THEORETICAL FOUNDATIONS OF MACHINE LEARNING (CSA-CC-521)

ASSOCIATION RULE MINING

M.Sc. Computer Science (With Specialization in Artificial Intelligence)

Dept. of Computer Science-University of Kerala
2023

Syllabus

Association Rule mining

- Concepts and Terminology,
- Apriori Algorithm,
- Probabilistic Correlation Algorithm,
- FP-growth Algorithm,
- Eclat Algorithm,
- Sparse Eclat,
- Tertius Algorithm,
- Treap Mining Algorithm

Association Rule Mining / Learning (ARM/ARL)

- ARM is a type of unsupervised learning method.
- Finds frequent patterns, associations, correlations, or causal structures among large sets of data items or objects in transaction databases, relational databases, and other information repositories.
- Involves the use of machine learning models to analyze the data for finding patterns for knowledge discovery.

Definition: Association Rule

- Association Rule
  - An implication expression of the form X → Y, where X and Y are itemsets
  - Contains two parts

    Antecedent, Left Hand Side (LHS), Body

    Consequent, Right Hand Side (RHS), Head

Example:

{Milk, Bread} → {Eggs}

{Coke, Fruits} → {Cereals, Oats}

T-ID	ITEMS
T1	Milk, Bread, Butter
T2	Bread, Butter, Eggs, Milk
T3	Milk, Bread, Eggs, Cereals, Fruits
T4	Milk, Coke, Bread, Butter
T5	Coke, Fruits, Cereals, Oats

Rule Evaluation Metrics:

The strength of an association rule is measured using

Support - Frequency of occurrence of an itemset in the rule

Confidence - Indicates the number of times the rule statements (if-then) are found true.

Lift - Ratio between Confidence & Support

Negative Correation

Positive Correalation

No Correlation

Use Cases:

Recommendation Systems, Cross Marketing, Catalog Design, Anomaly Detection, Web usage Mining etc…

Retail - Extract Purchasing Patterns to find which items are most likely to be purchased together. (Eg: Amazon, Flipkart…)

Entertainment - ML models analyze the past user behavior data for identifying frequent patterns and recommend contents accordingly. (Eg: Netflix, YouTube…)

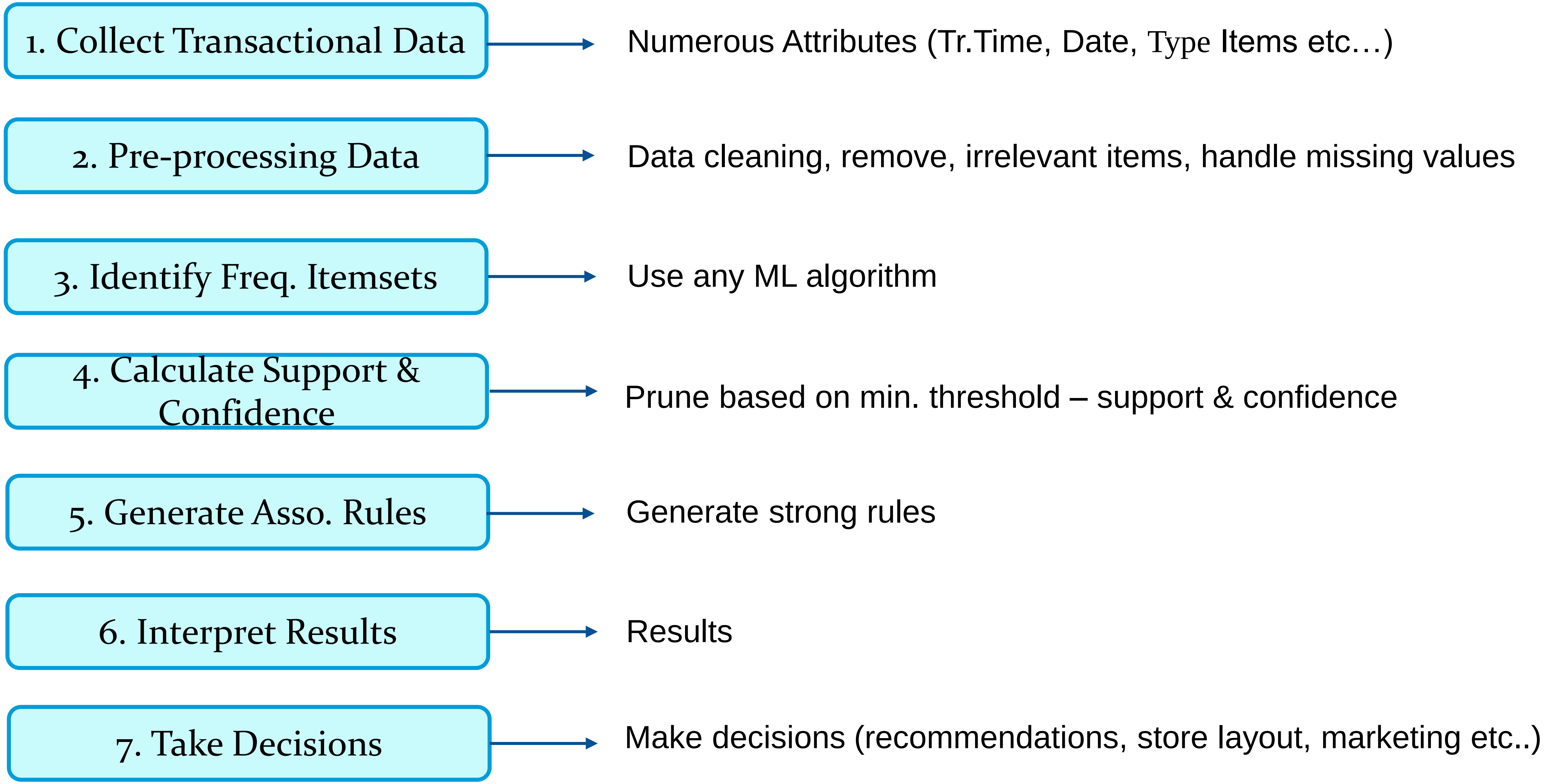
Medical - Help diagnose patients using the association rules-several diseases share common symptoms.

Fraud Detection - Companies and banks can identify fraudulent patterns, detect anomalies and prevent fraud.

Example of Association Rule: Market Basket Analysis

- Simple Data Mining Technique
- Given a set of transactions, find rules that will predict the occurrence of an item based on the occurrences of items in the transaction
- Involves analyzing large datasets (purchase history).
- Helps in:
  - ✓ Increase sales by understanding customer purchase patterns.
  - ✓ Selective Marketing - advertising intended to create demand for a specific brand rather than for the whole product category or class.
  - ✓ Cross Selling - suggesting products with a “customers also bought”
  - ✓ Optimal Product Placements - proper goods placement, proper analysis of customer preferences, promoting sales and satisfaction of customers

Market Basket Analysis: Steps



1. Collect Transactional Data → Numerous Attributes (Tr.Time, Date, Type Items etc…)

2. Pre-processing Data → Data cleaning, remove, irrelevant items, handle missing values

3. Identify Freq. Itemsets → Use any ML algorithm

4. Calculate Support & Confidence → Prune based on min. threshold – support & confidence

5. Generate Asso. Rules → Generate strong rules

6. Interpret Results → Results

7. Take Decisions → Make decisions (recommendations, store layout, marketing etc..)

Eg: Market Basket Analysis

T-ID	ITEMS
T1	Milk, Bread, Butter
T2	Milk, Bread, Butter, Eggs
T3	Milk, Bread, Eggs, Cereals, Fruits
T4	Milk, Coke, Bread, Butter
T5	Coke, Fruits, Cereals, Oats
T6	Milk, Fruits, Cereals, Eggs

Rule : {Milk, Bread} => {Eggs}

$$s = \frac{\text{freq}(\{\text{Milk, Bread, Eggs}\})}{N} = 2/6 = \mathbf{0.33}$$

$$c = \frac{\text{freq}(\{\text{Milk, Bread, Eggs}\})}{freq(Milk, Bread)} = 2/4 = \mathbf{0.5}$$

$$L = \frac{support}{support\ (LHS) X\ support\ (RHS)} = \frac{0.33}{\frac{4}{6} X \frac{3}{6}} = \mathbf{0.25}$$

Find:

LHS, RHS, Frequency, Support, Confidence, Lift

We need:

support, s ≥ min_sup threshold

confidence, c ≥ min_conf threshold

LHS = {Milk, Bread}

RHS = {Eggs}

N = Total no. of transactions

MBA

- ➢ Extract information on purchasing behavior

  (IF buys Milk and Bread, THEN also buy Eggs with high probability)

- ➢ Actionable information: can suggest...

