Data Mining

UNIT-III

Mining Frequent Patterns, Associations, and Correlations

B.Tech(CSE)-VI SEM

Course Outcomes

After Successful completion of the Course, the student will be able to:

CO1: Explain the concept of Data Mining and its functionalities (K2)

CO2: Discuss various Data Preprocessing Techniques (K2)

CO3: Demonstrate Association Analysis Techniques (K3)

CO4: Illustrate varions Classification Techniques (K3)

CO5: Demonstrate Alternative techniques for Classification (K3)

CO6: Use different Clustering techniques to cluster data (K3)

UNIT III: Mining Frequent Patterns, Associations, and Correlations:

- **Basic Concepts**
  - **Market Basket Analysis:** A Motivating Example
  - Frequent Itemsets, Closed Itemsets, and Association Rules
- **Frequent Itemset Mining Methods**
  - Apriori Algorithm: Finding Frequent Itemsets by Confined Candidate Generation
  - Generating Association Rules from Frequent Itemsets
  - Improving the Efficiency of Apriori
  - Pattern-Growth Approach for Mining Frequent Itemsets

Introduction

- ***Association rule mining*** searches for interesting associations and relationships among large sets of data items. This rule shows how frequently a itemset occurs in a transaction. A typical example is Market Based Analysis.

  The relationships can be represented in the form of

  {Milk} -> {Diaper}

- ***Applications:***
  - Purchasing Behavior
  - Marketing Promotion
  - Inventory Management
  - CRM etc…

Introduction (Contd…)

- **Frequent patterns** are patterns (e.g., itemsets, subsequences, or substructures) that appear frequently in a data set.

  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*.

Introduction (Contd…)

- A ***substructure*** can refer to different structural forms, such as subgraphs, subtrees, or sublattices, which may be combined with itemsets or subsequences.

- Finding frequent patterns plays an essential role in mining associations, correlations, and many other interesting relationships among data.

- Frequent patterns helps in data classification, clustering, and other data mining tasks.

Basic Concepts

- Frequent pattern mining searches for recurring relationships in a given data set .

  Example: Market Basket Analysis

- This example makes you to know concepts of frequent pattern mining for the discovery of interesting associations and correlations between itemsets in transactional and relational Databases.

- Frequent itemset mining leads to the discovery of associations and correlations among items in large transactional or relational data sets. Huge amounts of data is continuously being collected and stored, many industries are becoming interested in mining such patterns from their databases.

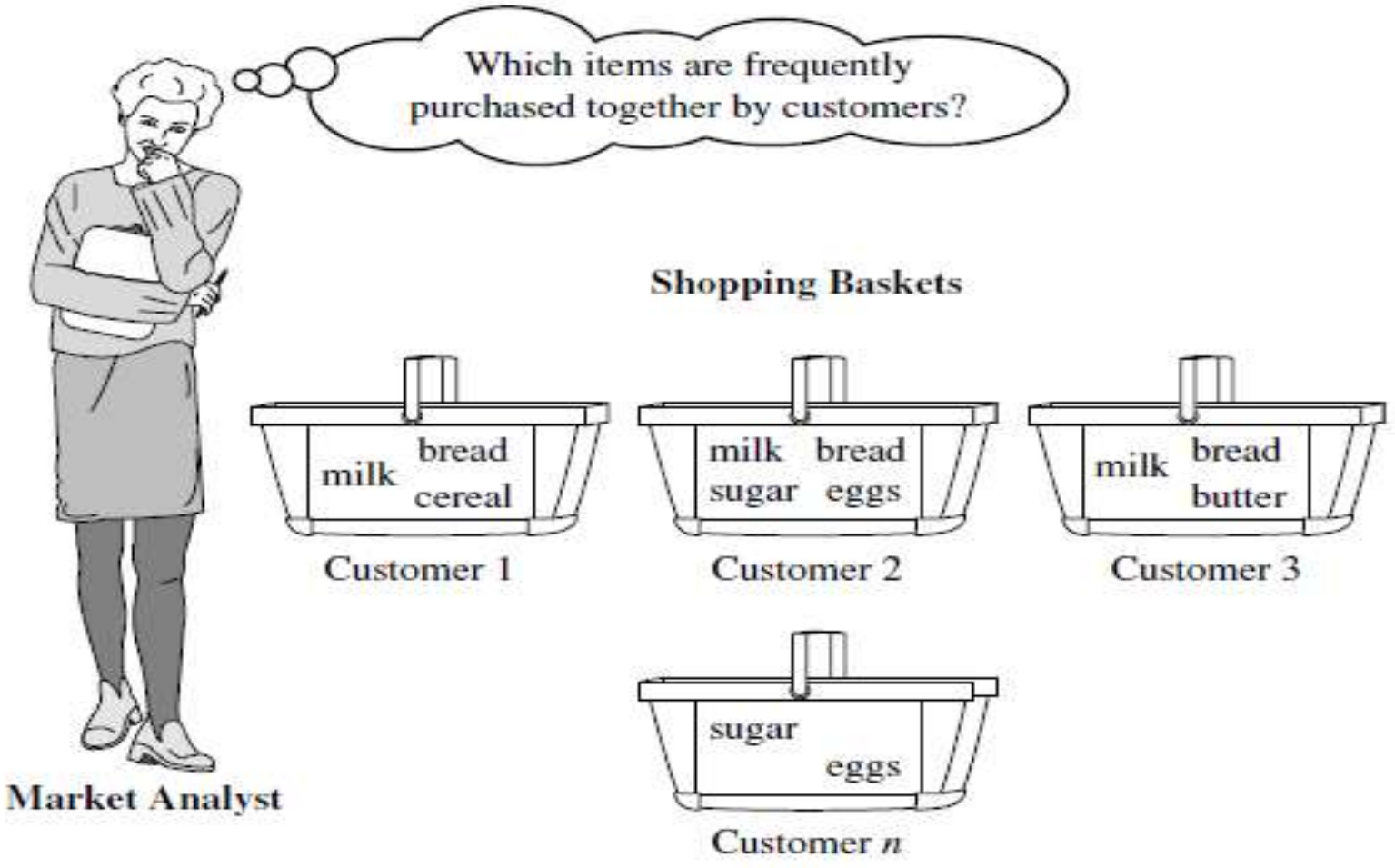
Basic Concepts (Contd…)

- The discovery of interesting correlation relationships among huge amounts of business transaction records can help in many business decision-making processes such as catalog design, cross-marketing, and customer shopping behavior analysis.

