UNIT 4

Association Analysis: Basic Concepts and Algorithms: Problem Definition, Frequent Item Set Generation, Apriori Principle, Apriori Algorithm, Rule Generation, Compact Representation of Frequent Itemsets, FP- Growth Algorithm. (Tan & Vipin)

INTRODUCTION

Que 1: What is association analysis?

Association analysis is used to discover interesting relationships hidden in large data sets.

The relationship hidden between data elements are represented in the form of association rules.

TID	Items
1	{Bread, Milk}
2	{Bread, Diapers, Beer, Eggs}
3	{Milk, Diapers, Beer, Cola}
4	{Bread, Milk, Diapers, Beer}
5	{Bread, Milk, Diapers, Cola}

Consider the dataset above the corresponding association rule is:

{milk} →{Bread} [s=2%,c=50%]

This type of analysis is called market basket analysis.

Applications of Association Analysis:

1) Market basket analysis: Buying trends of customers can be analyzed and retailers can make an offer between set of items to increase sales.
2) Earth science: Association between ocean, land, pollution and effect on temperature can be analyzed.
3) Medical diagnosis: Relation between food habits, exercise and decease can be analyzed.

Que 2: Write short note on:

a) **Binary representation:**
b) **Item set**
c) **Transaction width**
d) **support count**
e) **Support**
f) **Confidence**
g) **Frequent Itemset**
h) **Anti monotone property.**
i) **Support based pruning**
j) **Apriori principle**

a) Binary representation of association data:

TID	Bread	Milk	Diapers	Beer	Eggs	Cola
1	1	1	0	0	0	0
2	1	0	1	1	1	0
3	0	1	1	1	0	1
4	1	1	1	1	0	0
5	1	1	1	0	0	1

Each transaction is represented as '1's and'0's. Where '1' represents presence of an item in the transaction and '0' represent absence of an item in the transaction.

b) Item set: Consider I be set of items. In association rule, collection of item(s) forms an item-set.

{}→ Null item set

{brush, paste)→ 2 Itemset

c) Transaction width: Number of items in itemset is called transaction width.

d) Support count: Number of times the itemset is found in the transaction set. (Or) Number of transactions containing the itemset.

Consider the table below,

TID	Items
1	{Bread, Milk}
2	{Bread, Diapers, Beer, Eggs}
3	{Milk, Diapers, Beer, Cola}
4	{Bread, Milk, Diapers, Beer}
5	{Bread, Milk, Diapers, Cola}

Support count of itemset {Bread, Milk} is 3.
Because, {bread, Milk} is present in 3 transactions (1, 4 and 5).

e) Support: Support represents how often a rule is applicable to a dataset.

$$\text{Support, } s(X \longrightarrow Y) \quad = \quad \frac{\sigma(X \cup Y)}{N};$$

Where X and Y are items and N represents total number of transactions. Means, Out of all transactions (N) how many transactions contains both X and Y.

f) Confidence: Confidence represents, how often Y appear in transaction containing X.

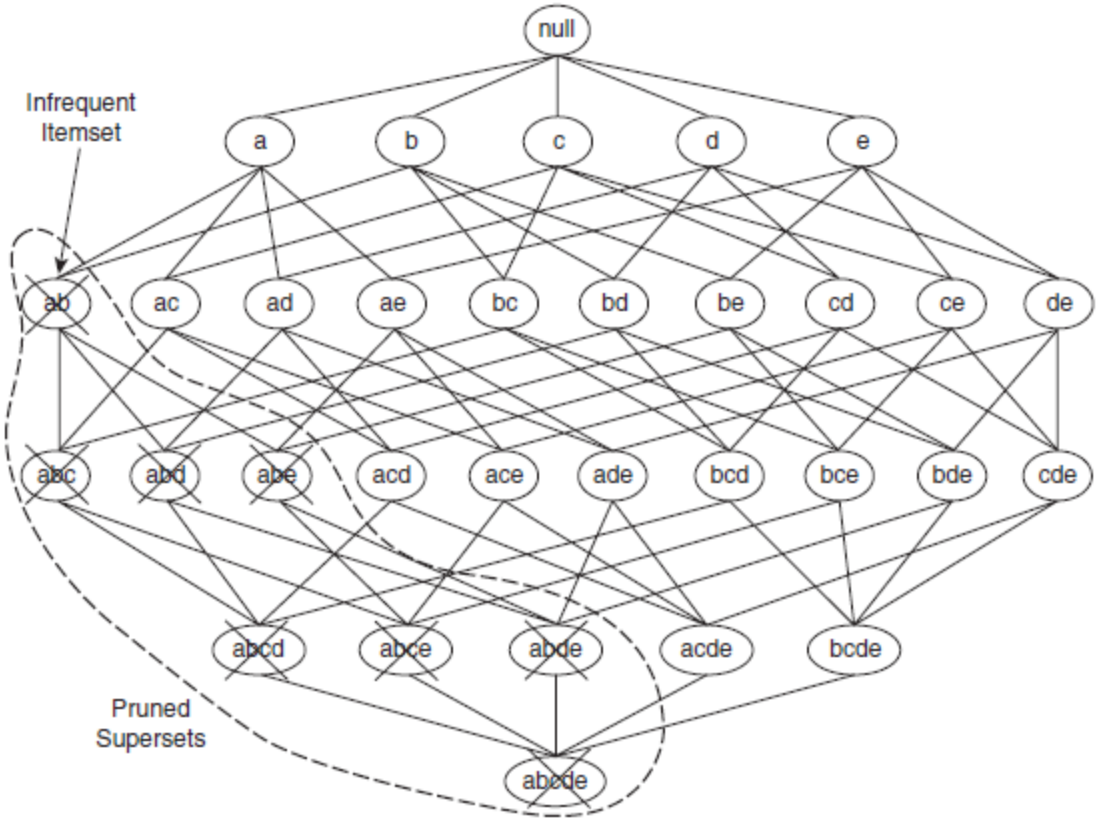
$$\text{Confidence, } c(X \longrightarrow Y) \quad = \quad \frac{\sigma(X \cup Y)}{\sigma(X)}.$$