  New store layouts and product collections
  Which products to put on promotion

- ➢ MBA approach is applicable
  - Credit cards
  - Services of telecommunication companies
  - Banking services
  - Medical treatments

- ➢ MBA Disadvantages
  - Computationally intensive, especially for large datasets
  - Does not provide information on why certain products are purchased together (Causality)

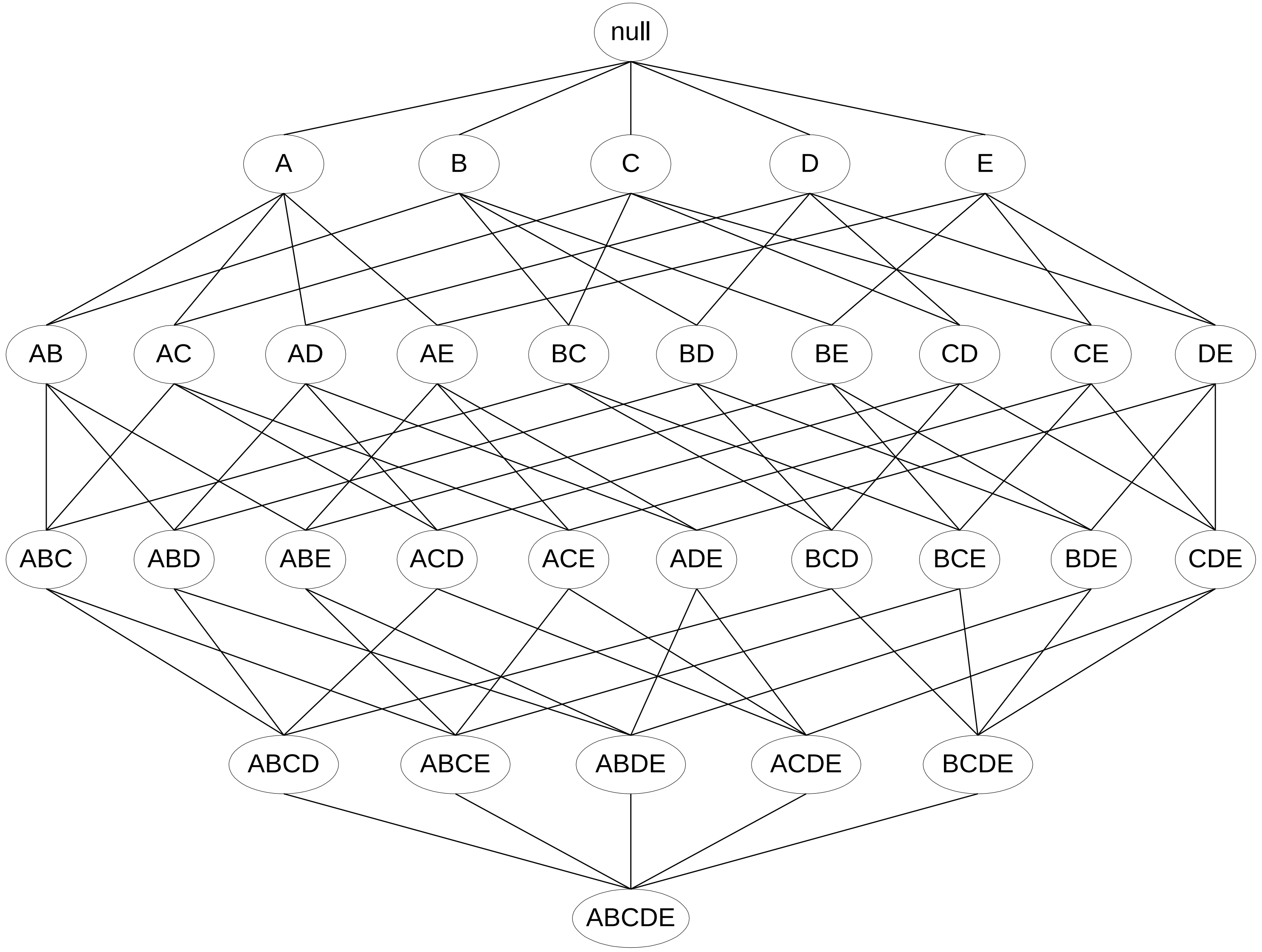
Algorithms for Association Rule Mining

1. Generating Association Rules and Apriori Algorithm

The general algorithm for generating association rules is a two-step approach:

1. Frequent Itemset Generation
   - Generate all possible itemsets whose support ≥ min_sup
2. Rule Generation
   - Generate high confidence rules from each frequent itemset, where each rule is having confidence ≥ min_conf

Frequent Itemset Generation:



Given d items, there are 2^d possible candidate itemsets

Complexity ~ O(NMw)

N - No. of transactions

M – No. of candidate itemsets

w – Attribute width in db

Reducing Association Rule Complexity

Two properties are used to reduce the search space for association rule generation.

- **Downward Closure**

  A subset of a large itemset must also be large

- **Anti-monotonicity**

  A superset of a small itemset is also small. This implies that the itemset does not have sufficient support to be considered for rule generation.

Frequent Itemset Generation Strategies

Reduce the number of candidates (M)

- Complete search: $M=2^d$
- Use pruning techniques to reduce M

Reduce the number of transactions (N)

- Reduce size of N as the size of itemset increases
- Use direct hashing and pruning (DHP) and vertical-based mining algorithms

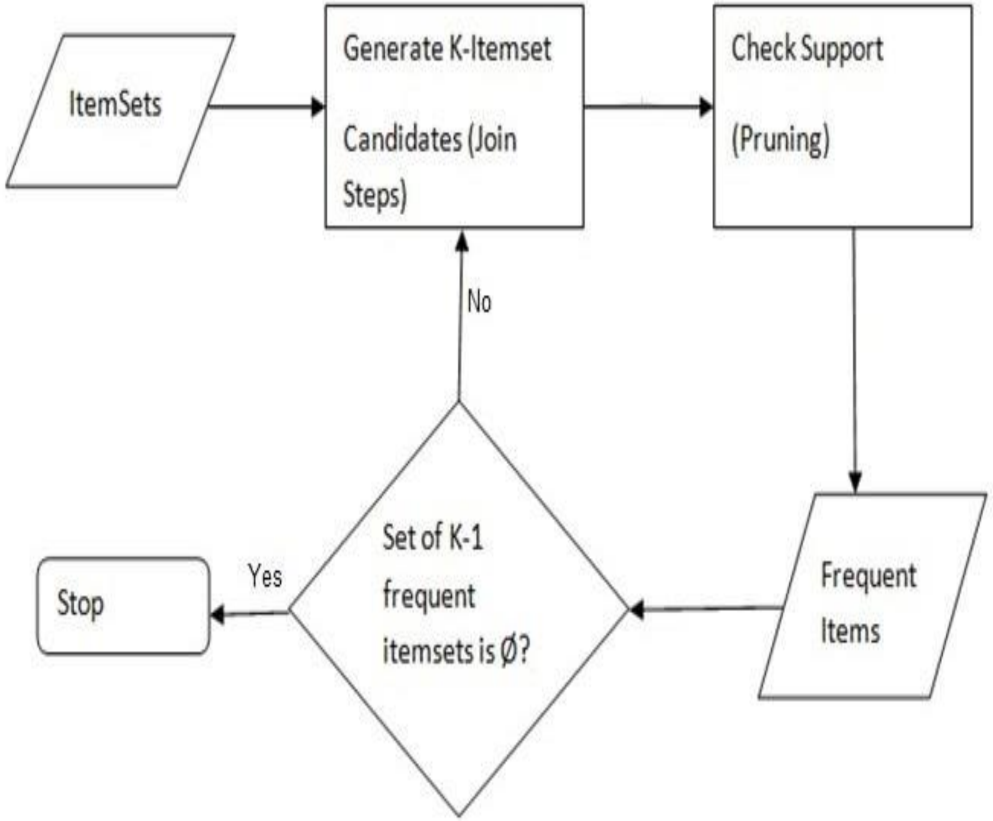
Reduce the number of comparisons (NM)

- Use efficient data structures to store the candidates or transactions
- No need to match every candidate against every transaction

Apriori Algorithm

- Let k=1
- Generate frequent itemsets of length 1
- Repeat until no new frequent itemsets are identified
  - 1. Generate candidate (k+1)-itemsets from frequent k-itemsets
  - 2. Prune candidate (k+1)-itemsets containing some infrequent k-itemset
  - 3. Count the support of each candidate by scanning the DB
  - 4. Eliminate infrequent candidates, leaving only those that are frequent

Apriori: Flowchart


ItemSets
Generate K-Itemset Candidates (Join Steps)
Check Support (Pruning)
Frequent Items
Set of K-1 frequent itemsets is Ø?
No
Yes
Stop