- **market basket analysis** process analyzes customer buying habits by finding associations between the different items that customers place in their "shopping baskets".

- The discovery of these associations can help retailers develop marketing strategies by gaining knowledge, which items are purchased together frequently by customers.

Basic Concepts (Contd…)



Basic Concepts (Contd…)

- Retailers can use the result by placing the items that are frequently purchased together in proximity to further encourage the combined sale of such items.

- In the above example(in the figure), Milk and bread is frequent, so it can be kept in proximity.

market basket analysis: A Motivating example

Example: Customers who purchase computers also tend to buy antivirus software at the same time is represented in the following association rule:

$$computer \Rightarrow antivirus\ software\ [support = 2\%, confidence = 60\%]$$

- Rule support and confidence are two measures of rule interestingness. They respectively reflect the usefulness and certainty of discovered rules.
- A support of 2% in the above Rule means that 2% of all the transactions under analysis show that computer and antivirus software are purchased together. A confidence of 60% means that 60% of the customers who purchased a computer also bought the software.
- Typically, association rules are considered interesting if they satisfy both a **minimum support threshold** and a **minimum confidence** threshold.

Basic Concepts: Frequent Itemsets, Closed Itemsets, and Association Rules

- Let I = {I1, I2,..., Im} be an itemset.

- Let D={ T1 , T2 , T3,.....,Tn } , a set of n transactions where each transaction Ti is non- empty item set such that $T \sqsubseteq I$.

- Each transaction is associated with an identifier, called a TID.

  [or]

  for each i, $Ti \neq \Phi$ and $Ti \sqsubseteq I$

Let A and B are set of items. [ex- A= { I1 , I3 , I7,I8 } and B= { I4 , I5 , I6 }]

An Association rule is an implication of the form

$$A \Longrightarrow B$$

where $A \subset I$, $B \subset I$, $A \neq \Phi$ and $B \neq \Phi$ & $A \cap B \neq \Phi$

The rule $A \Longrightarrow B$ holds in the transaction set D with Support s and Confidence c.

Basic Concepts: **Frequent Itemsets, Closed Itemsets, and Association Rules (Contd..)**

- **Support:**

  This is the percentage of transaction in D that contain AUB. Here AUB means every item in A and every item in B. Support is also written as P(AUB). It is also called Relative support. [Note: (AUB) ≠ A or B]

  Therefore,

  $$Support\ (A \Longrightarrow B) = support\ count\ (A \cup B)\ /N = \sigma(A \cup B)/N$$

- **Confidence:**

  This is the percentage of transactions in D containing A that also contain B. It is also written as P(B/A).

  $$Confidence(A \Longrightarrow B) = P(B/A).$$

  $$= \frac{support(A \cup B)}{support(A)}$$

  $$= \frac{support\ count\ (A \cup B)}{support\ count\ (A)}$$

Basic Concepts: **Frequent Itemsets, Closed Itemsets, and Association Rules (Contd..)**

- **Support count or Frequency**: Number of transactions that contain the item set. It is also called Absolute support.
- Rules that satisfy both a **minimum support threshold(min-sup)** and **minimum confidence threshold(min-conf)** are called **strong.**
- The support and confidence values occur between 0% and 100%, rather than 0 to 1.0

Basic Concepts: Frequent Itemsets, Closed Itemsets, and Association Rules (Contd..)

TID	Items
1	Bread, Milk
2	Bread, Cheese, Eggs,Sugar
3	Milk, Cheese, Sugar, Coke
4	Bread, Milk,Cheese, Sugar
5	Bread, Milk,Cheese,Coke

- **Itemset:**
  - In Association analysis, zero or more items called Itemset.
    - E.g.: {Milk, Bread, Cheese} -3 itemset
  - ***k-itemset***:
    - An itemset that contains k items.
- **Support count (σ)**
  - Frequency of occurrence of an itemset (number of transactions that contains a particuar itemset).
  - Mathematically Support count **σ(X)** for an **itemset X** can be stated as follows:

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

- **E.g.** σ({Milk, Bread,Cheese}) = 2
  - σ({Milk, Bread}) =

Basic Concepts: Frequent Itemsets, Closed Itemsets, and Association Rules (Contd..)

- **Support:**
  - Fraction of the transactions in which an itemset appears.
  - **E.g:**

    Rule: {Milk,Bread} -> {cheese}

TID	*Items*
1	Bread, Milk
2	Bread, Cheese, Eggs,Sugar
3	Milk, Cheese, Sugar, Coke
4	Bread, Milk,Cheese, Sugar
5	Bread, Milk,Cheese,Coke

σ({Milk, Bread, Cheese}) / N= 2/5

- **Frequent Itemset:**
  - An itemset whose support is greater than or equal to a ***minsup*** threshold

Basic Concepts: Frequent Itemsets, Closed Itemsets, and Association Rules *(contd…)*

TID	Items
1	Bread, Milk
2	Bread, Cheese, Eggs,Sugar
3	Milk, Cheese, Sugar, Coke
4	Bread, Milk,Cheese, Sugar
5	Bread, Milk,Cheese,Coke

Example: {Milk, cheese} => sugar

Suppor(s)= σ (Milk,cheese,sugar) / |T|

Suppor(s)= 2/5 = 0.4 = 40%

Confidence(c) = σ (Milk,cheese,sugar) / σ (Milk,cheese)

= 2/3 = 0.67 =67%

NOTE:

- **Support(s) : Fraction of transactions that contain both A and B**
- **Confidence(c): Measures how often items in B appear in transactions that contain A.**

Why use support & confidence:

Support: It is often used to eliminate uninteresting rules.

Confidence: It provides an estimate of the conditional probability of y given x.

Basic Concepts: Frequent Itemsets, Closed Itemsets, and Association Rules *(contd…)*

- ***Closed Frequent Itemset:*** It is a frequent itemset for which none of its immediate supersets have the same support count as itself.

- ***Maximal Frequent Itemset:*** It is a frequent itemset for which none of its immediate supersets are frequent..

