DMDW – Module-3

By

Dr. Pulak Sahoo

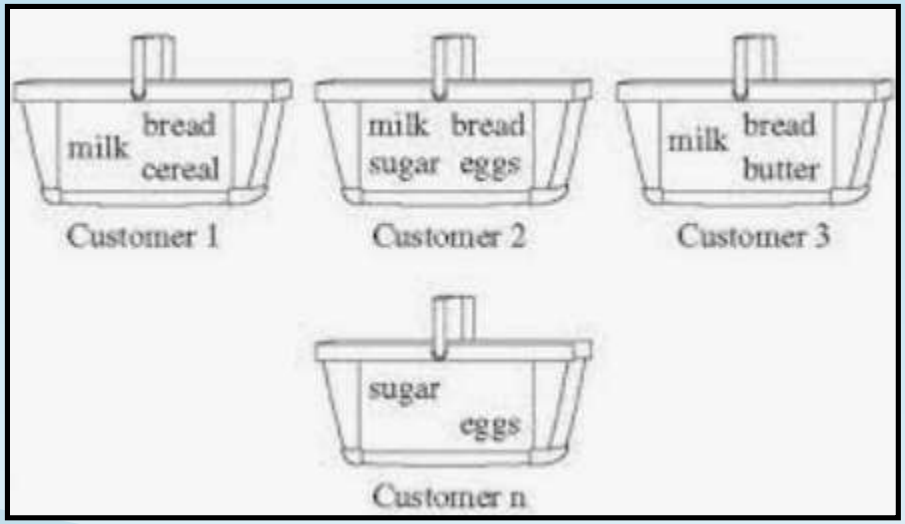
Associate Professor
Dept of CSE, SIT, BBSR

Module-3 Syllabus

Mining Frequent Patterns, Associations and Correlations: Introduction, Market Basket Analysis, Frequent Item-set generation using Apriori algorithm, Rule generation; Alternative methods for generating frequent item-sets using FP-Growth algorithm, Evaluation of association patterns; From association analysis to correlation analysis.

Mining Frequent Patterns, Associations, & Correlations:

Basic Concepts & Methods



What is Frequent Pattern Mining?

- **Frequent pattern**: a pattern (a set of items, subsequences, substructures etc.) that occurs frequently in a data set
- **Motivation**: Finding inherent regularities in data
  - What products were often purchased together? Ex: Milk & Bread?
  - What are the subsequent purchases after buying a PC?
  - What kinds of DNA are sensitive to this new drug?
  - Can we automatically classify web documents?
- **Applications**
  - Basket data analysis, cross-marketing, catalog design, sale campaign analysis, Web log analysis, & DNA sequence analysis

Why Is Freq. Pattern Mining Important?

- **Frequent Pattern Mining** searches for recurring relationships in a given data set
- It discloses intrinsic & important properties of data sets
- Finding frequent patterns plays an essential role in mining associations, correlations & many other interesting relationships between item sets in transactional & relational databases
- It helps in data classification, clustering & other data mining tasks

Market Basket Analysis

- **Market Basket Analysis** is a typical example of frequent item set mining
- This process analyses 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
- **Ex**: if customers are buying milk, how likely are they to also buy bread (what kind of bread) on same trip to the supermarket?

Example: Market Basket Analysis

- Manager of an All Electronics branch, would like to learn more about the buying habits of his customers specifically ***"Which groups or sets of items are customers likely to purchase on a given trip to the store?"***
- To answer this, market basket analysis may be performed on the retail data of customer transactions
- The results can then be used to plan marketing/advertising strategies or in the design of a new catalog or design different store layouts

Association Rule Mining

- Given a set of transactions, find rules that will predict the occurrence of an item based on the occurrences of other items in the transaction

Market-Basket transactions

TID	*Items*
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Example of Association Rules

{Diaper} → {Beer},
{Milk, Bread} → {Diaper},
{Beer, Bread} → {Milk}

Implication means co-occurrence

Association Rule Mining Cont…

- Association rules are written as **X ->Y** (whenever X appears Y also tends to appear)
- **X** is called **antecedent** & **Y is called consequent**
- **X->Y** indicates that X & Y have been found together frequently in the given data
- Suppose X & Y appear together in only 10% of the transactions
- But whenever X appears there is a 80% chance that Y also appears
- The 10% presence is called ***support of the rule & 80% chance is called confidence of rule***

Association Rule Mining Example

Association Rules

buys(X, “computer”)=>buys(X, “software”) [support =1%, confidence = 50%]

1% of all the transactions under analysis show that computer & software are purchased together

if a customer buys a computer, there is a 50% chance that he will buy software as well

Single Dimensional Association Rule

age(X, “20..29”) ^ income(X, “40K..49K”) => buys(X, “laptop”)

[support = 2%, confidence=60%]

Multi-Dimensional Association Rule

Association Rule Mining Cont…

- **Support**: it gives % of transactions from a transaction database that the given rule satisfies
- It is probability **P(XUY)**, where XUY indicates that a transaction contains both x & y
- **Confidence**: it access the % of certainty of the detected association
- It is taken as the conditional probability P(Y/X), that is the probability that a transaction containing X also contains Y
- Hence,
  - **Support (X → Y) = P(X U Y),**
  - **Confidence(X → Y) = P (Y/X)**

Association Rule Mining Cont…

- **Association Rule**
  - An implication expression of the form $X \rightarrow Y$, (X & Y are itemsets)
  - **Example**:
    {Milk, Diaper} → {Beer}
- **Rule Evaluation Metrics**
  - **Support (s)**
    - Fraction of transactions that contain **both X & Y**
  - **Confidence (c)**
    - Measures how often items in **Y appear in transactions that contain X**

TID	*Items*
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Example:

$$\{\text{Milk}, \text{Diaper}\} \Rightarrow \text{Beer}$$

$$s = \frac{\sigma(\text{Milk}, \text{Diaper}, \text{Beer})}{|T|} = \frac{2}{5} = 0.4$$

$$c = \frac{\sigma(\text{Milk}, \text{Diaper}, \text{Beer})}{\sigma(\text{Milk}, \text{Diaper})} = \frac{2}{3} = 0.67$$

Formal definition of Support & Confidence

- Let the total **No. of transactions is N**
- **Support(X)** = **(No. of times X appears) / N = P(X)**
- **Support(X->Y)** = **(No. of times X & Y appear together)/N = P(X ∪ Y)**
- **Confidence (X->Y)**= **Support(X->Y)/Support(X)= P(X∪Y)/P(X)= P(Y/X)**

Association Rule Mining Cont…

- Rule **support** & **confidence** are two measures of **rule interestingness**
- They respectively reflect the **usefulness** & **certainty** of **discovered rules**
- **Association rules** are considered **interesting** if they satisfy both a ***minimum support threshold*** *&* ***a minimum confidence threshold***
- **Rules** that satisfy both a minimum support threshold ***(min sup)*** ***&*** *a minimum confidence threshold* ***(min conf) are called strong***

Important Terminologies

- A **set of items** is referred to as an ***itemset***
- An **itemset** that contains **k items** is a ***k-itemset***
- The ***occurrence frequency*** of an itemset is the no. of transactions that contain the itemset. This is also known as ***(1)frequency, (2)support count or (3)count of the itemset***
- We know that **Support (X → Y) = P(X U Y)**. Here this **support** is referred to as ***relative support***. Whereas the **occurrence frequency** is called the ***absolute support.***

Support

- *(absolute) support*, or, *support count* of X (σ) :
- Frequency or occurrence of an itemset X
  - E.g. σ({Milk, Bread,Diaper}) = 2
- *(relative) support*, *s*, is the fraction of transactions that contains X (i.e., the probability that a transaction contains X)
  - E.g. s({Milk, Bread, Diaper}) = 2/5
- *Frequent Itemset*

  An itemset X is ***frequent*** if X's support >= a *minsup* threshold

TID	*Items*
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Important Terminologies

- An **itemset** ***X*** *is* ***closed*** *in a* ***data set*** ***S****, if there exists no proper* ***super-itemset*** ***Y*** *such* that *Y has the same support count as X*.

- ***Y*** is a **proper super-itemset** of ***X,*** if X is a proper sub-itemset of Y, i.e. $\mathbf{X \subset Y}$. i.e. every item of X is contained in Y but there is at least one item of Y that is not in X

- An **itemset** ***X*** *is a* ***closed frequent itemset*** in set ***S,*** *if X is both closed & frequent in S*

- An **itemset** ***X*** *is a* **maximal frequent itemset (or max-itemset)** in a data set *S, if X is frequent, & there exists no super itemset Y such that* $X \subset Y$ *& Y is frequent in S*

Apriori Algorithm

- Proposed by R. Agrawal & R. Srikant in 1994 for **mining frequent itemsets** for Boolean association rules
- **Apriori** employs an iterative approach known as a ***level-wise search***, *where k-itemsets are used to explore (k+1)-itemsets*
- To improve the efficiency of the level-wise generation of frequent itemsets, the **Apriori property** , is used to reduce the search space
- **Apriori property**: *All **nonempty subsets** of a **frequent itemset** must also be **frequent***

Apriori Algorithm

- If an "**itemset *I*"** *does not satisfy the minimum support threshold*, ***min-sup***, *then* ***'I' is not frequent****; i.e., P(I) < min-sup*.
- *If an "***item A***" is added to the "***itemset *I*****" , then the ***resulting itemset*** *cannot occur more frequently than I*. *Therefore,* **(I U A)** *is not frequent either; P(I U A) < min sup*

- **Apriori pruning principle**: If there is any itemset which is infrequent, its superset should not be generated/tested

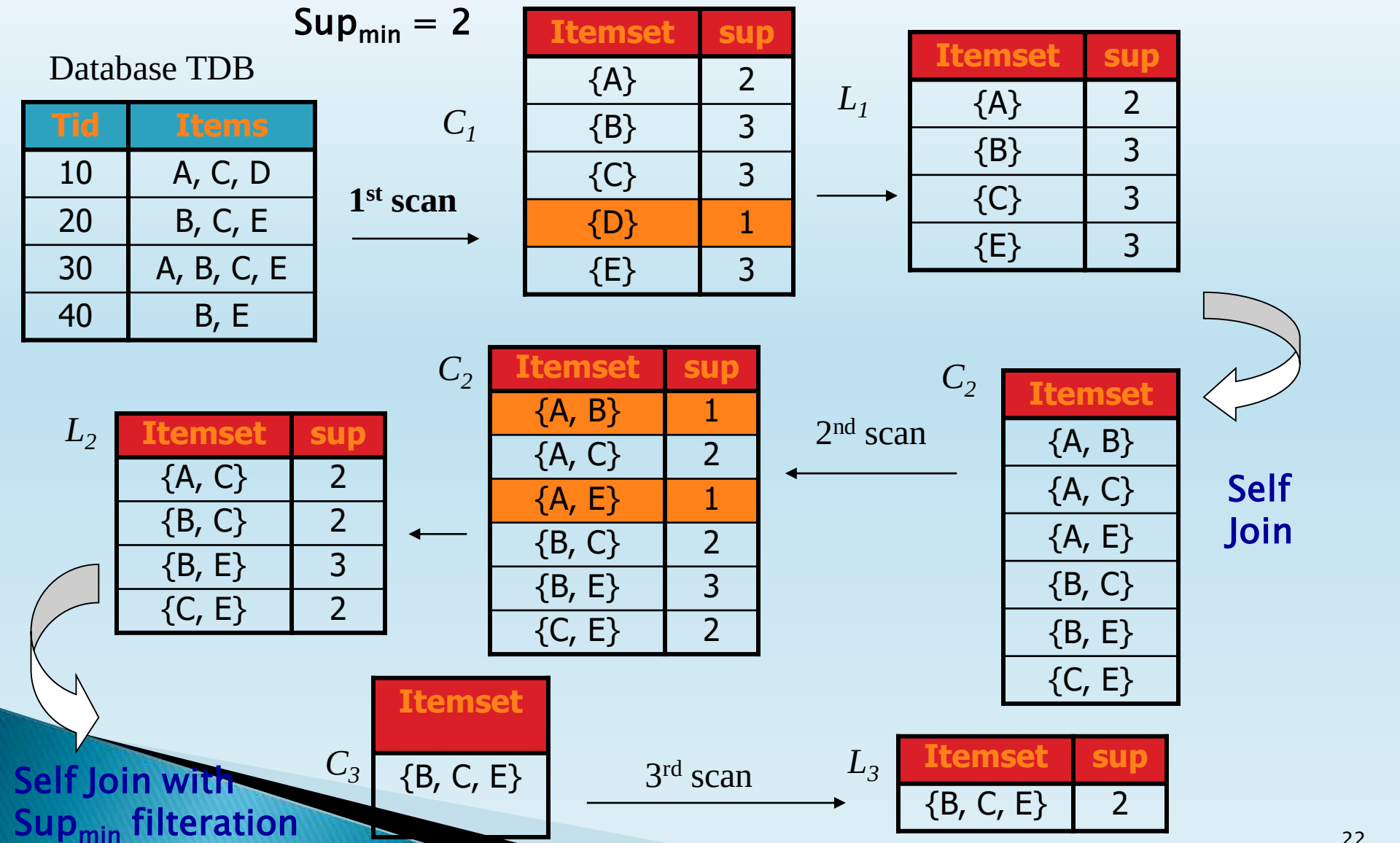
Apriori Algorithm

- **Method**:
  - Initially, scan the DB to get **frequent 1-itemset**
  - Generate **length (k+1)** candidate itemsets from length **k frequent itemsets**
  - **Test the candidates** against DB
  - **Terminate** when no frequent or candidate set can be generated