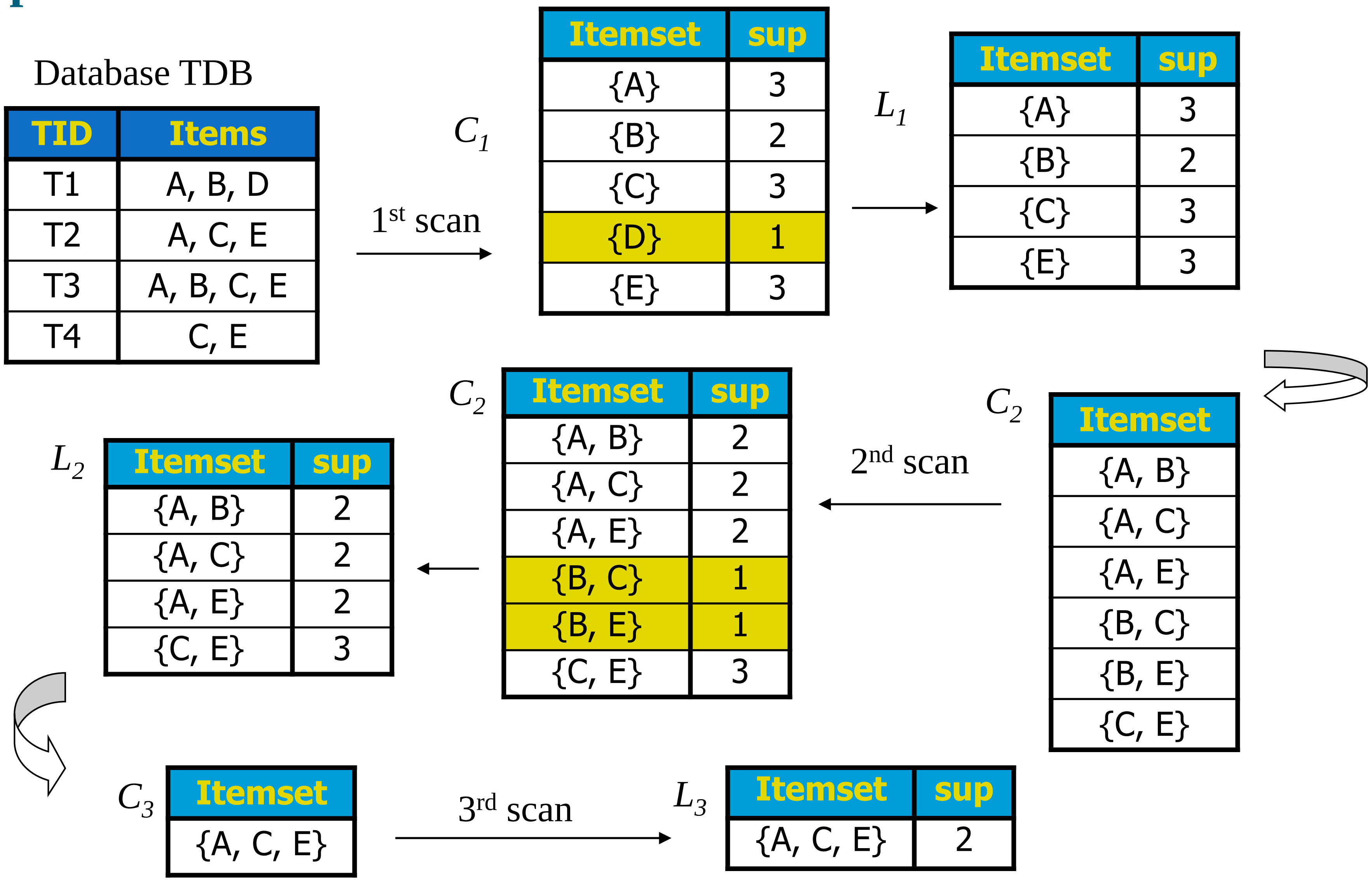
How to set the appropriate *min_sup* threshold?

- If *min_sup* is set too high, we could miss itemsets involving interesting rare items (e.g., expensive products)

- If *min_sup* is set too low, it is computationally expensive and the number of itemsets is very large

Example: Apriori

C **– candidate itemset,** ***L*** **– Frequent Itemset, Min_Sup = 2**



Database TDB

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

1st scan

C_1

Itemset	sup
{A}	3
{B}	2
{C}	3
{D}	1
{E}	3

L_1

Itemset	sup
{A}	3
{B}	2
{C}	3
{E}	3

C_2

Itemset
{A, B}
{A, C}
{A, E}
{B, C}
{B, E}
{C, E}

2nd scan

C_2

Itemset	sup
{A, B}	2
{A, C}	2
{A, E}	2
{B, C}	1
{B, E}	1
{C, E}	3

L_2

Itemset	sup
{A, B}	2
{A, C}	2
{A, E}	2
{C, E}	3

C_3

Itemset
{A, C, E}

3rd scan

L_3

Itemset	sup
{A, C, E}	2

Rule Generation

If {A,B,C,D} is a frequent itemset, candidate rules: $M=2^d$

ABC →D, ABD →C, ACD →B, BCD →A,

A →BCD, B →ACD, C →ABD, D →ABC

AB →CD, AC → BD, AD → BC, BC →AD,

BD →AC, CD →AB,

Ignoring the null set

Apriori: Limitations

- Apriori algorithm can be very slow and the bottleneck is candidate generation.
- The database need to be scanned at every level.
- It needs (*n* +*1*) scans, where *n* is the length of the longest pattern

Direct Hashing and Pruning

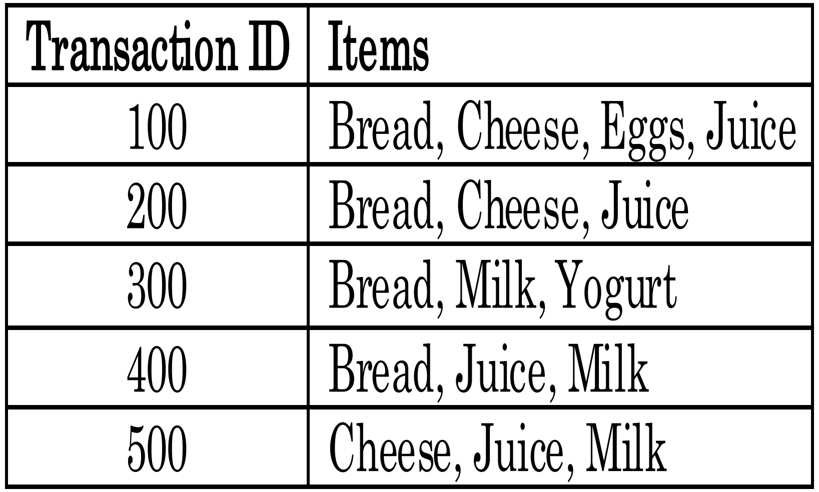
- DHP is very popular association rule mining technique to improve the performance of traditional Apriori algorithm.
- Uses hash function to filter unnecessary candidate itemset & reduce the size.
- Two basic hashing algorithms
  - ✓ Direct Hashing & Pruning (DHP),
  - ✓ Perfect Hashing &Pruning (PHP).
- Many algorithms have been also proposed by researchers like
  - ✓ Perfect Hashing Scheme (PHS),
  - ✓ Sorting-Indexing and Trimming (SIT) etc.

DHP: Working

When generating L_1, the DHP algorithm also generates all the 2-itemsets for each transaction, hashes them to a hash table and keeps a count.

Eg: Consider the transaction database in the first table below. The second table below shows all possible 2-itemsets for each transaction.

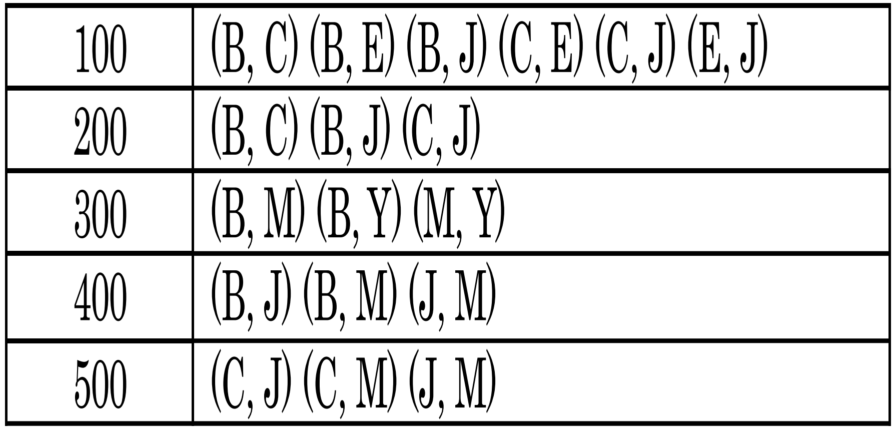
Find Min_Sup count = 50% = 2.5= **3**



Transaction ID	Items
100	Bread, Cheese, Eggs, Juice
200	Bread, Cheese, Juice
300	Bread, Milk, Yogurt
400	Bread, Juice, Milk
500	Cheese, Juice, Milk

Item	Count
B	4
C	3
E	1
J	4
M	3
Y	1

Transaction ID	Items
100	B,C, E, J
200	B,C, J
300	B, M, Y
400	B, J, M
500	C, J, M



100	(B, C) (B, E) (B, J) (C, E) (C, J) (E, J)
200	(B, C) (B, J) (C, J)
300	(B, M) (B, Y) (M, Y)
400	(B, J) (B, M) (J, M)
500	(C, J) (C, M) (J, M)

- For each pair, a numeric value is assigned by first representing

B - 1	J - 4
C - 2	M -5
E - 3	Y - 6

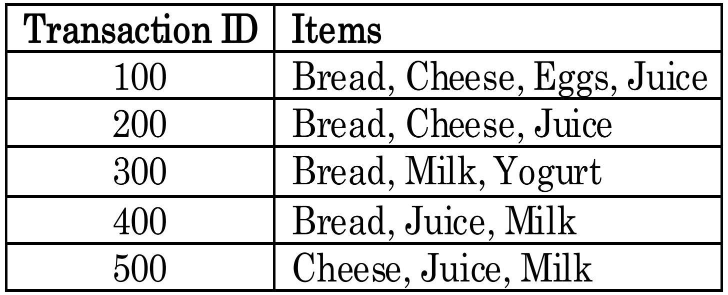
- Now each pair can be represented by a two digit number.
- The two digits are then coded as modulo 8 number (dividing by 8 and use the remainder).
- This is the bucket address.
- Those addresses that have a count above the support value have the bit vector set to 1 otherwise 0
- All pairs in rows that have zero bit are removed.