Example:

TID	Items
T1	{A,B,C,D}
T2	{A,,D}
T3	{A,E}
T4	{C,E}

{A} - 3	{C,E} - 1
{B} - 1	{D,E} - 0
{C} - 2	{A,B,C} - 1
{D} - 2	{A,B,D} - 1
{E} - 2	{A,B,E} - 0
{A,B} - 1	{B,C,D} - 1
{A,C} - 1	{B,C,E} -0
{A,D} - 2	{C,D,E} - 0
{A,E} - 1	{A,B,C,D} -1
{B,C} - 1	{A,B,C,E} – 0
{B,D} - 0	{B,C,D,E} – 0
{B,E} - 0	{A,B,C,D,E} - 0
{C,D} -1	

2. {A},{C},{E},{A,D} are closed frequent itemsets.

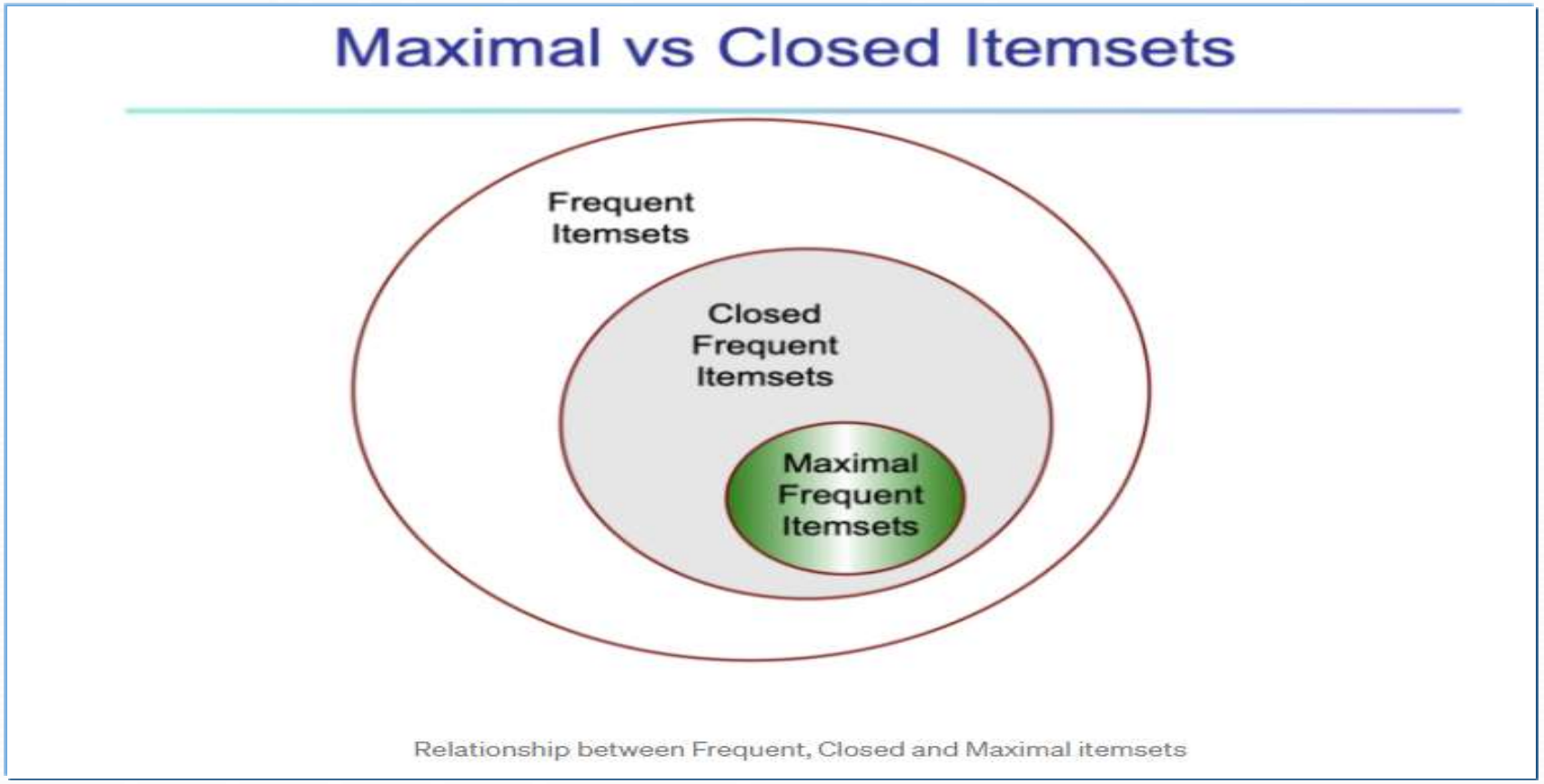
{C}, {E}, {A,D} are maximal frequent itemsets.

1. {A} is closed because none of its supersets have the same support as itself. But {A} is not maximal because {A,D} is a superset of {A} and its frequent.

3. Why D is not Closed, Because its immediate superset {A,D} also have the same support count.

Basic Concepts: Frequent Itemsets, Closed Itemsets, and Association Rules *(contd…)*

All Maximal Frequent Itemsets are Closed Frequent Itemsets. But , all Closed Frequent Itemsets are not Maximal Frequent Itemsets.



Basic Concepts: Frequent Itemsets, Closed Itemsets, and Association Rules *(contd…)*

In general, Association rule mining can be viewed as a two-step process:

1. **Find all frequent itemsets:** By definition, each of these itemsets will occur at least as frequently as a predetermined minimum support count, min sup.
    1. Apriori Method
    2. FP-Growth

2. **Generate strong association rules from the frequent itemsets:** By definition, these rules must satisfy minimum support and minimum confidence .

Frequent Itemset Mining Methods

- **Apriori Algorithm:** Finding Frequent Itemsets by Confined Candidate Generation
- Generating Association Rules from Frequent ItemsetsImproving the Efficiency of Apriori.
- Pattern-Growth Approach for Mining Frequent Itemsets

Frequent Itemset Mining Methods: Apriori Algorithm

- 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 iterates an iterative approach known as a *level-wise* search, where *k*-itemsets are used to explore (k+1)-itemsets

Apriori Algorithm

- 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 by 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.

