BITS Pilani

BITS Pilani
Hyderabad Campus

Dr. Manik Gupta
Assistant Professor
Department of CSIS

innovate achieve lead

BITS Pilani
Hyderabad Campus

Data Mining (CS F415)
Lecture 8 – Association Rule Mining

Thursday, 23rd January 2020

Today's Agenda

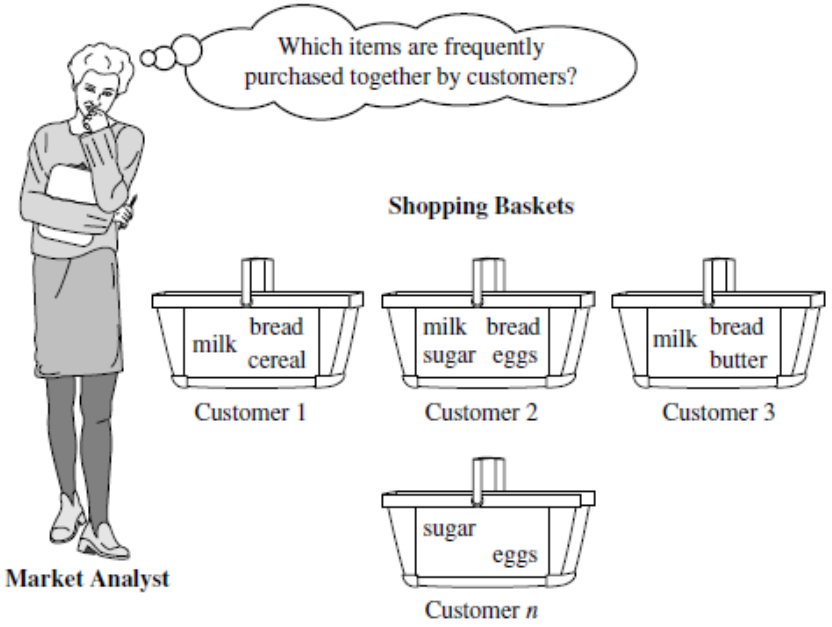
- Mining Frequent Patterns
- Associations Rules

Introduction

- **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*.
  - A *substructure* can refer to different structural forms, such as subgraphs, subtrees, or sublattices. If a substructure occurs frequently, it is called a (*frequent*) *structured pattern*.
- **Finding frequent patterns** plays an essential role in mining associations, correlations, and many other interesting relationships among data.
- **Frequent pattern mining** is an important DM task!

Market Basket Analysis: A Motivating Example

- Typical example of frequent itemset mining is **market basket analysis**. This process analyzes customer buying habits by **finding associations** between the different items that customers place in their "shopping baskets"



How would you represent market basket data?

- Binary format
  - Each row corresponds to a transaction and each column corresponds to an item.
  - An item can be treated as a binary variable whose value is one if the item is present and zero otherwise
- Asymmetric Binary variable!

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

	Beer	Bread	Milk	Diaper	Eggs	Coke
T_1	0	1	1	0	0	0
T_2	1	1	0	1	1	0
T_3	1	0	1	1	0	1
T_4	1	1	1	1	0	0
T_5	0	1	1	1	0	1

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.
- These rules represent patterns containing items that are frequently associated or purchased together.

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} → {Eggs,Coke},
{Beer, Bread} → {Milk},

Definition - Frequent Itemset

- **Itemset**
  - A collection of one or more items
    - Example: {Milk, Bread, Diaper}
  - k-itemset
    - An itemset that contains k items
- **Support count (σ)**
  - Frequency of occurrence of an itemset
  - E.g. σ({Milk, Bread, Diaper}) = 2
- **Support**
  - Fraction of transactions that contain an itemset
  - E.g. s({Milk, Bread, Diaper}) = 2/5
- **Frequent Itemset**
  - An itemset whose support is greater than or equal to 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

Definition - Association Rule

- Association Rule
  - An implication expression of the form $X \rightarrow Y$, where X and Y are itemsets
  - Example:
    {Milk, Diaper} $\rightarrow$ {Beer}

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

- Rule Evaluation Metrics
  - Support (s)
    - Fraction of transactions that contain both X and Y
  - Confidence (c)
    - Measures how often items in Y appear in transactions that contain X

Example:

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

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

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

Rule Interestingness

innovate achieve lead

- Rule **support** and **confidence** are two measures of rule interestingness. They respectively reflect the **usefulness and certainty** of discovered rules.
  - A support of 2% for 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 antivirus software.
- Typically, association rules are considered interesting if they satisfy both a **minimum support threshold** and a **minimum confidence threshold**

Support and Confidence

$$\text{SUPPORT} = \frac{\text{number of transactions containing X and Y}}{\text{total number of transactions}}$$

$$\text{CONFIDENCE} = \frac{\text{number of transactions containing X and Y}}{\text{number of transactions containing X}}$$

Example – Find s and c for the given Rules

- Example of Rules:
  - {Milk,Diaper} → {Beer}
  - (s=0.4, c=0.67)
  - {Milk,Beer} → {Diaper}
  - (s=0.4, c=1.0)
  - {Diaper,Beer} → {Milk}
  - (s=0.4, c=0.67)
  - {Beer} → {Milk,Diaper}
  - (s=0.4, c=0.67)
  - {Diaper} → {Milk,Beer}
  - (s=0.4, c=0.5)
  - {Milk} → {Diaper,Beer}
  - (s=0.4, c=0.5)

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

Mining Association Rules

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 Rules:

{Milk,Diaper} → {Beer} (s=0.4, c=0.67)
{Milk,Beer} → {Diaper} (s=0.4, c=1.0)
{Diaper,Beer} → {Milk} (s=0.4, c=0.67)
{Beer} → {Milk,Diaper} (s=0.4, c=0.67)
{Diaper} → {Milk,Beer} (s=0.4, c=0.5)
{Milk} → {Diaper,Beer} (s=0.4, c=0.5)

Observations:

- All the above rules **are binary partitions of the same itemset** {Milk, Diaper, Beer}
- Rules originating from the same itemset have identical support but can have different confidence
- Thus, we may **decouple the support and confidence** requirements

Mining Association Rules

- Given a set of transactions T, the goal of association rule mining is to find all rules having
  - support ≥ *minsup* threshold
  - confidence ≥ *minconf* threshold

  What happens when minsup is less ?

- Brute-force approach
  - List all possible association rules
  - Compute the support and confidence for each rule
  - Prune rules that fail the *minsup* and *minconf* thresholds
  - Computationally prohibitive!

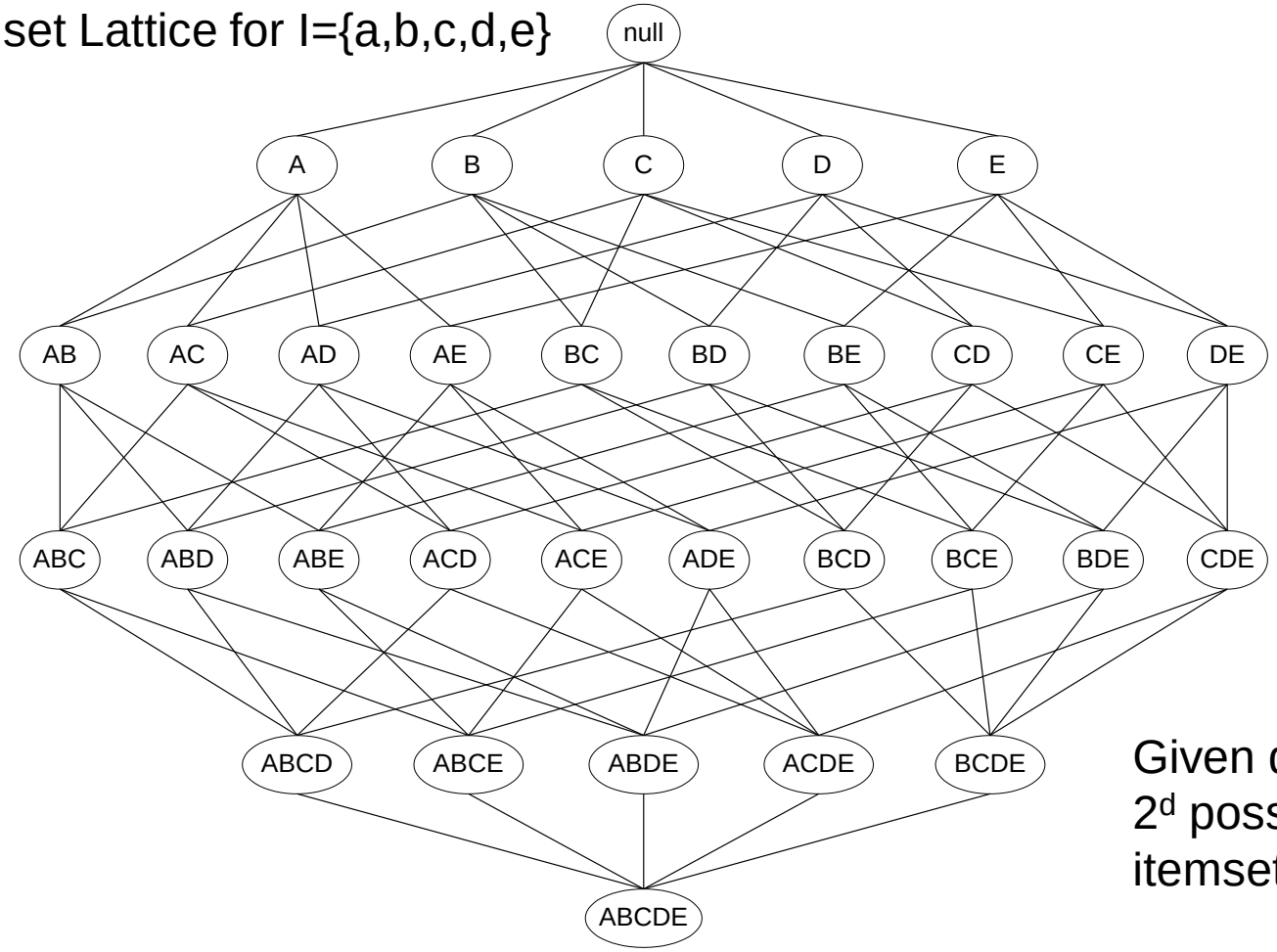
Mining Association Rules

- Two-step approach
  - **Frequent Itemset Generation**
    - Generate all itemsets whose support ≥ minsup
  - **Rule Generation**
    - Generate high confidence rules from each frequent itemset, where each rule is a binary partitioning of a frequent itemset
- Frequent itemset generation is still computationally expensive!!

Frequent Itemset Generation

Lattice structure can be used to enumerate the list of possible itemsets.

Itemset Lattice for I={a,b,c,d,e}

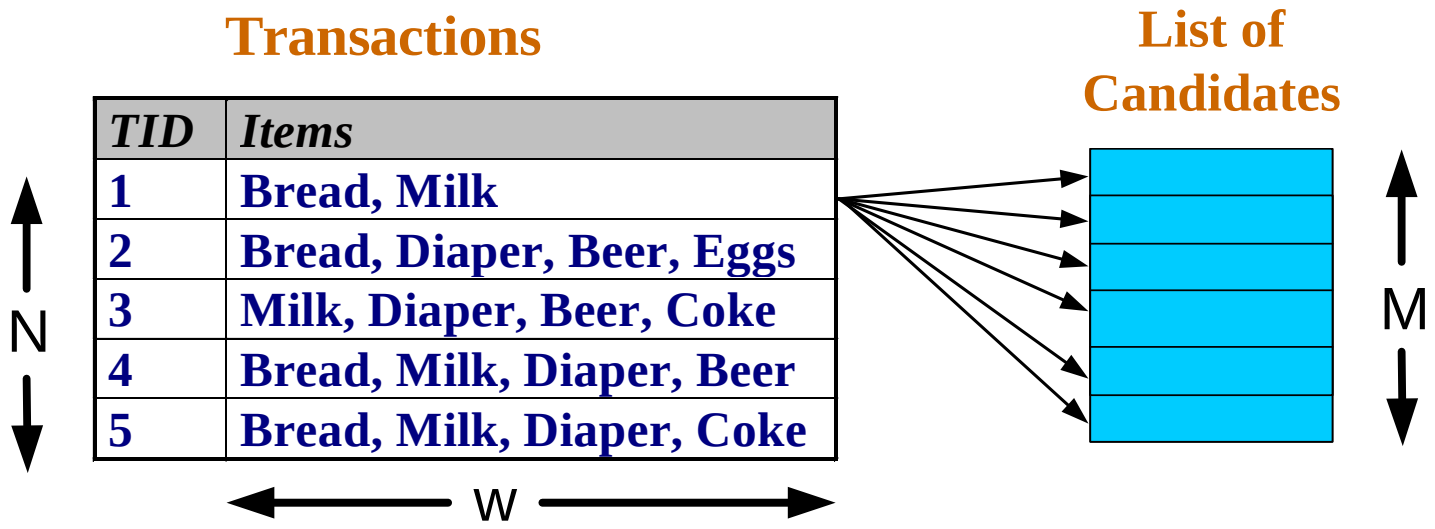


Given d items, there are 2^d possible candidate itemsets

What would be a brute force approach for finding frequent itemsets?

