Association Rule Mining

- Frequent Itemsets, Association Rules
- Apriori Algorithm
- Compact Representation of Frequent Itemsets
- FP-Growth Algorithm: An Alternative Frequent Itemset Generation Algorithm
- Evaluation of Association Patterns

Frequent Pattern

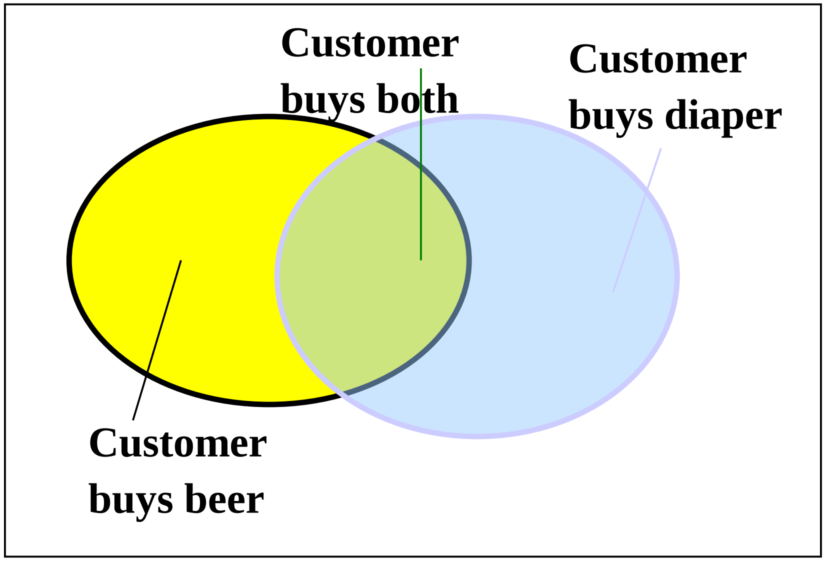
- **Frequent Pattern**: a pattern (a set of items, subsequences, substructures, etc.) that occurs frequently in a data set.
- For example, a set of items, such as milk and bread, that appear frequently together in a transaction data set is a ***frequent itemset***.
- A subsequence, such as buying first 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, which may be combined with itemsets or subsequences. If a substructure occurs frequently, it is called a **(*frequent*) *structured pattern***.

Frequent Pattern
Market Basket Analysis

- **Frequent Pattern**: a pattern that occurs frequently in a data set.
  - A set of items that appear frequently together in a transaction data set is called as a ***frequent itemset***.
- An 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”.
  - If we think of the universe as the set of items available at the store, then each item has a Boolean variable representing the presence or absence of that item.
  - Each basket can then be represented by a Boolean vector of values assigned to these variables.
  - The Boolean vectors can be analyzed for buying patterns that reflect items that are frequently associated or purchased together.
  - These ***patterns*** can be represented in the form of ***association rules***.

Basic Concepts: Frequent Patterns

Tid	Items bought
10	Beer, Nuts, Diaper
20	Beer, Coffee, Diaper
30	Beer, Diaper, Eggs
40	Nuts, Eggs, Milk
50	Nuts, Coffee, Diaper, Eggs, Milk



- **itemset**: A set of one or more items
  - **k-itemset** $X = \{x_1, \ldots, x_k\}$
- **(absolute) support of X**: Frequency of an itemset X.
  - Absolute Support of {Beer} is 3
- **(relative) support of X** is the fraction of transactions that contains X (i.e., the probability that a transaction contains X).
  - Relative Support of {Beer} is 3/5
- An itemset X is **frequent** if **X's support** is no less than a **minsup** threshold.

Basic Concepts: Association Rules

Association Rule

- **An implication expression of the form X $\rightarrow$ Y, where X and Y are itemsets**

Association Rule Mining:

- Find all the rules X $\rightarrow$ Y with **minimum support** and **minimum confidence**
  - **support**, probability that a transaction contains $X \cup Y$: $P(X \cup Y)$
    - Fraction of transactions that contain both X and Y
  - **confidence**, conditional probability that a transaction having X also contains *Y* : $P(Y/X) = support(X \cup Y) / support(X)$
    - Measures how often items in Y appear in transactions that contain X

Basic Concepts: Association Rules

Association Rule

– **An implication expression of the form X → Y, where X and Y are itemsets**

Association Rule Mining:

- Find all the rules X → Y with **minimum support** and **minimum confidence**

Let minsup = 50%, minconf = 50%

Frequent Pattterns:

{Beer}:3, {Nuts}:3, {Diaper}:4, {Eggs}:3,

{Beer, Diaper}:3

Association Rules:

– { Beer } → { Diaper } (60%, 100%)
– { Diaper } → { Beer } (60%, 75%)

Tid	Items bought
10	Beer, Nuts, Diaper
20	Beer, Coffee, Diaper
30	Beer, Diaper, Eggs
40	Nuts, Eggs, Milk
50	Nuts, Coffee, Diaper, Eggs, Milk

Why Use Support and Confidence?

- **Support** is an important measure because a rule that has very low support may occur simply by chance.
  - A low support rule may be uninteresting from a business perspective because it may not be profitable to promote items that customers seldom buy together
  - For these reasons, support is often used to eliminate uninteresting rules

- **Confidence** measures the reliability of the inference made by a rule.
  - For a given rule X → Y, the higher the confidence, the more likely it is for Y to be present in transactions that contain X.

- Association analysis results should be interpreted with caution.
  - The inference made by an association rule does not necessarily imply causality.
  - Instead, it suggests a strong co-occurrence relationship between items in the antecedent and consequent of the rule.

Association Rule Mining Task

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

- 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 not feasible!

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} $\rightarrow$ {Beer} (s=0.4, c=0.67)
{Milk,Beer} $\rightarrow$ {Diaper} (s=0.4, c=1.0)
{Diaper,Beer} $\rightarrow$ {Milk} (s=0.4, c=0.67)
{Beer} $\rightarrow$ {Milk,Diaper} (s=0.4, c=0.67)
{Diaper} $\rightarrow$ {Milk,Beer} (s=0.4, c=0.5)
{Milk} $\rightarrow$ {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

Association Rule Mining

- The problem of mining association rules can be reduced to that of mining frequent itemsets.

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

1. **Find all frequent itemsets**: By definition, each of these itemsets will occur at least as frequently as a predetermined minimum support count, **minsup**.
   - Generate all itemsets whose **support ≥ minsup**
2. **Generate strong association rules from the frequent itemsets**: By definition, these rules must satisfy minimum support and minimum confidence.
   - 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

Association Rules - Example

Transactions
A,B,D
A,B,C,D
A
A,B,C
B,C
B

minsup = 0.5 minconf=0.7

- Find frequent itemsets and association rules satisfying minsup and minconf.

Association Rules - Example

Transactions
A,B,D
A,B,C,D
A
A,B,C
B,C
B

minsup = 0.5 minconf=0.7

- Find frequent itemsets and association rules satisfying minsup and minconf.

Frequent Itemsets:

1-itemsets: **{A}** **support({A}) = 4/6**
{B} **support({B}) = 5/6**
{C} **support({C}) = 3/6**

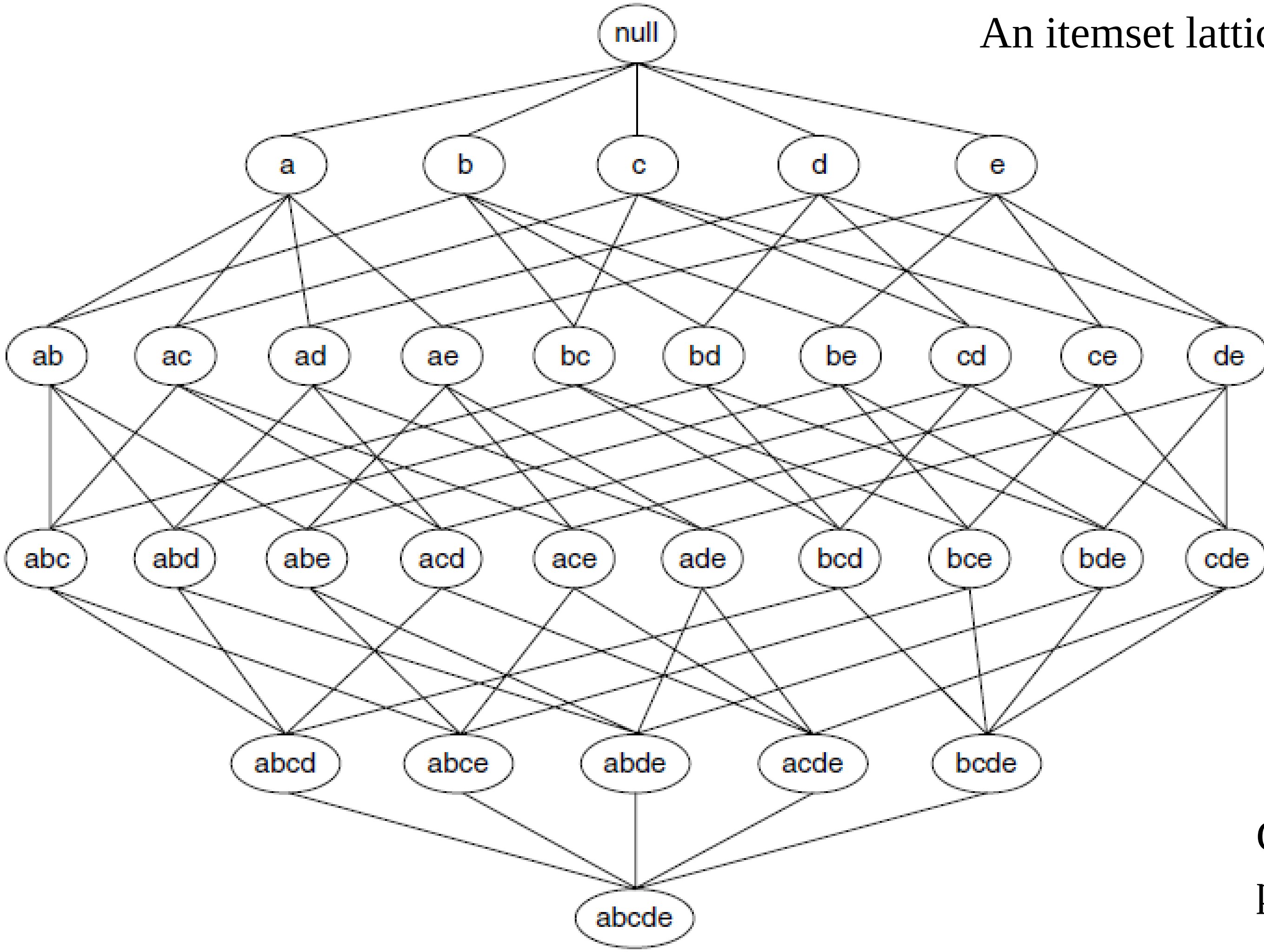
2-itemsets: **{A,B}** **support({A,B}) = 3/6**
{B,C} **support({B,C}) = 3/6**

Association Rules:

A→B **conf(A→B) = 3/4**
C→B **conf(C→B) = 3/3**

Frequent Itemset Generation

An itemset lattice



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

Frequent Itemset Generation

- **Brute-force approach:**
  - Each itemset in the lattice is a candidate frequent itemset
  - Count the support of each candidate by scanning the database