- To improve the efficiency of the level-wise generation of frequent itemsets, an important property called the **Apriori property** is used to reduce the search space.

Apriori Algorithm

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

- 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 U *A*) cannot occur more frequently than *I*. Therefore, *I* [*A* is not frequent either, that is, ***P (I U A) < min sup*.**

- This property belongs to a special category of properties called **antimonotonicity** in the sense that *if a set cannot pass a test, all of its supersets will fail the same test as well.*

Frequent Itemset Mining Methods: Apriori Algorithm (Contd..)

The Apriori Algorithm : Pseudo code

Join Step: C k is generated by joining Lk-ıwith itself

Prune Step: Any (k-1)-itemset that is not frequent cannot be a subset of a frequent k-itemset

- Pseudo-code:

C_k: Candidate itemset of size k
L_k : frequent itemset of size k
L_1 = {frequent items};
for (k = 1; L_k != ∅; k++) do begin
C_{k+1} = candidates generated from L_k;
for each transaction t in database do
increment the count of all candidates in C_{k+1}

that are contained in t
L_{k+1} = candidates in C_{k+1} with min_support
End
return $\cup_k L_k$;

Example for Apriori Algorithm

- In a database D, there are 9 transactions. that is, $|D| = 9$.
- Consider the following dataset and for this we have to find frequent itemsets and also have to generate association rules for them.
- **Let** *min_sup=2*

TID	List of Itemset
T1	I1,I2,I5
T2	I2,I4
T3	I2, I3
T4	I1,I2,I4
T5	I1,I3
T6	I2,I3
T7	I1,I3
T8	I1,I2,I3,I5
T9	I1,I2,I3

Example problem on Apriori Algorithm

- Let us consider the transaction database D as shown in below.
- There are 9 Transactions in the database use Apriori Algorithm for finding frequent itemsets in D.
- Minimum support count =2.

Tid	List of Item id's
T_1	I_1, I_2, I_5
T_2	I_2, I_4
T_3	I_2, I_3
T_4	I_1, I_2, I_4
T_5	I_1, I_3
T_6	I_2, I_3
T_7	I_1, I_3
T_8	I_1, I_2, I_3, I_5
T_9	I_1, I_2, I_3

Scan D for count of each candidate

C_1

Items	Support Count
$\{I_1\}$	6
$\{I_2\}$	7
$\{I_3\}$	6
$\{I_4\}$	2
$\{I_5\}$	2

Compare candidate support count with minimum support count

L_1

Items	Support Count
$\{I_1\}$	6
$\{I_2\}$	7
$\{I_3\}$	6
$\{I_4\}$	2
$\{I_5\}$	2

Generate C_2 from L_1

C_2

Itemset	Support Count
$\{I_1, I_2\}$	4
$\{I_1, I_3\}$	4
$\{I_1, I_4\}$	1
$\{I_1, I_5\}$	2
$\{I_2, I_3\}$	4
$\{I_2, I_4\}$	2
$\{I2, I_5\}$	2
$\{I_3, I_4\}$	0
$\{I_3, I_5\}$	1
$\{I_4, I_5\}$	0

Generate C_3 from L_2

L_2

Items	Support Count
$\{I_1, I_2\}$	4
$\{I_1, I_3\}$	4
$\{I_1, I_5\}$	2
$\{I_2, I_3\}$	4
$\{I_2, I_4\}$	2
$\{I_2, I_5\}$	2

Compare candidate support count with minimum support count

C_3

Itemset	Support Count
$\{I_1, I_2, I_3\}$	2
$\{I_1, I_2, I_5\}$	2
$\{I_1, I_2, I_4\}$	1
$\{I_1, I_3, I_5\}$	1
$\{I_2, I_3, I_4\}$	0
$\{I_2, I_3, I_5\}$	1
$\{I_2, I_4, I_5\}$	0

Generate C_4 from L_3

L_3

Itemset	Support Count
$\{I_1, I_2, I_3\}$	2
$\{I_1, I_2, I_5\}$	2

C_4

Itemset	Support Count
$\{I_1, I_2, I_3, I_4\}$	0

L_4

ϕ

Note: Whenever $L_4=\varphi$
The algorithm terminated.
The itemsets in L_3 are frequent i.e.,

$\{I_1, I_2, I_3\}$
$\{I_1, I_2, I_5\}$

Generating Strong Association Rules from Frequent Itemsets

Generate strong association rules from itemsets (where strong association rules satisfy both minimum support and minimum confidence).

Association rules can be generated as follows:

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

- For every nonempty subset *s* of *l*, output the rule "*s=>(l-s)*" *if* *Support_count(l)* / *Support_count(s)* ≥ *min_conf*, Where min-conf is minimum confidence threshold.

$$confidence(A \Rightarrow B) = P(B|A) = support\_count(AUB) \; / \; support\_count(A) \; .$$

Generating Association Rules from Frequent Itemsets

Example: Suppose the data contain the frequent itemset l = {I1, I2, I5}. What are the Association rules that can be generated from l? If the minimum confidence threshold is 70%, then which rules are strong?

- {I1, I2}=>I5, *confidence* = 2/4= 50%
- {I1, I5}=>I2, *confidence* = 2/2 = 100%
- {I2, I5}=>I1, *confidence* = 2/2 = 100%
- I1=>{I2,I5}, *confidence* = 2/6= 33%
- I2=>{I1,I5}, *confidence* = 2/7 = 29%
- I5=>{I1,I2}, *confidence* = 2/2 =100%

TID	List of Itemset
T1	I1,I2,I5
T2	I2,I4
T3	I2, I3
T4	I1,I2,I4
T5	I1,I3
T6	I2,I3
T7	I1,I3
T8	I1,I2,I3,I5
T9	I1,I2,I3

•If the minimum confidence threshold is, say, 70%, then only the second, third, and last rules are output, because these are the only ones generated that are strong.

•Similarly rules can be generated for {I1,I2,I3}

Frequent Itemset Mining Methods: Apriori Algorithm (Contd..)

- Advantages:
  - Easy to understand algorithm
  - Join and Prune steps are easy to implement on large itemsets in large databases
- Disadvantages
  - It requires high computation if the itemsets are very large and the minimum support is kept very low.
  - The entire database needs to be scanned.

Frequent Itemset Mining Methods: Apriori Algorithm (Contd..)