Implementation of Apriori

- **How to generate candidates?**
  - Step 1: self-joining L_k
  - Step 2: pruning
- **Example of Candidate-generation**
  - L_3={*abc, abd, acd, ace, bcd*}
  - **Self-joining: L_3*L_3**
    - *abcd* from *abc* & *abd*
    - *acde* from *acd* & *ace*
  - **Pruning:**
    - ***acde*** **is removed because subset** ***ade*** **is not in L_3**
  - **C_4 = {*abcd*}**

The Apriori Algorithm—An Example



The Apriori Algorithm

- 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 L_k

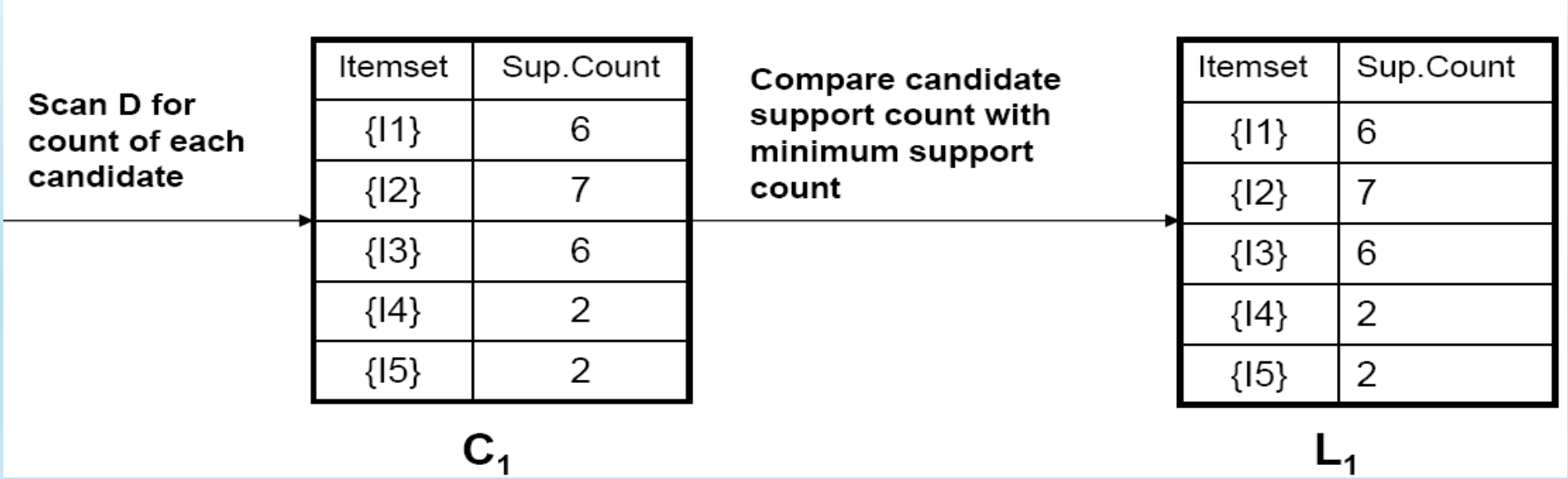
The Apriori Algorithm: Example

TID	List of Items
T100	I1, I2, I5
T100	I2, I4
T100	I2, I3
T100	I1, I2, I4
T100	I1, I3
T100	I2, I3
T100	I1, I3
T100	I1, I2 ,I3, I5
T100	I1, I2, I3

Consider a database, D , consisting of 9 transactions.

- **Suppose min. support count required is 2 (i.e. min_support = 2/9 = 22 %)**
- **Let minimum confidence required is 70%.**
- **We have to first find out the frequent itemset using Apriori algorithm.**
- **Then, Association rules will be generated using min. support & min. confidence.**

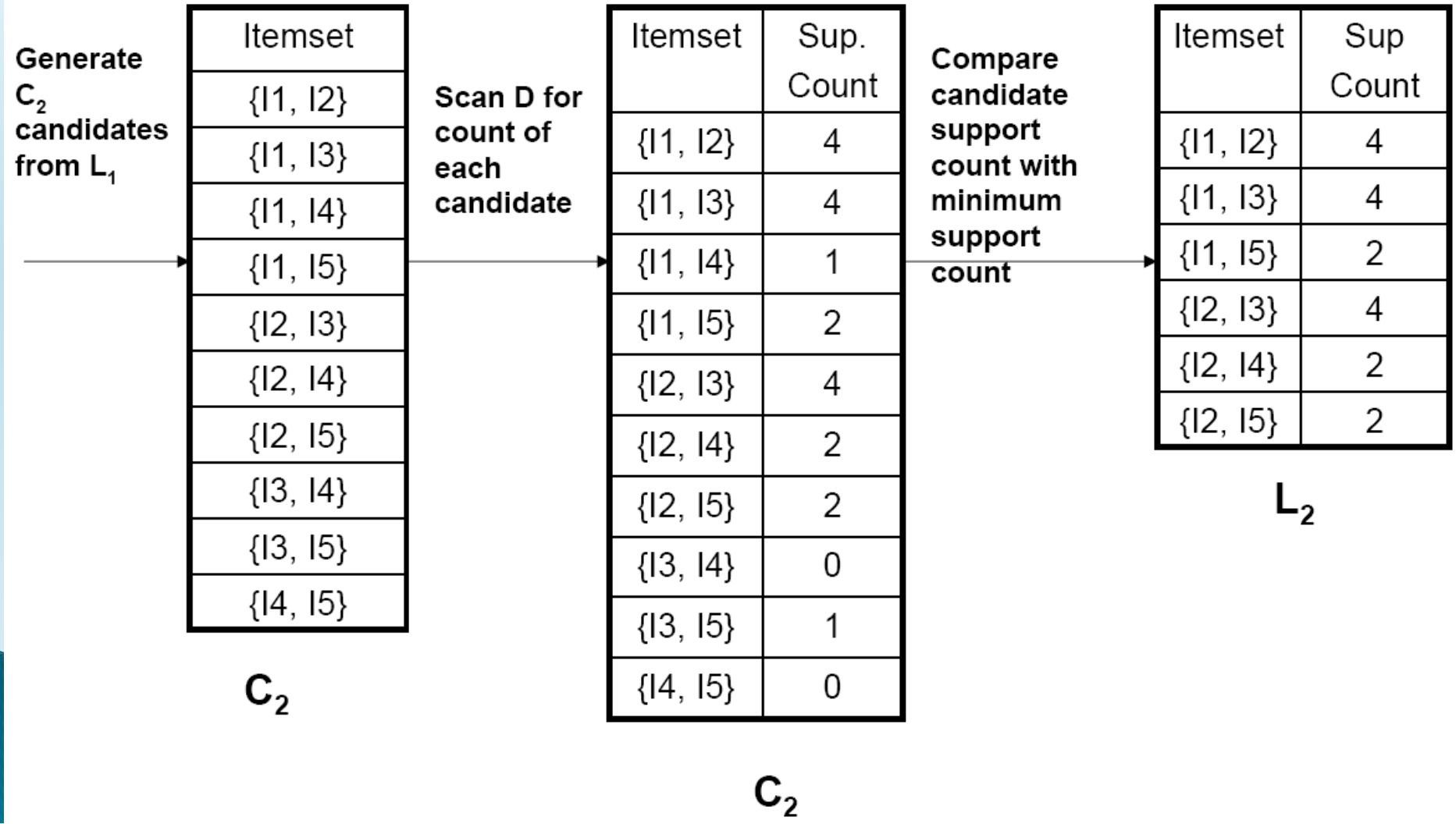
Step 1: Generating 1-itemset Frequent Pattern



The set of frequent 1-itemsets, L1, consists of the candidate 1-itemsets satisfying minimum support.

In the first iteration of the algorithm, each item is a member of the set of candidate.

Step 2: Generating 2-itemset Frequent Pattern



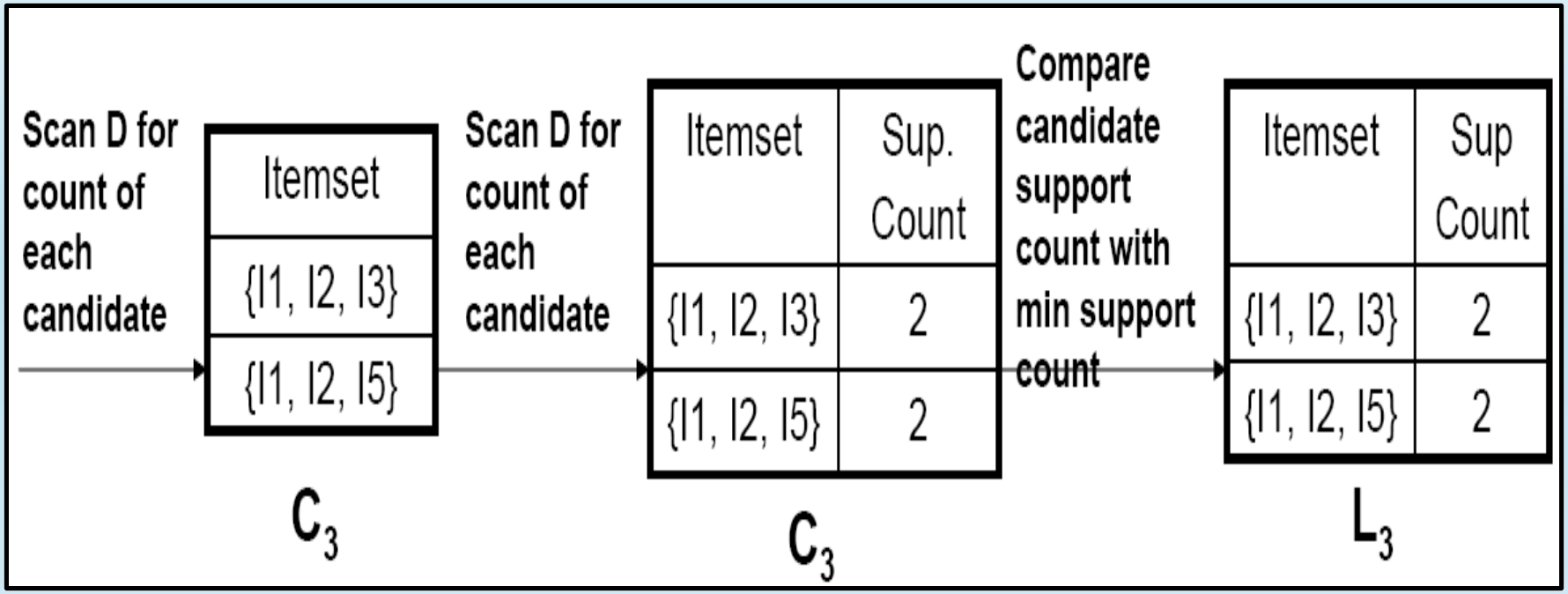
Step2 Cont…

- To discover the set of frequent 2-itemsets, **L_2**, the algorithm uses **L_1** ***Join L_1 to generate a candidate set of 2-itemsets, C_2***.
- Next, the **transactions in D are scanned** & the **support count** for each candidate itemset in C_2 is calculated
- The **set of frequent 2-itemsets, L_2**, is then determined, consisting of those candidate 2-itemsets in C_2 having minimum support.