Out of all transactions containing X, how many transactions contains Y.

g) Frequent itemset: An itemset is called frequent itemset if has minimum support and minimum confidence defined by the user.

h) Anti monotone property: Support of an itemset never exceeds its subset. For example, {brush, toothpaste} may have support count of 5. The item set (brush, toothpaste, bread} will never exceed 5.

i) Support based pruning: If an itemset is infrequent then all supersets containing the itemsets should also be infrequent.



If an item set {ab} is infrequent itemset, then all supersets containing both 'a' and 'b' are also infrequent and can be pruned(removed).

j) Apriori principle: If an itemset is frequent then, all its subsets are also frequent. Say for example, {c, d, e} is frequent itemset, then its subsets {c}, {d}, {e},{cd}, {de},{ce} should also be frequent.

Que 3: Explain Apriori algorithm in detail with example.
(Or)
How to find frequent item sets using candidate generation?

Algorithm 6.1 Frequent itemset generation of the *Apriori* algorithm.

1: $k = 1$.
2: $F_k = \{ i \mid i \in I \wedge \sigma(\{i\}) \geq N \times minsup\}$. {Find all frequent 1-itemsets}
3: **repeat**
4: $k = k + 1$.
5: $C_k = \text{apriori-gen}(F_{k-1})$. {Generate candidate itemsets}
6: **for** each transaction $t \in T$ **do**
7: $C_t = \text{subset}(C_k, t)$. {Identify all candidates that belong to t}
8: **for** each candidate itemset $c \in C_t$ **do**
9: $\sigma(c) = \sigma(c) + 1$. {Increment support count}
10: **end for**
11: **end for**
12: $F_k = \{ c \mid c \in C_k \wedge \sigma(c) \geq N \times minsup\}$. {Extract the frequent k-itemsets}
13: **until** $F_k = \emptyset$
14: Result $= \bigcup F_k$.

In the above algorithm,

K= Size of itemset

F_k= Frequent k^{th} item set.

I= Set of items (In below example, I={ I1,I2,I3,I4,I5})

i= Each element in I

σ{i}= Support count of items

Ck= Candidate set

T= Transaction database

t= individual transaction

Ct= candidates in Ck present in transaction data.

c= each candidate (or) each itemset in the candidate set.

σc = Support count of each candidate (or) Itemset.

Apriori_gen= A function call to generate candidate sets. (For example 2 item candidate set can be generated from one item frequent set).