Applications of Apriori Algorithm : Some fields where Apriori is used;

1. **In Education Field:** Extracting association rules in data mining of admitted students through characteristics and specialties.
2. **In the Medical field:** For example Analysis of the patient's database.
3. **In Forestry:** Analysis of probability and intensity of forest fire with the forest fire data.
4. Apriori is used by many companies like Amazon in the **Recommender System** and by Google for the auto-complete feature.

Conclusion

- Apriori algorithm is an efficient algorithm that scans the database only once.
- It reduces the size of the itemsets in the database considerably providing a good performance. Thus, data mining helps consumers and industries better in the decision-making process.

Improving the efficiency of Apriori Algorithms

The following are variations of Apriori Algorithm to improve its efficiency:

a) Hash based Technique – A k-itemset whose corresponding hashing bucket count is below the threshold cannot be frequent.

b) Transaction Reduction – A Transaction that does not contain any frequent k-item is useless in subsequent scans.

c) Partitioning – Any itemset that can potentially frequent in DB must be frequent in atleast one of the partions of DB.

d) Sampling – Mining on a subset of given data, lower support threshold + a method to determine the completeness.

e) Dynamic Itemset Counting – Add new candidate itemsets only when all of their subsets are estimated to be frequent.

Improving the efficiency of Apriori Algorithms

a) Hash Based Technique:

- Hash table is used as a data structure.
- A hash-based technique can be used to reduce the size of the candidate *k*-itemsets, *Ck*, for $k > 1$.
- First iteration is required to count support of each itemset.
- From second iteration, efforts are made to enhance execution of Apriori by utilizing hash table concept.
- Hash table mininmizes the number of itemset generated in second iteration.
- In second iteration i.e. 2-itemset generation, for every combination of two item, we map them into the diverse bucket of hash table structure and increment the bucket count.
- If count of bucket is less than min.sup. Count , we remove them from candidate sets.

TID	List of Items
T1	I1, I2, I5
T2	I2, I4
T3	I2, I3
T4	I1, I2, I4
T5	I1, I3
T6	I2, I3
T7	I1, I3
T8	I1, I2, I3, I5
T9	I1, I2, I3

Min. Support Count=3

C1

Itemset	Support Count
I1	6
I2	7
I3	6
I4	2
I5	2

Hash Function

Itemset	Count	Hash Function
I1, I2	4	[1*10+2] mod 7=**5**
I1, I3	4	[1*10+3] mod 7=**6**
I1, I4	1	[1*10+4] mod 7=**0**
I1, I5	2	[1*10+5] mod 7=**1**
I2, I3	4	[2*10+3] mod 7=**2**
I2, I4	2	[2*10+4] mod 7=**3**
I2, I5	2	[2*10+5] mod 7=**4**
I3, I4	0	--
I3, I5	1	[3*10+5] mod 7=**0**
I4, I5	0	--

Order of Items I1=1, I2=2, I3=3, I4=4, I5=5

H(x, y)=((Order of First)* 10+(Order of Second))mod 7

Hash Table Structure to generate L2

Bucket address	0	1	2	3	4	5	6
Bucket Count	2	2	4	2	2	4	4
Bucket Contents	{I1-I4}-1 {I3-I5}-1	{I1-I5}-2	{I2-I3}-4	{I2-I4}-2	{I2-I5}-2	{I1-I2}-4	{I1-I3}-4
L2	No	No	Yes	No	No	Yes	Yes

Advantages:

- Reduce the number of scans
- Remove the large candidates that cause high Input/output cost

Improving the efficiency of Apriori Algorithms(Contd….)

b) Transaction Reduction(reducing the number of transactions scanned in future iterations): A transaction that does not contain any frequent *k*-itemsets cannot contain any frequent (*k* +1)-itemsets. Therefore, such a transaction can be marked or removed from further consideration.

Trans.	Items
T1	I1, I2, I5
T2	I2, I3, I4
T3	I3, I4
T4	I1, I2, I3, I4

	I1	I2	I3	I4	I5
T1	1	1	0	0	1
T2	0	1	1	1	0
T3	0	0	1	1	0
T4	1	1	1	1	0

	I1	I2	I3	I4	I5
T1	1	1	0	0	1
T2	0	1	1	1	0
T3	0	0	1	1	0
T4	1	1	1	1	0

Minimum Support Count=2

	I1	I2	I3	I4
T1	1	1	0	0
T2	0	1	1	1
T3	0	0	1	1
T4	1	1	1	1

Trans.	Items
T1	I1, I2, I5
T2	I2, I3, I4
T3	I3, I4
T4	I1, I2, I3, I4

	I1,I2	I1,I3	I1,I4	I2,I3	I2,I4	I3,I4
T1	1	0	0	0	0	0
T2	0	0	0	1	1	1
T3	0	0	0	0	0	1
T4	1	1	1	1	1	1

	I1,I2	I2,I3	I2,I4	I3,I4
T2	0	1	1	1
T4	1	1	1	1

	I2,I3, I4
T2	1
T4	1

Improving the efficiency of Apriori Algorithms

c) Partitioning (partitioning the data to find candidate itemsets):

- A partitioning technique can be used that requires just two database scans to mine the frequent itemsets.