- Determine the support count for every candidate itemset in the lattice structure!

Brute-force approach for finding all frequent itemsets



- Compare each candidate itemset against each transaction
- Complexity?
  - For **M** candidates and **N** transactions, the complexity is ~**O(MNw)** => Expensive since M = 2^d !!!

Speeding-up the brute-force algorithm

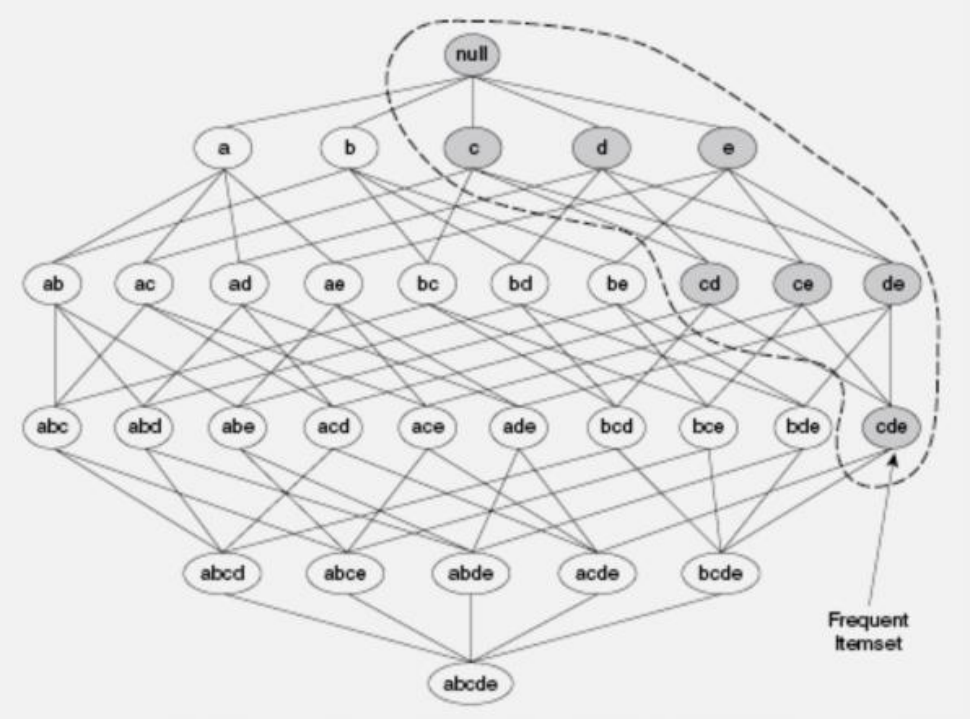
- Reduce the number of candidate itemsets (M)
  - Complete search: $M=2^d$
  - Use pruning techniques to reduce M
- Reduce the number of comparisons (NM)
  - Use efficient data structures to store the candidates or transactions
  - No need to match every candidate against every transaction

Frequent Itemset Mining Methods

- **Apriori**, the basic algorithm for finding frequent itemsets and generates strong association rules from frequent itemsets.
- **Pattern-growth methods** for mining frequent itemsets that confine the subsequent search space to only the data sets containing the current frequent itemsets.
- Methods for mining frequent itemsets that take advantage of the vertical data format.

Reduce the number of candidate itemset

- Apriori principle
  - If an itemset is frequent, then all of its subsets must also be frequent



  - If {c,d,e} is frequent, then all subsets of this itemset are also frequent

Anti-Monotone property of Support measure

- Apriori principle holds due to the following property of the support measure:

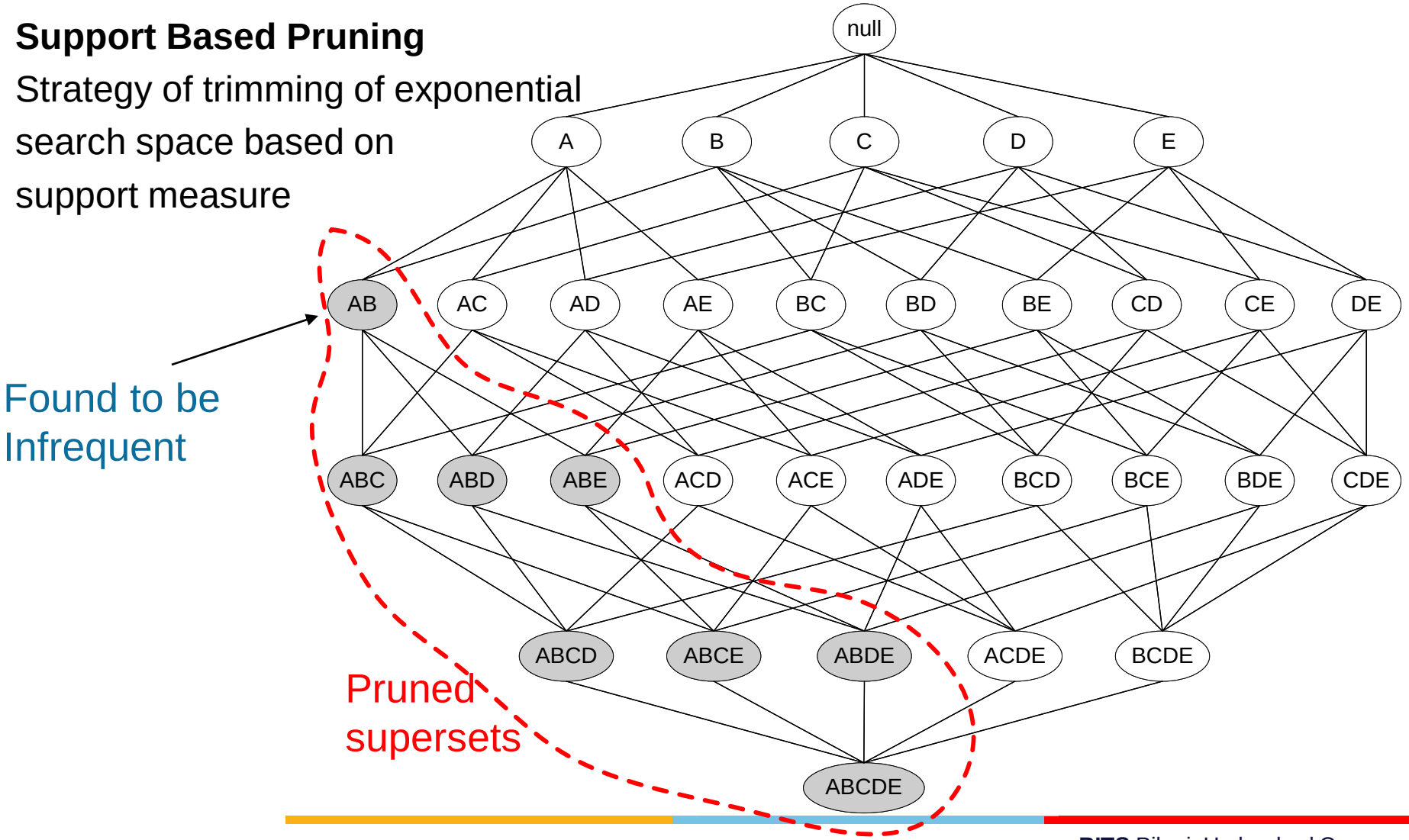
$$\forall X, Y : (X \subseteq Y) \Rightarrow s(X) \geq s(Y)$$

  - The support of an itemset ***never exceeds*** the support of its subsets
  - This is known as the ***anti-monotone*** property of the support measure

Illustrating Apriori Principle

Support Based Pruning

Strategy of trimming of exponential search space based on support measure



Illustrating Apriori Principle

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

Items (1-itemsets)

Item	Count
Bread	4
Coke	2
Milk	4
Beer	3
Diaper	4
Eggs	1

Minimum Support = 3

Illustrating Apriori Principle

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

Items (1-itemsets)

Item	Count
Bread	4
Coke	2
Milk	4
Beer	3
Diaper	4
Eggs	1

Minimum Support = 3

Illustrating Apriori Principle

Item	Count
Bread	4
Coke	2
Milk	4
Beer	3
Diaper	4
Eggs	1

Items (1-itemsets)

Itemset
{Bread,Milk}
{Bread, Beer }
{Bread,Diaper}
{Beer, Milk}
{Diaper, Milk}
{Beer,Diaper}

Pairs (2-itemsets)

(No need to generate candidates involving Coke or Eggs)

Minimum Support = 3

Illustrating Apriori Principle

Item	Count
Bread	4
Coke	2
Milk	4
Beer	3
Diaper	4
Eggs	1

Items (1-itemsets)

Itemset	Count
{Bread,Milk}	3
{Beer, Bread}	2
{Bread,Diaper}	3
{Beer,Milk}	2
{Diaper,Milk}	3
{Beer,Diaper}	3

Pairs (2-itemsets)

(No need to generate candidates involving Coke or Eggs)

Minimum Support = 3

Illustrating Apriori Principle

Item	Count
Bread	4
Coke	2
Milk	4
Beer	3
Diaper	4
Eggs	1

Items (1-itemsets)

Itemset	Count
{Bread,Milk}	3
{Bread,Beer}	2
{Bread,Diaper}	3
{Milk,Beer}	2
{Milk,Diaper}	3
{Beer,Diaper}	3

Pairs (2-itemsets)

(No need to generate candidates involving Coke or Eggs)

Minimum Support = 3

Triplets (3-itemsets)

Itemset	Count
{ Beer, Diaper, Milk}	2
{ Beer,Bread, Diaper}	2
{Bread, Diaper, Milk}	2
{Beer, Bread, Milk}	1

If every subset is considered,
${}^6C_1 + {}^6C_2 + {}^6C_3 = 41$
With support-based pruning,
${}^6C_1 + {}^4C_2 + {}^4C_3$
$6 + 6 + 4 = 16$
$6 + 6 + 1 = 13$

Apriori Algorithm

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

- L_k: frequent k-itemsets
- C_k: candidate k-itemsets

Algorithm

- Let k=1
- Generate L_1 = {frequent 1-itemsets}
- Repeat until L_k is empty
  - **Candidate Generation**: Generate C_{k+1} from L_k
  - **Candidate Pruning**: Prune candidate itemsets in C_{k+1} containing subsets of length k that are infrequent
  - **Support Counting**: Count the support of each candidate in C_{k+1} by scanning the DB
  - **Candidate Elimination**: Eliminate candidates in C_{k+1} that are infrequent, leaving only those that are frequent => L_{k+1}

Exercise

Customer ID	Transaction ID	Items Bought
1	0001	$\{a, d, e\}$
1	0024	$\{a, b, c, e\}$
2	0012	$\{a, b, d, e\}$
2	0031	$\{a, c, d, e\}$
3	0015	$\{b, c, e\}$
3	0022	$\{b, d, e\}$
4	0029	$\{c, d\}$
4	0040	$\{a, b, c\}$
5	0033	$\{a, d, e\}$
5	0038	$\{a, b, e\}$

a) Compute the support for itemsets *{e}*, *{b, d}*, and *{b, d, e}* by treating each transaction ID as a market basket.
b) Use the results in part (a) to compute the confidence for the association rules *{b, d}* → *{e}* and *{e}* → *{b, d}*.

Solution

Customer ID	Transaction ID	Items Bought
1	0001	$\{a, d, e\}$
1	0024	$\{a, b, c, e\}$
2	0012	$\{a, b, d, e\}$
2	0031	$\{a, c, d, e\}$
3	0015	$\{b, c, e\}$
3	0022	$\{b, d, e\}$
4	0029	$\{c, d\}$
4	0040	$\{a, b, c\}$
5	0033	$\{a, d, e\}$
5	0038	$\{a, b, e\}$

$$s(\{e\}) = \frac{8}{10} = 0.8$$

$$s(\{b, d\}) = \frac{2}{10} = 0.2$$

$$s(\{b, d, e\}) = \frac{2}{10} = 0.2$$

$$c(bd \longrightarrow e) = \frac{0.2}{0.2} = 100\%$$

$$c(e \longrightarrow bd) = \frac{0.2}{0.8} = 25\%$$

To Explore

- Think and explore other applications of association analysis.