Algorithm Explanation Step by step: //Optional

1. K is the size of itemset. Its value is initially set to 1 because 1-itemset should be generated first.
2. The support count of 1-itemset should be derived and the itemsets satisfying minimum support are kept and rests of them are deleted (F_1 is generated here).
    a. *Note: Here I is set of items (In below example, I={ I1,I2,I3,I4,I5})*
    b. *And i is individual item*
    c. *σ{i} is support count of items (Number of times the item i is found in our dataset)*
3. Repeat the process
4. Increment value of K to generate $k+1^{th}$ candidate set
5. After generating 1- item frequent set F_1, in step 2, Generate 2-Item candidate set using function call apriori_gen on F_1 .
    2- Item candidate set C_2 is generated from 1-item frequent set using one of the following approaches.
        i. *Brute force approach*
        ii. *F_{k-1}*F1 approach*
        iii. *F_{k-1} *F_{k-1} approach*
6. Consider each individual transaction (t) from our transaction dataset(T) and
7. Use function call 'subset' on C_k to generate itemsets present in our Transaction dataset. Let these itemsets be C_t.
    Note: delete itemsets from 2 item candidate sets whose support count is zero. And call rest of them as C_t
8. Out of all individual itemsets 'c' in 'C_t'
9. Find the support count of each candidate c in Ct. By scanning each transaction t and incrementing support count. $\sigma c = \sigma c+1$
10. End loop
11. End loop
12. Repeat step 4 to step 9 until biggest frequent item set is generated.
    a. *Support count for each candidate c is measured (σc). And candidates' c whose support count is greater than minimum support count is retained. This forms F_k*
13. This process is repeated until support count of all itemsets fall below minimum support count.
14. At the end, we have biggest frequent item sets.

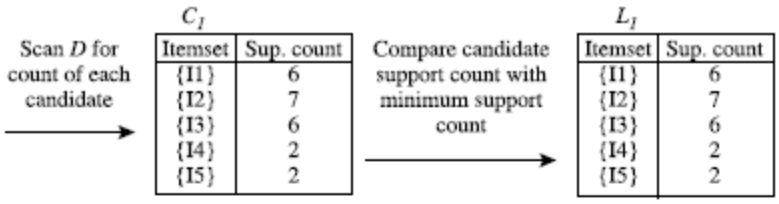
//

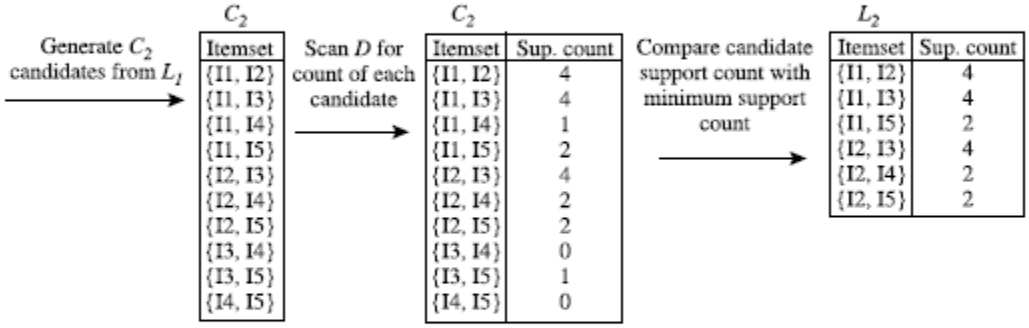
Example of apriori algorithm

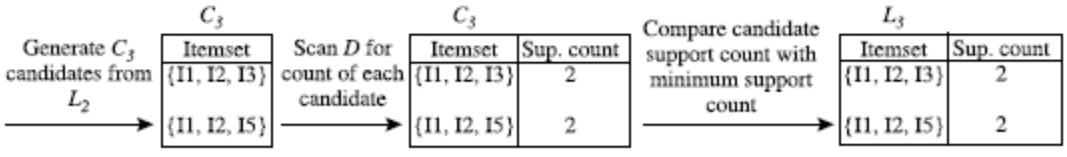
Consider a Transaction dataset T

TID	List of item_IDs
T100	I1, I2, I5
T200	I2, I4
T300	I2, I3
T400	I1, I2, I4
T500	I1, I3
T600	I2, I3
T700	I1, I3
T800	I1, I2, I3, I5
T900	I1, I2, I3

Apply apriori algorithm







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

Que 4: How to generate association rules from frequent item sets in apriori. (Or) What is confidence based pruning?

Once frequent itemsets are generated, then strong association rule can be generated by using confidence of the rule.

$$confidence(A \Rightarrow B) = P(B|A) = \frac{support\_count(A \cup B)}{support\_count(A)}.$$

Where,

support_count(AUB): Number of transactions containing both A & B together.

support_count(A): Number of transactions containing A.

Association rule generation is a two step process.