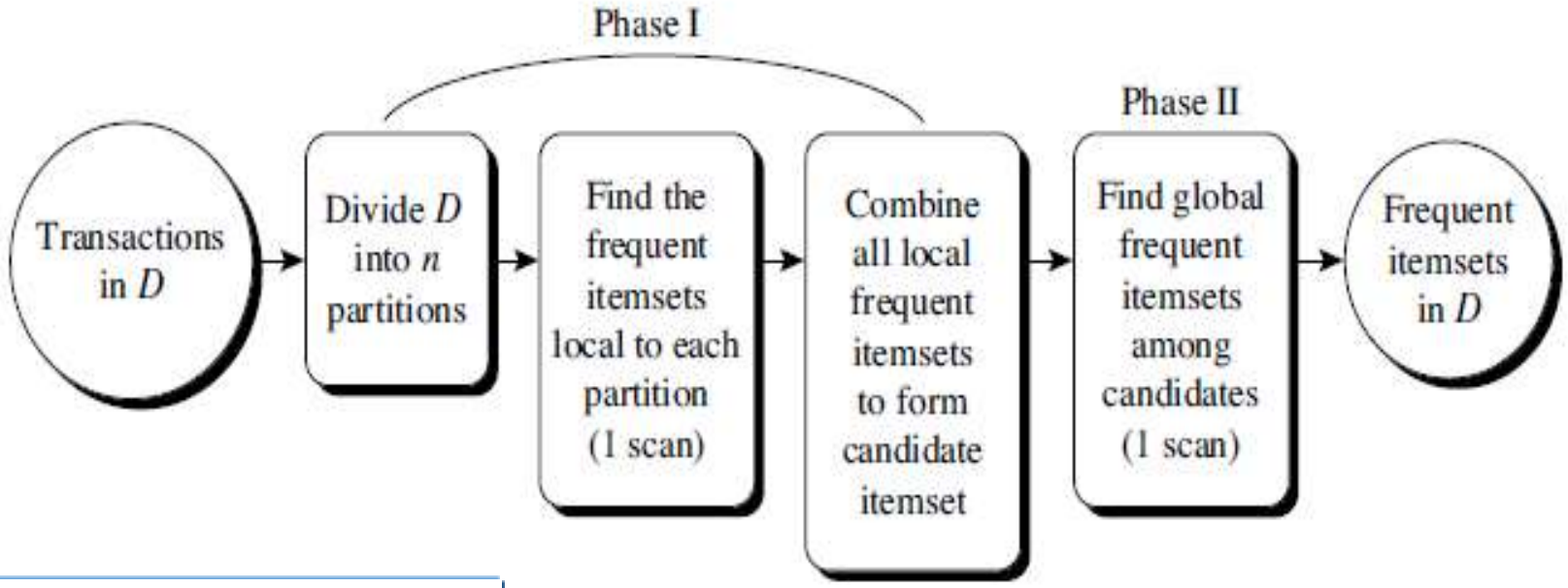
- It consists of two phases.

- **phase I**, the algorithm divides the transactions of *D* into *n* non overlapping partitions.

  If the minimum relative support threshold for transactions in *D* is *min sup*, then the minimum support count for a partition is *min_sup _ the number of transactions in that partition*.

- **Phase II :**

  A second scan of *Database D* is conducted in which the actual support of each candidate is assessed to determine the global frequent itemsets. Partition size and the number of partitions are set so that each partition can fit into main memory and therefore be read only once in each phase.



	I1	I2	I3	I4	I5
T1	1	0	0	0	1
T2	0	1	0	1	0
T3	0	0	0	1	1
T4	0	1	1	0	0
T5	0	0	0	0	1
T6	0	1	1	1	0

Database is divided into three partitions.

Each having two transactions with support of 20%

Trans.	Itemset	First Scan	Second Scan	Shortlisted
		Support =20% Min. sup=1	Support =20% Min. sup=2	
T1	I1, I5	I1-1, I2-1, I4-1, I5-1	I1-1, I2-3	I2-3, I3-2
T2	I2, I4	{I1,I5}-1 {I2, I4}-1	I3-2. I4-3	I4-3 ,I5-3
T3	I4, I5	I2-1, I3-1, I4-1, I5-1	I5-3, {I1, I5}-1	{I2, I4}-2
T4	I2, I3	{I4,I5}-1,{I2, I3}-1	{I2, I4}-2, {I4, I5}-1	{I2, I3}-2
T5	I5	I2-1, I3-1,I4-1, I5-1	{I2, I3}-2, {I3,I4}-1	
T6	I2, I3, I4	{I2, I3}-1, {I2,I4}-1 {I3, I4}-1 {I2, I3, I4}-1	{I2, I3, I4}-1	

Improving the efficiency of Apriori Algorithms

d) Sampling (mining on a subset of the given data):

- The basic idea is to pick up a random sample S of the given data D, and then search for frequent itemsets in S instead of D.
- Use a lower support threshold than minimum support to find local frequent items.
- Scan the database once to verify the frequent itemsets found in the sample.
- Only broader frequent itemsets are checked
  - Check abcd instead of ab, ac, …. Etc.
- Scan the database again to find missed frequent patterns.
- There is a trade off some degree of accuracy against efficiency.

Improving the efficiency of Apriori Algorithms

e) Dynamic itemset counting (adding candidate itemsets at different points during a scan):

- A dynamic itemset counting technique was proposed in which the database is partitioned into blocks marked by start points.
- Here, new candidate itemsets can be added at any start point, unlike in Apriori, which determines new candidate itemsets only immediately before each complete database scan.
- The technique uses the count-so-far as the lower bound of the actual count. If the count-so-far passes the minimum support, the itemset is added into the frequent itemset collection and can be used to generate longer candidates.
- This leads to fewer database scans than with Apriori for finding all the frequent itemsets.

Advantages of Apriori algorithm:

- This is the most simple and easy-to-understand algorithm among association rule learning algorithms
- The resulting rules are intuitive and easy to communicate to an end user
- It doesn't require labeled data as it is fully unsupervised; as a result, you can use it in many different situations because unlabeled data is often more accessible.

Disadvantages of Apriori algorithm:

- Requires many database scans
- Very slow

A Pattern-Growth Approach for Mining Frequent Itemsets (without candidate generation(FP-growth)

The Apriori candidate generate-and-test method significantly reduces the size of candidate sets, leading to good performance gain. However, it can suffer from two nontrivial costs:

1. ***It may still need to generate a huge number of candidate sets.*** For example, if there are 10^4 frequent 1-itemsets, the Apriori algorithm will need to generate more than 10^7 candidate 2-itemsets.

2. ***It may need to repeatedly scan the whole database and check a large set of candidates by pattern matching.*** It is costly to go over each transaction in the database to determine the support of the candidate itemsets.

A Pattern-Growth Approach for Mining Frequent Itemsets (without candidate generation(FP-growth)

The **Frequent Pattern growth** or **FP-Growth algorithm** is an alternate way to find frequent itemsets without using candidate generations, thus improving performance, for this it uses a ***divide-and-conquer*** strategy as follows:

1. First, compress the database representing frequent items into a **frequent-pattern tree, or FP-tree**, which retains the itemset association information.

2. Then divide the compressed database into a set of **conditional databases** (a special kind of projected database), each associated with one frequent item or "pattern fragment," and mines each such database separately.