Item	H(X)
BC	12%8 = 4
BE	13%8 = 5
BJ	14%8 = 6
CE	23%8 = 7
CJ	24%8 = 0
EJ	34%8 = 2
BM	15%8 = 7
BY	16%8 = 0
MY	56%8 = 0
JM	45%8 = 5
CM	25%8 = 1

Bit Vector	Bucket No.	Count	Pairs	C2
1	0	3 + 1 + 1 =5	(C,J) (B,Y), (M,Y)	(C, J)
0	1	1	(C,M)	
0	2	1	(E,J)	
0	3	0		
0	4	2	(B,C)	
1	5	1+2 =3	(B,E) (J,M)	(J,M)
1	6	3	(B, J)	(B,J)
1	7	1+2 =3	(C,E) (B,M)	(B,M)

If count>=3 , Bit Vector =1

Next from table…



Transaction ID	Items
100	Bread, Cheese, Eggs, Juice
200	Bread, Cheese, Juice
300	Bread, Milk, Yogurt
400	Bread, Juice, Milk
500	Cheese, Juice, Milk

Item	S-Count
B	4
C	3
E	~~1~~
J	4
M	3
Y	~~1~~

Transaction ID	Items	Itemset
100	B,C, E, J	(B,J), (C,J)
200	B,C, J	(B,J), (C,J)
300	B, M, Y	(B,M)
400	B, J, M	(B,J), (B,M), (J,M)
500	C, J, M	(C,J), (J,M)

From *C2 Hash table*

C2	Support	L2
(C,J)	3	(C,J)
(J,M)	2	
(B,J)	3	(B,J)
(B,M)	2	

Vertical Apriori Algorithm

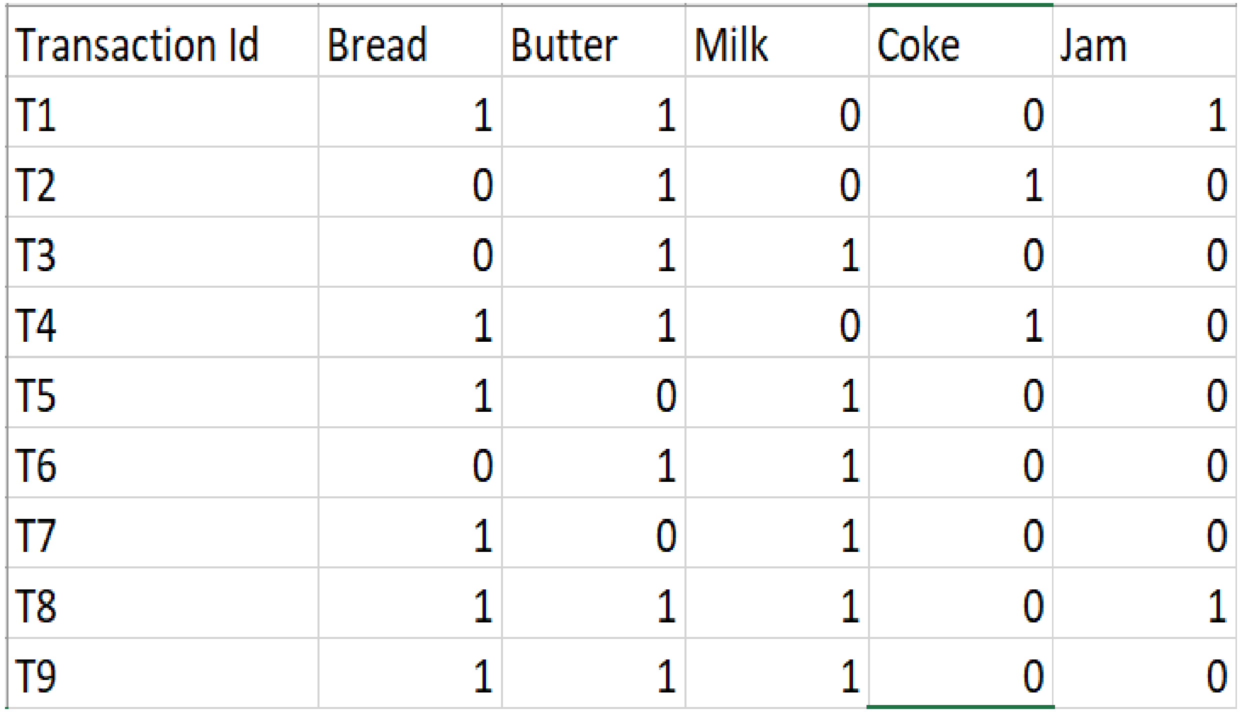
Eclat - Equivalence Class Clustering and bottom-up Lattice Traversal / Equivalence Class Transformation - (Zaki, 2000)

- Apriori algorithm use horizontal data format (Breadth-First Search of a graph)
- Eclat mines frequent itemsets using vertical data data format (Depth-First Search of a graph)
- It is a more efficient and scalable version of the Apriori algorithm
- Since the ECLAT algorithm uses a Depth-First Search approach, it uses less memory than Apriori algorithm
- Faster algorithm than the Apriori algorithm

- Eclat algorithm is used for frequent item set mining, aims at finding regularities in the shopping behavior of the customers
- The main operation of Eclat is intersecting tidsets
- The database is not required to be scanned multiple times in order to identify the (k + 1) itemsets while using Eclat.
- The size of tidsets is one of main factors affecting the running time and memory usage of Eclat
- The bigger tidsets are, the more time and memory are needed

Eg: Eclat Algorithm

Min_Sup = **2**



Transaction Id	Bread	Butter	Milk	Coke	Jam
T1	1	1	0	0	1
T2	0	1	0	1	0
T3	0	1	1	0	0
T4	1	1	0	1	0
T5	1	0	1	0	0
T6	0	1	1	0	0
T7	1	0	1	0	0
T8	1	1	1	0	1
T9	1	1	1	0	0

Horizontal Data Format

Iteration K=1 ⇨

Item	Tidset
Bread	{T1, T4, T5, T7, T8, T9}
Butter	{T1, T2, T3, T4, T6, T8, T9}
Milk	{T3, T5, T6, T7, T8, T9}
Coke	{T2, T4}
Jam	{T1, T8}

Vertical Data Format

K=2

Item	Tidset
{Bread, Butter}	{T1, T4, T8, T9}
{Bread, Milk}	{T5, T7, T8, T9}
{Bread, Coke}	{T4}
{Bread, Jam}	{T1, T8}
{Butter, Milk}	{T3, T6, T8, T9}
{Butter, Coke}	{T2, T4}
{Butter, Jam}	{T1, T8}
{Milk, Jam}	{T8}

K=3

Item	Tidset
{Bread, Butter, Milk}	{T8, T9}
{Bread, Butter, Jam}	{T1, T8}

K=4

Item	Tidset
{Bread, Butter, Milk, Jam}	{T8}

Items Bought	Recommended Products
Bread	Butter
Bread	Milk
Bread	Jam
Butter	Milk
Butter	Coke
Butter	Jam
{Bread, Butter}	Milk
{Bread, Butter}	Jam

Since minimum support = 2, we conclude the following rules from the given dataset

Eclat Algorithm

Step 1: Scan the database to create a vertical representation of the database.

Step 2: Create the first equivalence class wrt support count.

Step 3: Combine itemsets of the equivalence class to generate equivalence classes of size K+1

Step 4: For all other items, repeat the above steps.

Disadvantage: If the tidlist is too large, the Eclat algorithm may run out of memory

Sparse Eclat

- A convenient way to represent the transactions for the Eclat algorithm is a bit matrix, in which each row corresponds to an item, each column to a transaction (or the otherway round).
- A bit is set in this matrix if the item corresponding to the row is contained in the transaction corresponding to the column, otherwise it is cleared leading to a sparse matrix.
- Our focus is to verify on the vertical format, in particular its performance on sparse data.

Frequent Pattern Growth Algorithm (FP-Growth)

- 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.
- This tree structure will maintain the association between the itemsets.
- The database is fragmented using one frequent item.
- This fragmented part is called “pattern fragment”. The itemsets of these fragmented patterns are analyzed.
- Thus with this method, the search for frequent itemsets is reduced comparatively.

The algorithm consists of two steps:

Step 1 builds the FP-tree.

Step 2 uses the tree to find frequent itemsets.

- First, frequent 1-itemsets along with the count of transactions containing each item are computed.
- The 1-itemsets are sorted in non-increasing order.
- The root of the FP-tree is created with a "null" label.
- For each transaction T in the database, place the frequent 1-itemsets in T in sorted order. Designate T as consisting of a head and the remaining items, the tail.
- Insert itemset information recursively into the FP-tree as follows:
  - if the current node, N, of the FP-tree has a child with an item name = head, increment the count associated with N by 1 else create a new node, N, with a count of 1, link N to its parent and link N with the item header table.
  - if tail is nonempty, repeat the above step using only the tail, i.e., the old head is removed and the new head is the first item from the tail and the remaining items become the new tail.