Step 3: Generating 3-itemset Frequent Pattern

- The generation of the **set of candidate 3-itemsets, C_3**, involves use of the Apriori Property.
- In order to find **C_3**, we compute **L_2 join L_2.**
- $C_3 = L_2$ Join L_2 = {{I1, I2, I3}, {I1, I2, I5}, {I1, I3, I5}, {I2, I3, I4}, {I2, I3, I5}, {I2, I4, I5}}.
- Based on the Apriori property that **all subsets of a frequent itemset must also be frequent** we will determine that all of C_3 are not frequent.
- Therefore, C3 = {{I1, I2, I3}, {I1, I2, I5}} is remaining after checking for all members of result of Join operation & Pruning them
- Now, the transactions in D are scanned in order to determine L3, consisting of those candidates 3-itemsets in C3 having minimum support.

Step3 Cont…


Scan D for count of each candidate
Itemset
{I1, I2, I3}
{I1, I2, I5}
C3
Scan D for count of each candidate
Itemset
Sup. Count
{I1, I2, I3}
2
{I1, I2, I5}
2
C3
Compare candidate support count with min support count
Itemset
Sup Count
{I1, I2, I3}
2
{I1, I2, I5}
2
L3

Step 4: Generating 4-itemset Frequent Pattern

- The algorithm uses **L_3 Join L_3** to generate a candidate set of 4-itemsets, C4. Although the join results in {{I1, I2, I3, I5}}, this itemset is pruned since its subset {{I2, I3, I5}} is not frequent.

- **C4= ϕ & algorithm terminates**

- These **frequent itemsets will be used to generate strong association rules**

Step 5:Generating Association Rules from Frequent Itemsets

Procedure:

- 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 the minimum confidence threshold.**

- Lets take ***L= {I1,I2,I5}.***
- Its all nonempty subsets are {I1,I2}, {I1,I5}, {I2,I5}, {I1}, {I2}, {I5}.
- Let minimum confidence threshold is 70%

R1: I1 ^ I2 → I5

Confidence = sc{I1,I2,I5}/sc{I1,I2} = 2/4 = 50%

R1 is Rejected.

R2: I1 ^ I5 → I2

Confidence = sc{I1,I2,I5}/sc{I1,I5} = 2/2 = 100%

R2 is Selected.

R3: I2 ^ I5 → I1

Confidence = sc{I1,I2,I5}/sc{I2,I5} = 2/2 = 100%

R3 is Selected

R4: I1 → I2 ^ I5

Confidence = sc{I1,I2,I5}/sc{I1} = 2/6 = 33%

R4 is Rejected.

R5: I2 → I1 ^ I5

Confidence = sc{I1,I2,I5}/{I2} = 2/7 = 29%

R5 is Rejected.

Exercise

- A dataset has five transactions, let min-support=60% & min_confidence=80%
- Find all frequent itemsets using Apriori algorithm

TID	Items_bought
T1	M, O, N, K, E, Y
T2	D, O, N, K , E, Y
T3	M, A, K, E
T4	M, U, C, K ,Y
T5	C, O, O, K, I ,E

Association Rules with Apriori

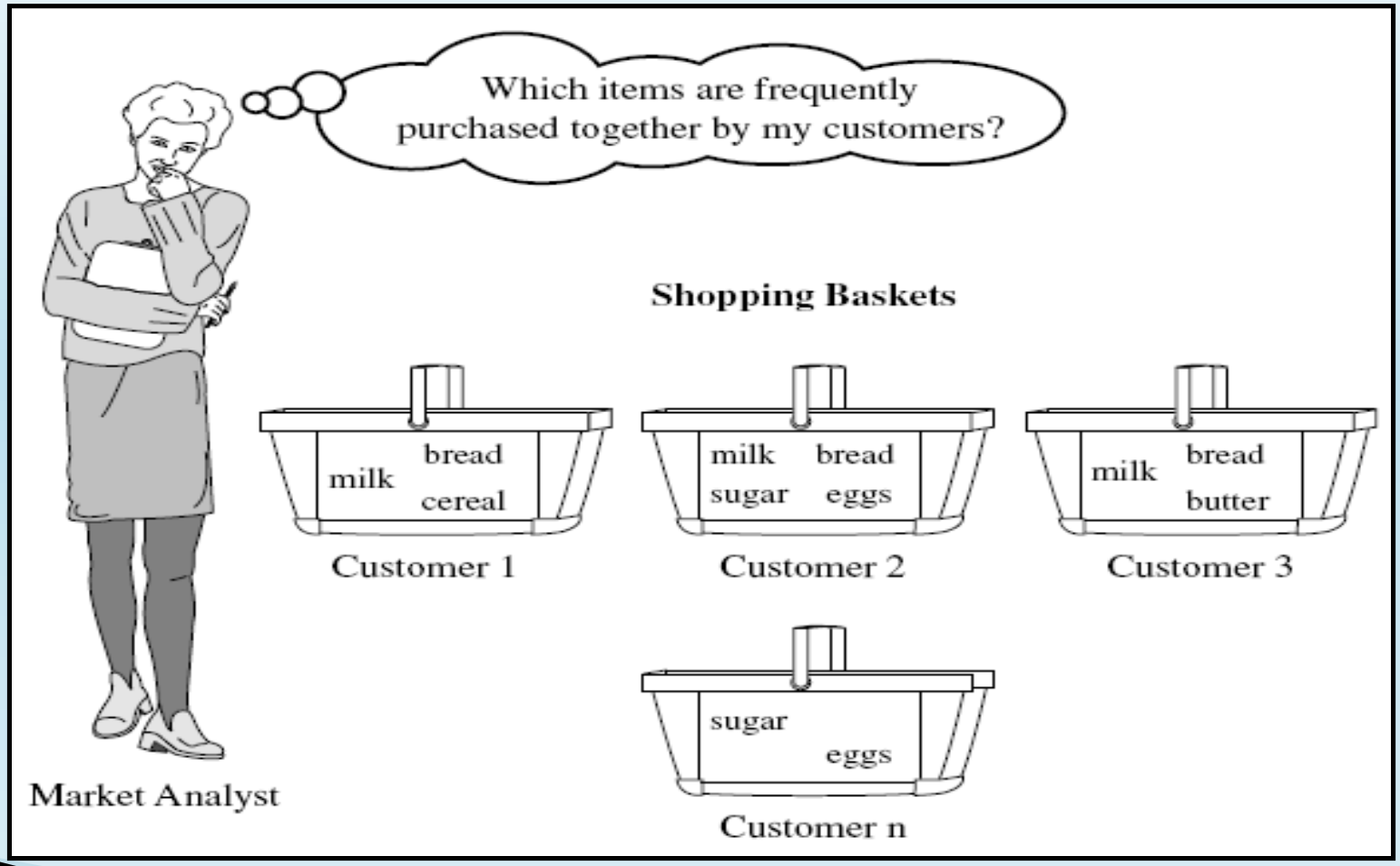
K:5		KE:4		KE		
E:4		KM:3		KM		
M:3		KO:3		KO		
O:3	=>	KY:3	=>	KY	=>	KEO
Y:3		EM:2		EO		
		EO:3				
		EY:2				
		MO:1				
		MY:2				
		OY:2				

Frequent Pattern Growth (FP- Growth)

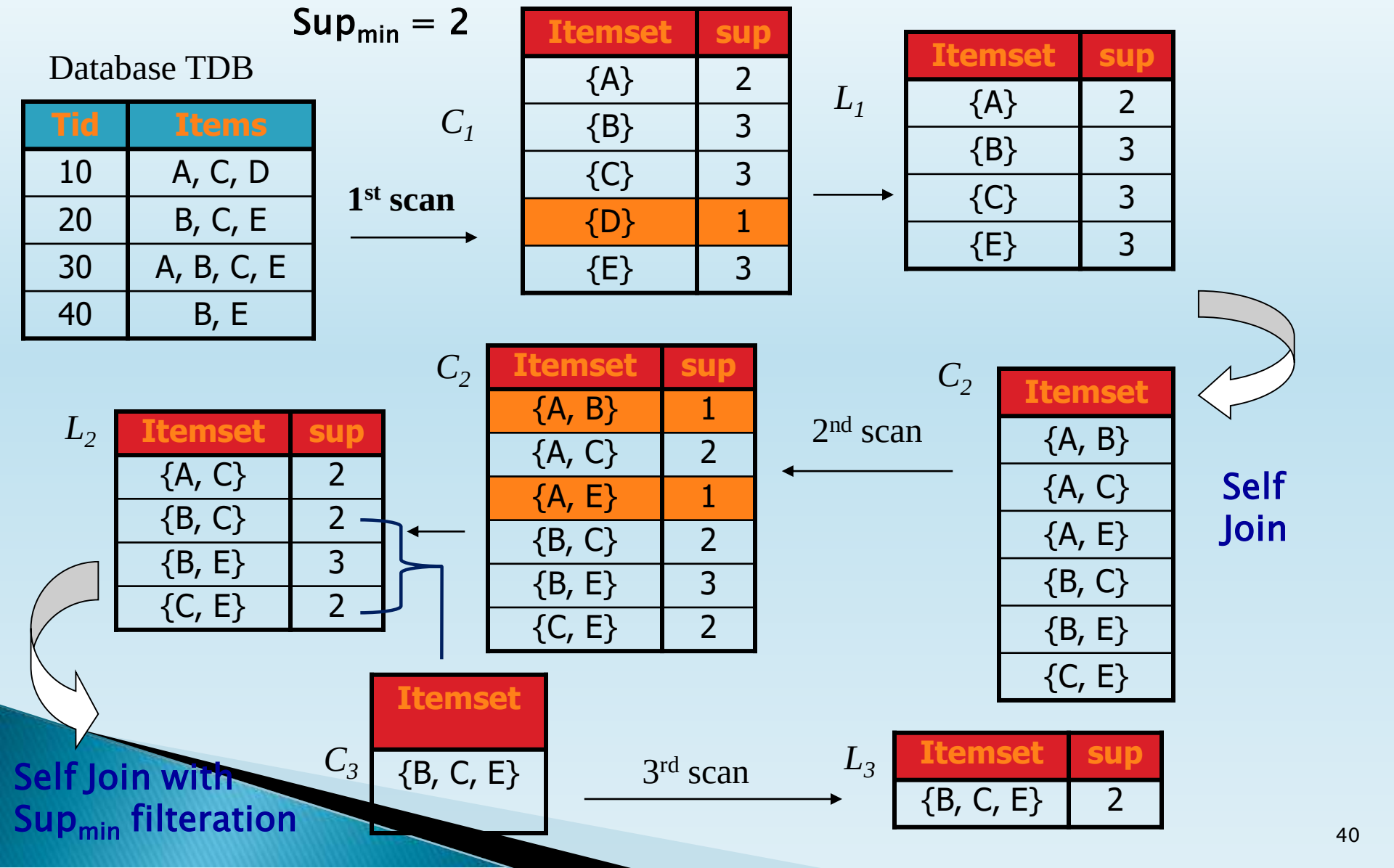
A Pattern- Growth Approach for Mining Frequent Itemsets