Apriori Algorithm

- L_k: frequent k-itemsets
- C_k: candidate k-itemsets

Algorithm

- Let k=1
- Generate L_1 = {frequent 1-itemsets}
- Repeat until L_k is empty
  - **Candidate Generation**: Generate C_{k+1} from L_k
  - **Candidate Pruning**: Prune candidate itemsets in C_{k+1} containing subsets of length k that are infrequent
  - **Support Counting**: Count the support of each candidate in C_{k+1} by scanning the DB
  - **Candidate Elimination**: Eliminate candidates in C_{k+1} that are infrequent, leaving only those that are frequent => L_{k+1}

Candidate Generation: $L_{k-1} \bowtie L_{k-1}$ Method

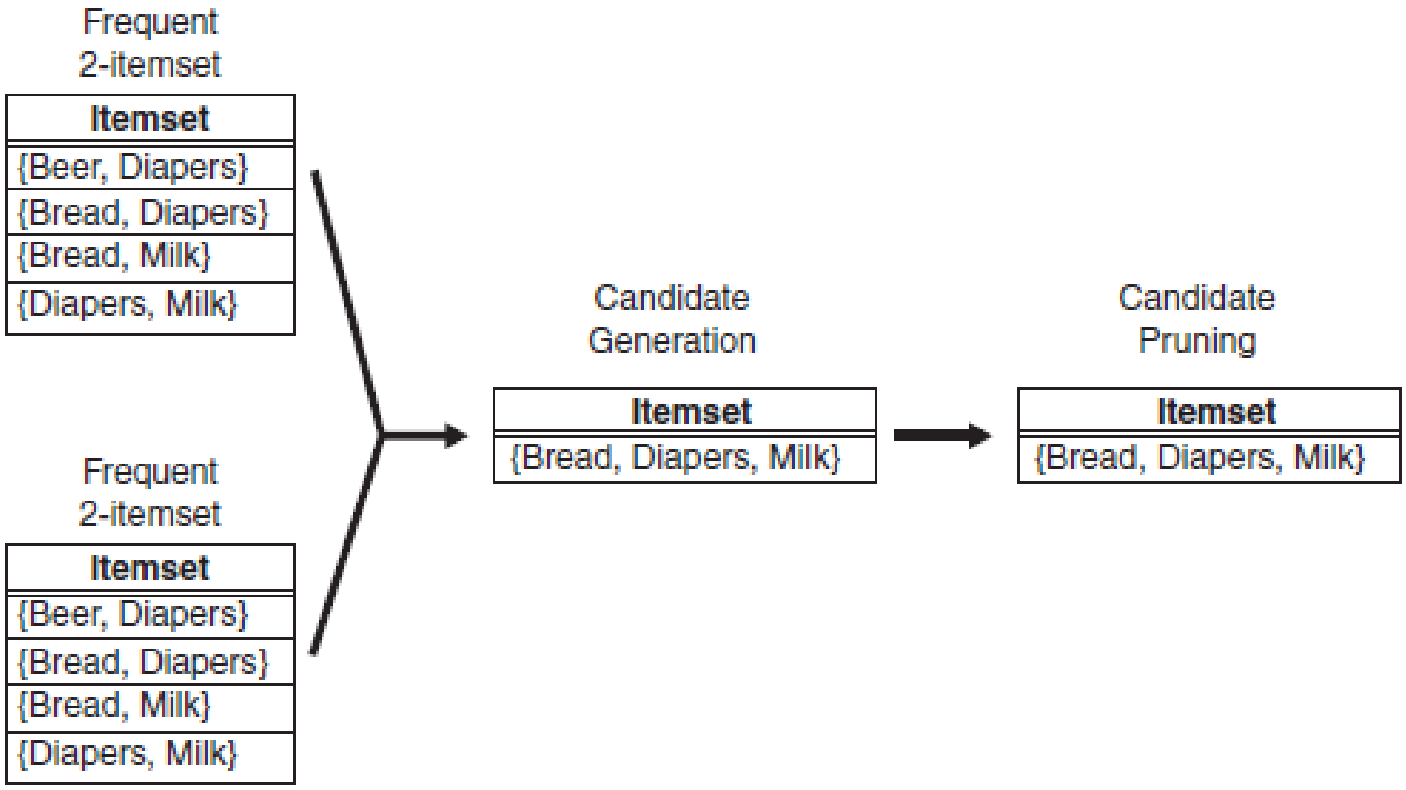


Figure 6.8. Generating and pruning candidate k-itemsets by merging pairs of frequent $(k-1)$-itemsets.

Candidate Generation: $L_{k-1} \bowtie L_{k-1}$ Method

- Merge two frequent (k-1)-itemsets if their first (k-2) items are identical

- L_3 = {ABC,ABD,ABE,ACD,BCD,BDE,CDE}
  - Merge(**AB**C, **AB**D) = **AB**CD
  - Merge(**AB**C, **AB**E) = **AB**CE
  - Merge(**AB**D, **AB**E) = **AB**DE

  - Do not merge(**A**BD,**A**CD) because they share only prefix of length 1 instead of length 2

Candidate Pruning

- Let L_3 = {ABC,ABD,ABE,ACD,BCD,BDE,CDE} be the set of frequent 3-itemsets
- C_4 = {ABCD,ABCE,ABDE} is the set of candidate 4-itemsets generated (from previous slide)
- Candidate pruning
  - Prune ABCE because ACE and BCE are infrequent
  - Prune ABDE because ADE is infrequent
- After candidate pruning: L_4 = {ABCD}

Apriori Example

- Illustrate the Apriori algorithm for finding frequent itemsets in *D*

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

Apriori Example

- In the first iteration of the algorithm, each item is a member of the set of candidate 1-itemsets, C_1. The algorithm simply scans all of the transactions to count the number of occurrences of each item.
- Suppose that the minimum support count required is 2, that is, *min_sup* = 2. The set of frequent 1-itemsets, L_1, can then be determined. It consists of the candidate 1-itemsets satisfying minimum support. In the example, all of the candidates in C_1 satisfy minimum support.

Scan D for count of each candidate →

C_1

Itemset	Sup. count
{I1}	6
{I2}	7
{I3}	6
{I4}	2
{I5}	2

Compare candidate support count with minimum support count →

L_1

Itemset	Sup. count
{I1}	6
{I2}	7
{I3}	6
{I4}	2
{I5}	2

Apriori Example

- To discover the set of frequent 2-itemsets, L_2, the algorithm uses the join $L_1 \bowtie L_1$ to generate a candidate set of 2-itemsets, C_2. Note that **no candidates are removed from C_2 during the prune step** because **each subset of the candidates is also frequent**.
- Next, the transactions in *D* are scanned and the support count of each candidate itemset in C_2 is accumulated, as shown below.
- The set of frequent 2-itemsets, L_2, is then determined, consisting of those candidate 2-itemsets in C_2 having minimum support.

Generate C_2 candidates from L_1 →

C_2

Itemset
{I1, I2}
{I1, I3}
{I1, I4}
{I1, I5}
{I2, I3}
{I2, I4}
{I2, I5}
{I3, I4}
{I3, I5}
{I4, I5}

Scan *D* for count of each candidate →

C_2

Itemset	Sup. count
{I1, I2}	4
{I1, I3}	4
{I1, I4}	1
{I1, I5}	2
{I2, I3}	4
{I2, I4}	2
{I2, I5}	2
{I3, I4}	0
{I3, I5}	1
{I4, I5}	0

Compare candidate support count with minimum support count →

L_2

Itemset	Sup. count
{I1, I2}	4
{I1, I3}	4
{I1, I5}	2
{I2, I3}	4
{I2, I4}	2
{I2, I5}	2

Apriori Example

- The next step is the generation of the set of the candidate 3-itemsets, C_3.
- From the join step, we first get $C_3 = L_2 \bowtie L_2$ = {{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 can determine that the four latter candidates cannot possibly be frequent. We therefore remove them from C_3, thereby saving the effort of unnecessarily obtaining their counts during the subsequent scan of D to determine L_3.
- Note that when given a candidate k-itemset, we only need to check if its (k − 1)-subsets are frequent since the Apriori algorithm uses a level-wise search strategy.

Apriori Example – Generation and Pruning of candidate 3-itemsets

(**a**) Join: $C_3 = L_2 \bowtie L_2 = \{\{I1, I2\}, \{I1, I3\}, \{I1, I5\}, \{I2, I3\}, \{I2, I4\}, \{I2, I5\}\}$
$\bowtie \{\{I1, I2\}, \{I1, I3\}, \{I1, I5\}, \{I2, I3\}, \{I2, I4\}, \{I2, I5\}\}$
$= \{\{I1, I2, I3\}, \{I1, I2, I5\}, \{I1, I3, I5\}, \{I2, I3, I4\}, \{I2, I3, I5\}, \{I2, I4, I5\}\}.$

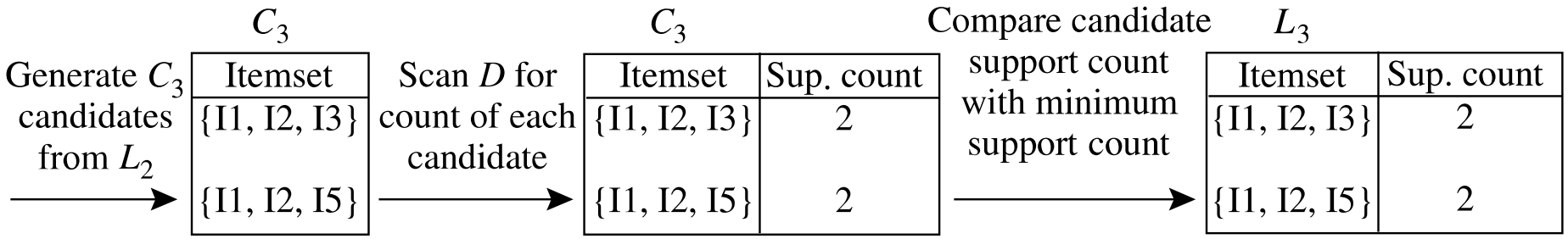
(**b**) Prune using the Apriori property: All nonempty subsets of a frequent itemset must also be frequent. Do any of the candidates have a subset that is not frequent?

- The 2-item subsets of {I1, I2, I3} are {I1, I2}, {I1, I3}, and {I2, I3}. All 2-item subsets of {I1, I2, I3} are members of L_2. Therefore, keep {I1, I2, I3} in C_3.
- The 2-item subsets of {I1, I2, I5} are {I1, I2}, {I1, I5}, and {I2, I5}. All 2-item subsets of {I1, I2, I5} are members of L_2. Therefore, keep {I1, I2, I5} in C_3.
- The 2-item subsets of {I1, I3, I5} are {I1, I3}, {I1, I5}, and {I3, I5}. {I3, I5} is not a member of L_2, and so it is not frequent. Therefore, remove {I1, I3, I5} from C_3.
- The 2-item subsets of {I2, I3, I4} are {I2, I3}, {I2, I4}, and {I3, I4}. {I3, I4} is not a member of L_2, and so it is not frequent. Therefore, remove {I2, I3, I4} from C_3.
- The 2-item subsets of {I2, I3, I5} are {I2, I3}, {I2, I5}, and {I3, I5}. {I3, I5} is not a member of L_2, and so it is not frequent. Therefore, remove {I2, I3, I5} from C_3.
- The 2-item subsets of {I2, I4, I5} are {I2, I4}, {I2, I5}, and {I4, I5}. {I4, I5} is not a member of L_2, and so it is not frequent. Therefore, remove {I2, I4, I5} from C_3.

(**c**) Therefore, $C_3 = \{\{I1, I2, I3\}, \{I1, I2, I5\}\}$ after pruning.

Apriori Example

- The transactions in D are scanned to determine L_3, consisting of those candidate 3-itemsets in C_3 having minimum support
- The algorithm uses $L_3 \bowtie L_3$ to generate a candidate set of 4-itemsets, $C4$. Although the join results in {{I1, I2, I3, I5}}, itemset {I1, I2, I3, I5} is pruned because its subset {I2, I3, I5} is not frequent. Thus, $C_4 = \varphi$, and the algorithm terminates, having found all of the frequent itemsets.



Apriori Algorithm

Algorithm: Apriori. Find frequent itemsets using an iterative level-wise approach based on candidate generation.

Input:

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

Output: L, frequent itemsets in D.

Method:

(1) L_1 = find_frequent_1-itemsets(D);
(2) **for** ($k = 2; L_{k-1} \neq \phi; k$++) {
(3) C_k = apriori_gen(L_{k-1});
(4) **for each** transaction $t \in D$ { // scan D for counts
(5) C_t = subset(C_k, t); // get the subsets of t that are candidates
(6) **for each** candidate $c \in C_t$
(7) c.count++;
(8) }
(9) $L_k = \{c \in C_k | c.count \geq min\_sup\}$
(10) }
(11) **return** $L = \cup_k L_k$;

- In steps 2 through 10, L_{k-1} is used to generate candidates C_k to find L_k for $k \geq 2$.
- The apriori_gen procedure generates the candidates and then uses the Apriori property to eliminate those having a subset that is not frequent (step 3)
- Once all of the candidates have been generated, the database is scanned (step 4). For each transaction, a subset function is used to find all subsets of the transaction that are candidates (step 5)
- Count for each of these candidates is accumulated (steps 6 and 7).
- Finally, all the candidates satisfying the minimum support (step 9) form the set of frequent itemsets, L (step 11)

Apriori Algorithm (contd.)


```
procedure apriori_gen(L_{k-1}:frequent (k-1)-itemsets)
(1)     for each itemset l_1 ∈ L_{k-1}
(2)         for each itemset l_2 ∈ L_{k-1}
(3)             if (l_1[1] = l_2[1]) ∧ (l_1[2] = l_2[2])
                    ∧... ∧ (l_1[k-2] = l_2[k-2]) ∧ (l_1[k-1] < l_2[k-1]) then {
(4)                 c = l_1 ⋈ l_2; // join step: generate candidates
(5)                 if has_infrequent_subset(c, L_{k-1}) then
(6)                     delete c; // prune step: remove unfruitful candidate
(7)                 else add c to C_k;
(8)             }
(9)     return C_k;
```

- The apriori_gen procedure performs two kinds of actions, namely, a join and prune.
  - In the join component, L_{k-1} is joined with L_{k-1} to generate potential candidates (steps 1–4).
  - The prune component (steps 5–7) employs the Apriori property to remove candidates that have a subset that is not frequent.