1) Generate all non empty subsets of a frequent itemset
2) For every subset calculate and find

$$\frac{\text{support\_count of frequent itemset}}{\text{support\_count of subset}} > Minimum\ confidence$$

TID	List of item_IDs
T100	I1, I2, I5
T200	I2, I4
T300	I2, I3
T400	I1, I2, I4
T500	I1, I3
T600	I2, I3
T700	I1, I3
T800	I1, I2, I3, I5
T900	I1, I2, I3

For example,

Consider a frequent itemset {I1, I2, I5}

possible subsets are:

{I1, I2}, {I1, I5}, {I2, I5}, {I1}, {I2}, {I5} and the possible association rules are:

$$I1 \wedge I2 \Rightarrow I5, \quad confidence = 2/4 = 50\%$$
$$I1 \wedge I5 \Rightarrow I2, \quad confidence = 2/2 = 100\%$$
$$I2 \wedge I5 \Rightarrow I1, \quad confidence = 2/2 = 100\%$$
$$I1 \Rightarrow I2 \wedge I5, \quad confidence = 2/6 = 33\%$$
$$I2 \Rightarrow I1 \wedge I5, \quad confidence = 2/7 = 29\%$$
$$I5 \Rightarrow I1 \wedge I2, \quad confidence = 2/2 = 100\%$$

Here, I1^I2 =>I5 Means,

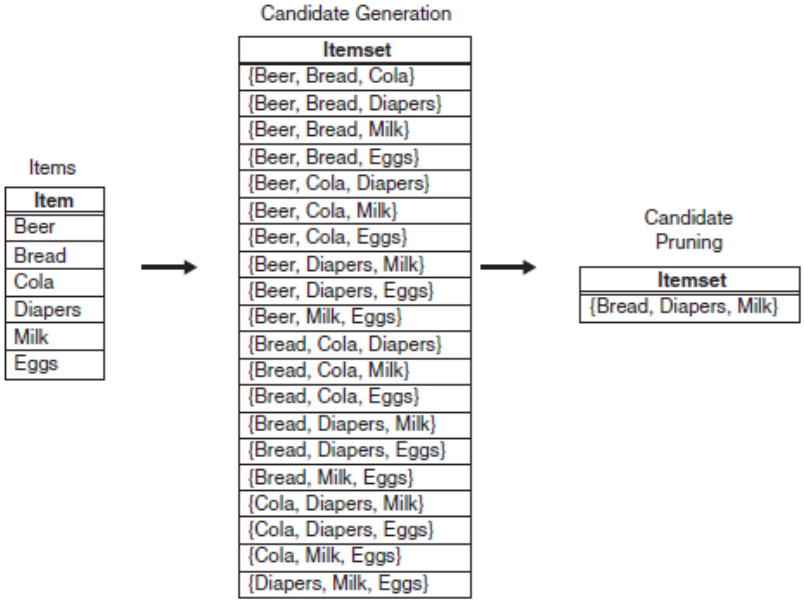
$$\frac{\text{Number of transaction containing I1,I2,I5}}{\text{Number of transactions containing I1,I2}} = 2/4 = 50\%$$

If, Minimum confidence is say, 70%, then 2nd, 3rd and 6th association rules will be retained.

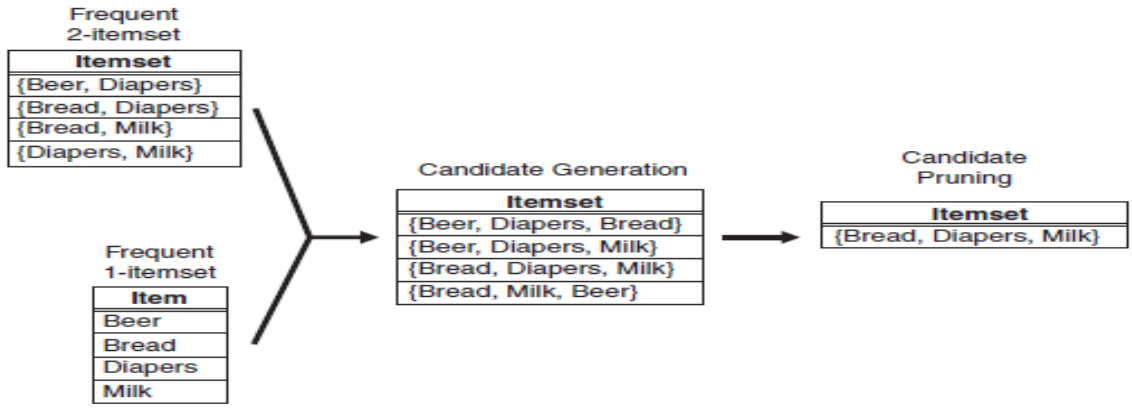
Que 5: What are the procedures for generating candidates in apriori algorithm?
(Or)
What are the techniques used in apriori_gen?

Ans:

Brute force method: All possible K- Itemsets are generated from k-1th item sets and infrequent k-item candidates are deleted.