Start from here

Market Basket Analysis



The Apriori Algorithm—An Example



Introduction

- **Apriori:** uses a generate-and-test approach – *generates candidate itemsets & tests if they are frequent*

- **Disadvantages:**
- Generation of **candidate itemsets** is **space** & **time** **extensive**
  - **Support count calc. & DB scans** after each iteration are overheads

- **FP-Growth approach:** Allows frequent itemset discovery without candidate itemset generation

- **It's a two step approach:**
  - **Step 1:** Build a data structure called the **FP-tree**
  - **Step 2:** Extracts frequent itemsets directly from FP-tree

Pattern-Growth Approach: Mining Frequent Patterns Without Candidate Generation

- **Bottlenecks of the Apriori approach**
  - **Breadth-first (level-wise) search (**using **self join)**
  - This candidate generation process often generates a huge no. of candidates
- **The FP-Growth Approach (J. Han, J. Pei, & Y. Yin, SIGMOD' 00)**
  - **Depth-first search, generates less candidates**

FP–Tree Construction

- **FP-Tree** is constructed using **2 passes over the data-set:**

Pass 1:

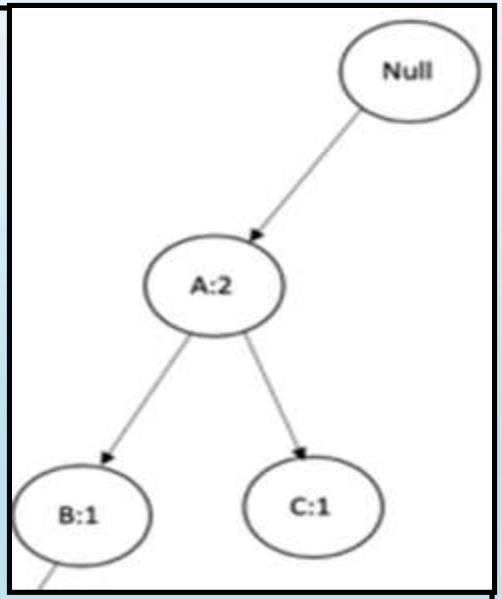
  - **Scan data** & **find support** for each item
  - **Discard** infrequent items
  - **Sort** frequent items in decreasing order of their support

- Use this order when building the **FP-Tree**

FP–Tree Construction Cont…

Pass 2:

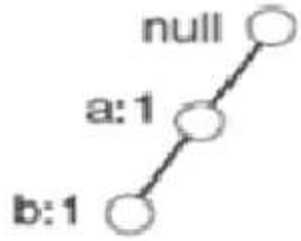
Create Nodes for each items along with a counter



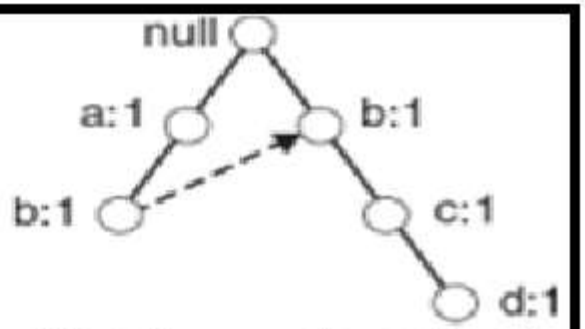
- Read one transaction at a time & maps it to a path
- Paths can overlap when transactions share items
  - In this case, **counters** are incremented
- **Pointers** are shown between nodes containing the same item
- **Frequent itemsets** extracted from the **FP-Tree**

FP-Tree Construction (Example)

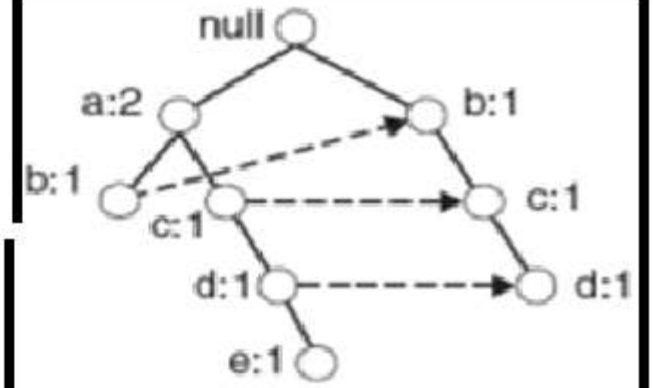
TID	Items
1	{a,b}
2	{b,c,d}
3	{a,c,d,e}
4	{a,d,e}
5	{a,b,c}
6	{a,b,c,d}
7	{a}
8	{a,b,c}
9	{a,b,d}
10	{b,c,e}



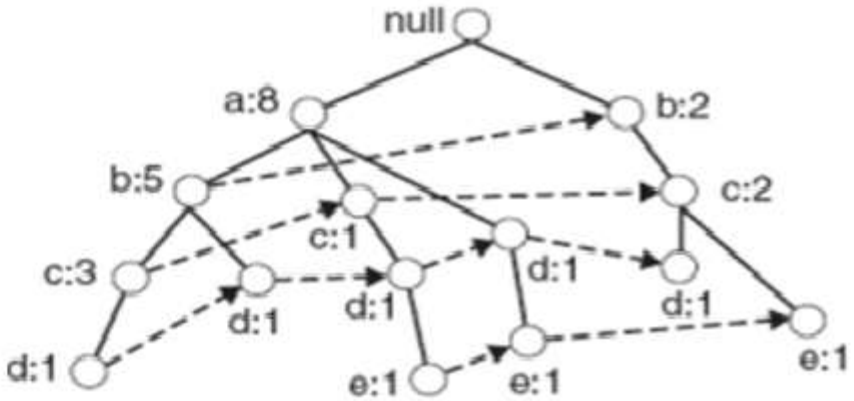
(i) After reading TID=1



(ii) After reading TID=2



(iii) After reading TID=3



(iv) After reading TID=10

FP–Tree Size

- **FP-Tree size** varies between:
  - **Best case scenario:** all transactions contain the same set of items
    - 1 path in the FP-tree
  - **Worst case scenario:** every transaction has unique set of items (no items in common)
    - Size of the FP-tree is as large as the original data
- The size of the FP-tree depends on the items ordering
- **Ordering** items by decreasing support normally generates **smaller trees**

Ex-1: Construct FP-tree from below Transaction Database

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, w}	{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}

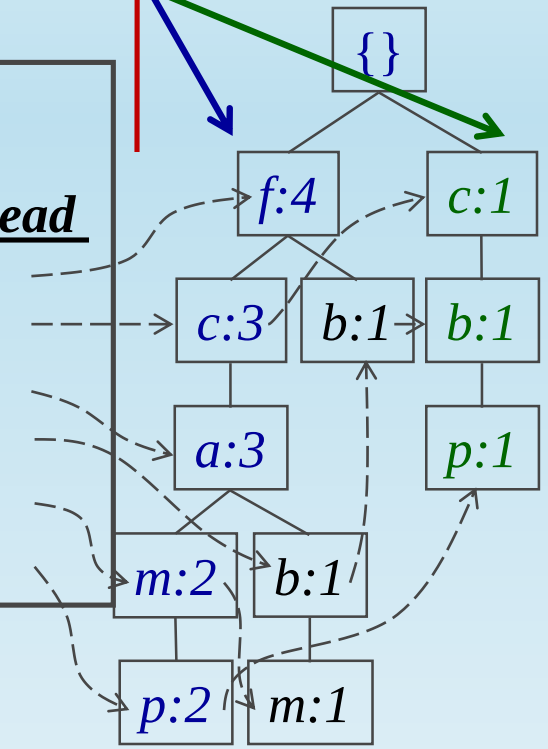
min_support = 3

1. Scan DB to find frequent 1-itemset
2. Sort items in descending order frequency, f-list
3. Construct FP-tree

Header Table

Item	frequency	head
f	4	
c	4	
a	3	
b	3	
m	3	
p	3	

F-list = f-c-a-b-m-p



Frequent Pattern Algorithm Steps

- **#1)** Scan the data to find the occurrences of 1-itemsets in the database along with the support count
- **#2)** Construct the FP tree. Create the root as null
- **#3)** The next step is examine the first transaction & find out the itemset in it. The itemset with the max count is taken at the top, the next itemset with lower count & so on
- **#4)** The next transaction is examined
- **#5)** The count of the itemset is incremented as it occurs in the transactions.

Frequent Pattern Algorithm Steps Cont…

- **#6)** Then the lowest node is examined first along with it's links. The lowest node represents the frequency pattern length 1. From this, traverse the path in the FP Tree.
- **#7)** Construct a **Conditional FP Tree**, which is formed by a count of itemsets (meeting the threshold support) in the path.
- **#8) Frequent Patterns** are generated from the **Conditional FP Tree**.

Repeat Explaination

FP-tree

FP-tree Pseudocode and Explanation

- Step 1: Deduce the ordered frequent items. For items with the same frequency, the order is given by the alphabetical order.
- Step 2: Construct the FP-tree from the above data
- Step 3: From the FP-tree above, construct the FP-conditional tree for each item (or itemset).
- Step 4: Determine the frequent patterns.

Ex–2: Construction of the FP tree

- **The transaction here consists of 5 items:**

- **Asparagus (A),**
- **Corn (C),**
- **Beans (B),**
- **Tomatoes (T)**
- **Squash (S)**

- Let the **min_support = 2**

Table-1

Table 1	
Transaction ID	**List of items in the tran**
T1	B , A , T
T2	A , C
T3	A , S
T4	B , A , C
T5	B , S
T6	A , S
T7	B , S
T8	B , A , S , T
T9	B , A , S

Table-3

Item	Support Count
Asparagus (A)	7
Beans (B)	6
Squash (S)	6
Corn (C)	2
Tomatoes (T)	2

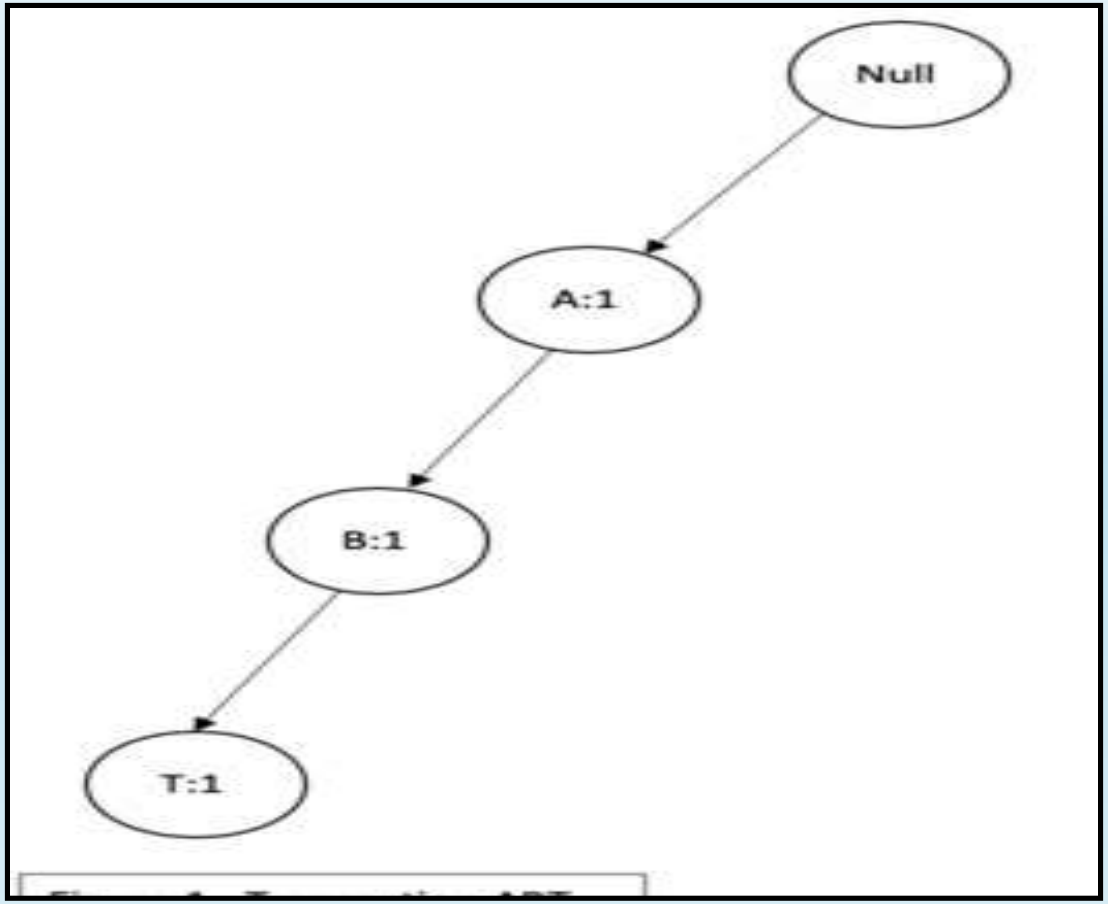
- Beans (B) & Squash (S) / Corn(C) & tomatoes(T) have the same support count & any of them can be written first