Apriori Algorithm (contd.)

procedure has_infrequent_subset(c: candidate k-itemset;
L_{k-1}: frequent $(k-1)$-itemsets); // use prior knowledge

(1) **for each** $(k-1)$-subset s **of** c
(2) **if** $s \notin L_{k-1}$ **then**
(3) **return** TRUE;
(4) **return** FALSE;

Generating Association Rules from Frequent Itemsets

- The 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

  "$s \Rightarrow (l - s)$" if $support\_count(l)/support\_count(s) \geq min\_conf$, where *min_conf* is the minimum confidence threshold.

- Because the rules are generated from frequent itemsets, each one automatically satisfies the minimum support.

Example

- The data contain frequent itemset X = {I1, I2, I5}. What are the association rules that can be generated from X?
  - The nonempty subsets of X are {I1, I2}, {I1, I5}, {I2, I5}, {I1}, {I2}, and {I5}. The resulting association rules are as shown below, each listed with its confidence:
    - {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%
  - 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.

Exercise

A database has 5 transactions. Let min_sup = 60% and min_conf = 80%.

TID	Items bought
T100	{M, O, N, K, E, Y}
T200	{D, O, N, K, E, Y }
T300	{M, A, K, E}
T400	{M, U, C, K, Y}
T500	{C, O, O, K, I ,E}

Find all frequent itemsets using Apriori.

Solution

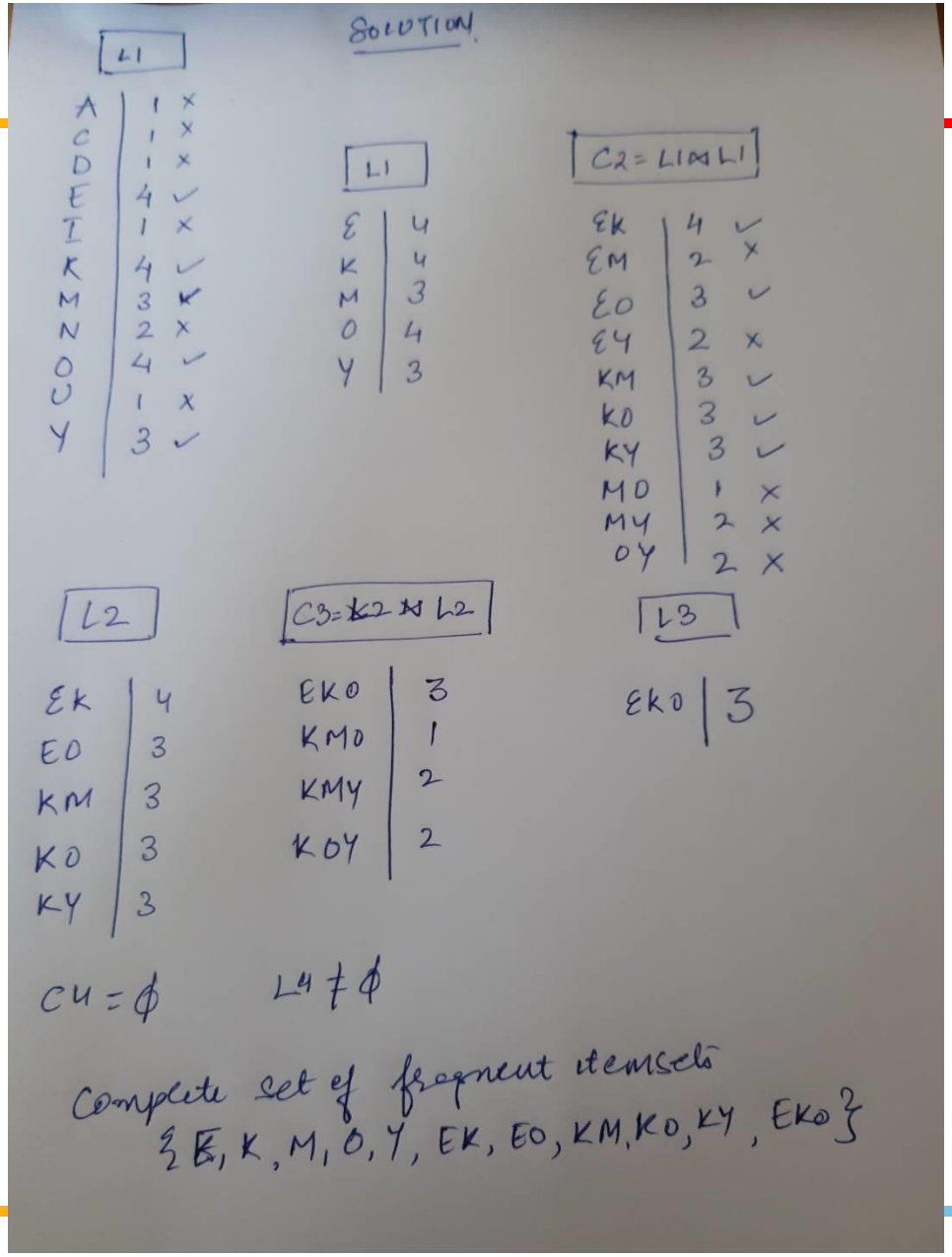
- For Apriori, one finds the following frequent itemsets, and candidate itemsets

- L1 = {E, K, M, O, Y}
- C2 = {EK, EM, EO, EY, KM, KO, KY, MO, MY, OY}
- L2 = {EK, EO, KM, KO, KY}
- C3 = {EKO}
- L3 = {EKO}
- C4 = ∅
- L4 = ∅

Finally resulting in the complete set of frequent itemsets:

- {E, K, M, O, Y, EK, EO, KM, KO, KY, EKO}

Solution


SOLUTION
L1
A 1 ×
C 1 ×
D 1 ×
E 4 ✓
I 1 ×
K 4 ✓
M 3 ✓
N 2 ×
O 4 ✓
U 1 ×
Y 3 ✓
L1
E 4
K 4
M 3
O 4
Y 3
C2 = L1 ⋈ L1
EK 4 ✓
EM 2 ×
EO 3 ✓
EY 2 ×
KM 3 ✓
KO 3 ✓
KY 3 ✓
MO 1 ×
MY 2 ×
OY 2 ×
L2
EK 4
EO 3
KM 3
KO 3
KY 3
C3 = L2 ⋈ L2
EKO 3
KMO 1
KMY 2
KOY 2
L3
EKO 3
C4 = ϕ
L4 ≠ ϕ
Complete set of frequent itemsets
{E, K, M, O, Y, EK, EO, KM, KO, KY, EKO}

Support Counting of Candidate Itemsets

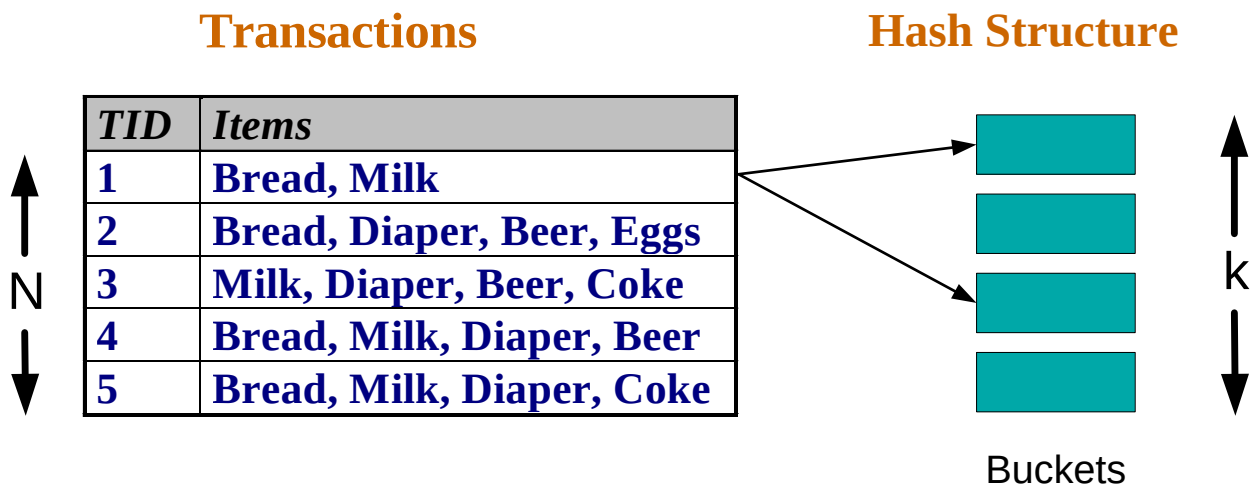
- Scan the database of transactions to determine the support of each candidate itemset
  - Must match every candidate itemset against every transaction, which is an expensive operation

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

Itemset
{ Beer, Diaper, Milk}
{ Beer,Bread,Diaper}
{Bread, Diaper, Milk}
{ Beer, Bread, Milk}

Computing Frequent Itemsets

- Given the set of candidate itemsets C_k, we need to compute the support and find the frequent itemsets L_k.
- Scan the data, and use a hash structure to keep a counter for each candidate itemset that appears in the data
  - Instead of matching each transaction against every candidate, match it against candidates contained in the hashed buckets

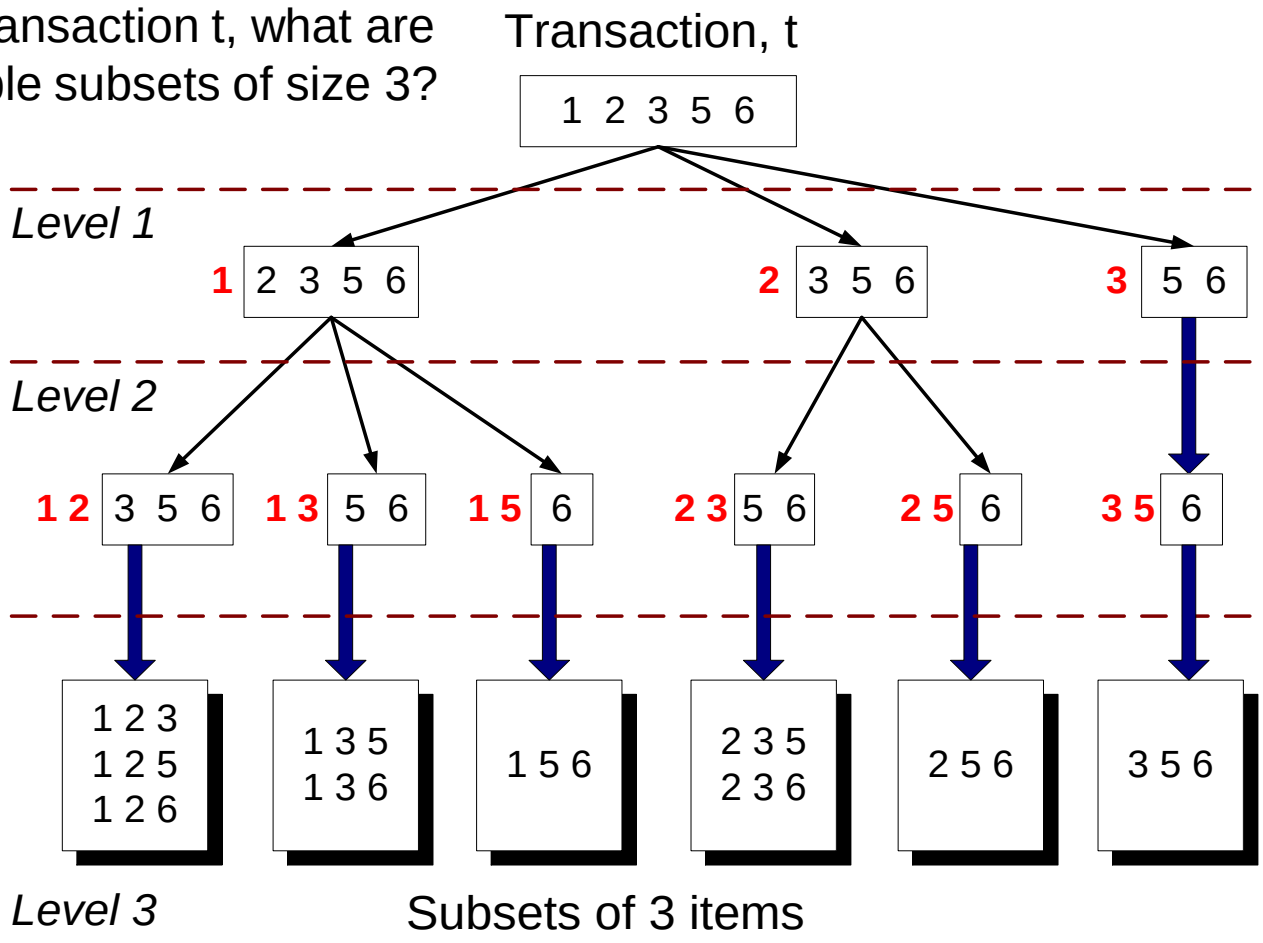


A simple hash structure

- Create a hash table that stores the candidate itemsets as keys, and the number of appearances as the value.
- Increment the counter for each itemset that you see in the candidate list.