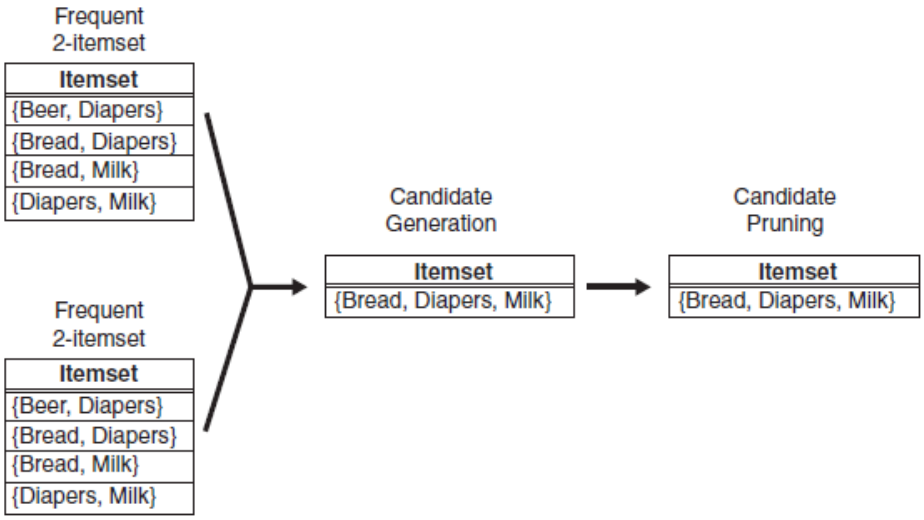
F_{k-1} * F_K method: As the name suggests, frequent 'K-1' item set is combined with 1 itemset to generate 'K' item-candidate set.



$F_{k-1} * F_{k-1}$ method: Frequent K-1th itemset is combined with itself to generate C_k.

$$\begin{aligned}\text{Join: } C_3 = L_2 \bowtie L_2 &= \{\{I1, I2\}, \{I1, I3\}, \{I1, I5\}, \{I2, I3\}, \{I2, I4\}, \{I2, I5\}\} \bowtie \\ &\quad \{\{I1, I2\}, \{I1, I3\}, \{I1, I5\}, \{I2, I3\}, \{I2, I4\}, \{I2, I5\}\} \\ &= \{\{I1, I2, I3\}, \{I1, I2, I5\}, \{I1, I3, I5\}, \{I2, I3, I4\}, \{I2, I3, I5\}, \{I2, I4, I5\}\}.\end{aligned}$$

In the above example two item sets are combined if starting elements in itemset are same and only last element in both item sets are different.



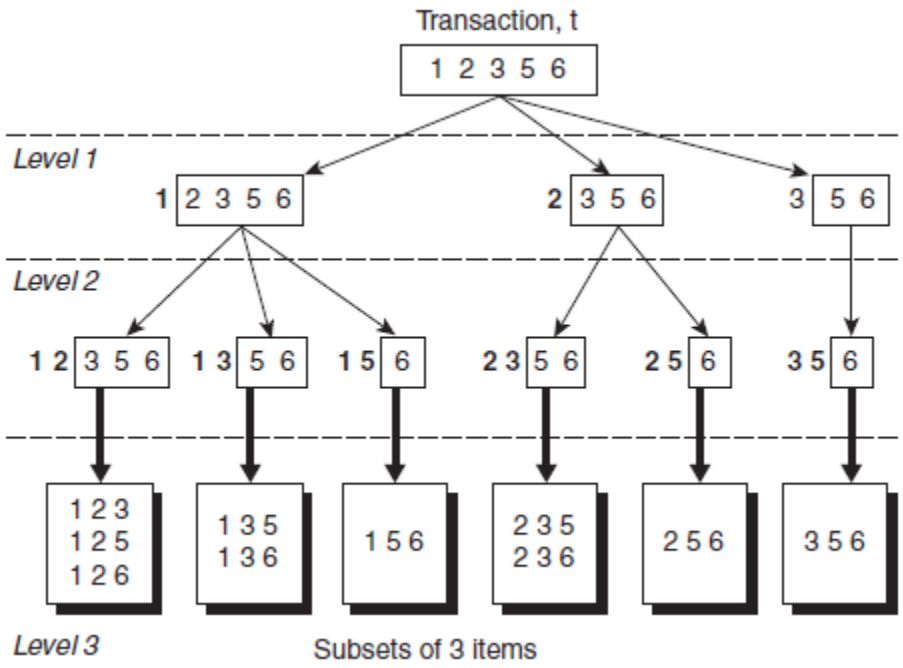
Que 6: What are the methods for generating support counts for candidates?
(Or)
Discuss about support counting using Hash Tree.

3 methods for generating support counts for candidates.

