

4.2.1. The Apriori algorithm : Finding Frequent itemsets by Confined candidate generation

- This algorithm uses the **prior** knowledge of frequent itemset properties.
- Uses 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. This set is denoted by L_1. L_1 is used to find L_2, the set of frequent 2-itemsets,whichis used to find L_3,and son 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 to reduce the search space**: " All nonempty subsets of a frequent itemset must also be frequent."
- P(I)<min_sup => I is not frequent.
- If an item A is added to the itemset I, then the resulting itemset (I U A) can not occur more frequently than I. Therefore, I U A is not frequent either, i.e. P(I+A)<min_sup
- **Anti-Monotone property** – "if a set cannot pass a test, all of its supersets will fail the same test as well "

- Using the apriori property in the algorithm:
    - Let us look at how Lk-1 is used to find Lk, for k>=2
- Two steps:
    - **Join :** C_k is generated by joining $L_{k-1}$with itself.
    - **Prune:** Any (k-1)-itemset that is not frequent cannot be a subset of a frequent k-itemset.
        - Scaning the database to determine the count of each candidate in Ck – heavy computation
        - To reduce the size of Ck the Apriori property is used: if any (k-1) subset of a candidate k-itemset is not in Lk-1, then the candidate cannot be frequent either,so it can be removed from Ck. – subset testing (hash tree)

Example: Transactional data for an All Electronics branch.