Subset Generation

Given a transaction t, what are the possible subsets of size 3?



Example

- Suppose you have 15 candidate itemsets of length 3:

  {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}

- Hash table stores the counts of the candidate itemsets as they have been computed so far.

Key	Value
{3 6 7}	0
{3 4 5}	1
{1 3 6}	3
{1 4 5}	5
{2 3 4}	2
{1 5 9}	1
{3 6 8}	0
{4 5 7}	2
{6 8 9}	0
{5 6 7}	3
{1 2 4}	8
{3 5 7}	1
{1 2 5}	0
{3 5 6}	1
{4 5 8}	0

Example

Transaction {1,2,3,5,6} generates the following itemsets of length 3:

{1 2 3}, {1 2 5}, {1 2 6}, {1 3 5}, {1 3 6},

{1 5 6}, {2 3 5}, {2 3 6}, {3 5 6}

Increment the counters for the itemsets in the hash table.

Key	Value
{3 6 7}	0
{3 4 5}	1
{1 3 6}	3
{1 4 5}	5
{2 3 4}	2
{1 5 9}	1
{3 6 8}	0
{4 5 7}	2
{6 8 9}	0
{5 6 7}	3
{1 2 4}	8
{3 5 7}	1
{1 2 5}	0
{3 5 6}	1
{4 5 8}	0

Example

Transaction {1,2,3,5,6} generates the following itemsets of length 3:

{1 2 3}, {1 2 5}, {1 2 6}, {1 3 5}, {1 3 6},

{1 5 6}, {2 3 5}, {2 3 6}, {3 5 6}

Increment the counters for the itemsets in the hash table

Key	Value
{3 6 7}	0
{3 4 5}	1
{1 3 6}	4
{1 4 5}	5
{2 3 4}	2
{1 5 9}	1
{3 6 8}	0
{4 5 7}	2
{6 8 9}	0
{5 6 7}	3
{1 2 4}	8
{3 5 7}	1
{1 2 5}	1
{3 5 6}	2
{4 5 8}	0

Hash Tree Structure

Suppose you have the same 15 candidate itemsets of length 3:

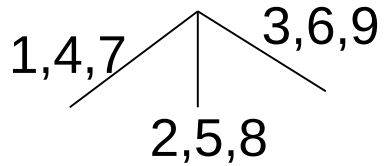
{1 4 5}, {1 2 4}, {4 5 7}, {1 2 5}, {4 5 8}, {1 5 9}, {1 3 6}, {2 3 4},

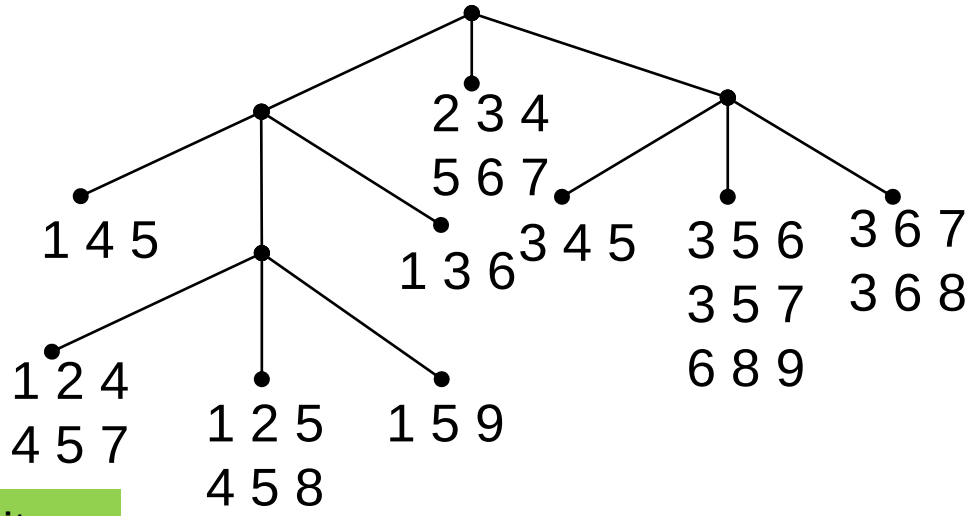
{5 6 7}, {3 4 5}, {3 5 6}, {3 5 7}, {6 8 9}, {3 6 7}, {3 6 8}

You need:

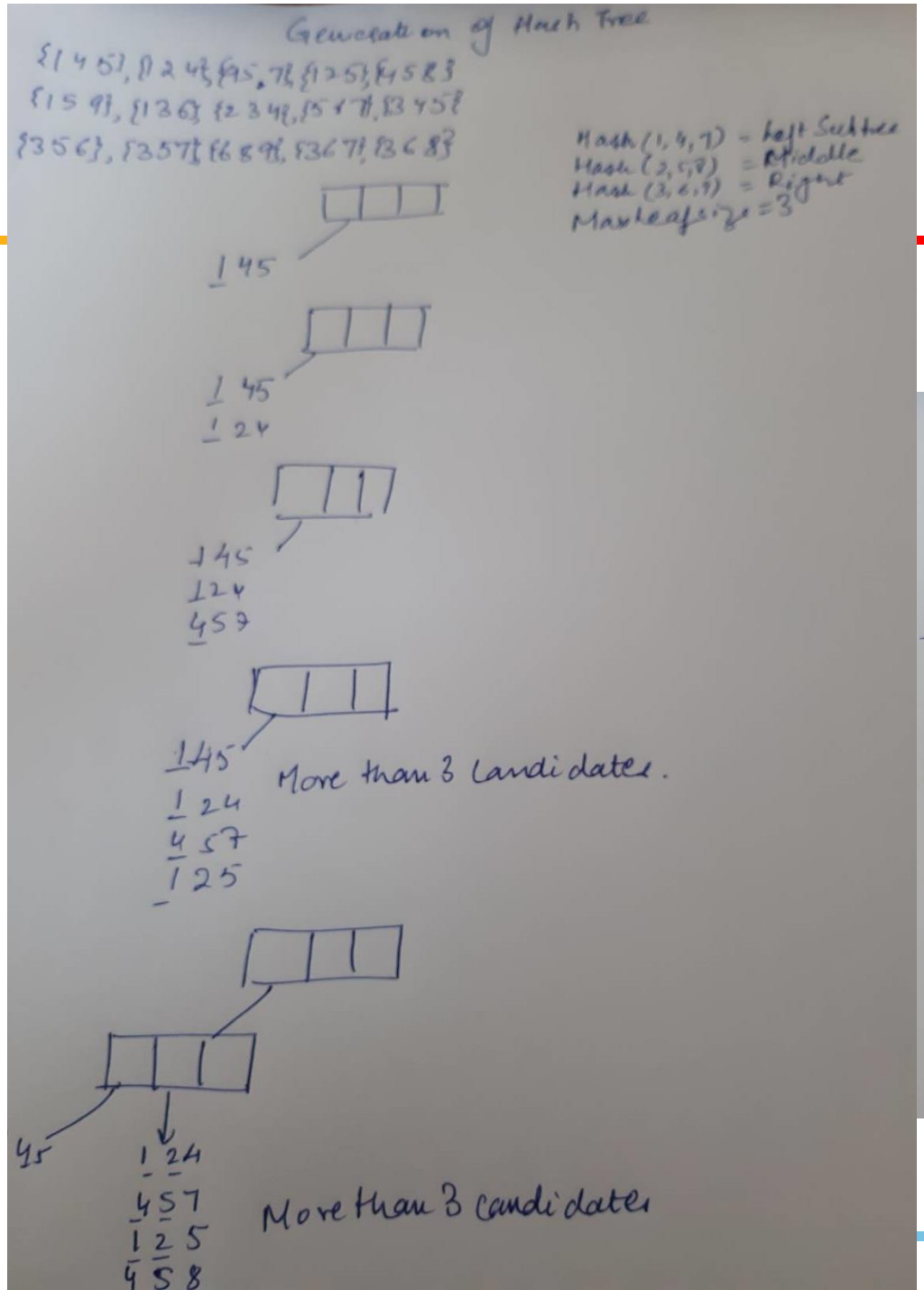
- Hash function
- Max leaf size
  - Max number of itemsets stored in a leaf node (if number of candidate itemsets exceeds max leaf size, split the node)