FP-Growth - Example

1. List individual items
2. Write priorities

Transaction ID	Items
1	E,A,D,B
2	D,A,E,C,B
3	C,A,B,E
4	B,A,D
5	D
6	D,B
7	A,D,E
8	B,C

A,B,C,D,E

Itemset	Frequency	Priority
A	5	3
B	6	1
C	3	5
D	6	2
E	4	4

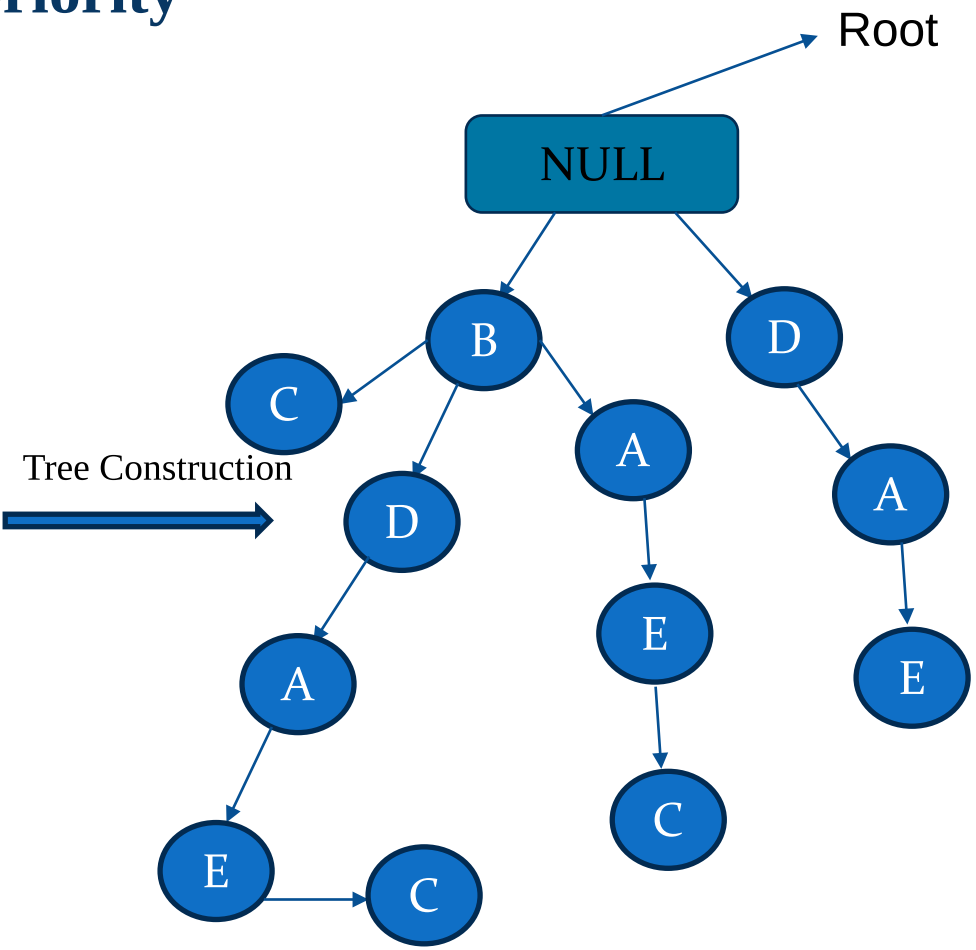
New priority list

B,D,A,E,C

More Frequent – More Priority
Same Frequency –First Come First Serve (FCFS)

Order items according to priority

Transaction ID	Items	Ordered items
1	E,A,D,B	B,D,A,E
2	D,A,E,C,B	B,D,A,E,C
3	C,A,B,E	B,A,E,C
4	B,A,D	B,D,A
5	D	D
6	D,B	B,D
7	A,D,E	D,A,E
8	B,C	B,C



No. of Occurrences

B-1,2,3,4,5,6
D-1,2,3,4
A-1,2,3
E-1,2
C-1
A-1
E-1
C-1
D-1,2
A-1
E-1
C-1

Stepwise - Example

Transaction ID	Items
T1	I1, I3, I4
T2	I2, I3, I5, I6
T3	I1, I2, I3, I5
T4	I2, I5
T5	I1, I3, I5

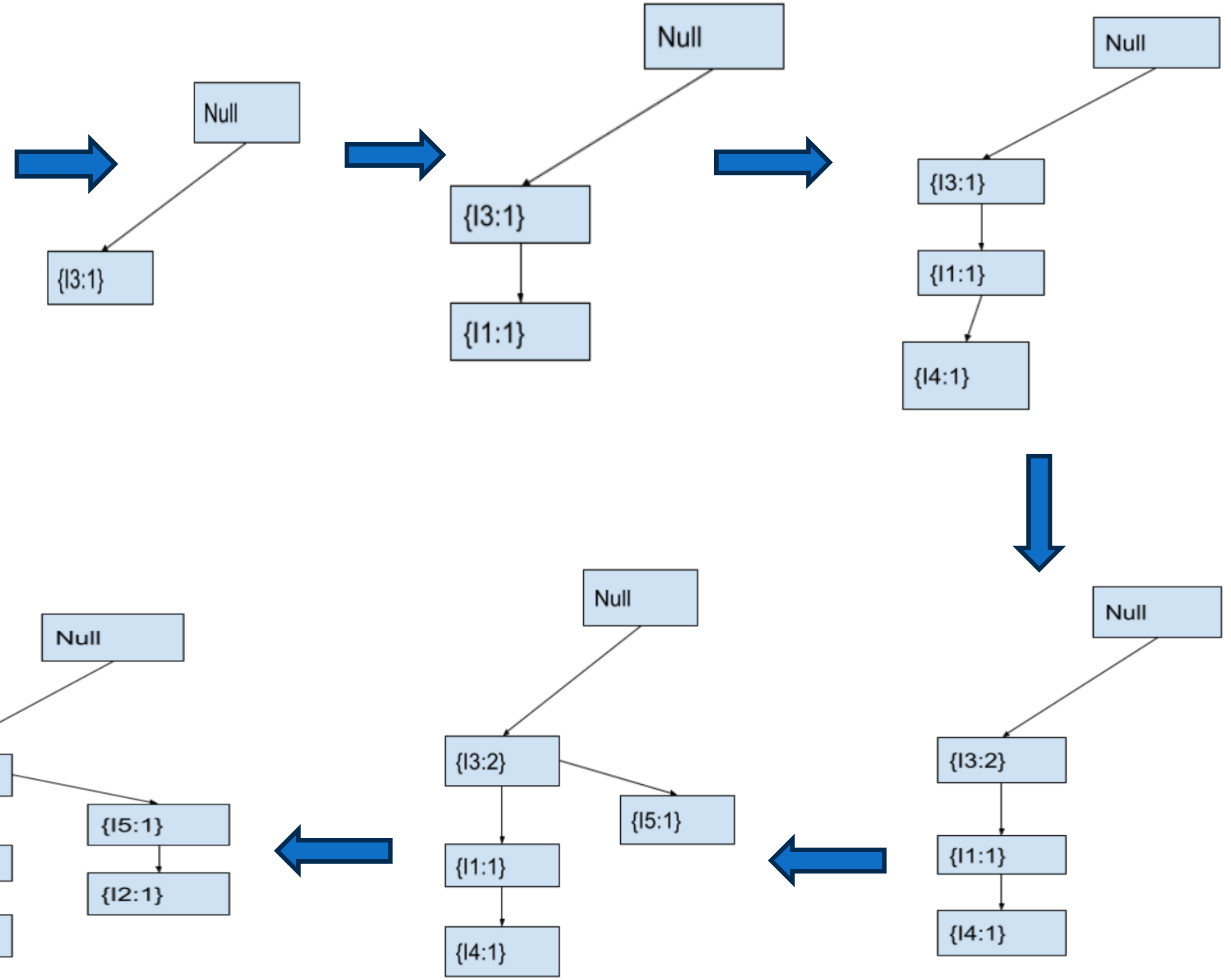
Item	Support Count
I1	3
I2	3
I3	4
I4	1
I5	4
I6	1