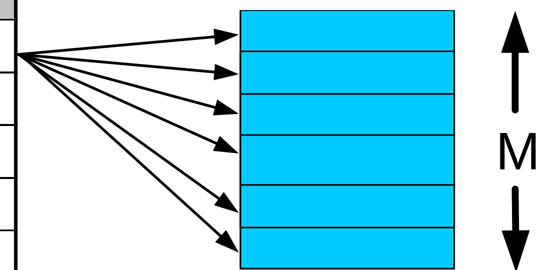
Transactions

N

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**

w

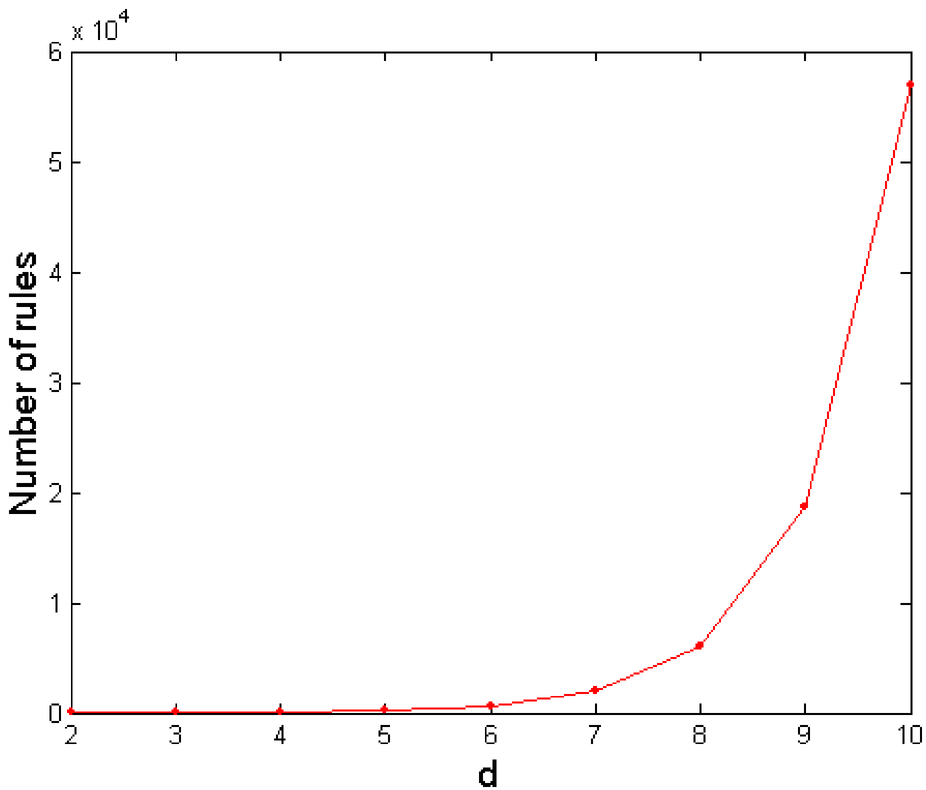
List of Candidates



  - Match each transaction against every candidate
  - Complexity ~ O(NMW) => Expensive since $M = 2^d$!!!

Computational Complexity

- Given d unique items:
  - Total number of itemsets = 2^d
  - Total number of possible association rules:



$$R = \sum_{k=1}^{d-1}\left[\binom{d}{k} \times \sum_{j=1}^{d-k}\binom{d-k}{j}\right]$$
$$= 3^d - 2^{d+1} + 1$$

If d=6, R = 602 rules

Frequent Itemset Generation Strategies

- Reduce the number of candidates (M)
  - Complete search: $M=2^d$
  - Use pruning techniques to reduce M
  - The **Apriori principle** is an effective way to eliminate some of the candidate itemsets without counting their support values.

- 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

- Reduce the number of transactions (N)
  - Reduce size of N as the size of itemset increases

- **Frequent Itemsets, Association Rules**
- **Apriori Algorithm**
- **Compact Representation of Frequent Itemsets**
- **FP-Growth Algorithm: An Alternative Frequent Itemset Generation Algorithm**
- **Evaluation of Association Patterns**

Reducing Number of Candidates
Apriori Principle

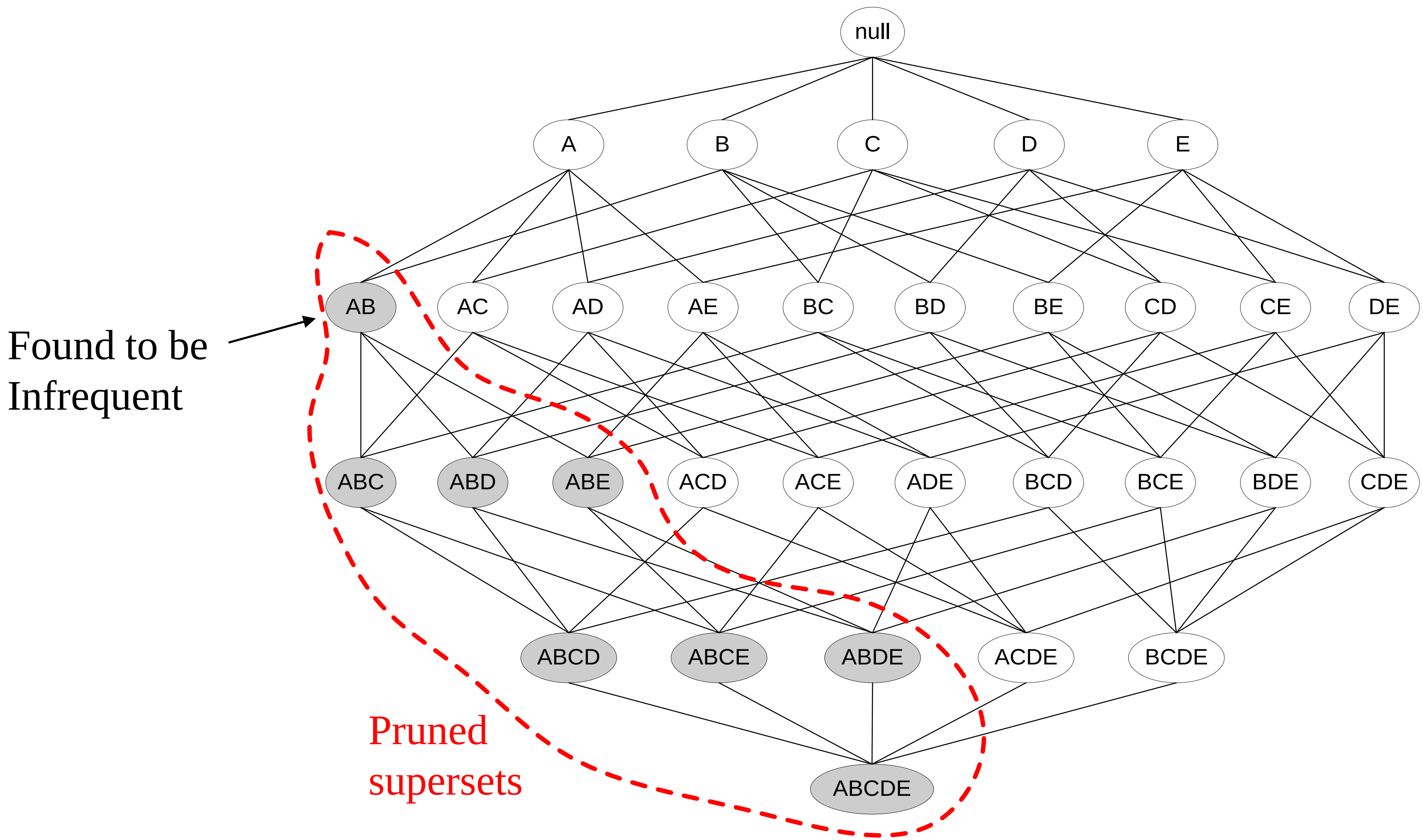
- **Apriori Principle: If an itemset is frequent, then all of its subsets must also be frequent.**

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

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

  – Support of an itemset never exceeds the support of its subsets
  – This is known as the anti-monotone property of support

Illustrating Apriori Principle



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

Generate 1-itemset candidates

If every subset is considered,

$^6C_1 + {}^6C_2 + {}^6C_3$

$6 + 15 + 20 = 41$

With support-based pruning,

$6 + 6 + 4 = 16$

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

Eliminate infrequent 1-itemset candidates

If every subset is considered,

$^6C_1 + {}^6C_2 + {}^6C_3$

$6 + 15 + 20 = 41$

With support-based pruning,

$6 + 6 + 4 = 16$

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

If every subset is considered,

$^6C_1 + {}^6C_2 + {}^6C_3$

$6 + 15 + 20 = 41$

With support-based pruning,

$6 + 6 + 4 = 16$

Generate 2-itemset candidates

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

Eliminate infrequent 2-itemset candidates

If every subset is considered,

$$^6C_1 + {}^6C_2 + {}^6C_3$$

$$6 + 15 + 20 = 41$$

With support-based pruning,

$$6 + 6 + 4 = 16$$

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

If every subset is considered,

$$^6C_1 + {}^6C_2 + {}^6C_3$$

$$6 + 15 + 20 = 41$$

With support-based pruning,

$$6 + 6 + 4 = 16$$

Generate 3-itemset candidates

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,

$${}^6C_1 + {}^6C_2 + {}^6C_3$$

$$6 + 15 + 20 = 41$$

With support-based pruning,

$$6 + 6 + 4 = 16$$

$$6 + 6 + 1 = 13$$

Prune 3-itemset candidates with infrequent 2-itemsets

Eliminate infrequent 3-itemset candidates

Apriori Algorithm:
Finding Frequent Itemsets Using Candidate Generation

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

Apriori Algorithm: F_k: frequent k-itemsets L_k: candidate k-itemsets

- Let k=1
- Generate F_1 = {frequent 1-itemsets}
- Repeat until F_k is empty
  - **Candidate Generation**: Generate L_{k+1} from F_k
  - **Candidate Pruning**: Prune candidate itemsets in L_{k+1} containing subsets of length k that are infrequent
  - **Support Counting**: Count the support of each candidate in L_{k+1} by scanning the DB
  - **Candidate Elimination**: Eliminate candidates in L_{k+1} that are infrequent, leaving only those that are frequent => F_{k+1}

Apriori Algorithm:
Candidate Generation: F_{k-1} x F_{k-1} Method

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

- F_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

- L_4 = {ABCD,ABCE,ABDE} is the set of candidate 4-itemsets **generated**

Apriori Algorithm:
Candidate Pruning

- Let F_3 = {ABC,ABD,ABE,ACD,BCD,BDE,CDE} be the set of frequent 3-itemsets

- L_4 = {ABCD,ABCE,ABDE} is the set of **candidate 4-itemsets generated**

- Candidate pruning
  – Prune ABCE because ACE and BCE are infrequent
  – Prune ABDE because ADE is infrequent

- After **candidate pruning**: L_4 = {ABCD}

Apriori Algorithm:
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

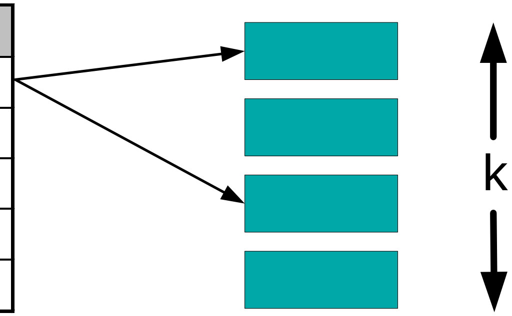
- To reduce the number of comparisons, store the candidates in a hash structure
  - Instead of matching each transaction against every candidate, match it against candidates contained in the hashed buckets

Transactions

N

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

Hash Structure



Buckets

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:

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

Apriori Algorithm

```
procedure apriori_gen(L_{k-1}:frequent (k-1)-itemsets)
    for each itemset l_1 ∈ L_{k-1}
        for each itemset l_2 ∈ L_{k-1}
            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 {
                c = l_1 ⋈ l_2; // join step: generate candidates
                if has_infrequent_subset(c, L_{k-1}) then
                    delete c; // prune step: remove unfruitful candidate
                else add c to C_k;
            }
    return C_k;
```

```
procedure has_infrequent_subset(c: candidate k-itemset;
            L_{k-1}: frequent (k-1)-itemsets); // use prior knowledge
    for each (k-1)-subset s of c
        if s ∉ L_{k-1} then
            return TRUE;
    return FALSE;
```