Processing Transactions & building FP Tree

Transaction - T1

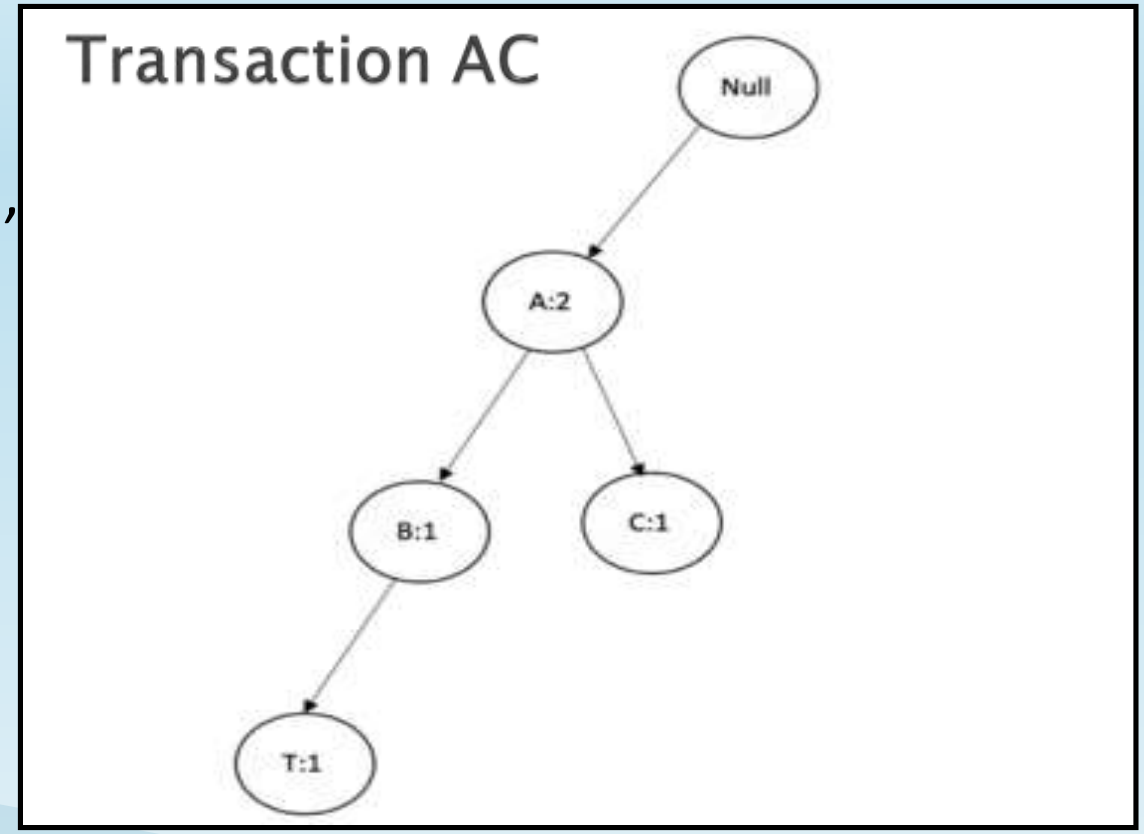
- The FP tree **root node is a NULL node**
- **T1** consists of **Beans (B), Asparagus (A) & Tomatoes (T)**
- Calculate **Support count** of 3 items:
- **Asparagus (A)** – Sup_cnt = **7 (max)**, Extend the tree from root node to A (Asparagus) with count denoted by **A:1**
- **Beans(B) –** Sup_cnt = **6,** From **Asparagus**, we can extend the tree to Beans **B:1** & after that **T:1** for **Tomatoes,** Sup_cnt = 2
- The tree structure is as below in **next page**

Transaction ABT



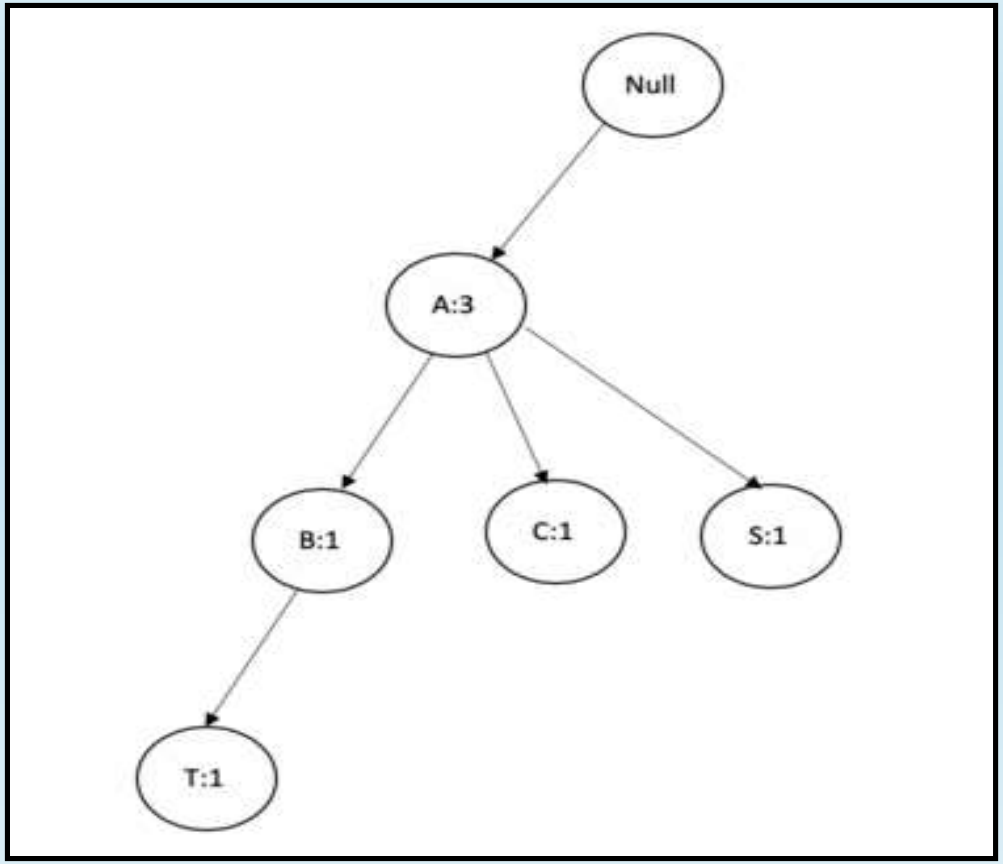
Transaction – T2: two items : Asparagus(A) & Corn (C)

- The support count for Asparagus(A) is **7** & Corn (C) is **2**
- Search for any branch to Asparagus (A), we have one & we can increase the count as A:2 (Figure 2)
- Next, since there is no node connecting (A) to (C), we need to create a branch for Corn as C:1



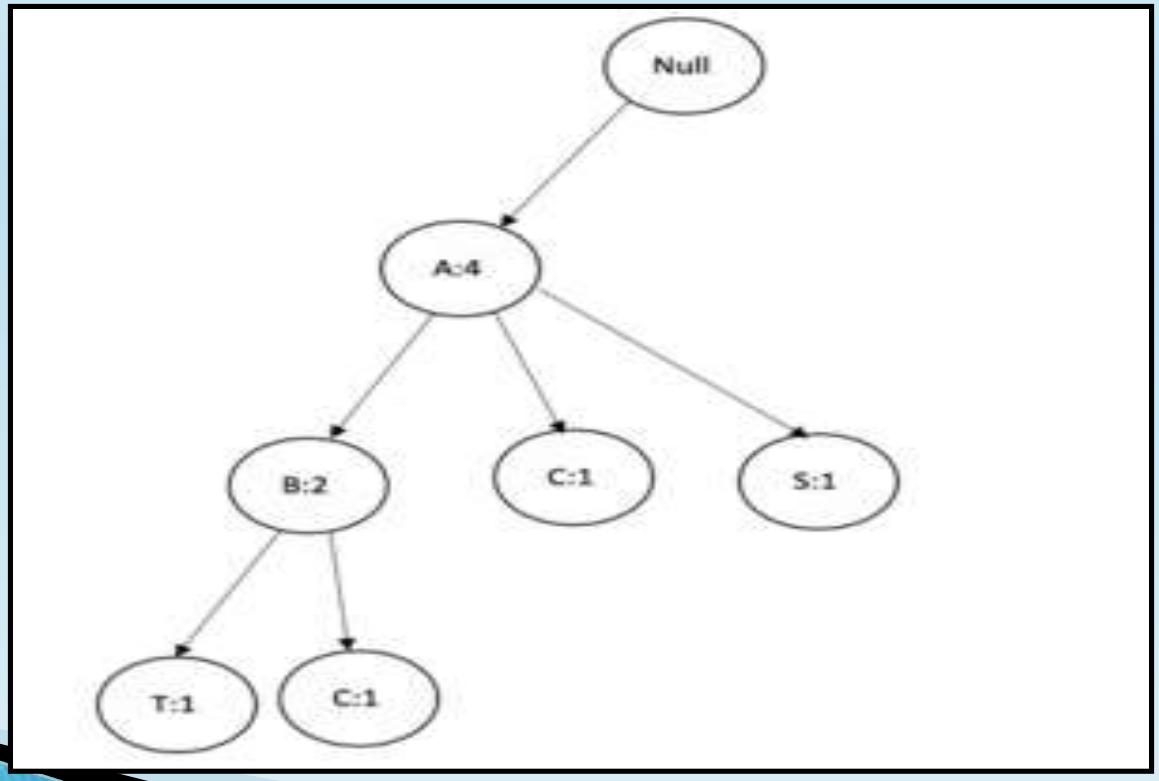
- **Transaction – T3:** two items : Asparagus(A) & Squash (S)
- So, Asparagus(A) count has been increased from A:2 to A:3
- Since no nodes from Asparagus to Squash, we need to create another branch for Squash S:1 (Figure 3)