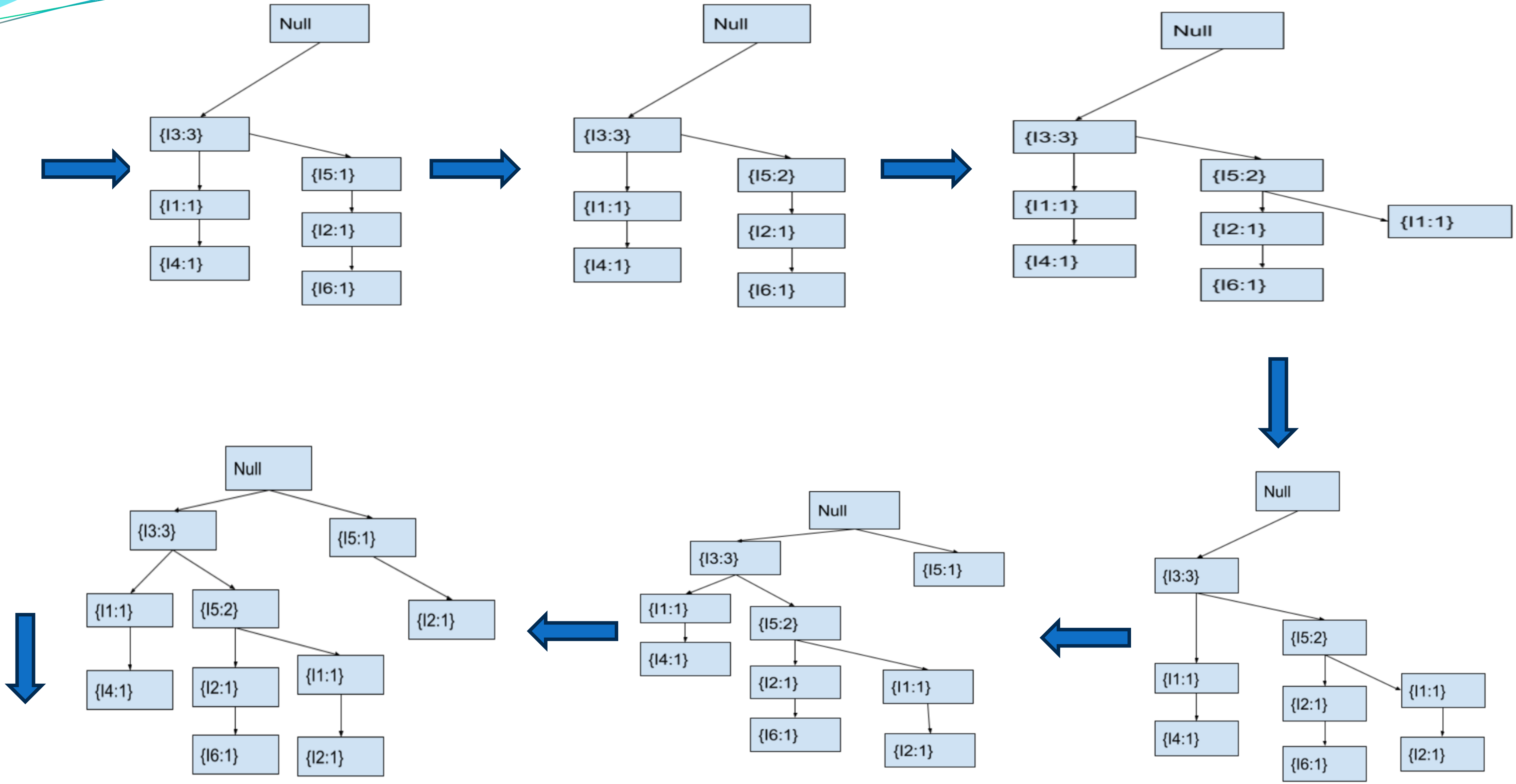
priority

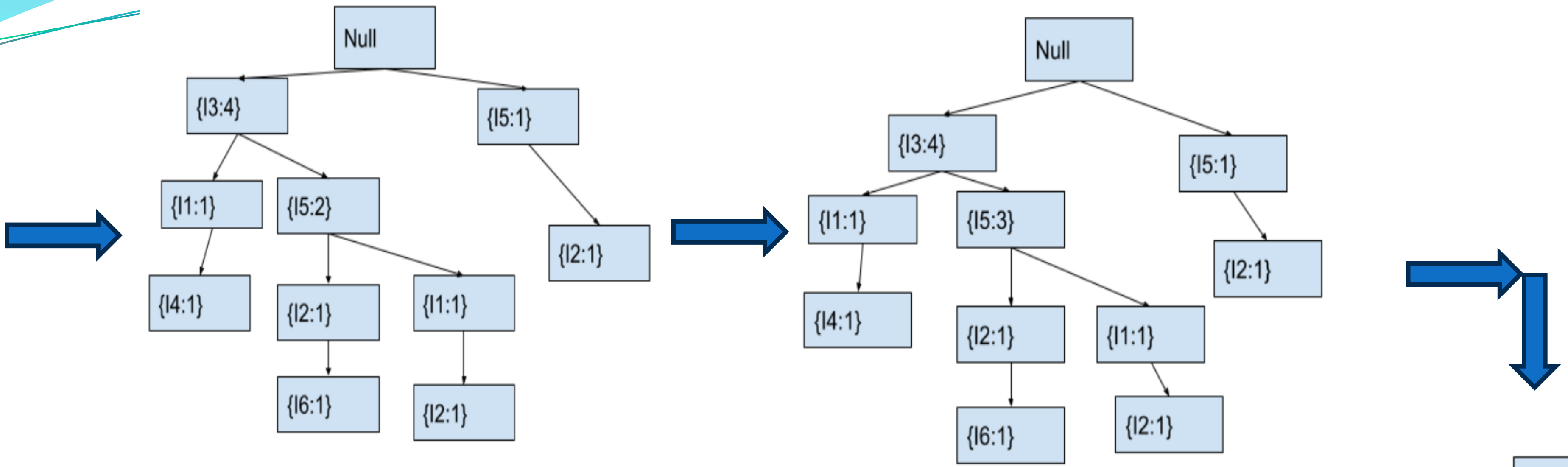
Item	Support Count
I3	4
I5	4
I1	3
I2	3
I4	1
I6	1

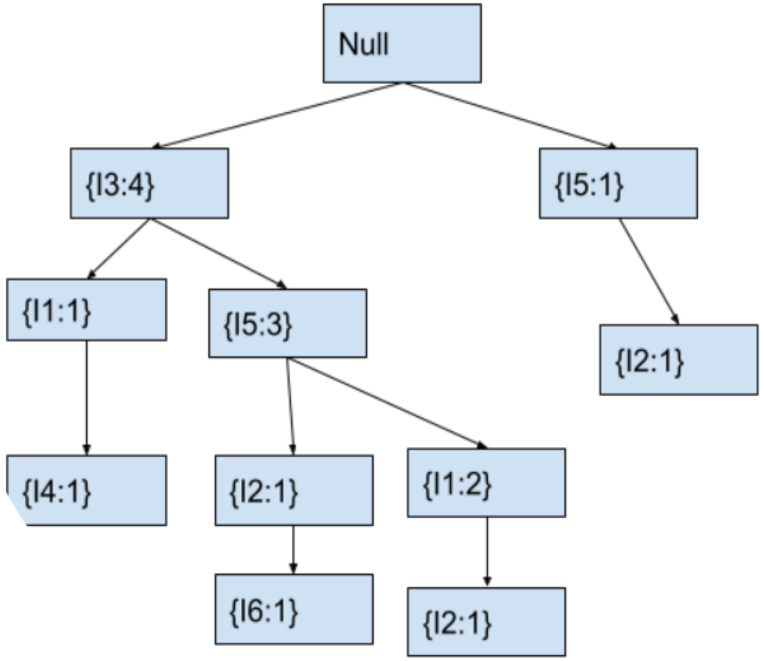
Ordered items wrt priority

Transaction ID	Items
T1	I3, I1, I4
T2	I3, I5, I2, I6
T3	I3, I5, I1, I2
T4	I5, I2
T5	I3, I5, I1




Null
{I3:3}
{I1:1}
{I4:1}
{I5:1}
{I2:1}
{I6:1}
Null
{I3:3}
{I1:1}
{I4:1}
{I5:2}
{I2:1}
{I6:1}
Null
{I3:3}
{I1:1}
{I4:1}
{I5:2}
{I2:1}
{I6:1}
{I1:1}
Null
{I3:3}
{I1:1}
{I4:1}
{I5:2}
{I2:1}
{I6:1}
{I1:1}
{I2:1}
Null
{I3:3}
{I5:1}
{I1:1}
{I4:1}
{I5:2}
{I2:1}
{I6:1}
{I1:1}
{I2:1}
Null
{I3:3}
{I5:1}
{I1:1}
{I4:1}
{I5:2}
{I2:1}
{I6:1}
{I1:1}
{I2:1}
{I2:1}





Final FP-Tree

Tertius Algorithm

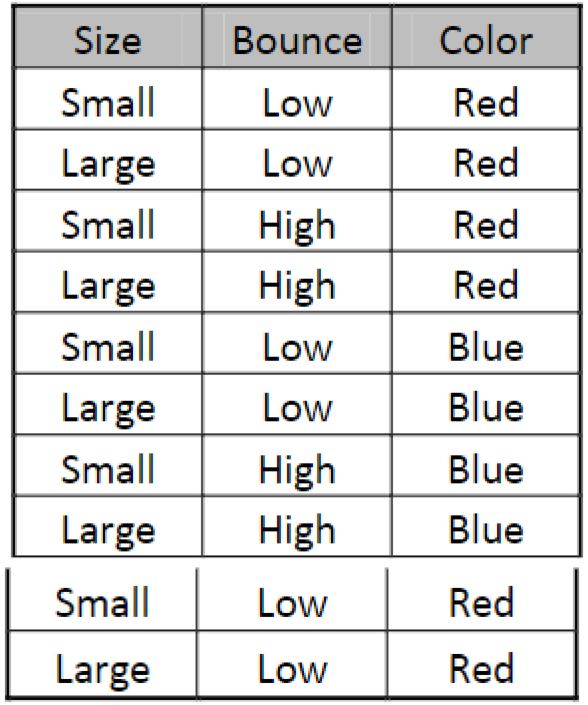
- Was designed to handle unsupervised learning using first-order representations for predictions (Eg: if x is a philosopher, then x is a scholar).
- Otherwise called as a first order logic discovery algorithm
- Tertius employs a complete top-down A* search over the space of possible rules
- Address both categorical and non-categorical confirmatory induction tasks
- The algorithm searches for the k most confirmed hypotheses; if a consistent set of hypotheses is requested, the additional requirement of the evidence forming a model of the hypothesis is enforced.
- The Tertius algorithm builds rules out of the attribute pair values in the training data and ranks them according to their reliability ,that is how many times the rule hold true in the training data.