Apriori Algorithm - An Example

$Sup_{min} = 2$

Database TDB

Tid	Items
10	A, C, D
20	B, C, E
30	A, B, C, E
40	B, E

1^{st} scan

C_1

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

L_1

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

C_2

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

2^{nd} scan

C_2

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

L_2

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

C_3

Itemset
{B, C, E}

3^{rd} scan

L_3

Itemset	sup
{B, C, E}	2

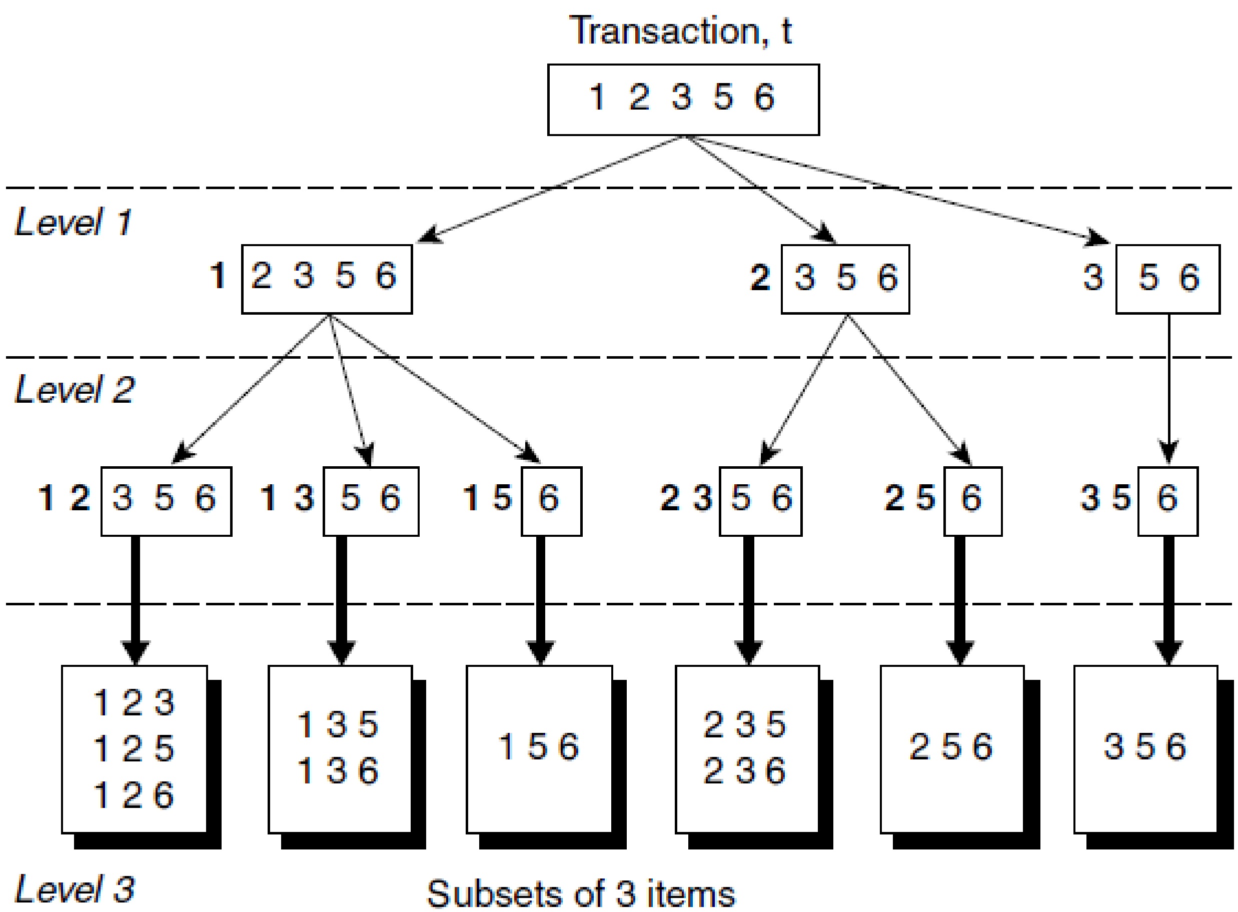
Support Counting Using Hash Tree

- Why counting supports of candidates a problem?
  - The total number of candidates can be very huge
  - One transaction may contain many candidates
  - Must match every candidate itemset against every transaction, which is an expensive operation

- Method:
  - Candidate itemsets are stored in a ***hash-tree***
  - ***Leaf* node** of hash-tree contains a list of itemsets and counts
  - ***Interior* node** contains a hash table
  - ***Subset function*:** finds all the candidates contained in a transaction

Support Counting Using Hash Tree
Subset Operation

- Enumerating subsets of three items from a transaction t



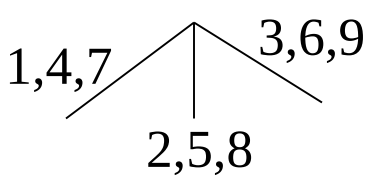
Support Counting Using Hash Tree
Generate Candidate Hash Tree

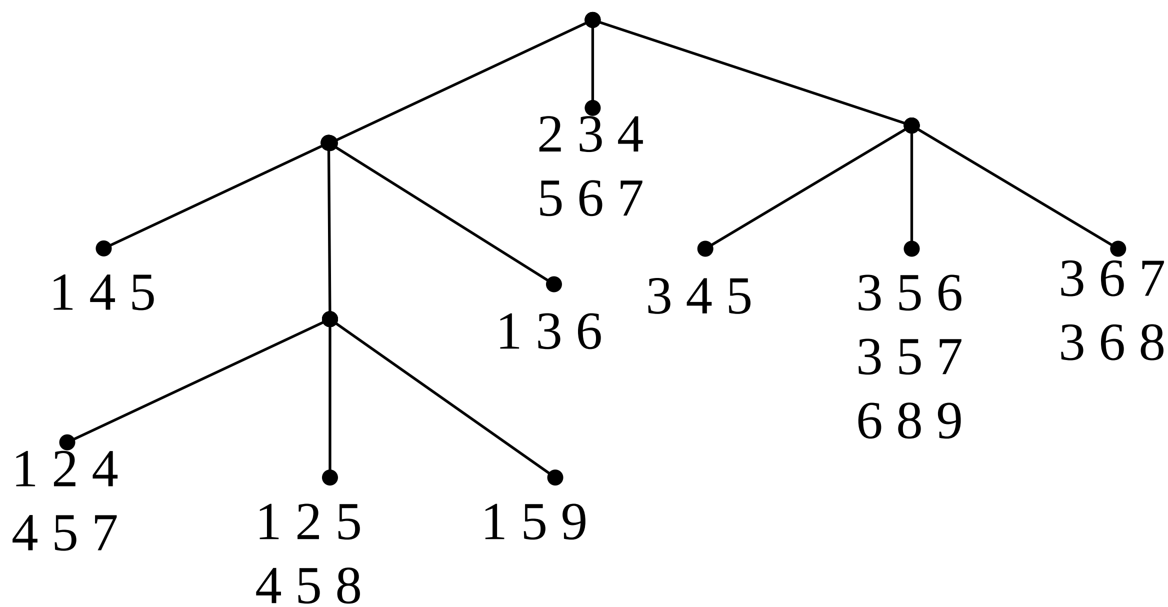
- Suppose you have 15 candidate itemsets of length 3:

  {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}

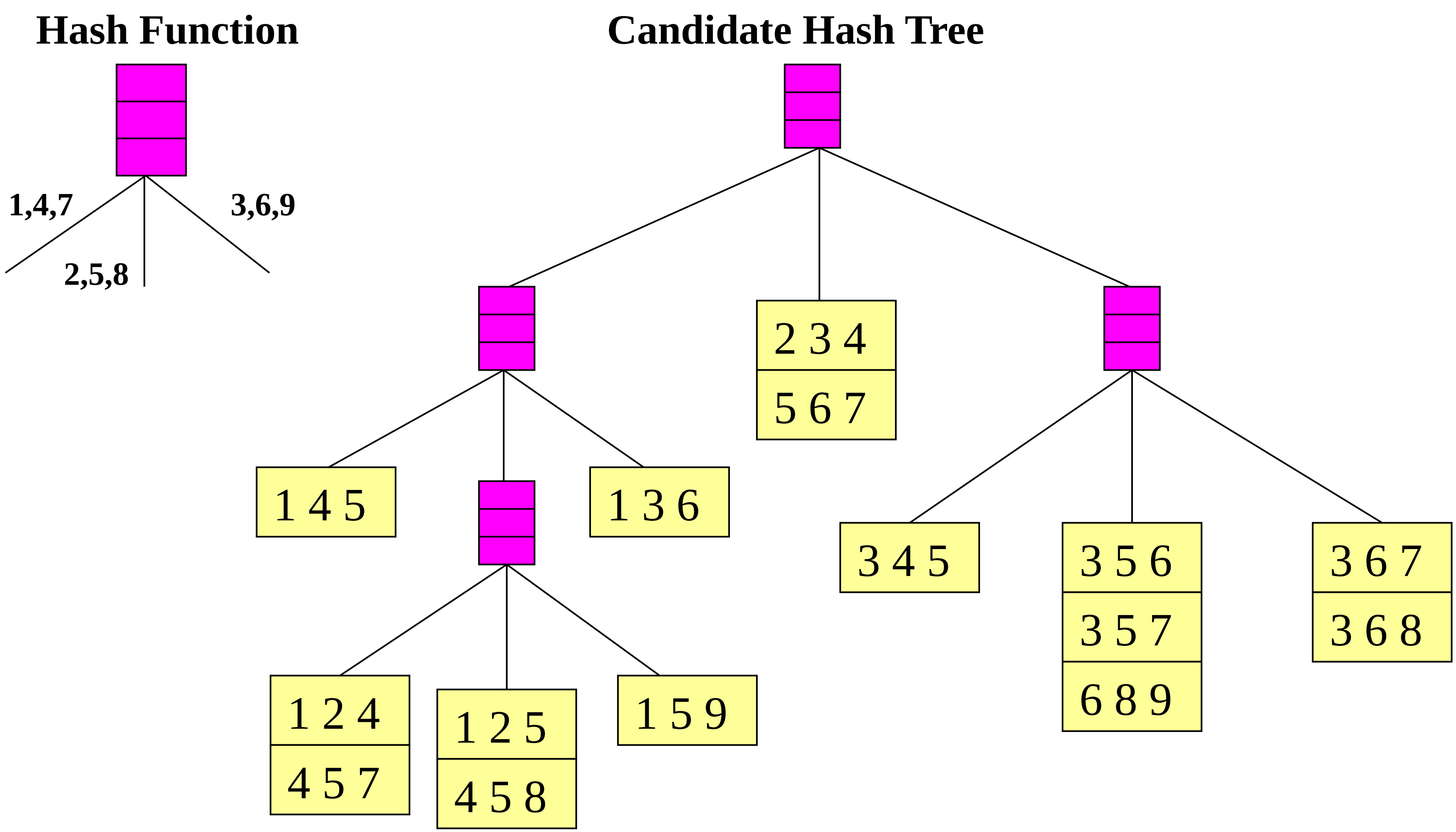
- We need: Hash function
  - HashFunc: **mod 3**