1) **Compare itemsets against each transaction**
2) **Enumerate the itemsets**
3) **Support counting using a hash tree.**

1) **Compare itemsets against each transaction:** Compare itemsets against each transaction and increment support count of the candidates contained in the transaction.

2) **Enumerate the itemsets:**

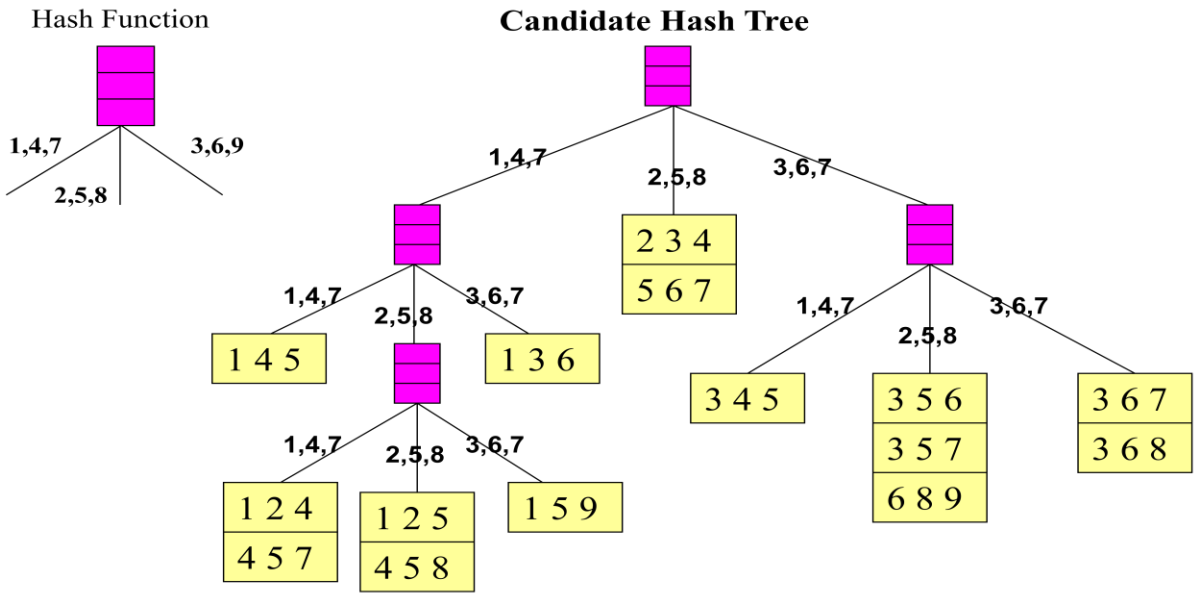


Arrange elements in increasing order of their occurrence in the transaction database.

In level 1, Take 1, 2, and 3 as prefix. And in level 2, take each set from level one and take the next elements are prefixes. This process repeats until 3 itemsets are generated.

3) Support counting using a hash tree:

{1 4 5}, {1 2 4}, {4 5 7}, {1 2 5}, {4 5 8}, {1 5 9}, {1 3 6}, {2 3 4}, {5 6 7}, {3 4 5}, {3 5 6}, {3 5 7}, {6 8 9}, {3 6 7}, {3 6 8}



Here, Items are stored in different buckets called as hash tables.

In hash tree, user needs to specify two things:

1) The hash function: here it is h(p)=p mod 3
2) Max leaf size: max number of itemsets stored in a leaf node (if number of candidate itemsets exceeds max leaf size, split the node)

For example,

1 mod 3=1	2 mod 3=2	3 mod 3=0
4 mod 3=1	5 mod 3=2	6 mod 3=0
7 mod 3=1	8 mod 3=2	9 mod 3=0

- In level 1
  - The itemsets containing 1, 4 and 7 as first elements are mapped to left side of the tree.

    - The itemsets containing 2, 5, and 8 as first elements are mapped to middle of the tree.
    - The itemsets containing 3, 6, and 9 as first element are mapped to right side of the tree.

- In level 2
    - Consider the leftmost sub tree, and apply the same hash function applied at level one.
    - The itemsets containing 1, 4 and 7 as second elements are mapped to left side of the tree.
    - The itemsets containing 2, 5, and 8 as second elements are mapped to middle of the tree.
    - The itemsets containing 3, 6, and 9 as second element are mapped to right side of the tree.

Repeat the same process for every sub tree until 3 itemsets are generated. Now, the candidate itemsets stored at leaf node are compared against each transaction. If a candidate is subset of transaction, its support count is incremented.

Que 7: Write the computational complexity of apriori algorithm.