Tertius Algorithm - Working

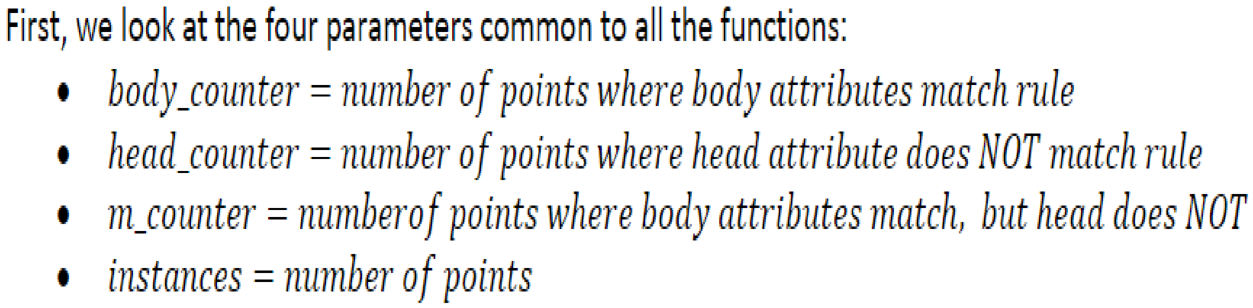
- A rule consists of two parts a body and a head.
- The body contains the conditions (which are known as literals) required for the rule to hold, and can consist of any number of literals.
- Literals can be combined into formulae by means of the usual logical connectives (negation ¬, conjunction ∧, disjunction ∨, implication → or ←, and equivalence ↔) Eg: H1 ∨ H2.
- The head contains the event that occurs when the rules hold true. blank head
- During rule learning, Tertius starts with an empty rule – means a blank body and a.
- The rule is then refined by adding attribute-value pairs in the order that they appear in the dataset.
- Once this completes, the algorithm counts the number of times the rule holds true (both body and head are true) and the times when the rule gives a false positive (when the body is true but the head is false) based on a confirmation measure.

Tertius Algorithm - Example

Table 1: Balls Dataset Used in Running Example.



Size	Bounce	Color
Small	Low	Red
Large	Low	Red
Small	High	Red
Large	High	Red
Small	Low	Blue
Large	Low	Blue
Small	High	Blue
Large	High	Blue
Small	Low	Red
Large	Low	Red



First, we look at the four parameters common to all the functions:

- $body\_counter = number\ of\ points\ where\ body\ attributes\ match\ rule$
- $head\_counter = number\ of\ points\ where\ head\ attribute\ does\ NOT\ match\ rule$
- $m\_counter = numberof\ points\ where\ body\ attributes\ match,\ \ but\ head\ does\ NOT$
- $instances = number\ of\ points$

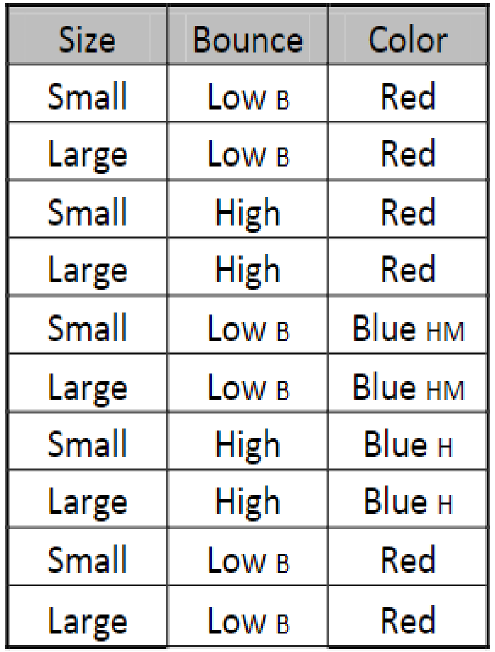
For a rule:

$$bounce = low \rightarrow color = red. \text{ (That is, } body \rightarrow head.\text{)}$$

The table gives the following parameter values for this rule:

- $body\_counter = 6$
- $head\_counter = 4$
- $m\_counter = 2$
- $instances = 10$

Table 2: Example of computing the four parameters for the rule: $bounce = low \rightarrow color = red$. The tiny B is for $body\_counter$, the tiny H is for $head\_counter$, and the tiny M is for $m\_counter$.



Size	Bounce	Color
Small	Low B	Red
Large	Low B	Red
Small	High	Red
Large	High	Red
Small	Low B	Blue HM
Large	Low B	Blue HM
Small	High	Blue H
Large	High	Blue H
Small	Low B	Red
Large	Low B	Red

Second, we consider the confirmation value used to measure the novelty of the association rules.

$expected = \frac{body\_counter * head\_counter}{instances^2}$ (roughly equal to the estimated number of counterexamples)

$observed = \frac{m\_counter}{instances}$ (roughly equal to the observed number of counterexamples)

$$confirmation = 0 \; if \; expected = 1 \; or \; 0$$

$$confirmation = \frac{expected - observed}{\sqrt{expected} - expected}$$

An example of computing the $expected$, $observed$, $confirmation$ on the Balls dataset is given in the bullets below for the rule: $bounce = low \rightarrow color = red$.

- $expected = 6 * 4/100 = 0.24$
- $observed = 2/10 = 0.2$
- $confirmation = \frac{0.24-0.2}{\sqrt{0.24}-0.24} = 0.16$

Third we consider the true positive and false positive rates given as the output of Tertius algorithm.

$$true\text{-}positive = \frac{body\_counter - m\_counter}{instances - head\_counter}$$

$$false\text{-}positive = \frac{m\_counter}{head\_counter}$$

Hence, for the rule: $bounce = low \rightarrow color = red.$

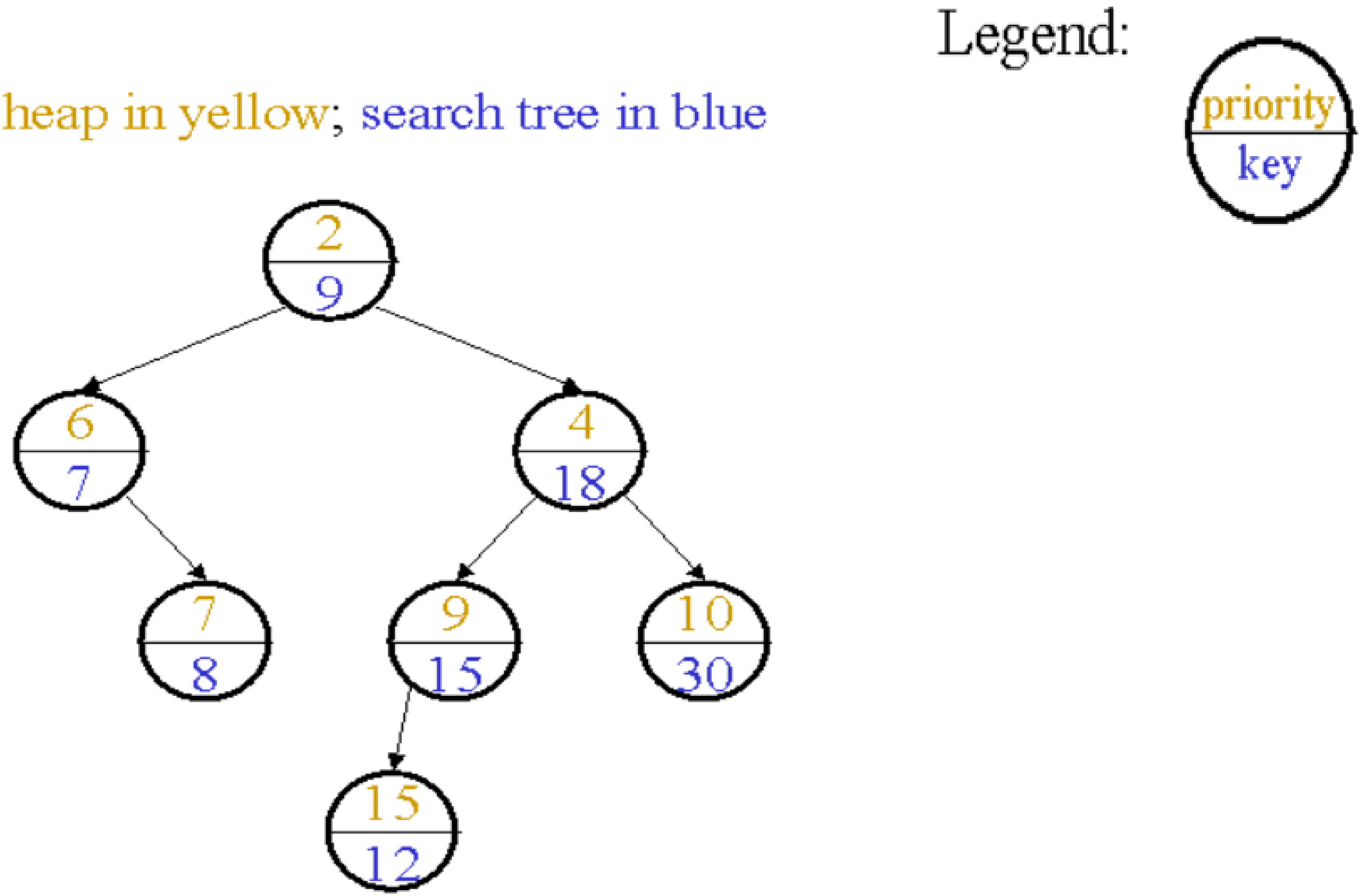
- $true\text{-}positive = \frac{6-2}{10-4} = 0.67$
- $false\text{-}positive = 2/4 = 0.5$