TID	List of items_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

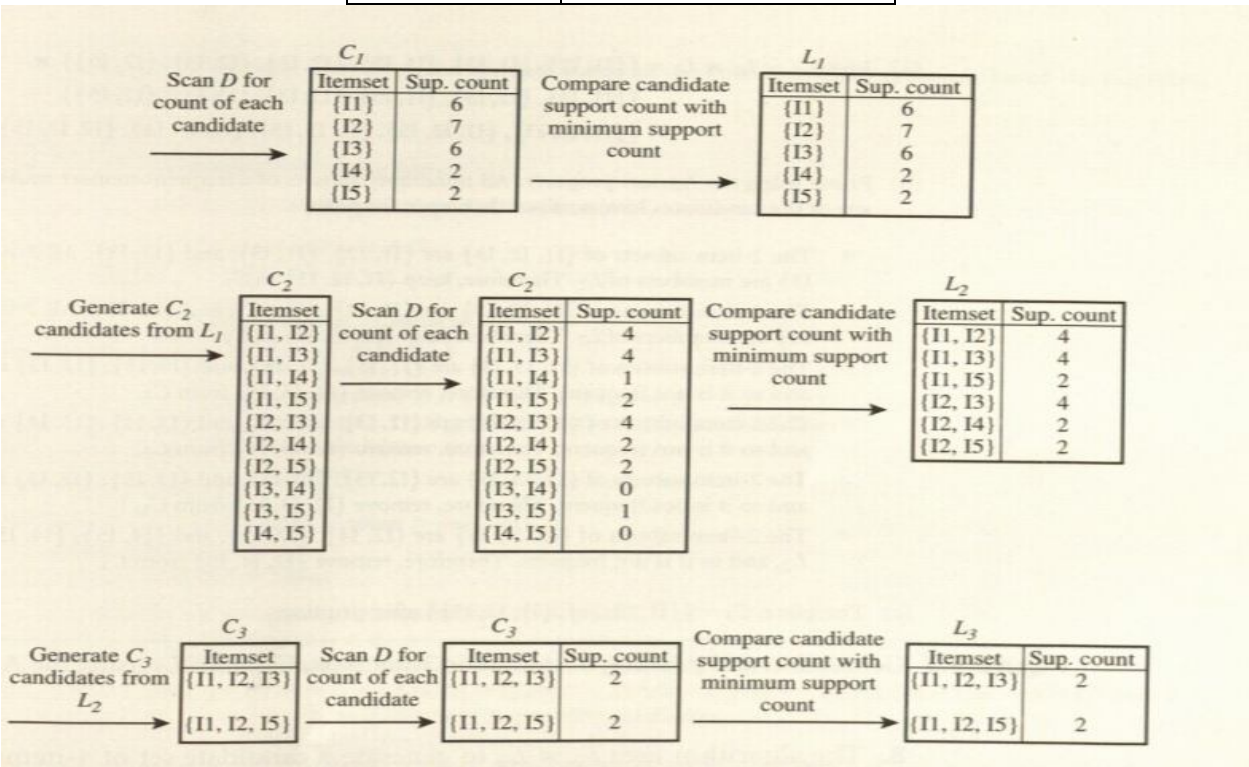


Figure: Generation of candidate itemsets and frequent itemsets, where the minimum support count is 2.

- Scan D for count of each candidate

  C1: I1 – 6, I2 – 7, I3 -6, I4 – 2, I5 - 2

- Compare candidate support count with minimum support count (min_sup=2)

  L1: I1 – 6, I2 – 7, I3 -6, I4 – 2, I5 - 2

- Generate C2 candidates from L1 and scan D for count of each candidate

  C2: {I1,I2} – 4, {I1, I3} – 4, {I1, I4} – 1, …

- Compare candidate support count with minimum support count

L2: {I1,I2} – 4, {I1, I3} – 4, {I1, I5} – 2, {I2, I3} – 4, {I2, I4} - 2, {I2, I5} – 2

- Generate C3 candidates from L2 using the join and prune steps:

  Join: C3=L2xL2={{I1, I2, I3}, {I1, I2, I5}, {I1, I3, I5}, {I2, I3, I4}, {I2, I3, I5}, {I2, I4, I5}}

  Prune: C3: {I1, I2, I3}, {I1, I2, I5}

- Scan D for count of each candidate

  C3: {I1, I2, I3} - 2, {I1, I2, I5} – 2

- Compare candidate support count with minimum support count

  L3: {I1, I2, I3} – 2, {I1, I2, I5} – 2

- Generate C4 candidates from L3

  C4=L3xL3={I1, I2, I3, I5}

  This itemset is pruned, because its subset {{I2, I3, I5}} is not frequent => C4=null