Hash function



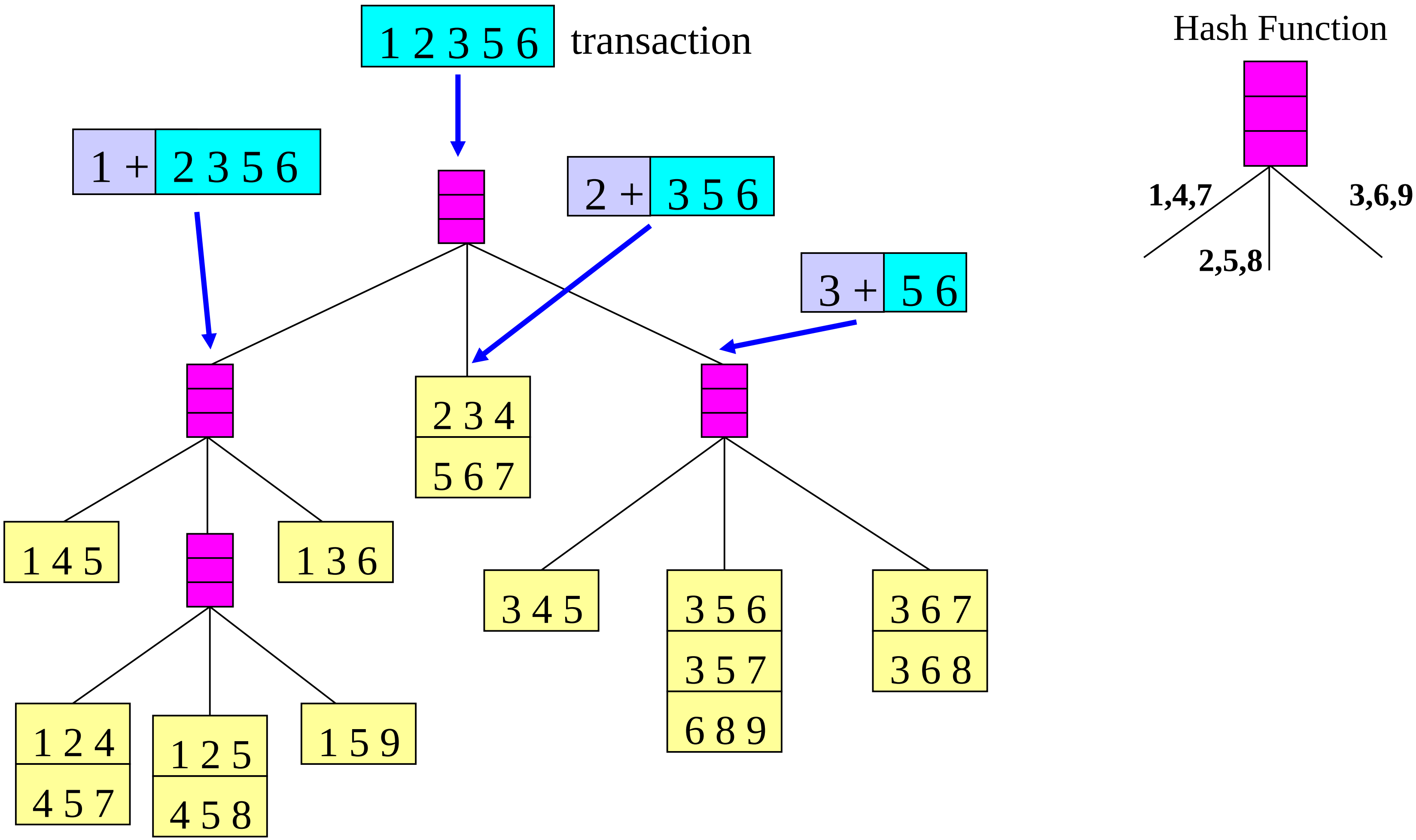


Support Counting Using Hash Tree
Generate Candidate Hash Tree



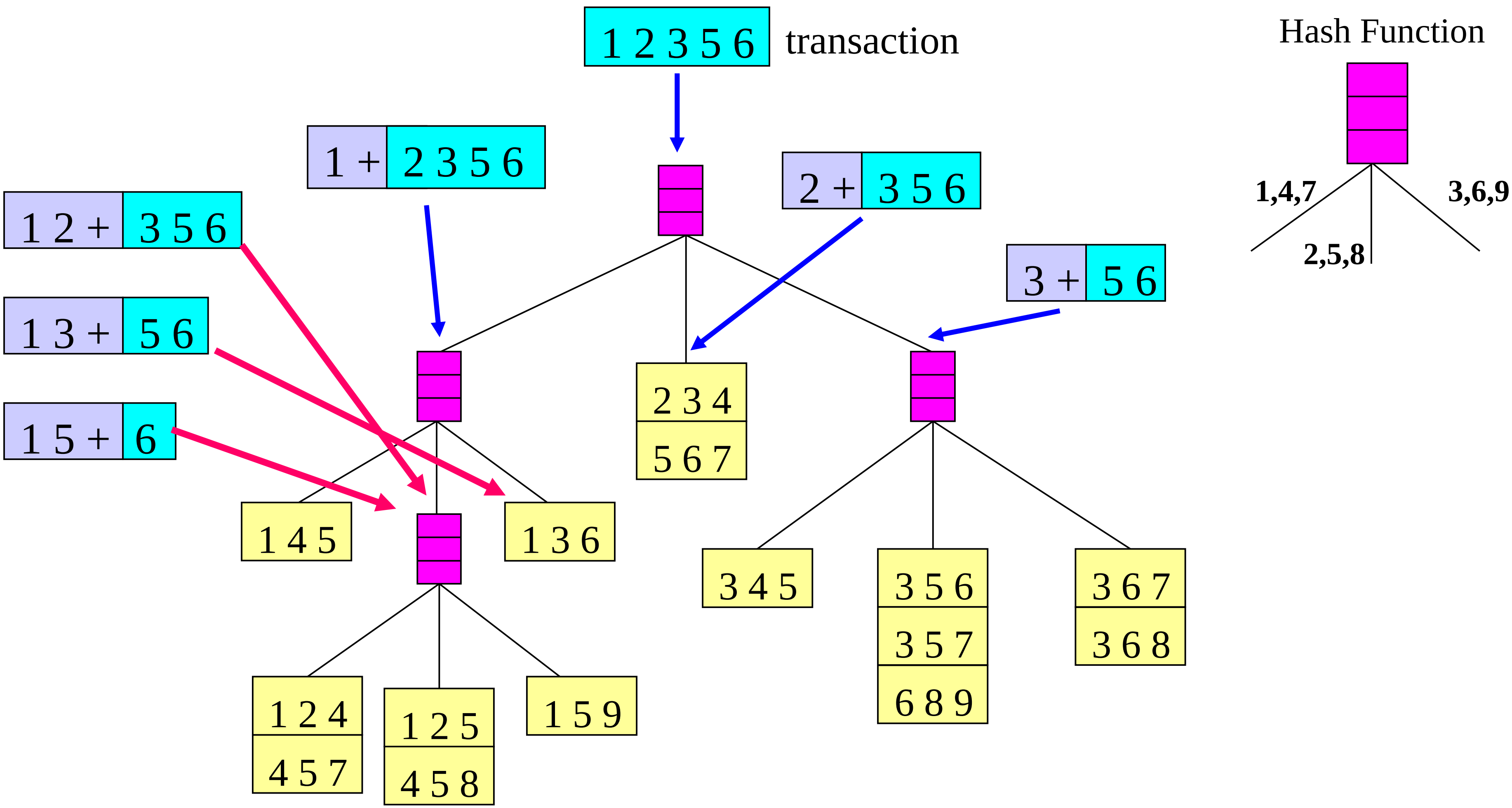
Support Counting Using Hash Tree

Traverse Candidate Hash Tree to Update Support Counts



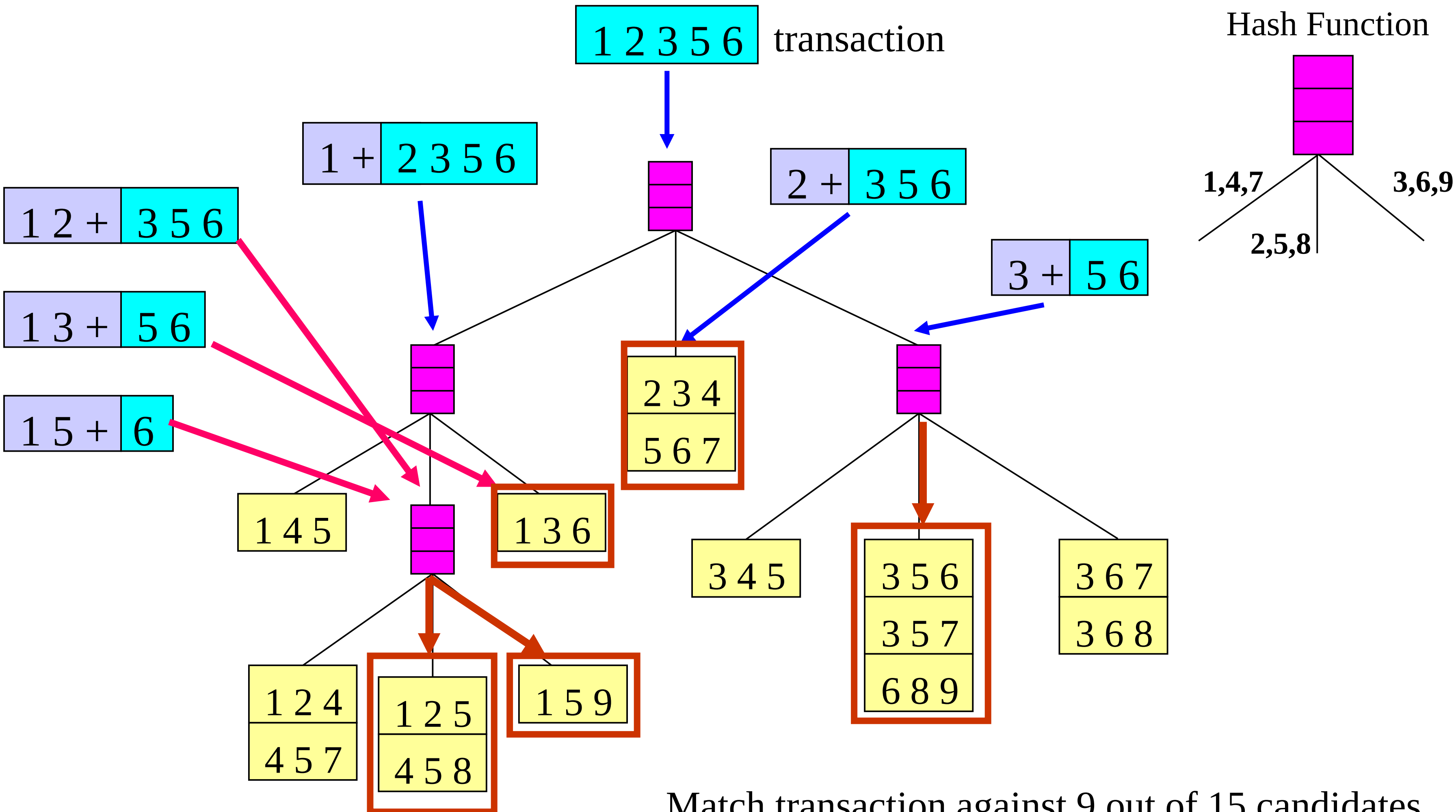
Support Counting Using Hash Tree

Traverse Candidate Hash Tree to Update Support Counts



Support Counting Using Hash Tree

Traverse Candidate Hash Tree to Update Support Counts



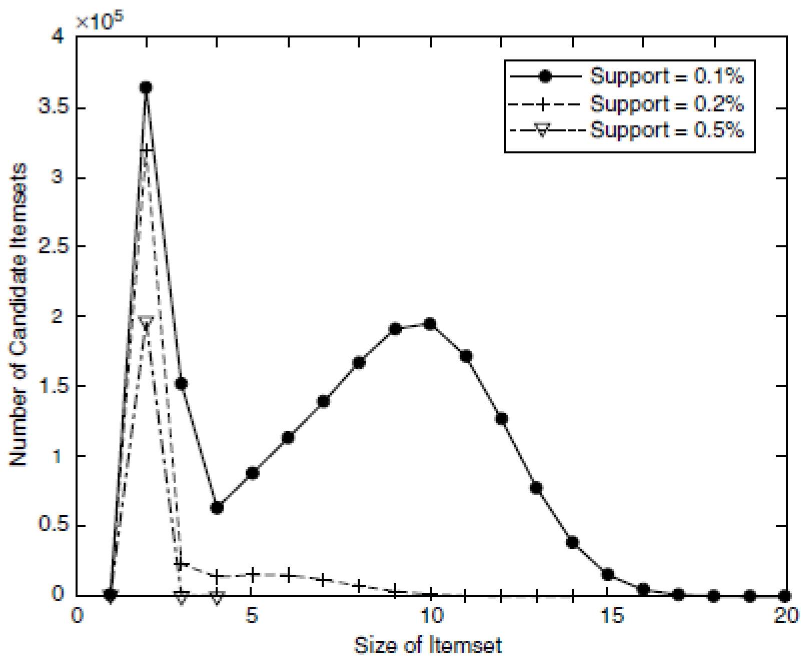
Match transaction against 9 out of 15 candidates

Factors Affecting Complexity of Apriori Algorithm

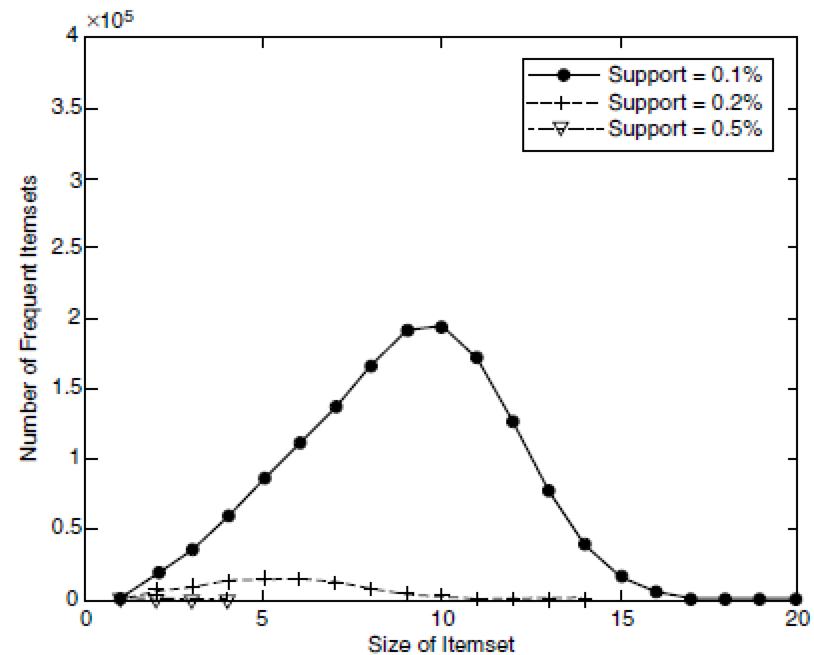
- **Choice of minimum support threshold**
  - lowering support threshold results in more frequent itemsets
  - this may increase number of candidates and max length of frequent itemsets
- **Dimensionality (number of items) of the data set**
  - more space is needed to store support count of each item
  - 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 number of subsets in a transaction increases with its width