Treap Mining Algorithm

- ➢ A treap is a data structure which combines binary tree and binary heap (hence the name: tree + heap ⇒ Treap).
  - ✓ In a binary search tree, for each node, all items' values in the left subtree are less than the node's value, and all items in the right subtree are greater
  - ✓ A binary heap is a binary tree where each node child's value is less than the node's value
- ➢ It conforms to a core search binary tree property and binary heap property at the same time
- ➢ Used for finding interesting relations in a database and works in a priority-based model

- Treap organize the data wrt priority and helps to mine high priority itemsets
- A node in a Treap is like a node in a Binary Search Tree (BST)
- In BST, it has a data value, x, but in treap it has a unique numerical priority, p, in addition to, x
- The nodes in a Treap also obey the heap property; that is, at every node u, except the root, u.parent.p < u.p.
- Treap structure can be classified as Mini Treap (minimum priority) and Max Treap (maximum priority).
- The mining task begins by calculating the priority of each variable in the database.
- After calculating the priority the Treap data structure is built and the frequent variables are identified by having a depth first Treap traversal.

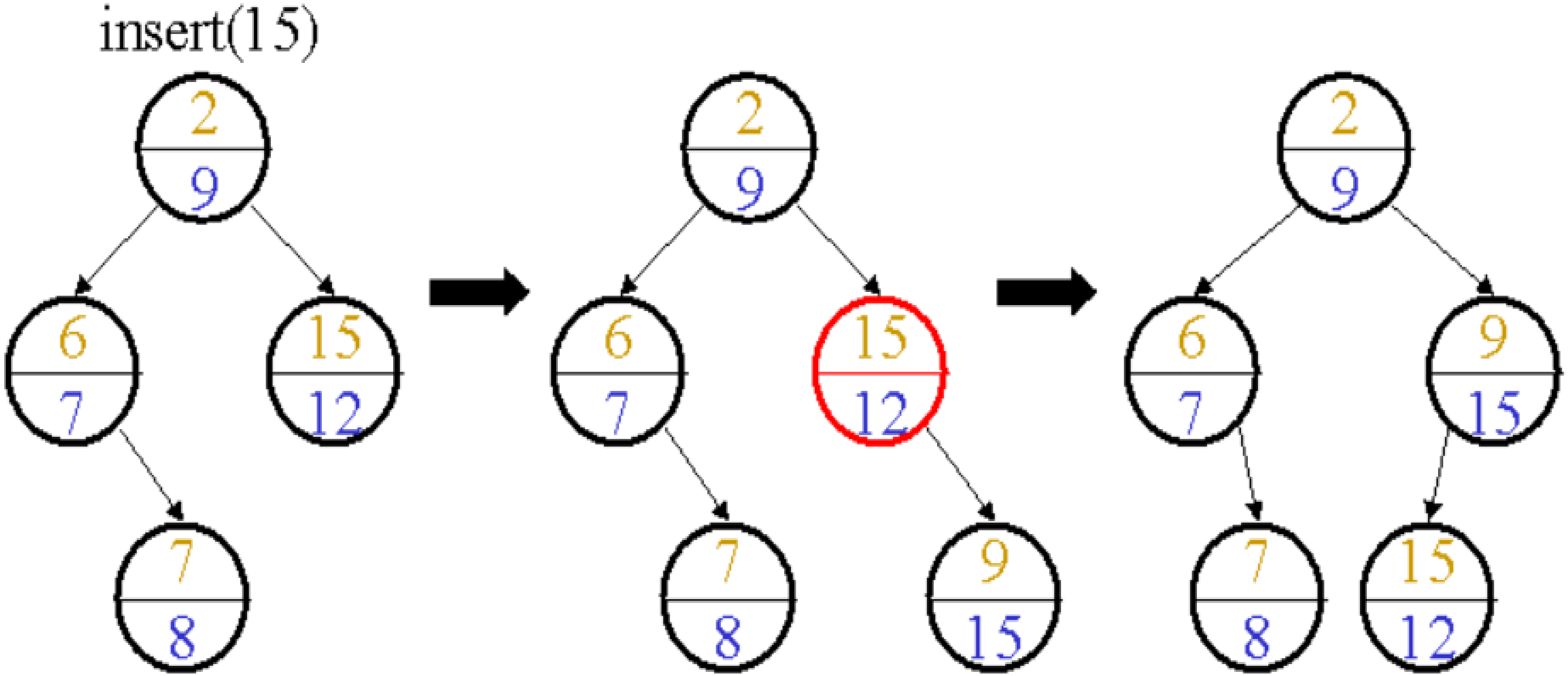
Example: Treap Mining


Legend:

heap in yellow; search tree in blue

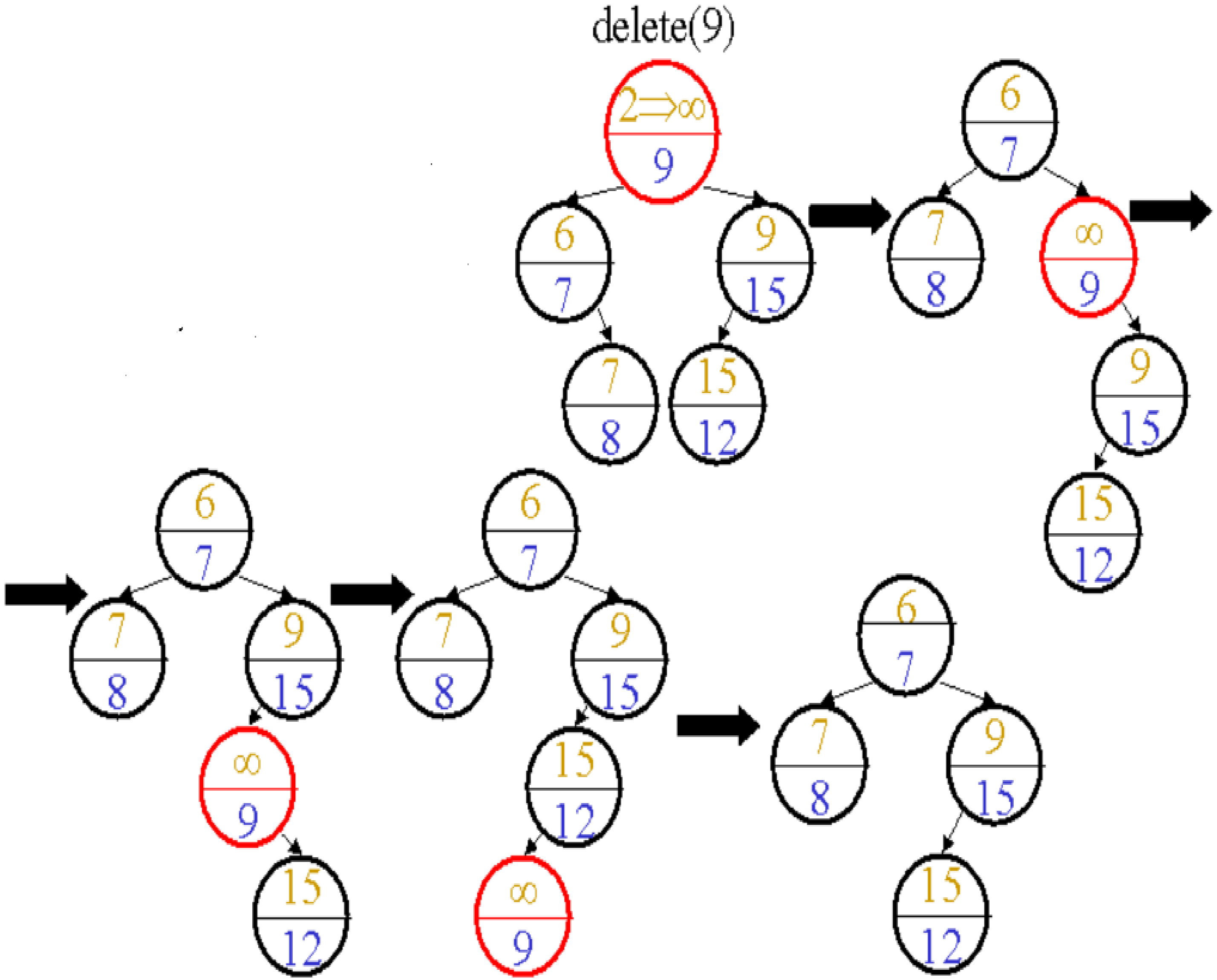
Treap Insert

- Choose a random priority
- Insert as in normal BST
- Rotate up until heap order is restored



Treap Delete

- Find the key
- Increase its value to ∞
- Rotate it to the fringe
- Snip it off


delete(9)
2⇒∞
9
6
7
9
15
7
8
15
12
6
7
7
8
∞
9
9
15
15
12
6
7
7
8
9
15
∞
9
15
12
6
7
7
8
9
15
15
12
∞
9
6
7
7
8
9
15
15
12