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;
```

The Apriori algorithm for discovering frequent itemsets for mining Boolean association rules.

4.2.2 Generating association rules from frequent itemsets

- Generating strong association rules:

  For each frequent itemset "l", generate all nonempty subsets of l.

  Confidence(A=>B)=P(B|A)= support_count(AUB)/support_count(A)

  - support_count(AUB) – number of transactions containing the itemsets AUB
  - support_count(A) - number of transactions containing the itemsets A

- for every nonempty susbset s of l, output the rule s=>(l-s) if support_count(l)/support_count(s)>=min_conf

Example: lets have l={I1, I2, I5}

The nonempty subsets are {I1, I2}, {I1, I5}, {I2, I5}, {I1}, {I2}, {I5}.

Generating association rules:

I1 and I2=>I5	conf=2/4=50%
I1 and I5=>I2	conf=2/2=100%
I2 and I5=> I1	conf=2/2=100%
I1=>I2 and I5	conf=2/6=33%
I2=>I1 and I5	conf=2/7=29%
I5=>I1 and I2	conf=2/2=100%

If min_conf is 70%, then only the second, third and last rules above are output.

4.3. A Pattern-Growth approach for mining frequent item sets

Two Steps:

1. Scan the transaction DB for the **first time**, find frequent items (single item patterns) and order them into a list L in frequency descending order.

    - In the format of (item-name, support)

2. For each transaction, order its frequent items according to the order in L; **Scan DB the second time**, construct FP-tree by putting each frequency ordered transaction onto it.

- **FP-Tree Definition**
    - FP-tree is **a frequent pattern tree** . Formally, FP-tree is a tree structure defined below:
        1. One root labeled as “null", a set of *item prefix sub-trees* as the children of the root, and a *frequent-item header table*.
        2. Each node in *the item prefix sub-trees* has three fields:
            - item-name : register which item this node represents,
            - count, the number of transactions represented by the portion of the path reaching this node,
            - node-link that links to the next node in the FP-tree carrying the same item-name, or null if there is none.
        3. Each entry in the *frequent-item header table* has two fields,
            - item-name, and head of node-link that points to the first node in the FP-tree carrying the item-name.
- **Advantages of the FP-tree Structure**
    - The most significant advantage of the FP-tree

- Scan the DB only twice and twice only.

- Completeness:
  - the FP-tree contains all the information related to mining frequent patterns (given the min-support threshold). Why?
- Compactness:
  - The size of the tree is bounded by the occurrences of frequent items
  - The height of the tree is bounded by the maximum number of items in a transaction

- **Mining Frequent Patterns Using FP-tree**
  - General idea (divide-and-conquer)
    - Recursively grow frequent patterns using the FP-tree: looking for shorter ones

      Recursively and then concatenating the suffix:
    - For each frequent item, construct its conditional pattern base, and then its conditional FP-tree;
    - Repeat the process on each newly created conditional FP-tree until the resulting FP-tree is empty, or it contains only one path (single path will generate all the combinations of its sub-paths, each of which is a frequent pattern)
- **FP-Growth Method : An Example**

TID	List of items_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

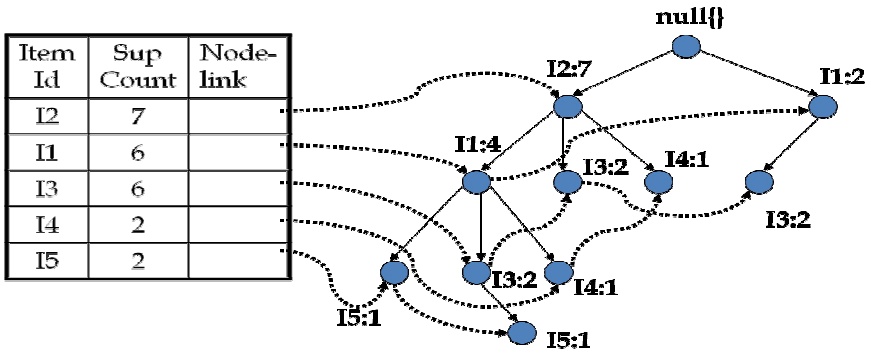
Consider the same previous example of a database, D , consisting of 9 transactions.

Suppose min. support count required is 2 (i.e. min_sup = 2/9 = 22 %)

The first scan of database is same as Apriori, which derives the set of 1-itemsets & their support counts. The set of frequent items is sorted in the order of descending support count.

The resulting set is denoted as L = {I2:7, I1:6, I3:6, I4:2, I5:2}

- **FP-Growth Method: Construction of FP-Tree**
  - First, create the root of the tree, labeled with “null”.
  - Scan the database D a second time. (First time we scanned it to create 1-itemset and then L).
  - The items in each transaction are processed in L order (i.e. sorted order).
  - A branch is created for each transaction with items having their support count separated by colon.