Effect of Support Threshold

- Effect of support threshold on the number of candidate and frequent itemsets



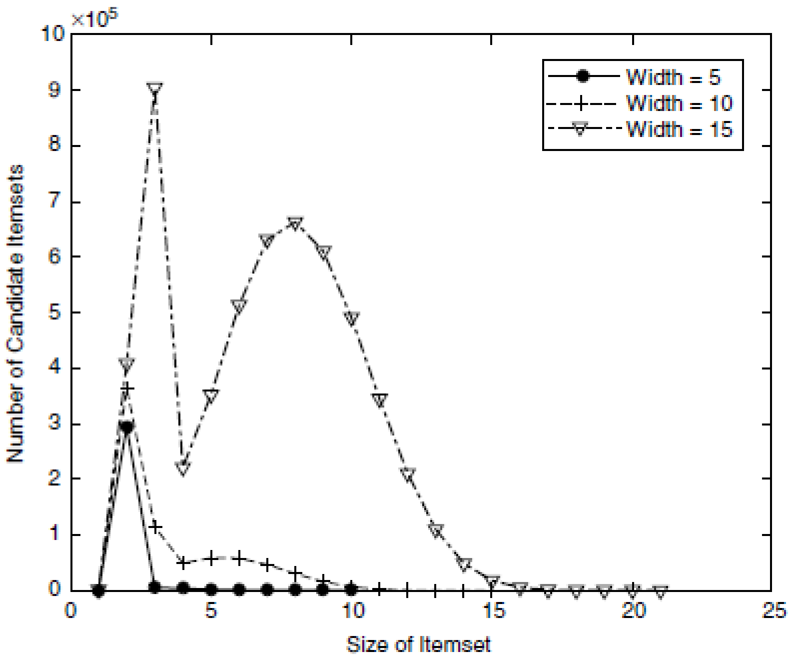
Number of candidate itemsets



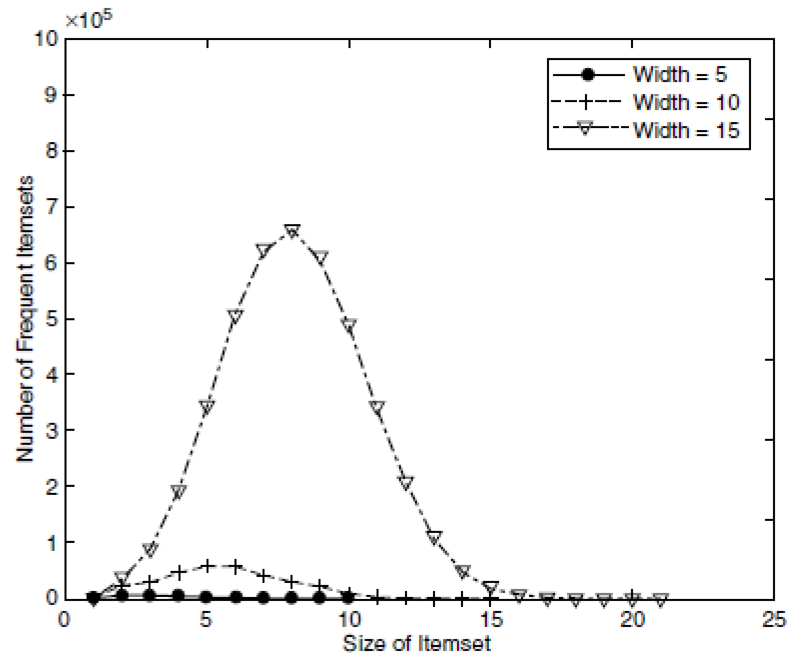
Number of frequent itemsets

Effect of Average Transaction Width

- Effect of average transaction width on the number of candidate and frequent itemsets



Number of candidate itemsets



Number of frequent itemsets

Effect of Support Distribution

- How to set the appropriate *minsup* threshold?
  - If *minsup* is set too high, we could miss itemsets involving interesting rare items (e.g., expensive products)
  - If *minsup* is set too low, it is computationally expensive and the number of itemsets is very large
- Using a single minimum support threshold may not be effective

Multiple Minimum Support

- How to apply multiple minimum supports?
  - MS(i): minimum support for item i
  - e.g.: MS(Milk)=5%, MS(Coke) = 3%,
    MS(Broccoli)=0.1%, MS(Salmon)=0.5%
  - MS({Milk, Broccoli}) = min (MS(Milk), MS(Broccoli))
    = 0.1%
  - Challenge: Support is no longer anti-monotone
    - Suppose: Support(Milk, Coke) = 1.5% and
      Support(Milk, Coke, Broccoli) = 0.5%
    - {Milk,Coke} is infrequent but {Milk,Coke,Broccoli} is frequent

Multiple Minimum Support

- Order the items according to their minimum support (in ascending order)
  - e.g.: MS(Milk)=5%, MS(Coke) = 3%, MS(Broccoli)=0.1%, MS(Salmon)=0.5%
  - Ordering: Broccoli, Salmon, Coke, Milk

- Need to modify Apriori such that:
  - L_1 : set of frequent items
  - F_1 : set of items whose support is ≥ MS(1) where MS(1) is $\min_i$(MS(i))
  - C_2 : candidate itemsets of size 2 is generated from F_1 instead of L_1

Multiple Minimum Support

- Modifications to Apriori:
  - In traditional Apriori,
    - A candidate (k+1)-itemset is generated by merging two frequent itemsets of size k
    - The candidate is pruned if it contains any infrequent subsets of size k
  - Pruning step has to be modified:
    - Prune only if subset contains the first item
    - e.g.: Candidate={Broccoli, Coke, Milk} (ordered according to minimum support)
    - {Broccoli, Coke} and {Broccoli, Milk} are frequent but {Coke, Milk} is infrequent
      - Candidate is not pruned because {Coke,Milk} does not contain the first item, i.e., Broccoli.

Rule Generation in Apriori Algorithm

- Given a frequent itemset **L**, find all non-empty subsets $\mathbf{f \subset L}$ such that **candidate rule $\mathbf{f \rightarrow L - f}$** satisfies the minimum confidence requirement
  - If {A,B,C,D} is a frequent itemset, candidate rules:

$$
\begin{array}{llll}
ABC \rightarrow D & ABD \rightarrow C & ACD \rightarrow B & BCD \rightarrow A \\
D \rightarrow ABC & C \rightarrow ABD & B \rightarrow ACD & A \rightarrow BCD \\
 & & & \\
AB \rightarrow CD & AC \rightarrow BD & AD \rightarrow BC & \\
CD \rightarrow AB & BD \rightarrow AC & BC \rightarrow AD &
\end{array}
$$

- If $|L| = k$, then there are $2^k - 2$ candidate association rules
  - (ignoring $L \rightarrow \varnothing$ and $\varnothing \rightarrow L$)

Rule Generation in Apriori Algorithm

- How to efficiently generate rules from frequent itemsets?

- In general, confidence does not have an anti-monotone property

  $c(ABC \rightarrow D)$ can be larger or smaller than $c(AB \rightarrow D)$

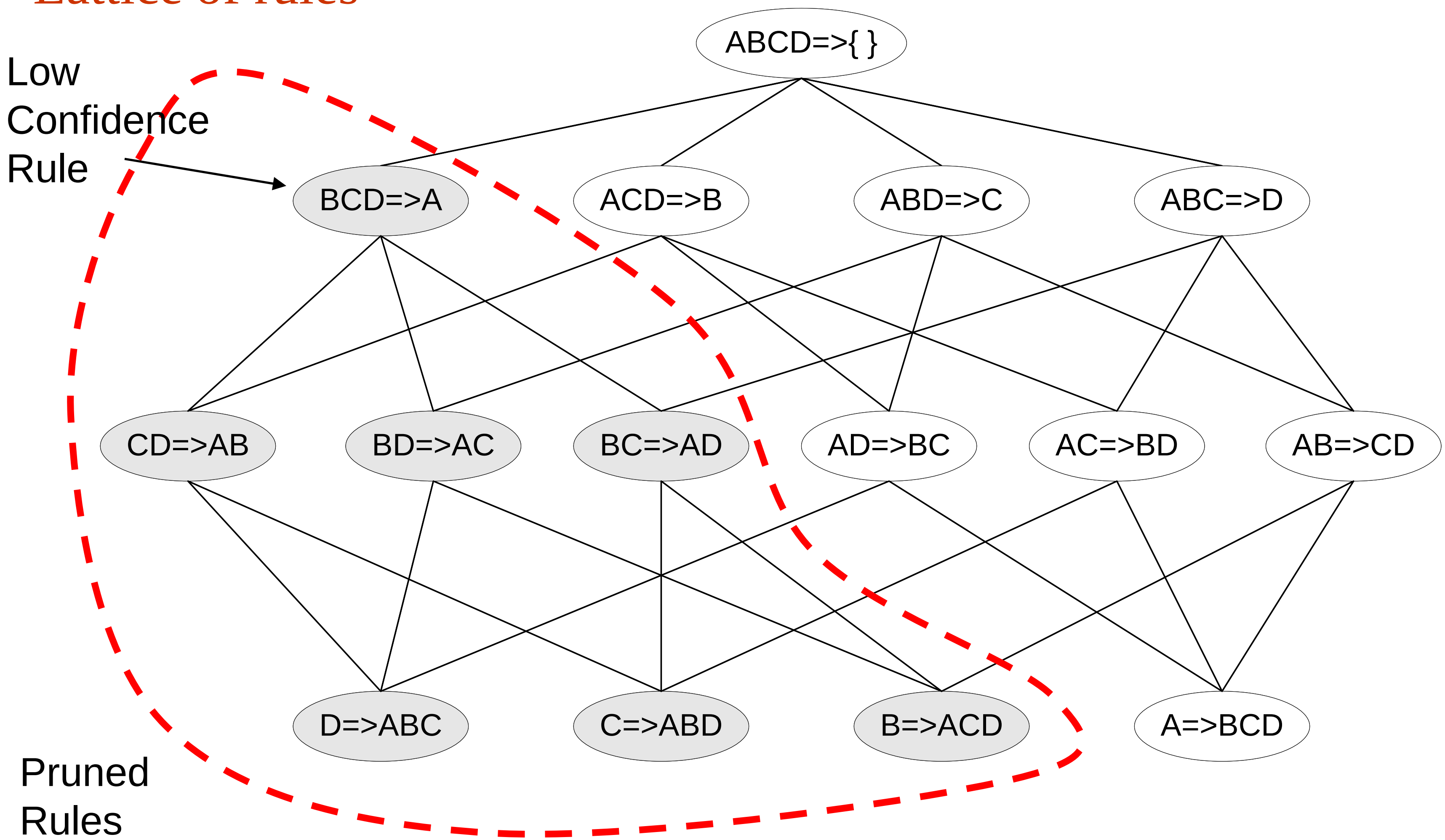
- But confidence of rules generated from the same itemset has an anti-monotone property
  - E.g., Suppose {A,B,C,D} is a frequent 4-itemset:

$$c(ABC \rightarrow D) \geq c(AB \rightarrow CD) \geq c(A \rightarrow BCD)$$

  - Confidence is anti-monotone w.r.t. number of items on the RHS of the rule

Rule Generation in Apriori Algorithm

Lattice of rules



- **Frequent Itemsets, Association Rules**
- **Apriori Algorithm**
- **Compact Representation of Frequent Itemsets**
- **FP-Growth Algorithm: An Alternative Frequent Itemset Generation Algorithm**
- **Evaluation of Association Patterns**

Compact Representation of Frequent Itemsets

- The number of frequent itemsets produced from a transaction data set can be very large.
- Some produced itemsets can be redundant because they have identical support as their supersets