- Choice of minimum support threshold
    - lowering support threshold results in more frequent itemsets
    - this may increase number of candidates and max length of frequent itemsets
- Dimensionality (number of items) of the data set
    - more space is needed to store support count of each item
    - if number of frequent items also increases, both computation and I/O costs may also increase
- Size of database

  - since Apriori makes multiple passes, run time of algorithm may increase with number of transactions
- Average transaction width
  - transaction width increases with denser data sets
  - This may increase max length of frequent itemsets and traversals of hash tree (number of subsets in a transaction increases with its width)

Que 8: What are the advantages and disadvantages of Apriori algorithm?

Disadvantages of Apriori

- The candidate generation could be extremely slow (pairs, triplets, etc.).
- The counting method iterates through all of the transactions each time.
- Constant items make the algorithm a lot heavier.
- Huge memory consumption

Advantages of Apriori

The Apriori Algorithm calculates more sets of frequent items.

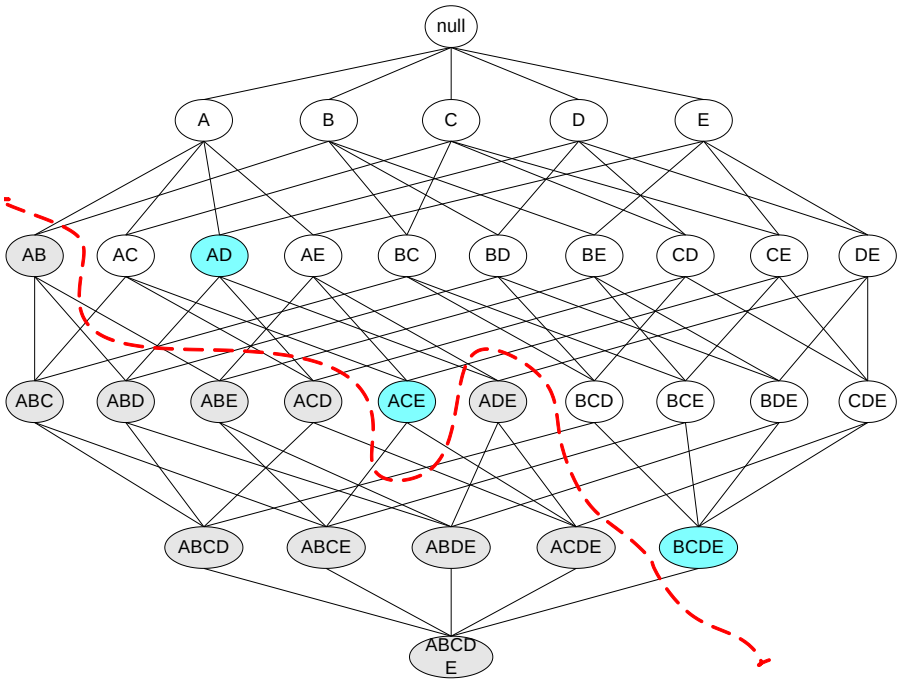
Que 9: Write a short note on:

1) Maximal frequent itemset

2) Closed itemset

3) Closed frequent itemset

Maximal frequent itemset: An itemset is maximal frequent if none of its immediate supersets is frequent.



In the above figure, consider 'AD' as frequent itemset, then if all supersets containing 'AD' are infrequent then 'AD' is called maximal frequent itemset.

Closed itemset: An itemset is closed if none of its immediate supersets has the same support as the itemset

Itemset	Support
{A}	4
{B}	5
{C}	3
{D}	4
{A,B}	4
{A,C}	2
{A,D}	3
{B,C}	3
{B,D}	4
{C,D}	3

- Consider itemset {A},
    - {A} has the support count of 4
    - And, {A,B} has also the support count of 4
    - So, {A} is not a closed itemset.

Closed frequent itemset: If an itemset is closed and its support count is greater than min_Sup, then it is called closed frequent item set.

Que: How to generate frequent itemsets without candidates?
(Or)
Explain FP Growth algorithm

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

1. Generation of candidate itemsets is expensive (in both space and time.
2. Support counting is expensive
    - Subset checking (computationally expensive)
    - Multiple Database scans (I/O)

FP-Growth: allows frequent itemset discovery without candidate itemset generation. This is a two step approach:

- Build a compact data structure called the FP-tree
- Extracts frequent itemsets directly from the FP-tree

Algorithm for FP-Tree

Step 1:

- Scan DB once, find frequent 1-itemset
- Sort frequent items in frequency descending order, f-list
- Scan DB again, construct FP-tree
    - FP-Growth reads 1 transaction at a time and maps it to a path
    - Fixed order is used, so paths can overlap when transactions share items

Step 2:

- Generating conditional pattern base
    - Starting at the frequent item header table in the FP-tree
    - Traverse the FP-tree by following the link of each frequent item
    - Accumulate all of *transformed prefix paths* of the item to form *its* conditional pattern base
- For each conditional pattern-base