  - Whenever the same node is encountered in another transaction, we just increment the support count of the common node or Prefix.
  - To facilitate tree traversal, an item header table is built so that each item points to its occurrences in the tree via a chain of node-links.
  - Now, The problem of mining frequent patterns in database is transformed to that of mining the FP-Tree.



An FP-Tree that registers compressed, frequent pattern information

Mining the FP-Tree by Creating Conditional (sub) pattern bases

Steps:

1. Start from each frequent length-1 pattern (as an initial suffix pattern).
2. Construct its conditional pattern base which consists of the set of prefix paths in the FP-Tree co-occurring with suffix pattern.
3. Then, Construct its conditional FP-Tree & perform mining on such a tree.
4. The pattern growth is achieved by concatenation of the suffix pattern with the frequent patterns generated from a conditional FP-Tree.
5. The union of all frequent patterns (generated by step 4) gives the required frequent itemset.

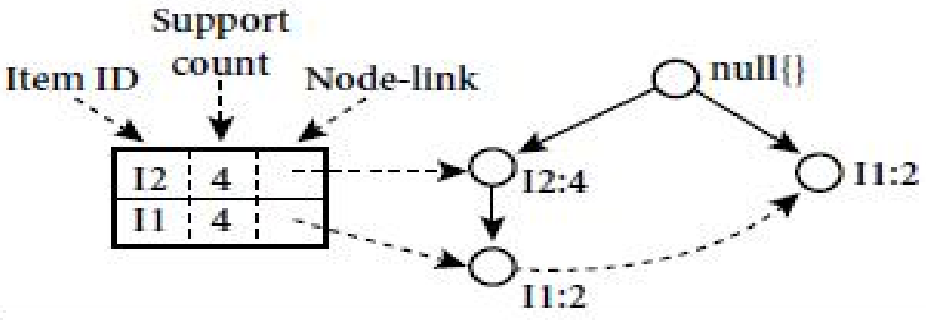
Item	Conditional pattern base	Conditional FP-Tree	Frequent patterns generated
I5	{(I2 I1: 1),(I2 I1 I3: 1)}	<I2:2 , I1:2>	I2 I5:2, I1 I5:2, I2 I1 I5: 2
I4	{(I2 I1: 1),(I2: 1)}	<I2: 2>	I2 I4: 2
I3	{(I2 I1: 1),(I2: 2), (I1: 2)}	<I2: 4, I1: 2>,<I1:2>	I2 I3:4, I1, I3: 2 , I2 I1 I3: 2

I2	{(I2: 4)}	<I2: 4>	I2 I1: 4

Mining the FP-Tree by creating conditional (sub) pattern bases

- Now, Following the above mentioned steps:
  - Lets start from I5. The I5 is involved in 2 branches namely {I2 I1 I5: 1} and {I2 I1 I3 I5: 1}.
  - Therefore considering I5 as suffix, its 2 corresponding prefix paths would be {I2 I1: 1} and {I2 I1 I3: 1}, which forms its conditional pattern base.

- **Properties of FP-Tree**
  - Node-link property: For any frequent item a_i, all the possible frequent patterns that contain a_i can be obtained by following a_i's node-links, starting from a_i's head in the FP-tree header.
  - Prefix path property : To calculate the frequent patterns for a node a_i in a path P, only the prefix sub-path of a_i in P need to be accumulated, and its frequency count should carry the same count as node a_i.



The conditional FP-tree associated with the conditional node I3.

- Out of these, Only I1 & I2 is selected in the conditional FP-Tree because I3 is not satisfying the minimum support count.
  - For I1 , support count in conditional pattern base = 1 + 1 = 2
  - For I2 , support count in conditional pattern base = 1 + 1 = 2
  - For I3, support count in conditional pattern base = 1

- Thus support count for I3 is less than required min_sup which is 2 here.

– Now , We have conditional FP-Tree with us.
– All frequent pattern corresponding to suffix I5 are generated by considering all possible combinations of I5 and conditional FP-Tree.
– The same procedure is applied to suffixes I4, I3 and I1.
– Note: I2 is not taken into consideration for suffix because it doesn't have any prefix at all.

Algorithm: FP_growth. Mine frequent itemsets using an FP-tree by pattern fragment growth.

Input:

- D, a transaction database;
- *min_sup*, the minimum support count threshold.

Output: The complete set of frequent patterns.

Method:

1. The FP-tree is constructed in the following steps:
   (a) Scan the transaction database D once. Collect F, the set of frequent items, and their support counts. Sort F in support count descending order as L, the *list* of frequent items.
   (b) Create the root of an FP-tree, and label it as "null." For each transaction *Trans* in D do the following. Select and sort the frequent items in *Trans* according to the order of L. Let the sorted frequent item list in *Trans* be $[p|P]$, where p is the first element and P is the remaining list. Call insert_tree($[p|P]$, T), which is performed as follows. If T has a child N such that $N.item\text{-}name = p.item\text{-}name$, then increment N's count by 1; else create a new node N, and let its count be 1, its parent link be linked to T, and its node-link to the nodes with the same *item-name* via the node-link structure. If P is nonempty, call insert_tree(P, N) recursively.
2. The FP-tree is mined by calling FP_growth(*FP_tree*, *null*), which is implemented as follows.