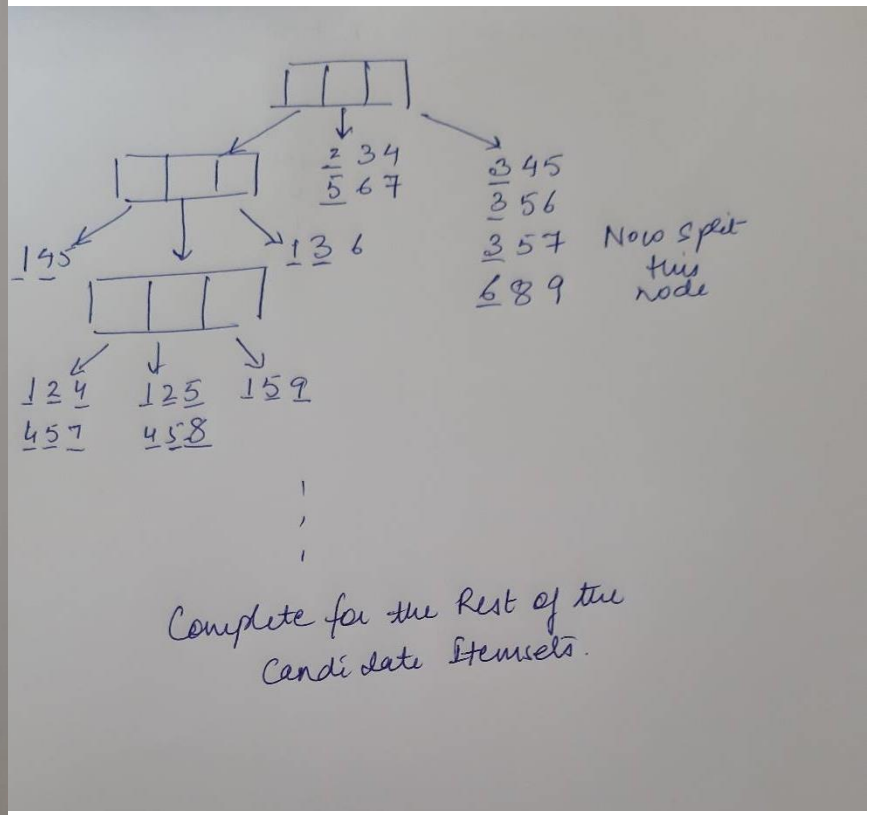
Hash function = x mod 3



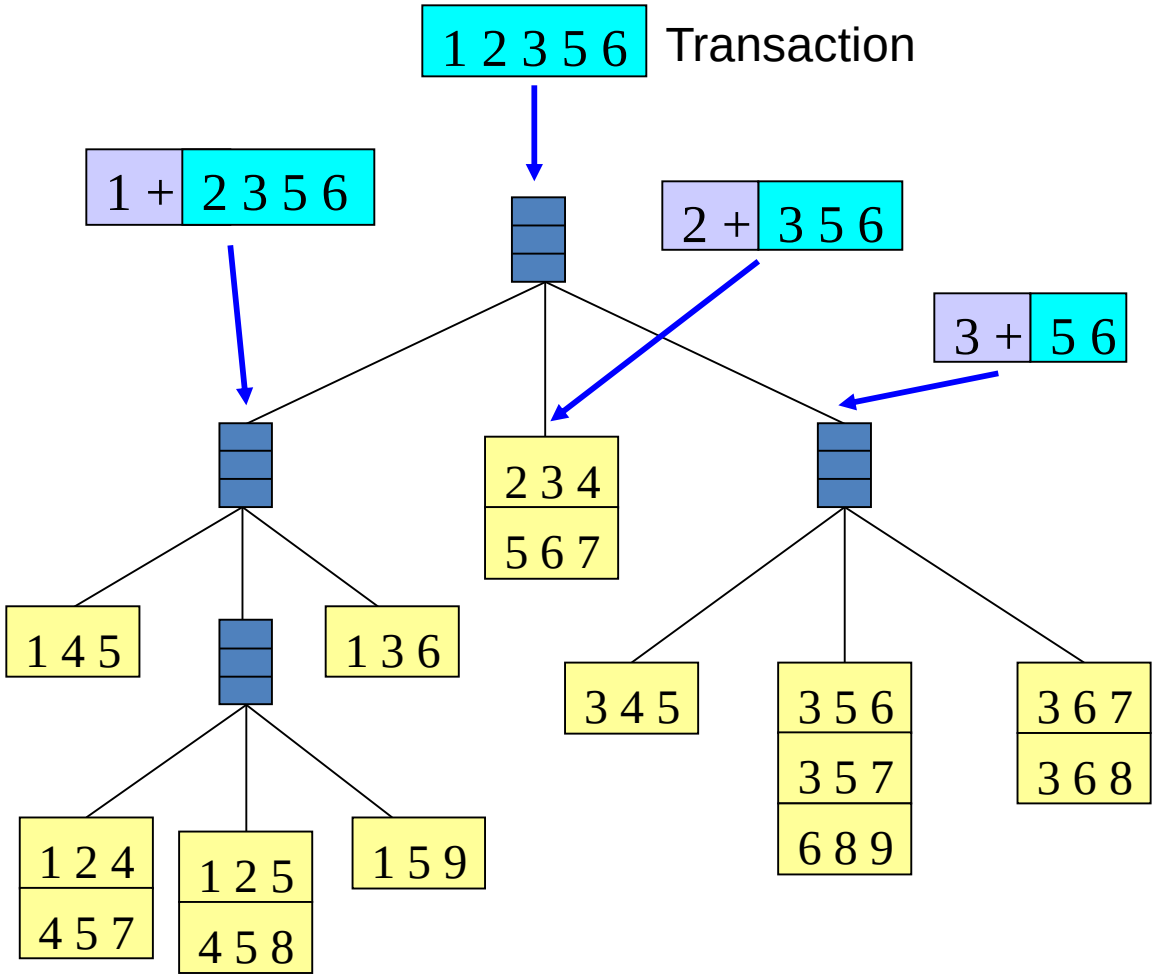


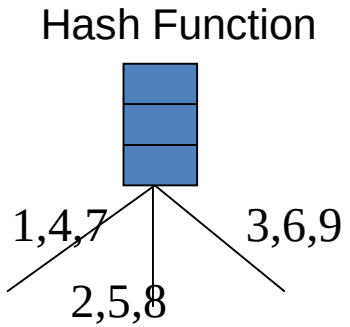
At the i-th level we hash on the i-th item


Generation of 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}
Hash (1, 4, 7) = Left Subtree
Hash (2, 5, 8) = Middle
Hash (3, 6, 9) = Right
Max leaf size = 3
1 4 5
1 4 5
1 2 4
1 4 5
1 2 4
4 5 7
1 4 5
1 2 4
4 5 7
1 2 5
More than 3 candidates.
4 5
1 2 4
4 5 7
1 2 5
4 5 8
More than 3 candidates.


2 3 4
5 6 7
3 4 5
3 5 6
3 5 7
6 8 9
Now split this node
1 4 5
1 3 6
1 2 4
4 5 7
1 2 5
4 5 8
1 5 9
Complete for the Rest of the Candidate Itemsets.

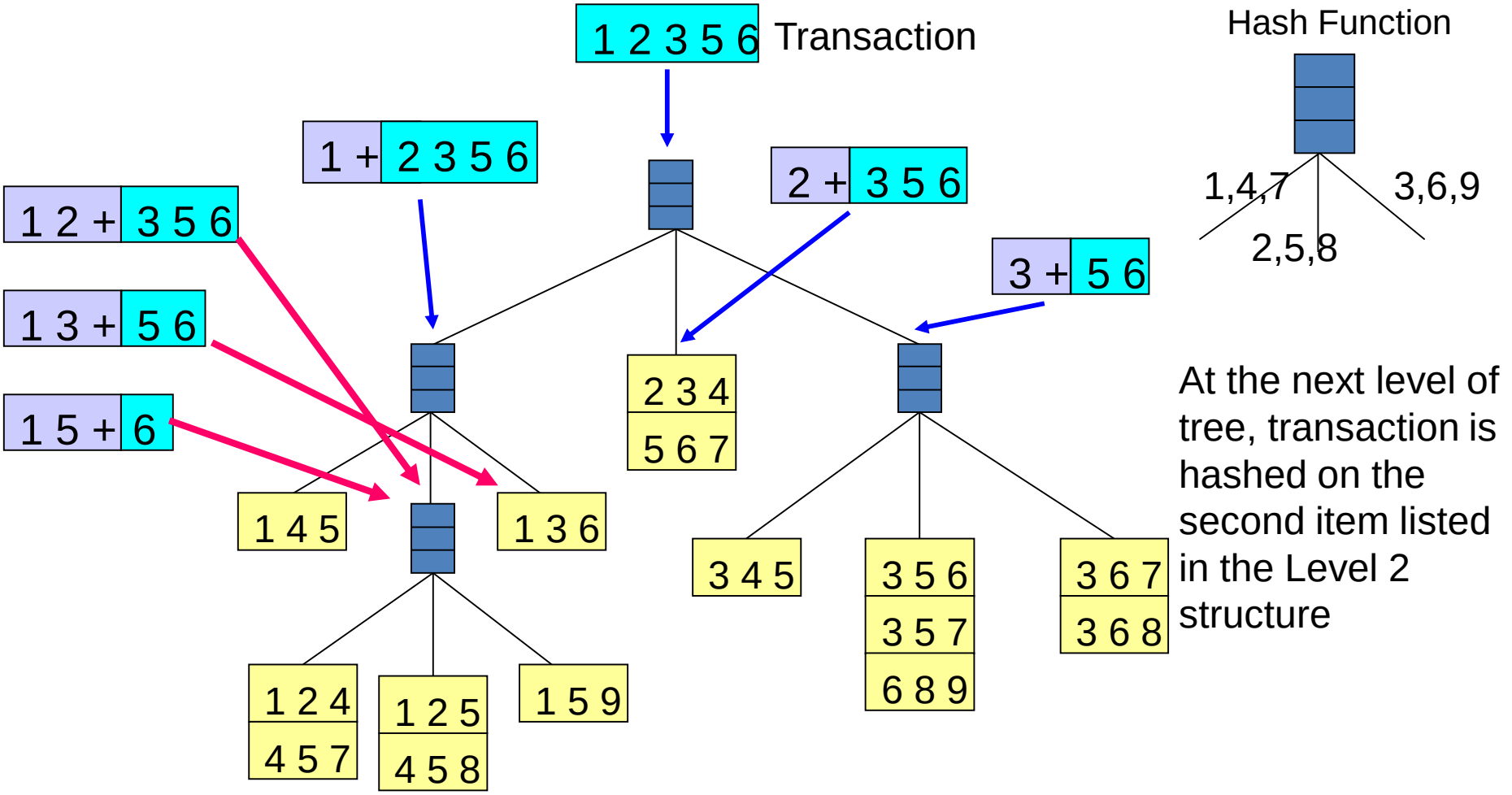
Subset Operation Using Hash Tree





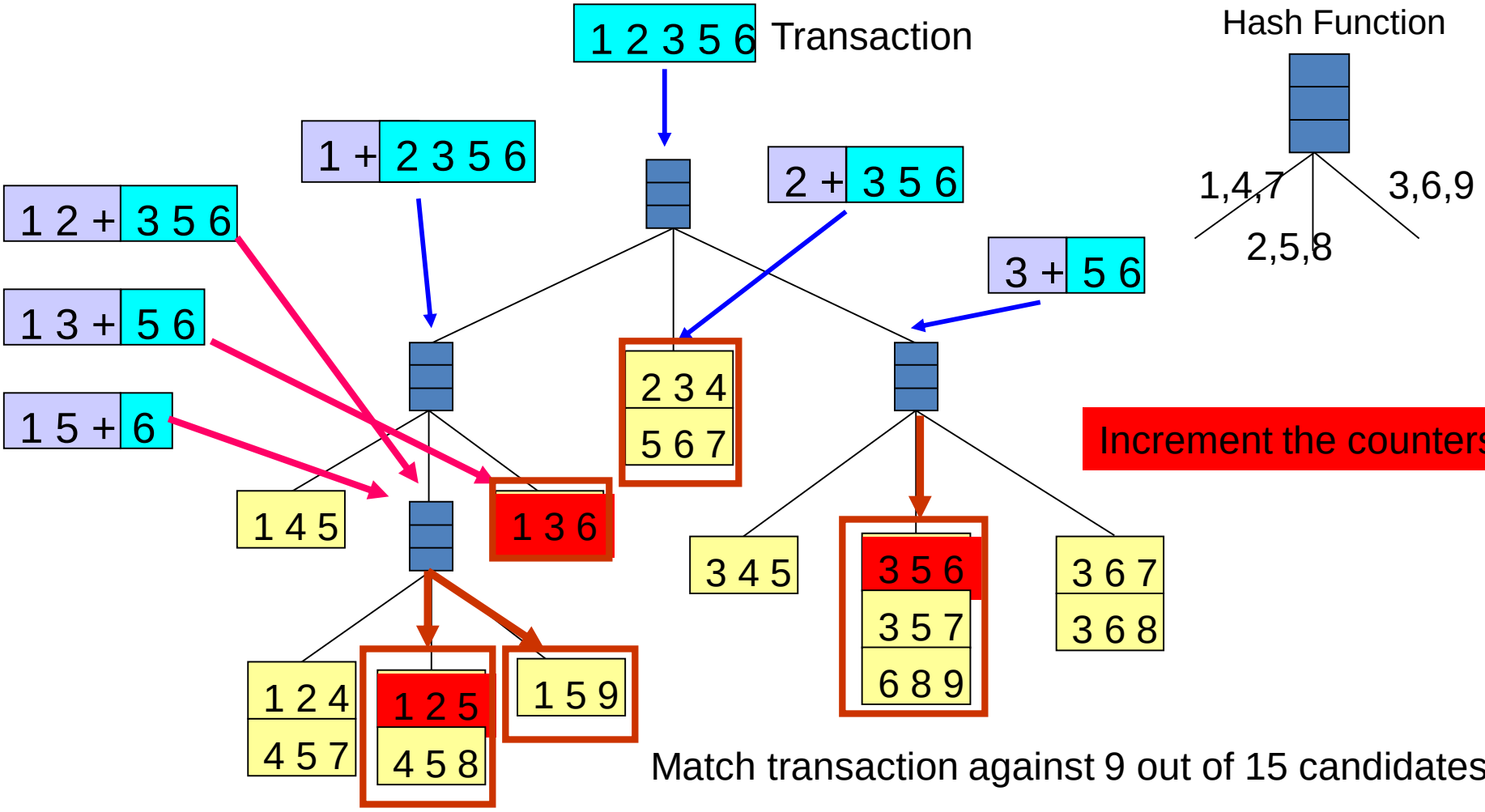
At root node of the hash tree, the Level 1 prefix structures, i.e items 1,2 and 3 are hashed separately

Subset Operation Using Hash Tree



At the next level of tree, transaction is hashed on the second item listed in the Level 2 structure

Subset Operation Using Hash Tree



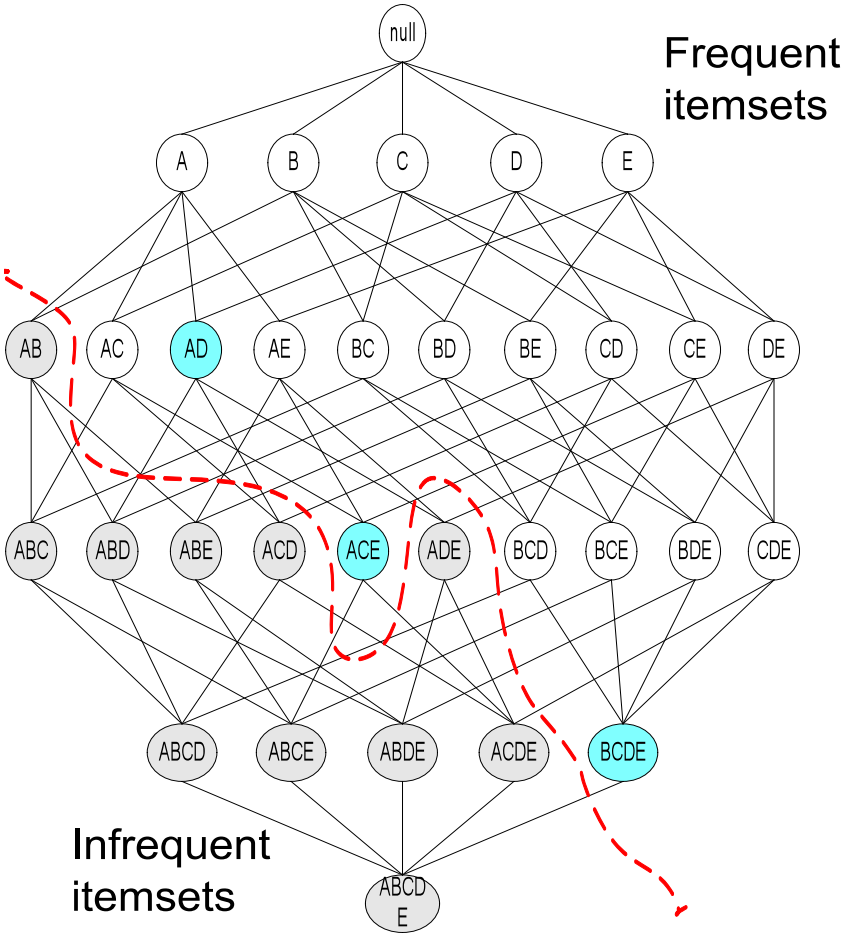
Hash-tree helps enumerate itemsets in transaction & match them against candidates stored at visited leaf nodes