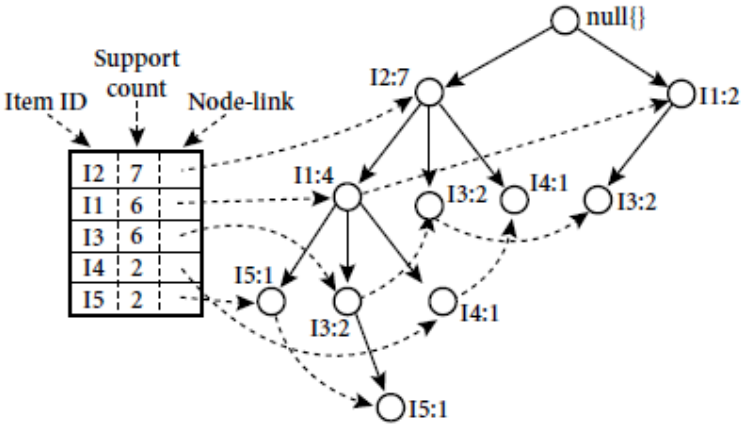
- Accumulate the count for each item in the conditional pattern base
- Construct the FP-tree for the frequent items of the pattern base

- Frequent itemsets is, all the combinations of its sub-paths, each of which is a frequent pattern

Example of FP Growth

TID	List of item_IDs
T100	I1, I2, I5
T200	I2, I4
T300	I2, I3
T400	I1, I2, I4
T500	I1, I3
T600	I2, I3
T700	I1, I3
T800	I1, I2, I3, I5
T900	I1, I2, I3

The first scan of the database is the same as Apriori, which derives the set of frequent items (1-itemsets) and their support counts (frequencies). Let the minimum support count be 2. The set of frequent items is sorted in the order of descending support count. This resulting set or *list* is denoted *L*. Thus, we have *L* = {{I2: 7}, {I1: 6}, {I3: 6}, {I4: 2}, {I5: 2}}.



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}

Mining of the FP-tree is summarized in the above table.

//Optional

- We first consider I5, which is the last item in *L*, rather than the first. I5 occurs in two branches of the FP-tree. The paths formed by these branches are {(I2, I1, I5: 1) and (I2, I1, I3, I5: 1)}. Therefore, considering I5 as a suffix, its corresponding two prefix paths are (I2, I1: 1) and (I2, I1, I3: 1), which form its conditional pattern base. Its conditional FP-tree contains only a single path, {I2: 2, I1: 2}; I3 is not included because its support count of 1 is less than the minimum support count. The single path generates all the combinations of frequent patterns: {I2, I5: 2}, {I1, I5: 2}, {I2, I1, I5: 2}.

- For I4, its two prefix paths form the conditional pattern base, {{I2 I1: 1}, {I2: 1}}, which generates a single-node conditional FP-tree, {I2: 2}, and derives one frequent pattern, {I2, I4: 2}.

- Similar to the above analysis, I3's conditional pattern base is {{I2, I1: 2}, {I2: 2}, {I1: 2}}. its conditional FP-tree has two branches, (I2: 4, I1: 2) and (I1: 2), as shown in Figure, which generates the set of patterns, {{I2, I3: 4}, {I1, I3: 4}, {I2, I1, I3: 2}}.

- Finally, I1's conditional pattern base is {{I2: 4}}, whose FP-tree contains only one node, (I2: 4), which generates one frequent pattern, {I2, I1: 4}.

Another version of algorithm (As in Kamber text book, students can learn any one of the two algorithm)

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

Input:

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

Output: The complete set of frequent patterns.

Method:

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

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

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

Que: What are the advantages and disadvantages of FP-Tree?

Advantages:

- no candidate generation, no candidate test
- compressed database: FP-tree structure
- no repeated scan of entire database
- basic ops—counting local freq items and building sub FP-tree, no pattern search and matching
- leads to focused search of smaller databases

Disadvantage:

Parallelization of FP Growth technique may end with bottlenecks in shared memory.

Que: Compare and contrast Apriori and FP Growth.

Algorithm	Technique	Runtime	Memory usage	Parallelizability
Apriori	Generate singletons, pairs, triplets, etc.	Candidate generation is extremely slow. Runtime increases exponentially depending on the number of different items.	Saves singletons, pairs, triplets, etc.	Candidate generation is very parallelizable
FP-Growth	Insert sorted items by frequency into a pattern tree	Runtime increases linearly, depending on the number of transactions and items	Stores a compact version of the database.	Data are very inter-dependent, each node needs the root.