Therefore, this approach may substantially reduce the size of the data sets to be searched, along with the "growth" of patterns being examined.

Frequent Pattern-Growth (FP-growth) Definition

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

- Each node in the ***item prefix subtree*** consists of three fields:
  - item-name
  - node-link
  - count
- Each entry in **the frequent-item header table** consists of two fields:
  - item-name
  - head of node-link

Frequent Pattern-Growth (FP-growth) Example

- In a database D, there are 9 transactions. that is, $|D| = 9$.
- Consider the following dataset:
- **Let** *min_sup=2*

TID	List of Itemset
T1	I1,I2,I5
T2	I2,I4
T3	I2, I3
T4	I1,I2,I4
T5	I1,I3
T6	I2,I3
T7	I1,I3
T8	I1,I2,I3,I5
T9	I1,I2,I3

Frequent Pattern-Growth (FP-growth) Example (Contd…)

An FP-tree is then constructed as follows:

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.

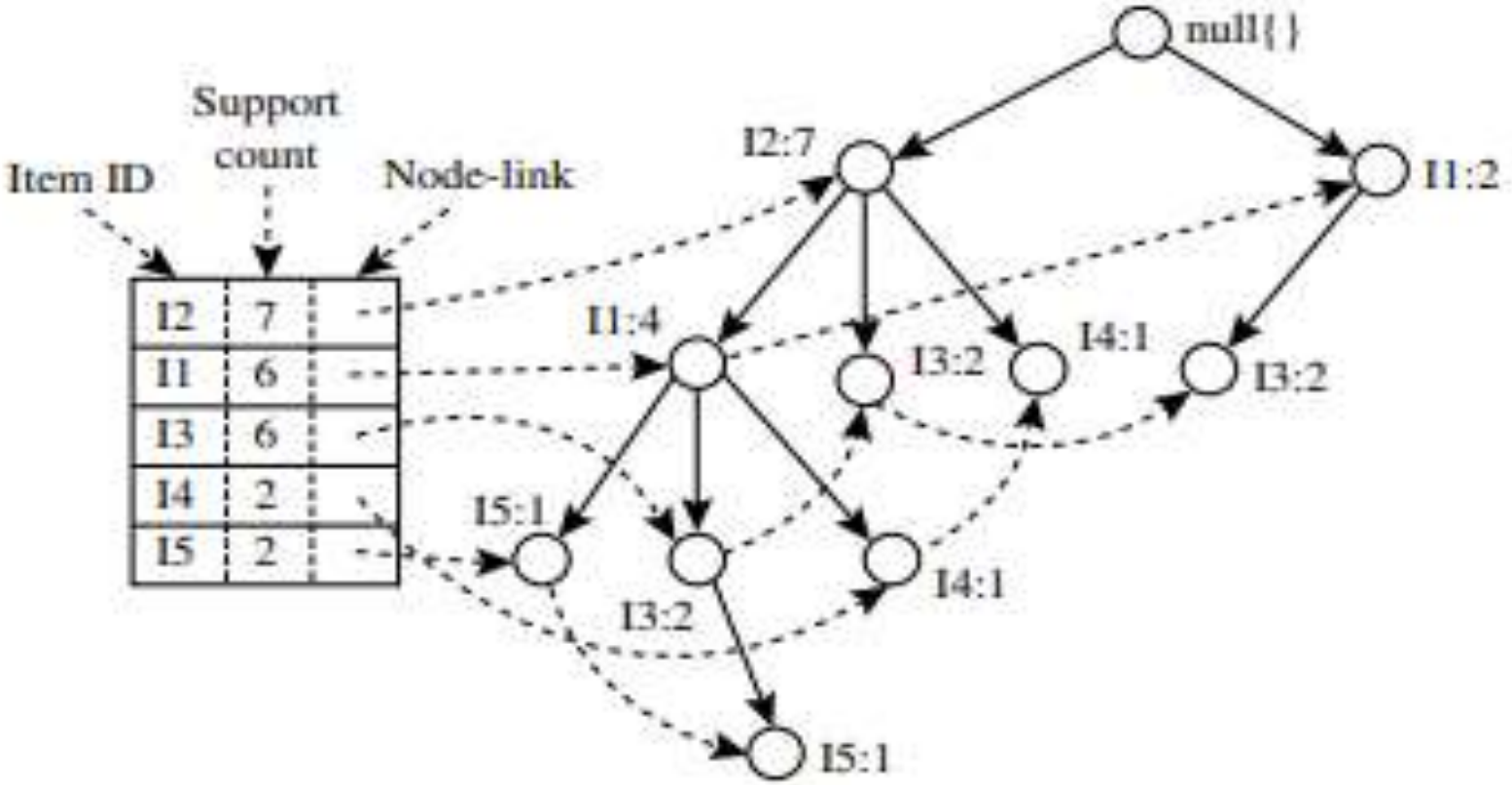
Items	Support Count
$\{I_1\}$	6
$\{I_2\}$	7
$\{I_3\}$	6
$\{I_4\}$	2
$\{I_5\}$	2

Example: L = {{I2: 7}, {I1: 6}, {I3: 6}, {I4: 2}, {I5: 2}}

In the format of (item-name, support)