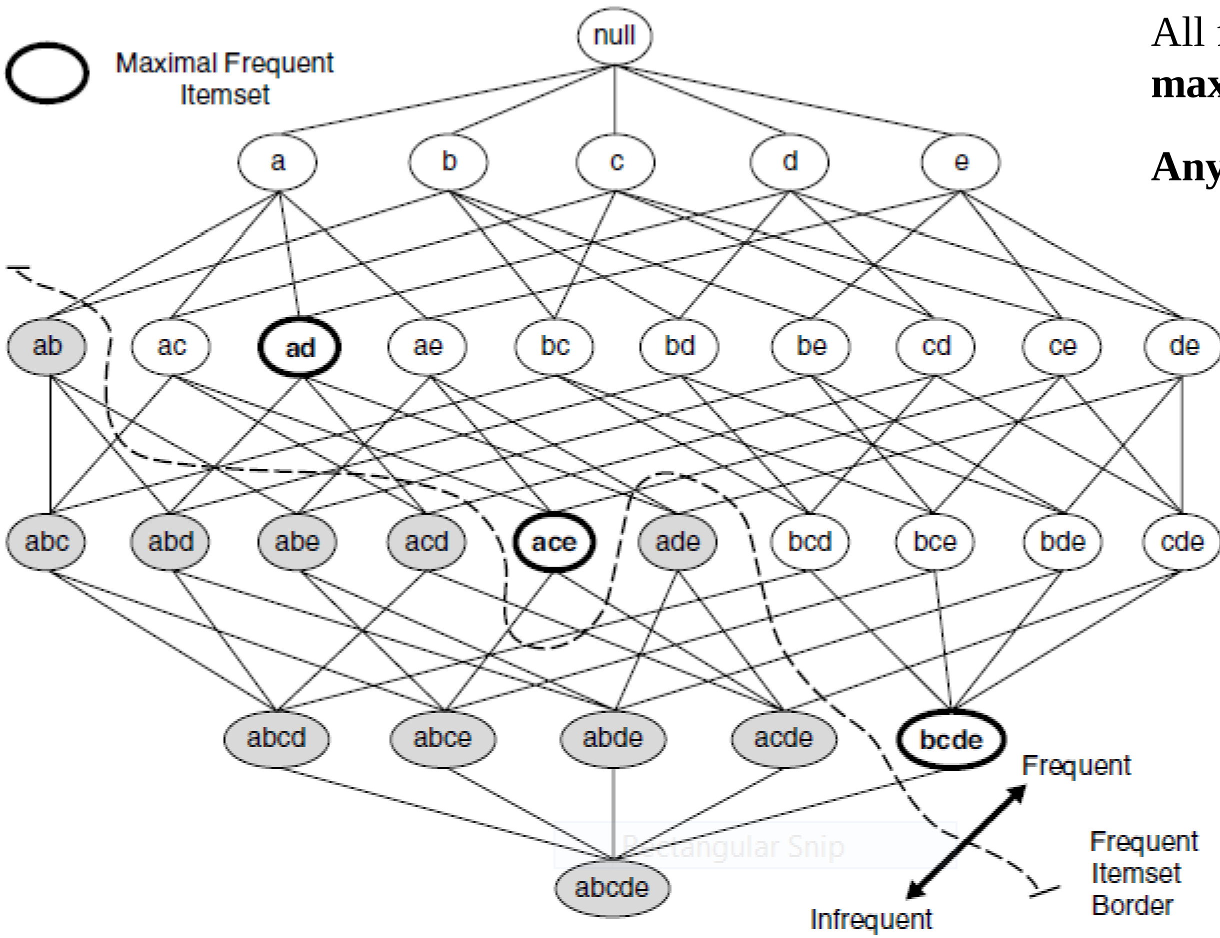
- It is useful to identify a small representative set of itemsets from which all other frequent itemsets can be derived. ➔ Need a compact representation
  - **Maximal Frequent Itemsets** and
  - **Closed Frequent Itemsets**

Maximal Frequent Itemsets

Maximal Frequent Itemset: A maximal frequent itemset is defined as a frequent itemset for which none of its immediate supersets are frequent.

- Maximal frequent itemsets effectively provide a compact representation of frequent itemsets.
- Maximal frequent itemsets form the smallest set of itemsets from which all frequent itemsets can be derived.

Maximal Frequent Itemsets



All frequent itemsets can be derived from **maximal frequent itemsets** **ad**, **ace**, **bcde**.

Any frequent itemset $\subseteq$
a maximal frequent itemset

Maximal Frequent Itemsets

- Despite providing a compact representation, maximal frequent itemsets do not contain the support information of their subsets.
- For example, the support of the maximal frequent itemsets {a, c, e}, {a, d}, and {b,c,d,e} do not provide any hint about the support of their subsets.
- An additional pass over the data set is therefore needed to determine the support counts of the non-maximal frequent itemsets.

- It might be desirable to have a minimal representation of frequent itemsets that preserves the support information. ➔ **Closed Frequent Itemsets**

Closed Frequent Itemsets

Closed Itemset: An itemset X is closed if none of its immediate supersets has exactly the same support count as X.

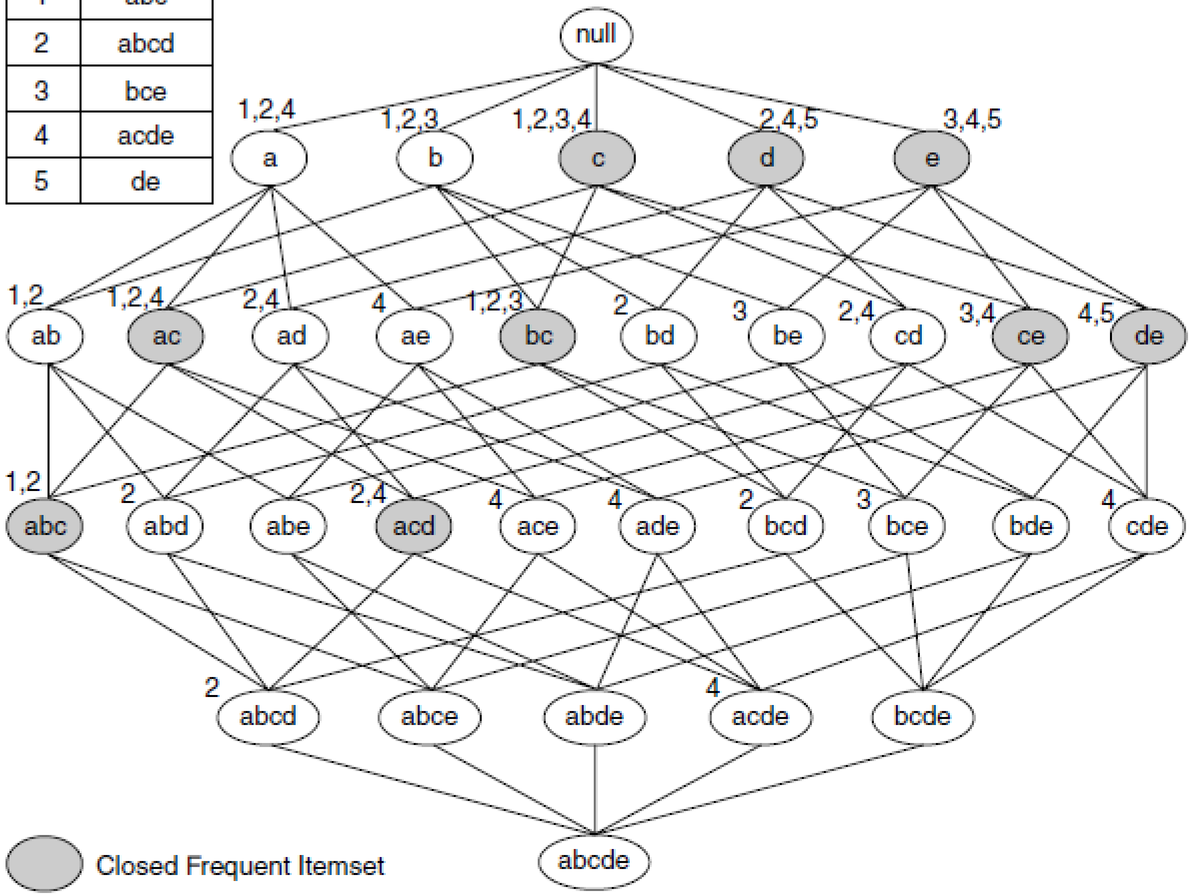
- **Closed itemsets** provide a minimal representation of itemsets without losing their support information.
- Put another way, X is not closed if at least one of its immediate supersets has the same support count as X.

Closed Frequent Itemset: An itemset is a closed frequent itemset if it is closed and its support is greater than or equal to *minsup*.

Closed Frequent Itemsets

TID	Items
1	abc
2	abcd
3	bce
4	acde
5	de

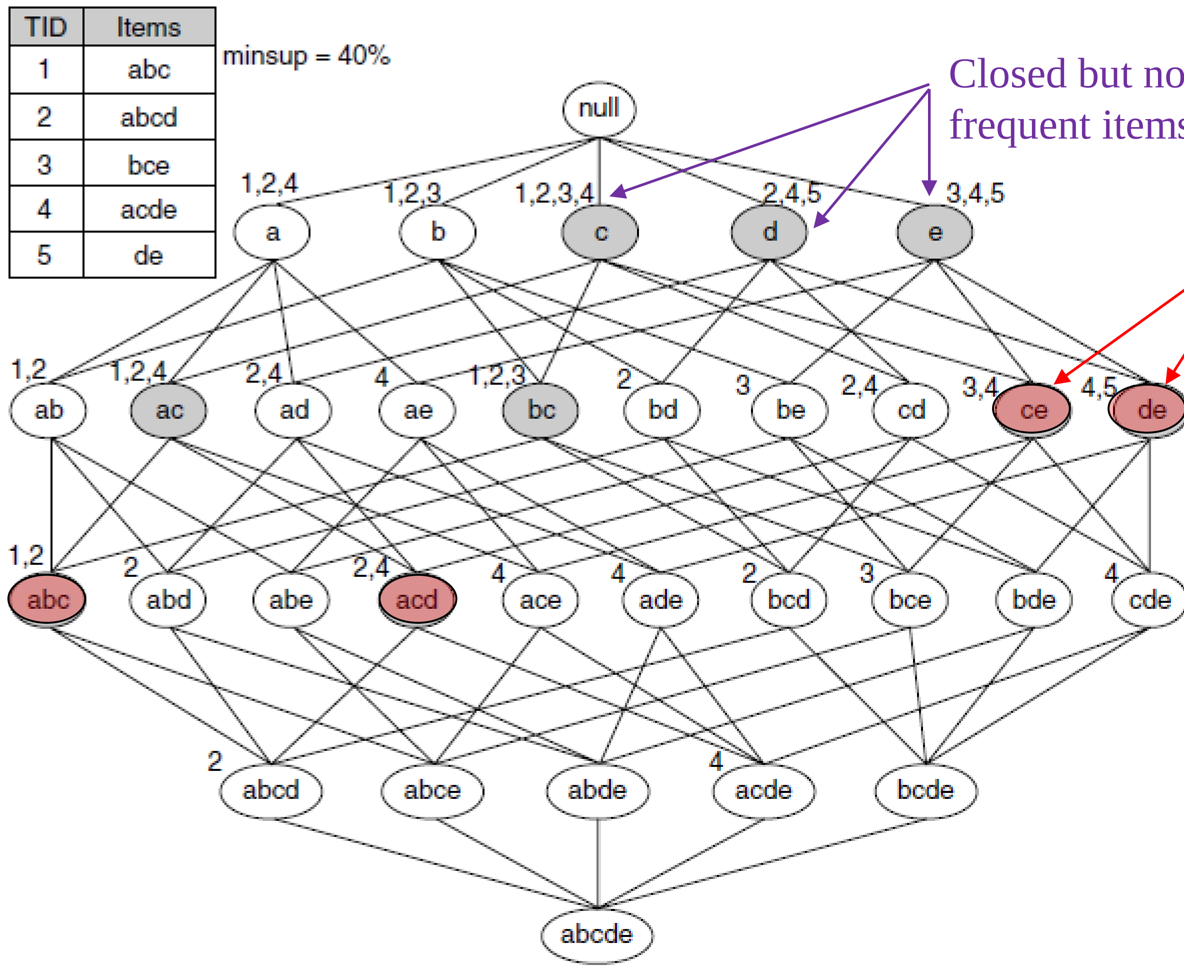
minsup = 40%



All subsets of a closed frequent itemset are frequent and their supports is greater than or equal to the support of that closed frequent itemset.

For example, all subsets of a closed frequent itemset **abc** are frequent and their supports ≥ support of **abc**.