Transaction AS



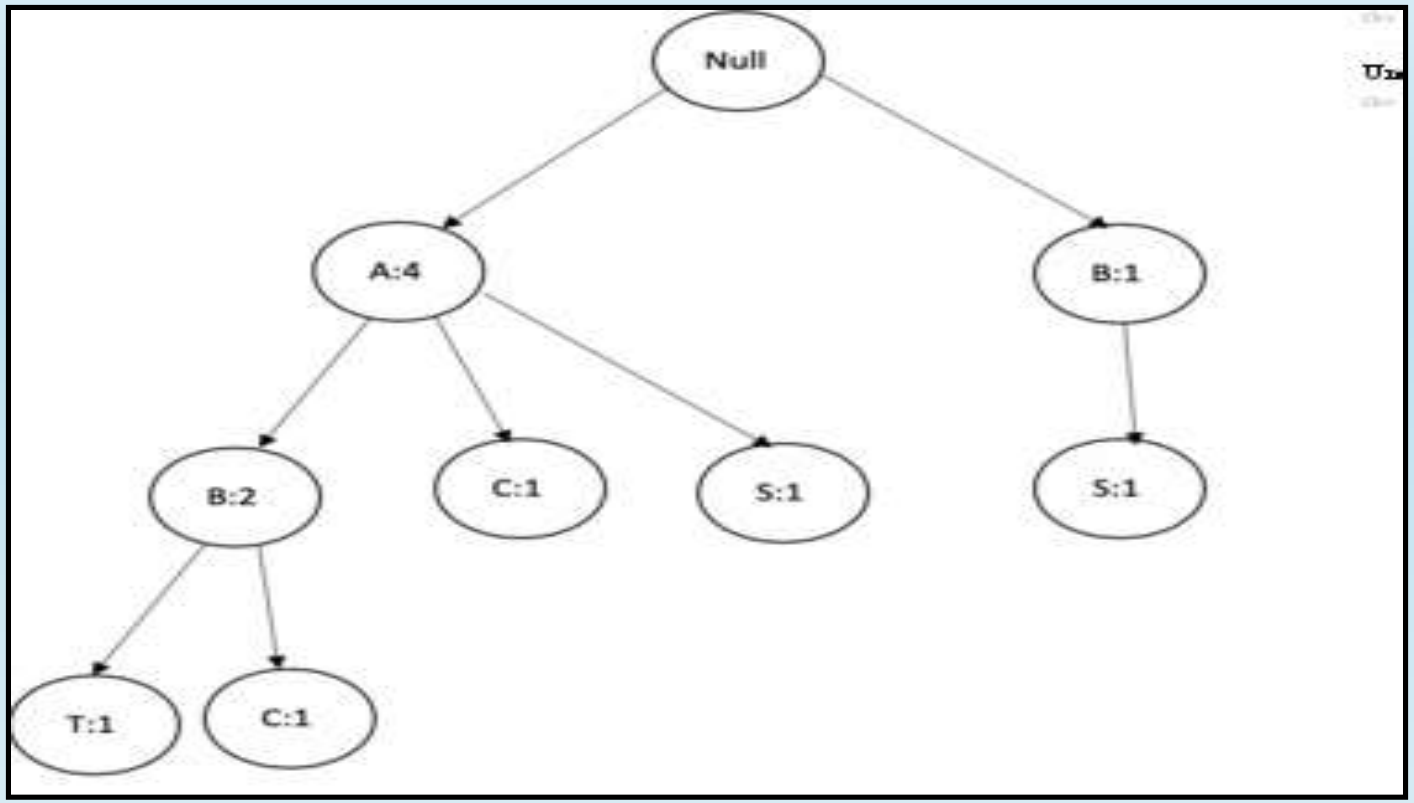
- For **Transaction 4**, the count for Asparagus & Beans has been increased to A:4 & B:2
- But after that, there isn't any branch that extends to corn (C), so we need to create a node for C:1

Transaction ABC



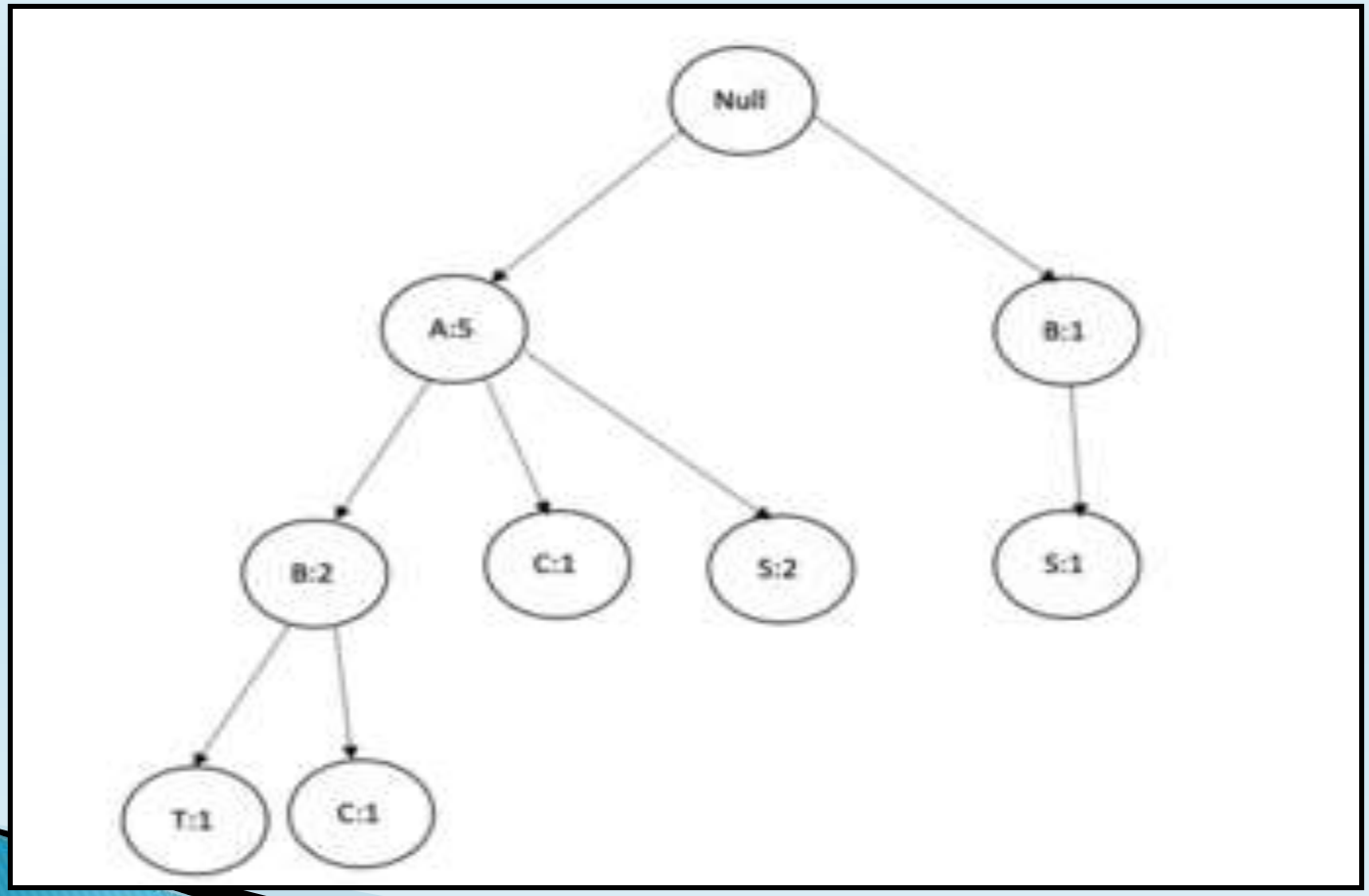
- **Transaction 5**, contains Beans (B) & Squash (S)
- But there aren't any node linked to Beans from the root node
- We need to create a new branch from the Null node with B:1 & S:1

Transaction BS

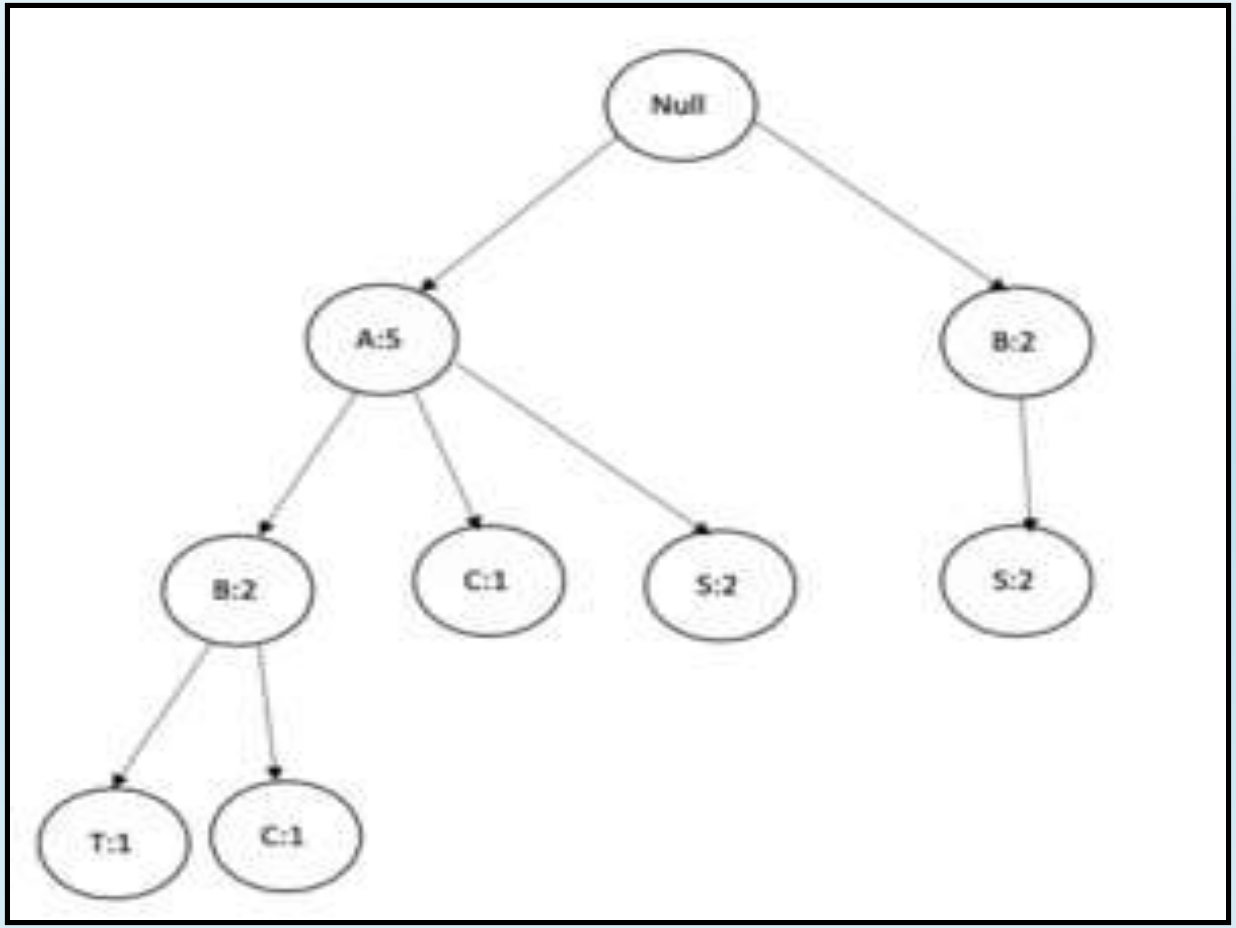


- **Transaction 6** to **transaction 10** are self-explanatory.
- The figs have been provided for each transaction.