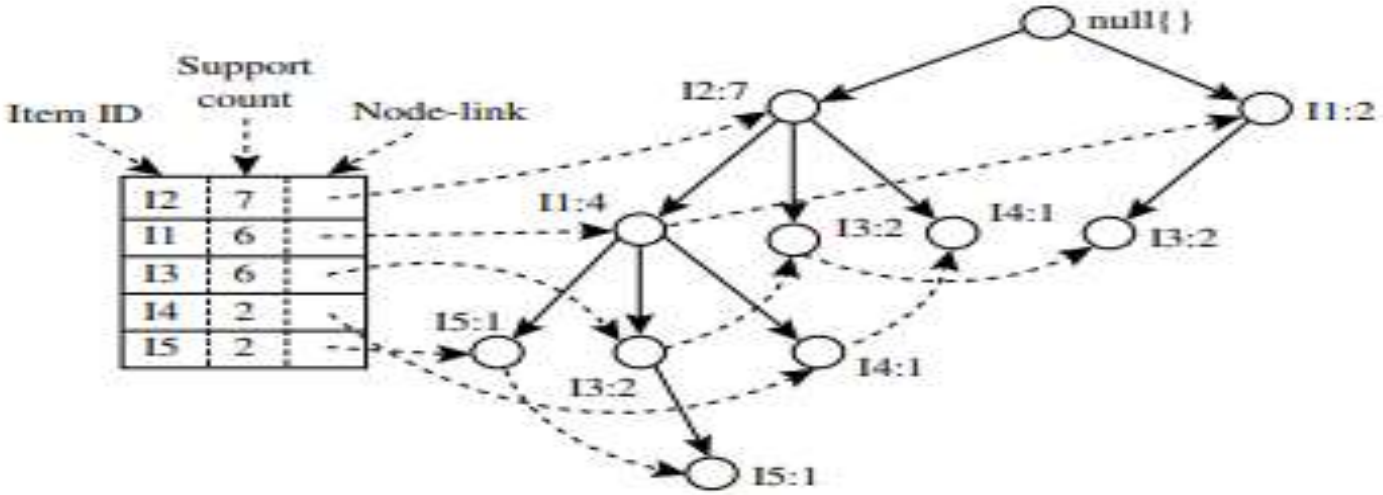
2. For each transaction, order its frequent items according to the order in L; Scan DB the second time, construct FP-tree by putting each *frequency ordered transaction onto* it. (repeat this step for all transactions)

Frequent Pattern-Growth (FP-growth) Example (Contd…)



An FP-tree registers compressed, frequent pattern information.

Frequent Pattern-Growth (FP-growth) Example (Contd…)

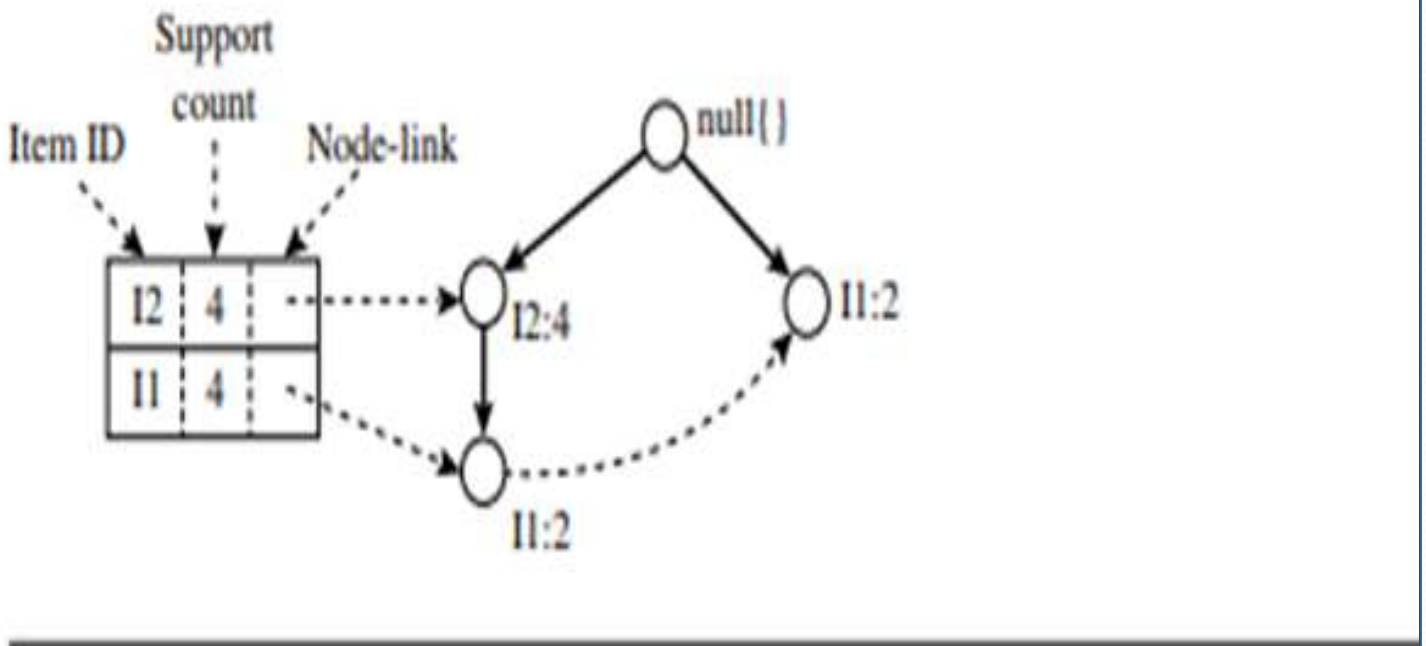


An FP-tree registers compressed, frequent pattern information.

Mining the FP-Tree by Creating Conditional (Sub-)Pattern Bases

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: 2}, {I2: 2}, {I1: 2}}	⟨I2: 4, I1: 2⟩, ⟨I1: 2⟩	{I2, I3: 4}, {I1, I3: 4}, {I2, I1, I3: 2}
I1	{{I2: 4}}	⟨I2: 4⟩	{I2, I1: 4}

Frequent Pattern-Growth (FP-growth) Example (Contd…)



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

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); }
```

A Pattern-Growth Approach for Mining Frequent Itemsets (without candidate generation(FP-growth)

- The FP-growth method transforms the problem of finding long frequent patterns into searching for shorter ones in much smaller conditional databases recursively and then concatenating the suffix.
- It uses the least frequent items as a suffix, offering good selectivity.
- The method substantially reduces the search costs.
- When the database is large, it is sometimes unrealistic to construct a main memory based FP-tree.

A Pattern-Growth Approach for Mining Frequent Itemsets (without candidate generation(FP-growth)

- An interesting alternative is to first partition the database into a set of projected databases, and then construct an FP-tree and mine it in each projected database. This process can be recursively applied to any projected database if its FP-tree still cannot fit in main memory.

- A study of the FP-growth method performance shows that it is efficient and scalable for mining both long and short frequent patterns, and is about an order of magnitude faster than the Apriori algorithm.

- **Reference:**

Data Mining Concepts and Techniques, Jiawei Han, Micheline Kamber, Jian Pei,3rd Edition, Morgan Kaufmann Publishers