```
procedure FP_growth(Tree, α)
(1)  if Tree contains a single path P then
(2)     for each combination (denoted as β) of the nodes in the path P
(3)        generate pattern β∪α with support_count = minimum support count of nodes in β;
(4)  else for each a_i in the header of Tree {
(5)     generate pattern β = a_i ∪ α with support_count = a_i.support_count;
(6)     construct β's conditional pattern base and then β's conditional FP_tree Tree_β;
(7)     if Tree_β ≠ ∅ then
(8)        call FP_growth(Tree_β, β); }
```

The FP-growth algorithm for discovering frequent itemsets without candidate generation.

Reasons for the fastness of the Frequent Pattern Growth

– No candidate generation, no candidate test
– Use compact data structure
– Eliminate repeated database scan
– Basic operation is counting and FP-tree building

UNIT-IV
Assignment-Cum-Tutorial Questions
SECTION-A

Objective Questions

1. The market basket analysis is a typical example of _____
2. The interestingness mesures of Association rule mining are _____ and _____.
3. Association rules are considered interesting if they satisfy_____ [ ]
   A) Minimum support threshold. B) Minimum confidence threshold.
   C) Both A & B. D) Either A or B.
4. The formula for Support(A ⇒ B)=
5. The formula for Confidence (A ⇒ B)=
6. A association rule mining is a two step process which contains__[ ]
   A) Finding support and confidence.
   B) Finding all frequent itemsets.
   C) Generate strong association rules from the frequent itemsets.
   D) Both A & B. E. Both B & C.
7. All nonempty subset of a frequent itemset must also be frequent is _____ property.
8. Apriori method mines the frequent itemsets without candidate generation [T/F] [ ]
9. For the given transactional data find the Support(I1I2)=_____. [ ]

ID	ITEMS
1	I1,I2,I4
2	I2,I4,I5
3	I1,I2
4	I1,I2,I3
5	I1,I2,I5

A) 1 B) 2 C) 3 D) 4

10.How many number of scans were required in FP-Growth for finding frequent itemsets with 10 distinct items _____. [ ]

A) 1 B) 2 C) 3 D) 100

11. The rules which involves items at different levels of abstraction are

A) Multidimensional Association rules. B) Multilevel Association rules.

C) Rules interested at different levels. D) Predefined ruels. [ ]

12. For the given transactional database find the Support(AB)=_____

TID	A	B	C	D	E
1	1	1	1	0	1
2	0	1	0	1	1
3	1	1	1	0	1
4	0	1	0	1	0
5	1	1	0	1	1

A) 1 B) 2 C) 3 D) 4 [ ]

SECTION-B

SUBJECTIVE QUESTIONS

1.What is Association Rule Mining? Define Support and Confidence with example.

2. Generate frequent itemsets using the Apriori algorithm for the following data with the minimum support count 2.

TID	List of items_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

3.Explain how the association rules were generated from the frequent itemsets.

4.List and brief several methods to improve the efficiency of Apriori.

5.Find all frequent item sets using FP-Growth for the following data with min sup = 60% and *min conf* = 80%.

TID	ITEMS_BOUGHT
T100	{K,A,D,B}
T200	{D,A,C,E,B}
T300	{C,A,B,E}
T400	{B,A,D}

6. Design the node structure for representing the FP-Tree.