Compact Representation of Frequent Itemsets

- The number of frequent itemsets can be very large
  - Useful to identify a small representative set of itemsets from which all other other frequent itemsets can be derived

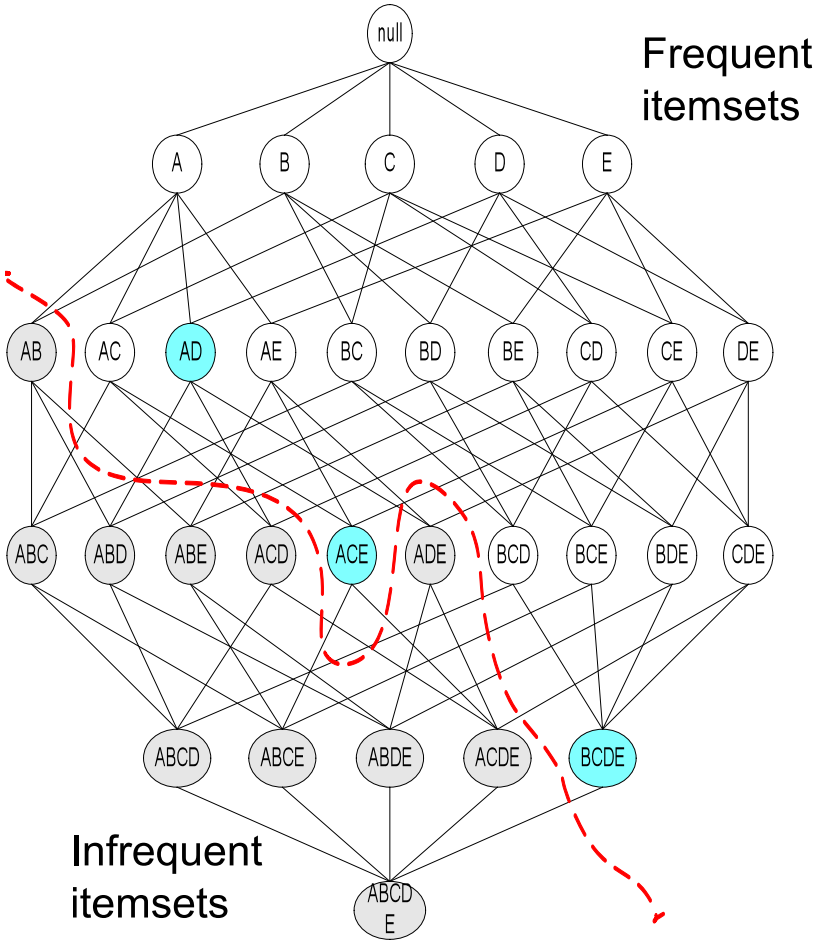
Maximal Frequent Itemset

An itemset is maximal frequent if none of its immediate supersets is frequent



Maximal Frequent Itemsets

- {A,C} is not maximal as it can be extended to frequent itemset {A,C,E} although its supersets {A,B,C}, {A,C,D} are infrequent
- {A,D} is maximal as all its immediate supersets {A,B,D}, {A,C,D} and {A,D,E} are infrequent
- {B,D} is not maximal as it can be extended to frequent itemsets {B,C,D} and {B,D,E}



Drawback of Maximal Frequent Itemsets

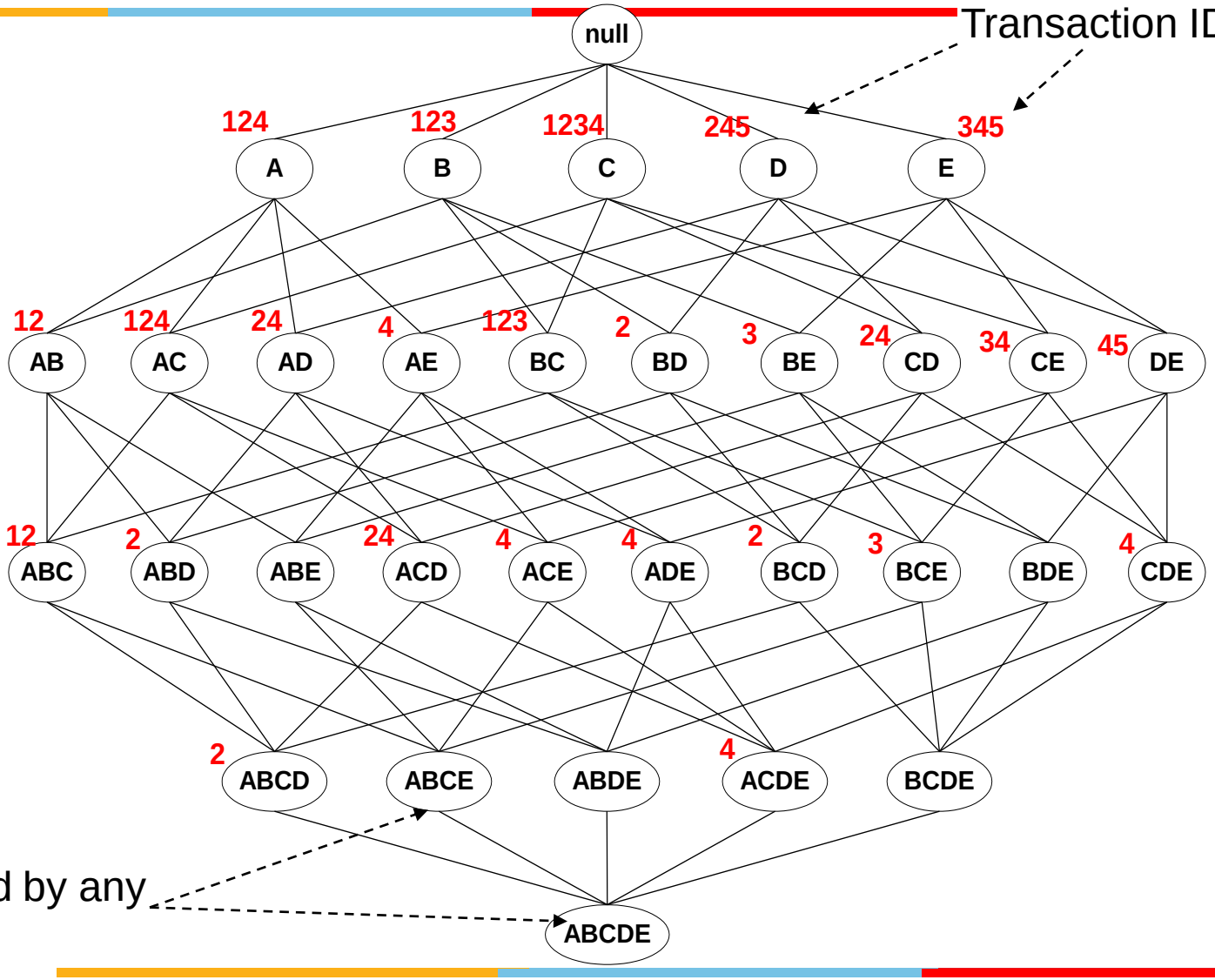
- Maximal frequent itemsets do not contain the support information of their subsets.
- Hence it is desirable to have a minimal representation of frequent itemsets that preserve the support information.

Closed Itemset

- An itemset X is closed if none of its immediate supersets has the same support count as X
- A closed frequent itemset is a closed itemset that satisfies the minimum support threshold
- Maximal frequent itemsets are closed by definition

Closed Itemsets

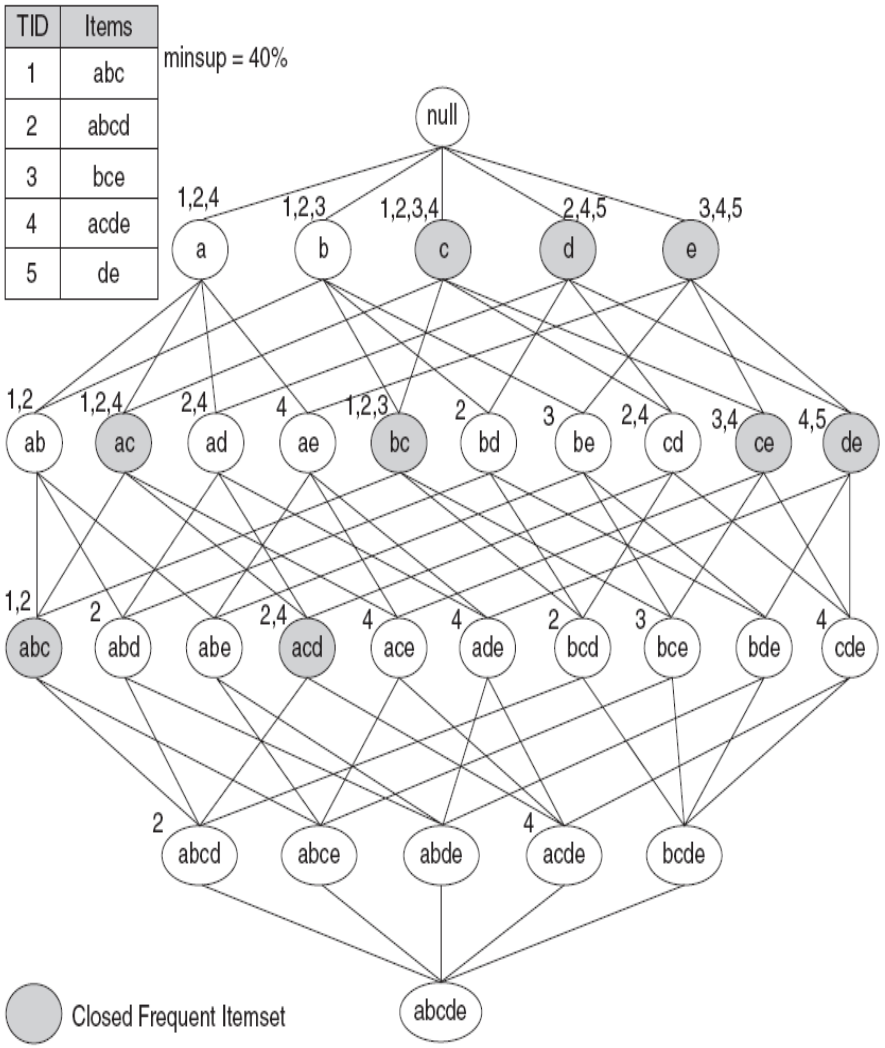
TID	Items
1	ABC
2	ABCD
3	BCE
4	ACDE
5	DE