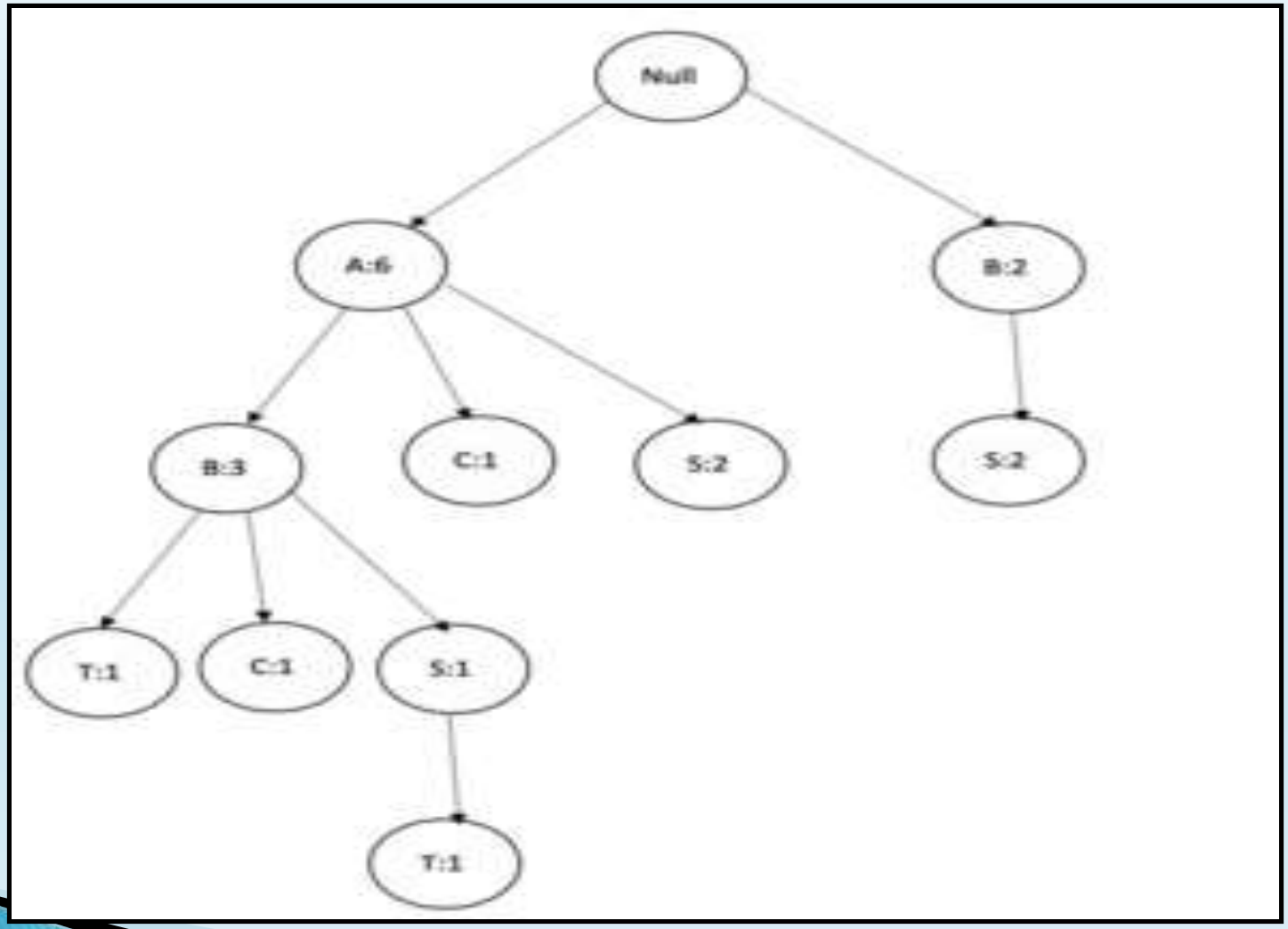
Transaction AS



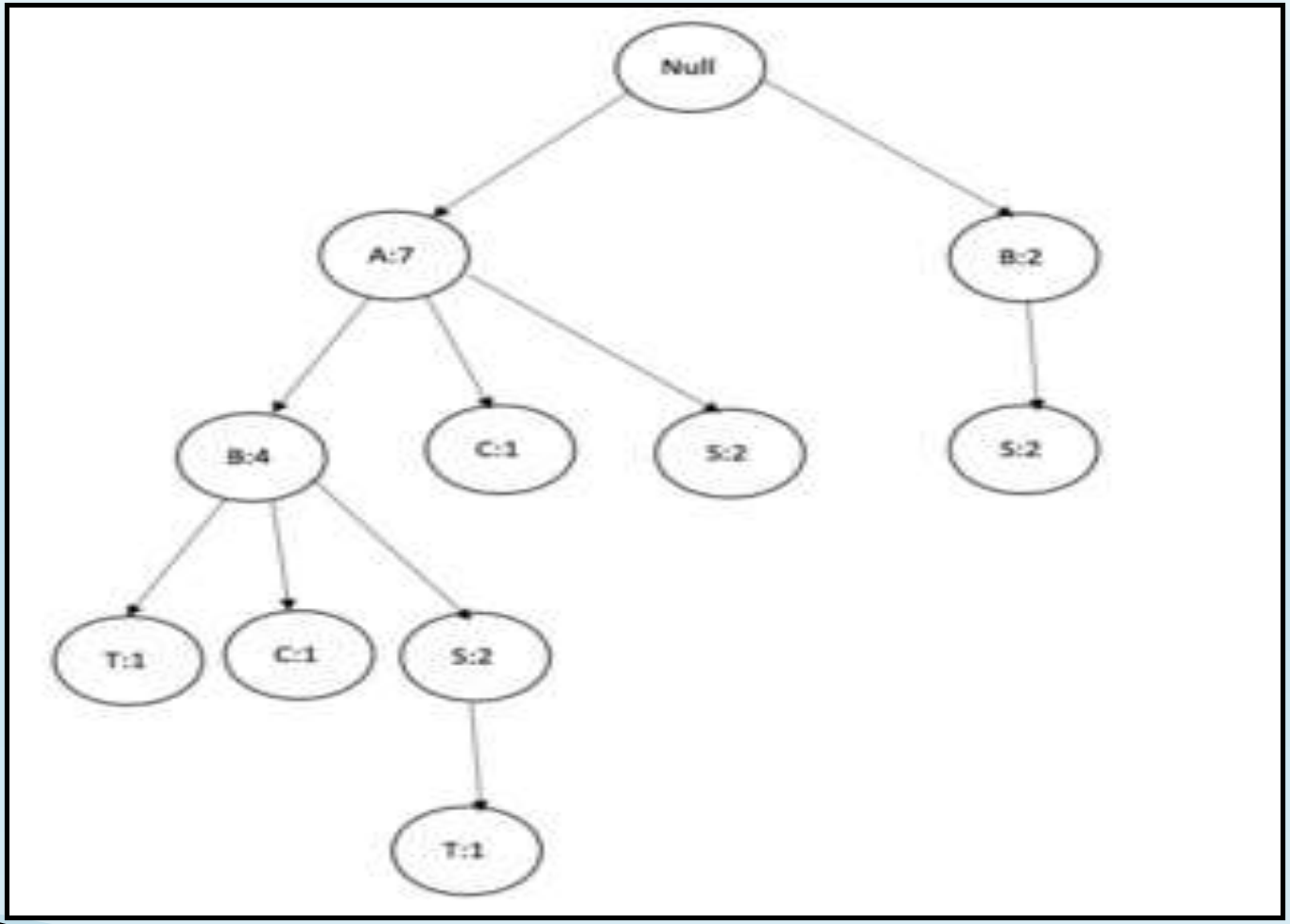
Transaction-7



Transaction-8(ABST)



Transaction-9



Item	Support Count	Node Link
Asparagus (A)	7	
Beans (B)	6	
Squash (S)	6	
Corn (C)	2	
Tomatoes (T)	2	

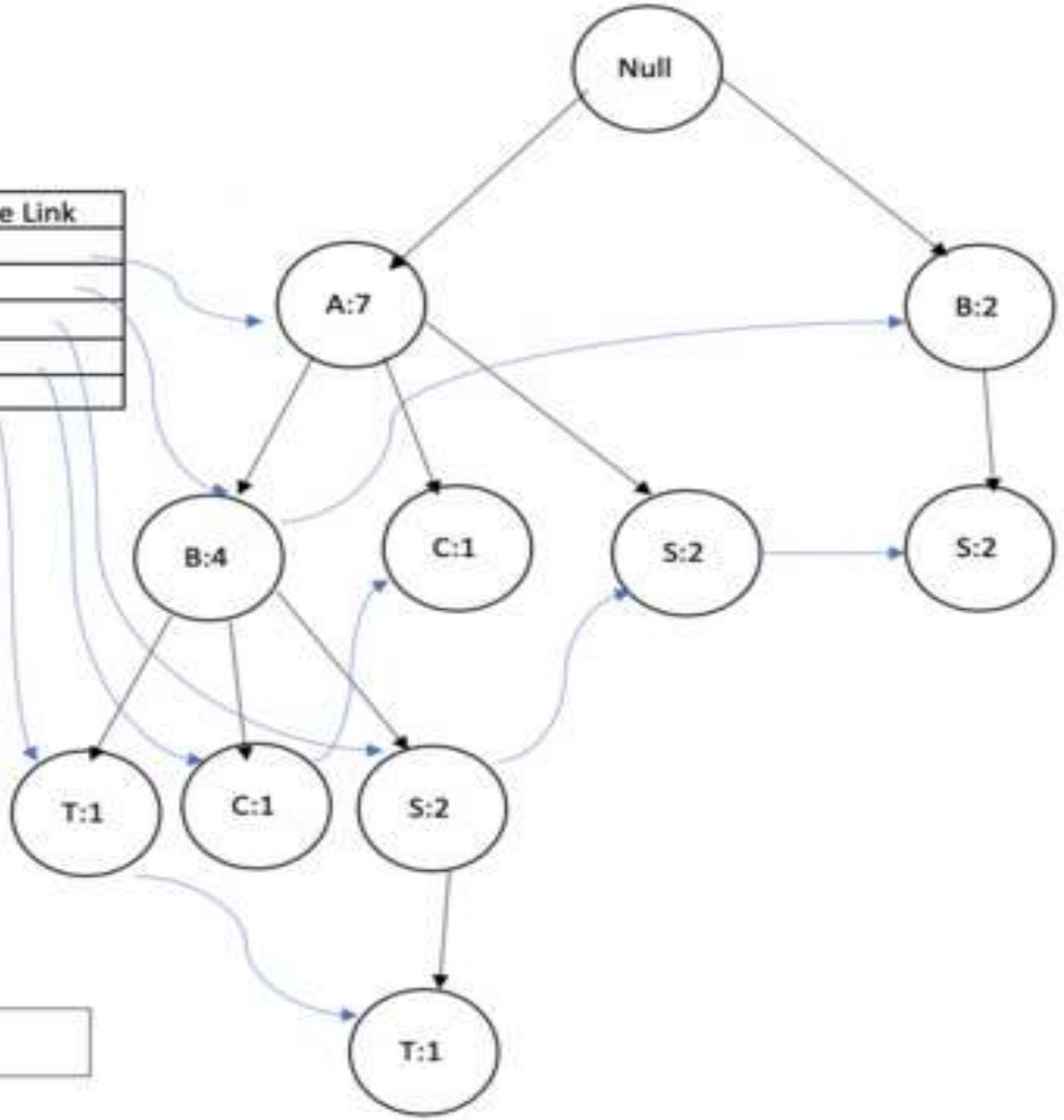


Figure 10

Table for Conditional Pattern Base

- Conditional Pattern Base table helps in for finding the frequent pattern generation
- We need to consider the item which has the lowest support count, Tomatoes (T) : 2
- **Tamato**
- There are 2 traversal paths for tomatoes (T) from root node
- **First being A-B-T**, where T is having count 1 i.e., T:1.
- **Second being A-B-S-T** & T has a count of 1 (T:1)
- The **conditional pattern base is {A, B, S:1}**.
- There are **two conditional pattern bases {{A,B:1},{A,B,S:1}}**

Conditional FP Tree

- Here **A is in {A,B:1} & {A,B,S:1}** so both have a final count of 1 each summing up to **<A:2>**
- In the similar fashion **<B:2> & <S:1>**
- So, it comes out to be **<A:2,B:2, S:1>, S:1** will not be considered as the **min support count =2**
- Finally the value should be **<A:2, B:2>** for the first row Tomatoes (T)

Table-4

Item	Conditional Pattern base	Conditional FP tree	Frequent Pattern Generation
Tomatoes (T)	{{A,B:1},{A,B,S:1}}	<A:2,B:2>	{A,T:2},{B,T:2},{A,B,T:2}
Corn (C)	{{A,B:1},{A:1}}	<A:2>	{A,C:2}
Squash (S)	{{A,B:2},{A:2},{B:2}}	<A:4,B:2>,<B:2>	{A,S:4},{B,S:4},{A,B,S:2}
Bean (B)	{{A:4}}	<A:4>	{A,B:4}

Frequent pattern for Tamato

- Joining <A:2> with Tomatoes (T) we can write {A,T:2}
- Joining <B:2> with tomatoes (T) it comes out to be {B,T:2}
- Then we need to join A & B both with T to get {A,B,T:2}
- So the frequent pattern generation for Tomatoes (T) row is: **{A,T:2},{B,T:2},{A,B,T:2}**

Corn (C) rows

- We can reach Corn (C) through 2 different paths A-B-C & A-C, for both the traversal paths, the count for Corn (C) is 1
- {{A,B:1},{A:1}} is our conditional pattern base
- Now for the Conditional FP tree column, we need to check for a count of each item in the conditional pattern base, for we have 2 as an account of A & 1 as count of B

So the conditional FP Tree columns stands to be <A:2>

- After this, join Conditional FP tree column with Item column for Corn (C) which comes out to be **{A, C:2}**

Squash (S)

- We can reach Squash through 3 paths A-B-S, A-S, & B-S
- Since the count of Squash is 2 in A-B-S, we can write {A,B:2}, for the other two paths we can write {A;2} & {B:2}
- So the Conditional pattern base stands at **{{A,B:2},{A:2},{B:2}}**
- We need to write this differently & not add in one **<A:4, B:2>,<B:2>**
- Now joining A:4 from <A:4,B:2> with Squash (S) gives {A,S:4}
- Now from <A:4,B:2> joining with Squash (S) gives,{B,S:2}
- But we have written **{B, S:4}** the count as 4 since we have another <B:2> in the Conditional FP tree column

- Similarly, the last row for **Bean** is constructed
- Also, Asparagus is directly from the Null node & since there aren't any in-between nodes to reach Asparagus (A), there is no need to go for another row of Asparagus

Generation of strong association rules from frequent item sets

- **FP growth generates strong association rules using a minimum support defined by the user** & what we have done till now is to get to the table-4 using minimum count=2
- And finally generated frequent Item sets which are in the **last column of the Frequent Pattern Generation in table 4**
- So considering table 4's last column of frequent Pattern generation, we have generated a 3-item frequent set as **{A,B,T:2} & A,B,S:2}**
- Similarly 2-Item frequent sets are:
  - **{A,T:2},**
  - **{B,T:2}, {A,C:2}, {A,S:4},{B,S:4} & {A,B:4}**
- We can see that we have arrived at distinct sets.
- Here we haven't generated Tomatoes (T) & Squash (S) or Corn (C) & Beans(B) since they are not frequently bought together items which is the main essence behind the association rules criteria & FP growth algorithm.

Advantages Of FP Growth Algorithm

- This algorithm needs to scan the database only twice when compared to Apriori which scans the transactions for each iteration
- **Pairing** is faster
- The database is stored in a compact version in memory
- It is efficient & scalable for mining both long & short frequent patterns

Disadvantages

- FP Tree is more cumbersome & difficult to build than Apriori because of pointers
- When the DBis large, the algorithm may not fit in the shared memory

FP Growth vs Apriori

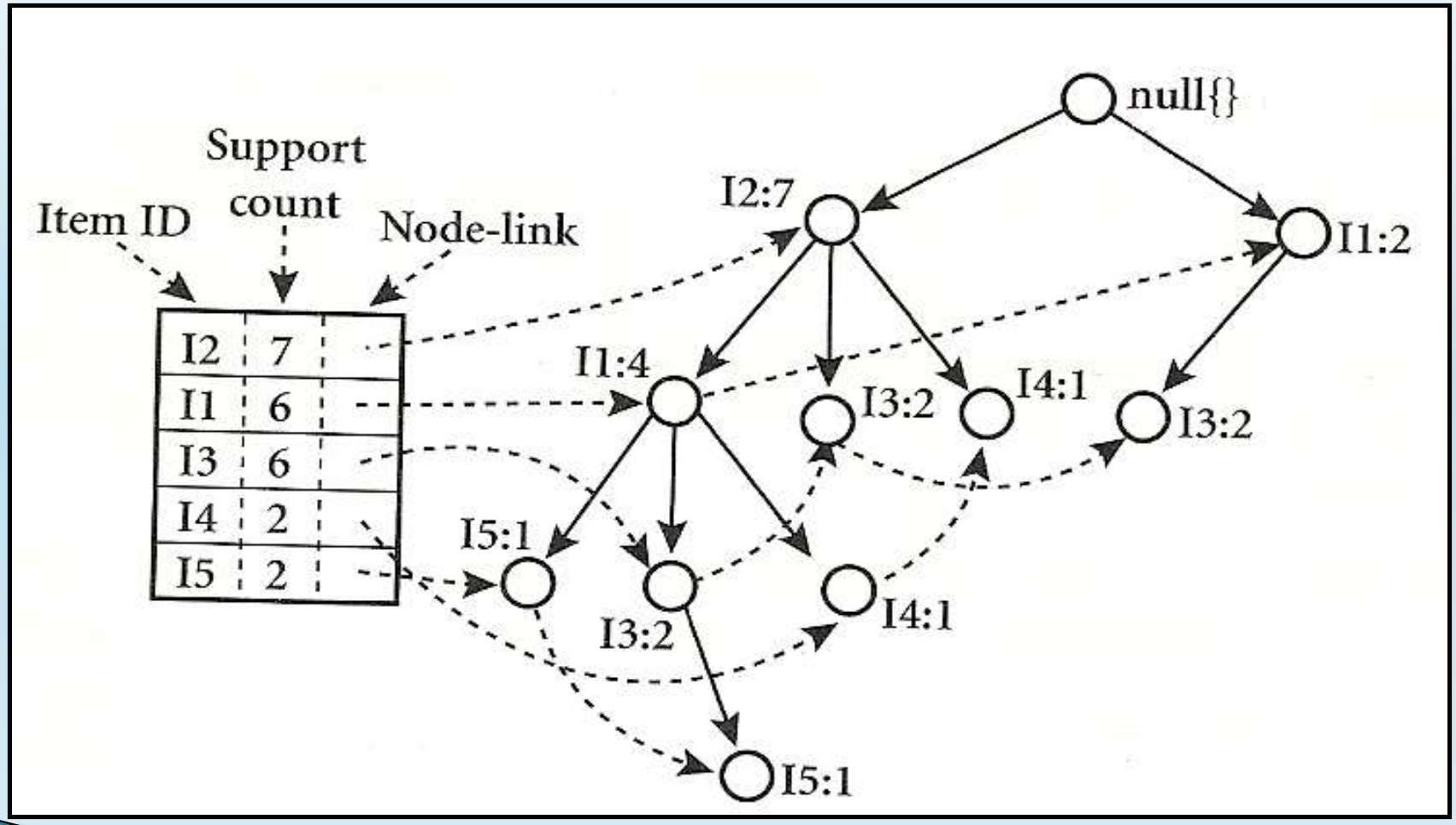
FP Growth	Apriori
Pattern Generation	
FP growth generates pattern by constructing a FP tree	Apriori generates pattern by pairing the items into singletons, pairs and triplets.
Candidate Generation	
There is no candidate generation	Apriori uses candidate generation
Process	
The process is faster as compared to Apriori. The runtime of process increases linearly with increase in number of itemsets.	The process is comparatively slower than FP Growth, the runtime increases exponentially with increase in number of itemsets
Memory Usage	
A compact version of database is saved	The candidates combinations are saved in memory

Example-3

- Let's have an example

T100	1,2,5
T200	2,4
T300	2,3
T400	1,2,4
T500	1,3
T600	2,3
T700	1,3
T800	1,2,3,5
T900	1,2,3

FP Tree


Item ID
Support count
Node-link
I2 7
I1 6
I3 6
I4 2
I5 2
null{}
I2:7
I1:2
I1:4
I3:2
I4:1
I3:2
I5:1
I3:2
I4:1
I5:1

Mining the FP tree

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}

Exercise-4

- A dataset has five transactions, let min-support=60% & min_confidence=80%
- Find all frequent itemsets using FP Tree

TID	Items_bought
T1	M, O, N, K, E, Y
T2	D, O, N, K , E, Y
T3	M, A, K, E
T4	M, U, C, K ,Y
T5	C, O, O, K, I ,E

Association Rules with FP Tree

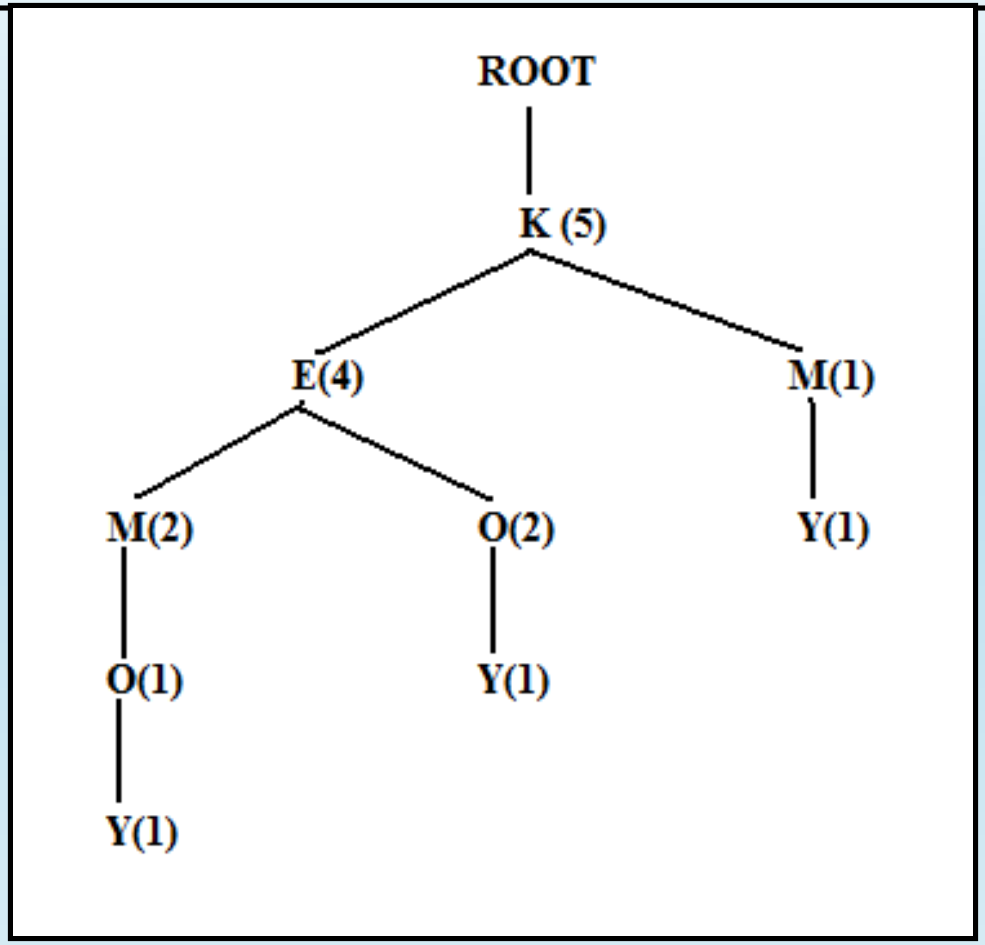
K:5

E:4

M:3

O:3

Y:3



Association Rules with FP Tree

Y: KEMO:1 KEO:1 KM:1

K:3 KY

O: KEM:1 KE:2

KE:3 KO EO KEO

M: KE:2 K:1

K:3 KM

E: K:4 KE


THAT'S
ALL FOLKS