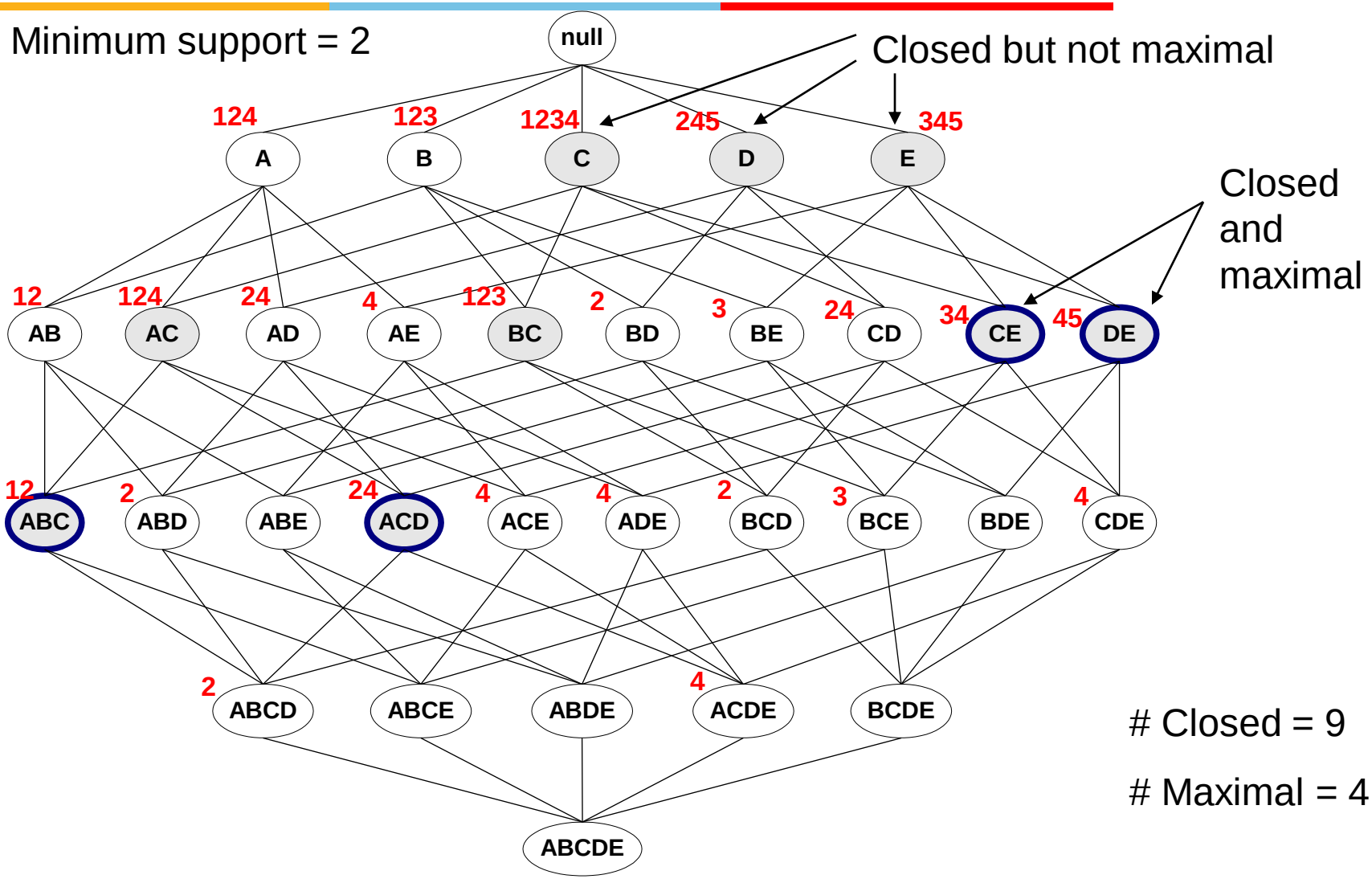
Closed Itemsets

Assume minimum support threshold 40%

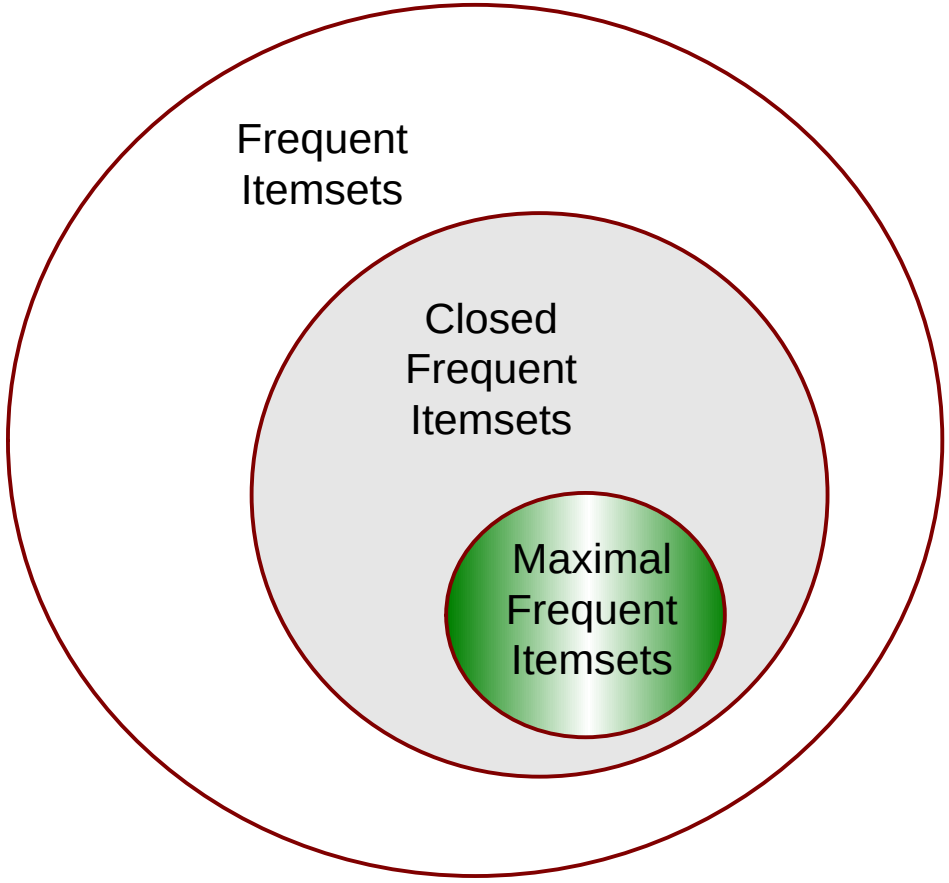
- {b} is frequent: σ({b})=3, but not closed: σ({b}) = σ({b,c}) = 3
- {b,c} is frequent: σ({b,c})= 3, and closed: σ({a,b,c}) = 2, σ({b,c,d})=1,σ({b,c,e})=1
- {b,c,d} is not frequent: σ({b,c,d}) = 1, and not closed : σ({a,b,c,d}) = 1



Maximal vs Closed Frequent Itemsets



Maximal vs Closed Itemsets



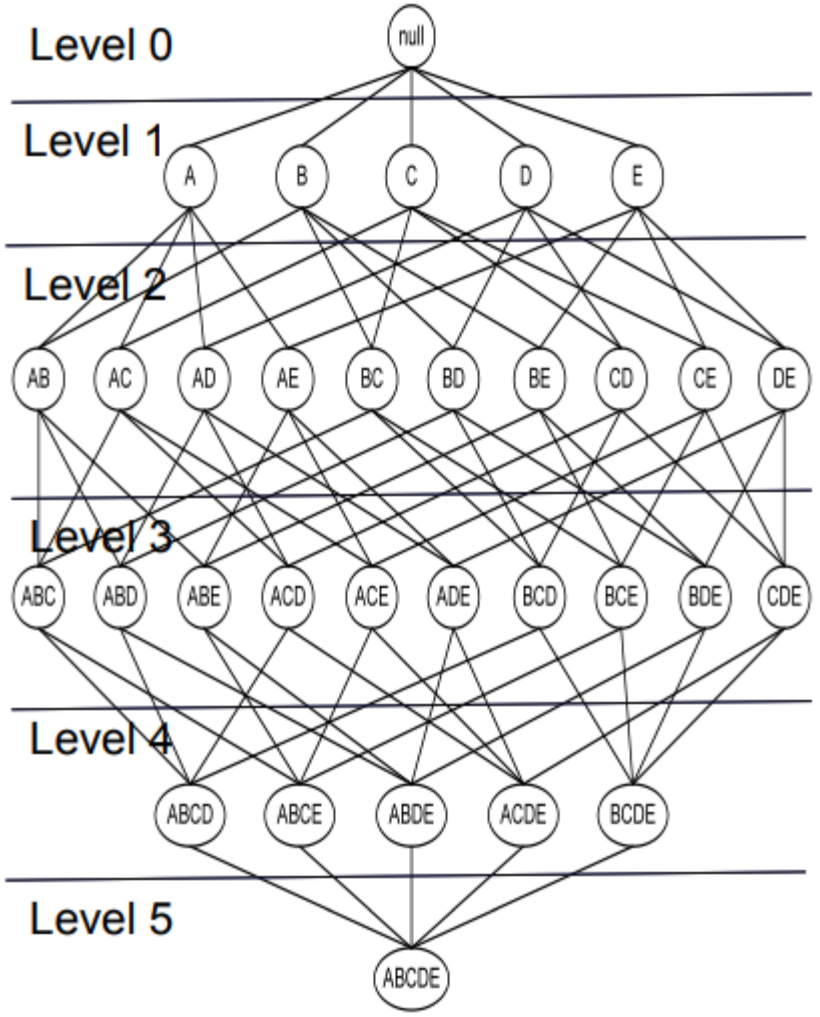
All maximal frequent itemsets are closed because none of the maximal frequent itemsets can have the same support count as their immediate supersets.

Factors Affecting Performance

- 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
  - Number of subsets in a transaction increases with its width
  - This may increase max length of frequent itemsets and traversals of hash tree

Characteristics of Apriori algorithm

- Breadth-first search algorithm
  - All frequent itemsets of given size are kept in the algorithms processing queue
- General-to-specific search
  - Start with itemsets with large support, work towards lower-support region
- Generate-and-test strategy
  - Generate candidates, test by support counting.



Weaknesses of Apriori

- Apriori is one of the first algorithms that succesfully tackled the exponential size of the frequent itemset space
- Nevertheless the Apriori suffers from two main weaknesses:
  - High I/O overhead from the generate-and-test strategy: several passes are required over the database to find the frequent itemsets
  - The performance can degrade significantly on dense databases, as large portion of the itemset lattice becomes frequent

To Explore

- Alternative methods for generating frequent itemsets

Thanks!

innovate achieve lead

Next Lecture:

- FP Growth Algorithm

Readings:

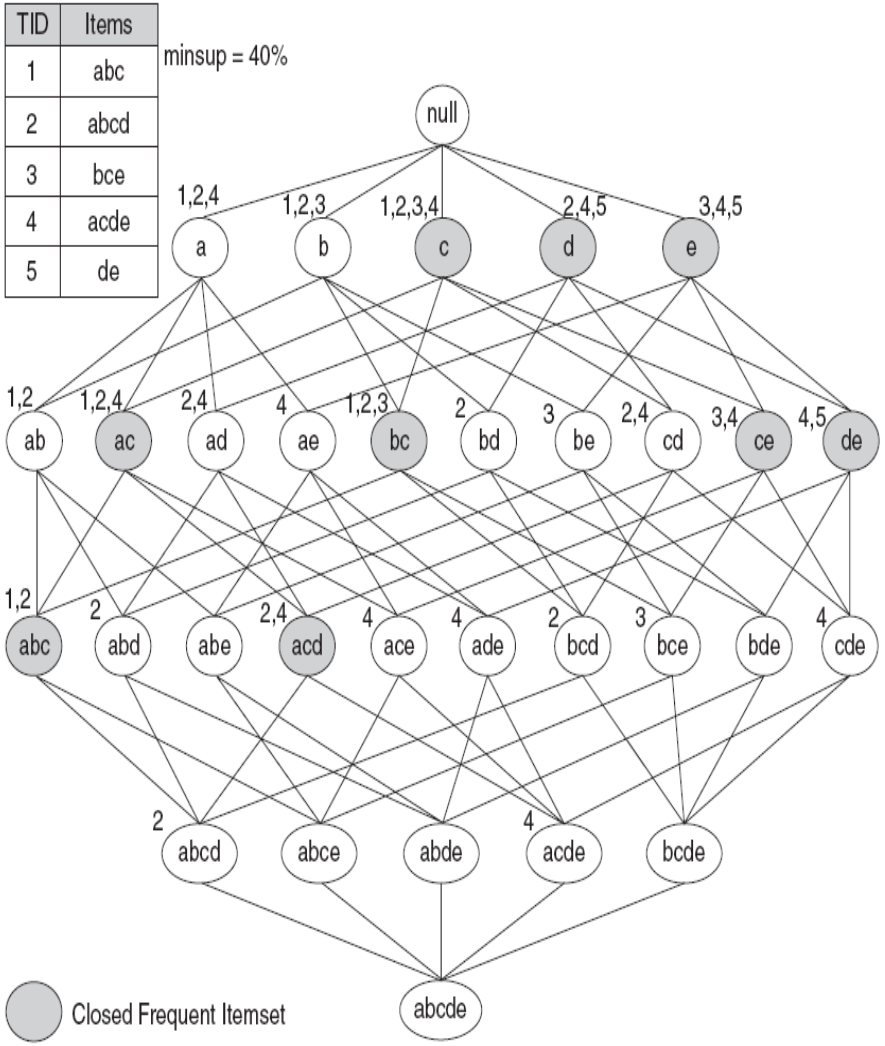
- Chapter 6 - Tan & Kumar
- Chapter 6 - Han & Kamber

Use of maximal and closed frequent itemset

- All frequent itemsets can be derived from the set of maximal frequent itemsets by using the Apriori principle backwards.
- Closed frequent itemsets can be used to determine the support counts for the non closed frequent itemsets
- They are useful for removing some of the redundant association rules.
- Example:
  - {b} is not closed frequent while {b,c} is closed.
  - {b}->{d,e} is redundant since it has same support and confidence as {b,c}->{d,e}
  - Such redundant rules are not generated if closed frequent itemsets are used for rule generation

Determining the support of non-closed frequent itemsets

- Consider a non-closed frequent itemset {a,d}
- Assume we have not stored its support count
- By definition, there must be at least one immediate superset that has the same support count
- It must be that σ({a,d}) = σ(X) for some immediate superset X of {a,d}



Determining the support of non-closed frequent itemsets

- From the Apriori principle we know that no superset can have higher support than {a,d}
- It must be that the support equals the support of the most frequent superset
- σ({a,d}) = max(σ(abd),σ(acd),σ(ade))