Maximal vs Closed Itemsets



TID	Items
1	abc
2	abcd
3	bce
4	acde
5	de

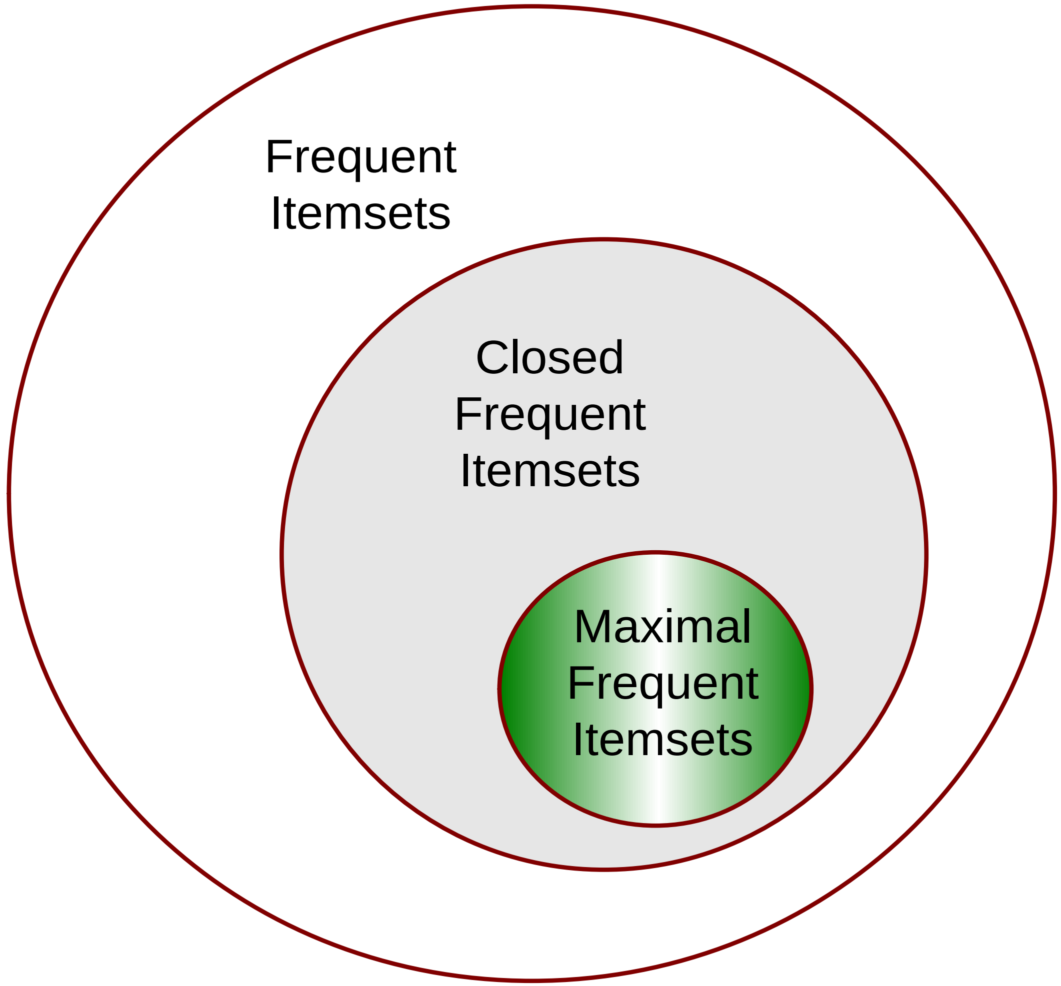
Closed but not maximal frequent itemsets

Closed and maximal frequent itemsets

Closed = 9

Maximal = 4

Maximal vs Closed Itemsets



- **Frequent Itemsets, Association Rules**
- **Apriori Algorithm**
- **Compact Representation of Frequent Itemsets**
- **FP-Growth Algorithm: An Alternative Frequent Itemset Generation Algorithm**
- **Evaluation of Association Patterns**

FP-Growth (Frequent Pattern Growth) Algorithm

- **FP-growth algorithm** that takes a radically different approach to discovering frequent itemsets.
  - The algorithm does not subscribe to the generate-and-test paradigm of Apriori

- **FP-growth algorithm** encodes the data set using a compact data structure called an **FP-tree** and extracts frequent itemsets directly from this structure.
  - Use a compressed representation of the database using an FP-tree
  - Once an FP-tree has been constructed, it uses a recursive divide-and-conquer approach to mine the frequent itemsets

FP-Tree Construction

- An FP-tree is a compressed representation of the input data.

- It is constructed by reading the data set one transaction at a time and mapping each transaction onto a path in the FP-tree.
  - Different transactions can have several items in common, their paths may overlap.
  - The more the paths overlap with one another, the more compression we can achieve using the FP-tree structure.

FP-Tree Construction

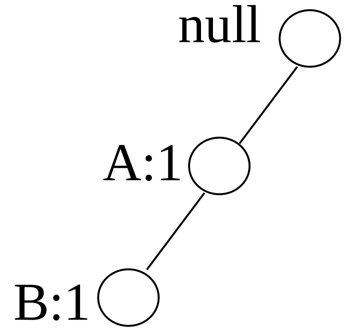
- Each node in the tree contains the label of an item along with a counter that shows the number of transactions mapped onto the given path.
  - Initially, the FP-tree contains only the root node represented by the null symbol.
  - Every transaction maps onto one of the paths in the FP-tree.

- The size of an FP-tree is typically smaller than the size of the uncompressed data because many transactions in market basket data often share a few items in common.
  - best-case scenario, all transactions have same set of items
    ➔ FP-tree contains only a single branch.
  - worst-case scenario happens when every transaction has a unique set of items
    ➔ FP-tree is effectively the same as the size of the original data.
  - physical storage requirement for FP-tree is higher because it requires additional space to store pointers between nodes and counters for each item.

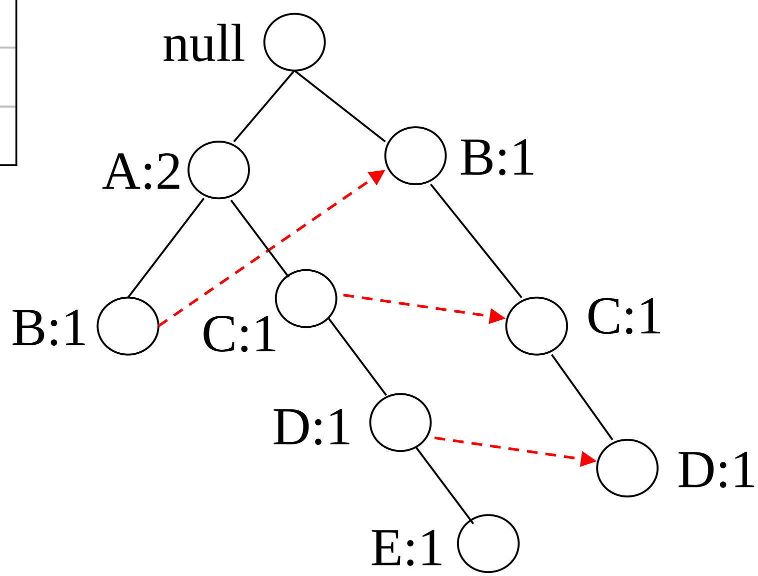
FP-Tree Construction

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	{B,C}
8	{A,B,C}
9	{A,B,D}
10	{B,C,E}

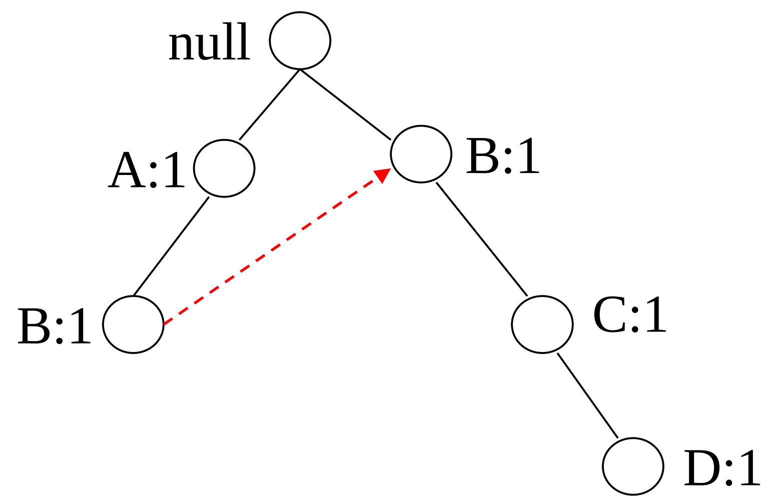
After reading TID=1:



After reading TID=3:



After reading TID=2:

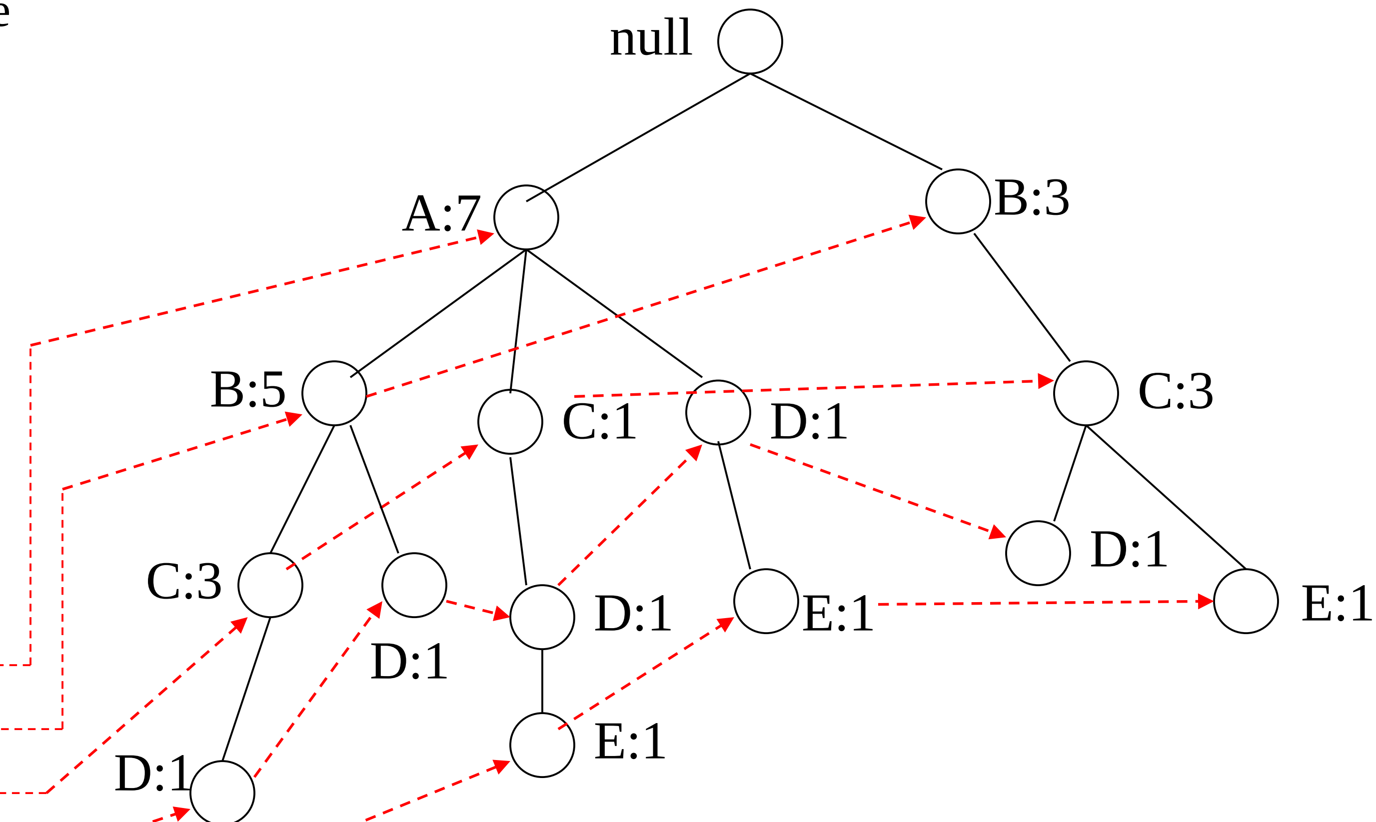


FP-Tree Construction

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	{B,C}
8	{A,B,C}
9	{A,B,D}
10	{B,C,E}

Transaction Database

After reading all transactions:



Header table

Item	Pointer
A	
B	
C	
D	
E	

sorted

Pointers are used to assist frequent itemset generation

Frequent Itemset Generation in FP-Growth Algorithm

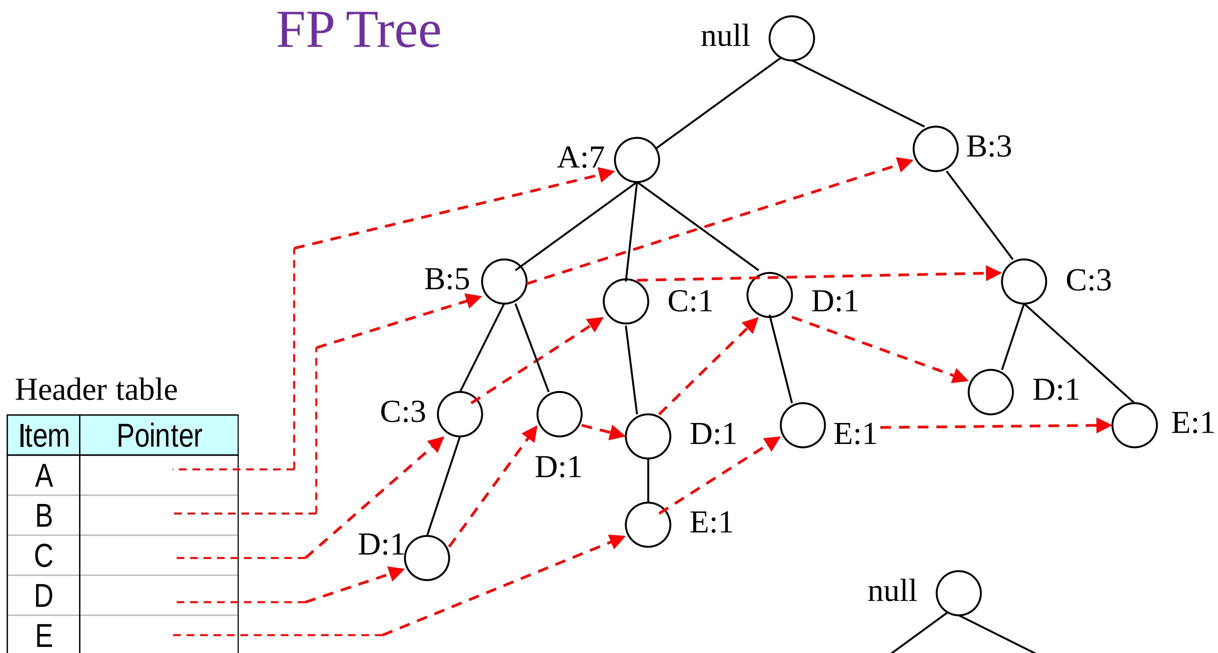
- FP-growth is an algorithm that generates frequent itemsets from an FP-tree by exploring the tree in a bottom-up fashion.
  - This bottom-up strategy for finding frequent itemsets ending with a particular item is equivalent to the suffix-based approach
  - Since every transaction is mapped onto a path in the FP-tree, we can derive the frequent itemsets ending with a particular item, say **e**, by examining only the paths containing node **e**.
  - The algorithm looks for frequent itemsets ending in **e** first, followed by **d**, **c**, **b**, and finally, **a**.
- FP-growth finds all the frequent itemsets ending with a particular suffix by employing a divide-and-conquer strategy to split the problem into smaller subproblems.
  - To find all frequent itemsets ending in **e**, we must first check whether the itemset **{e}** itself is frequent.
  - If it is frequent, we consider the subproblem of finding frequent itemsets ending in **de**, followed by **ce**, **be**, and **ae**.
  - In turn, each of these subproblems are further decomposed into smaller subproblems.
  - By merging the solutions obtained from the subproblems, all the frequent itemsets ending in **e** can be found.

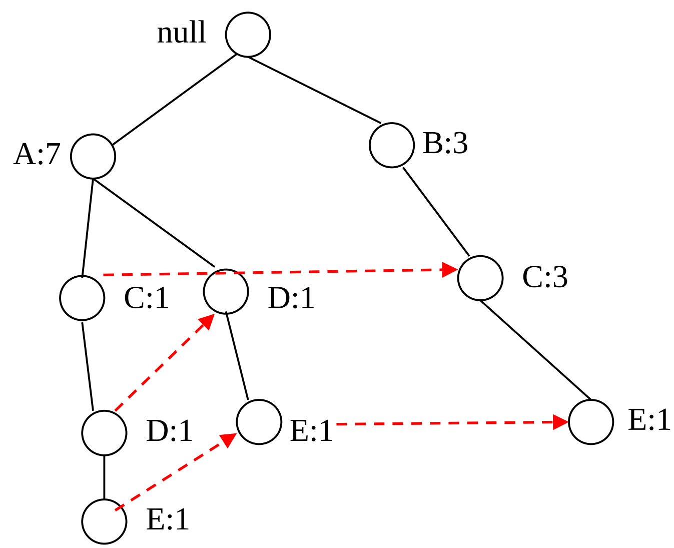
Finding Frequent Itemsets Ending with e

1. The first step is to gather all the paths containing node **e**. These initial paths are called ***prefix paths***
2. From the prefix paths, the support count for **e** is obtained by adding the support counts associated with node **e**. Assuming that the minimum support count is 2, **{e}** is declared a frequent itemset because its support count is 3.
3. Because **{e}** is frequent, the algorithm has to solve the subproblems of finding frequent itemsets ending in **de**, **ce**, **be**, and **ae**. Before solving these subproblems, it must first convert the prefix paths into a ***conditional FP-tree***, which is structurally similar to an FP-tree, except it is used to find frequent itemsets ending with a particular suffix.
   - First, the support counts along the prefix paths must be updated because some of the counts include transactions that do not contain item **e**.
   - The prefix paths are truncated by removing the nodes for **e**.
   - After updating the support counts along the prefix paths, some of the items may no longer be frequent
     - the node b appears only once and has a support count equal to 1, which means that there is only one transaction that contains both b and e. Item b can be safely ignored from subsequent analysis because all itemsets ending in be must be infrequent.
4. FP-growth uses the conditional FP-tree for **e** to solve the subproblems of finding frequent itemsets ending in **de**, **ce**, and **ae**.

Prefix Paths Ending with e

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	{B,C}
8	{A,B,C}
9	{A,B,D}
10	{B,C,E}





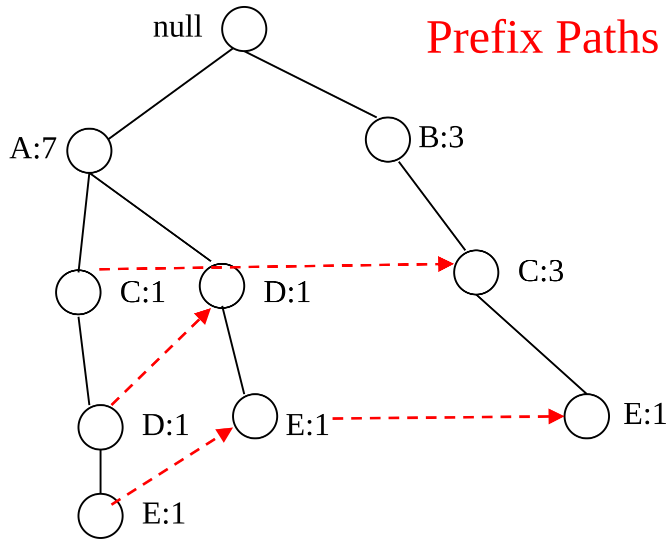
Prefix Paths Ending with e

Conditional FP-Tree for e

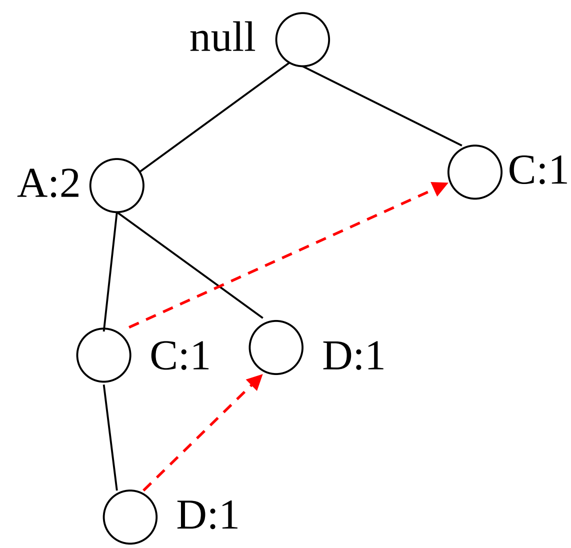
TID	Items
1	{A,B}
2	{B,C,D}
3	{A,C,D,E}
4	{A,D,E}
5	{A,B,C}
6	{A,B,C,D}
7	{B,C}
8	{A,B,C}
9	{A,B,D}
10	{B,C,E}

minsup=2

Prefix Paths Ending with e



Conditional FP-Tree for e



To create Conditional FP-Tree for e

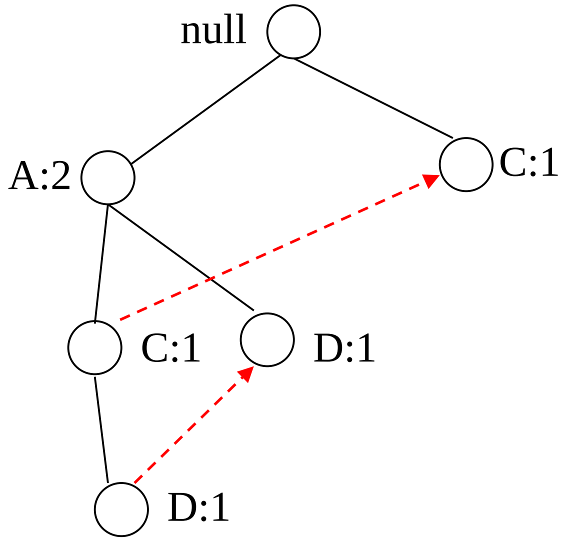
- Update support counts because paths without e are removed
- e is frequent (support=3), Remove e nodes from prefix paths
- Remove infrequent nodes

Conditional FP-Tree for de

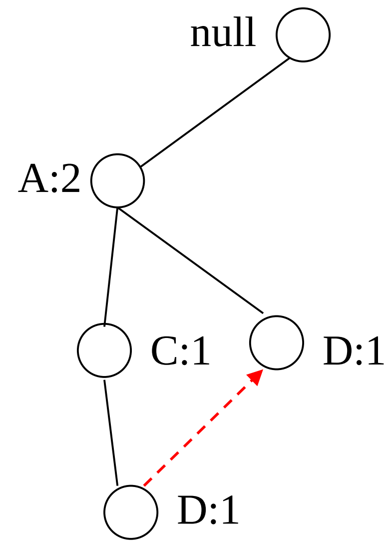
TID	Items
1	{A,B}
2	{B,C,D}
3	{A,C,D,E}
4	{A,D,E}
5	{A,B,C}
6	{A,B,C,D}
7	{B,C}
8	{A,B,C}
9	{A,B,D}
10	{B,C,E}

minsup=2

Conditional FP-Tree for e

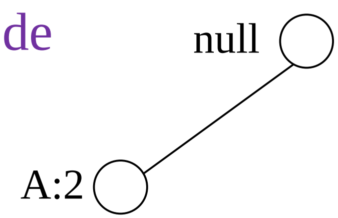


Prefix Paths Ending with de



de is frequent (support=2)

Conditional FP-Tree for de

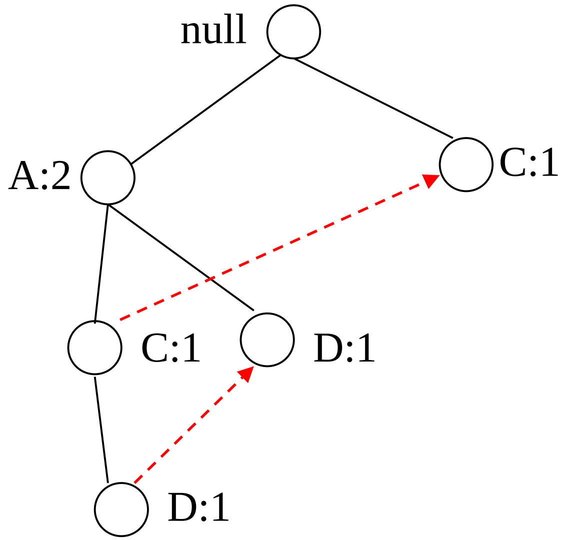


Conditional FP-Tree for ce

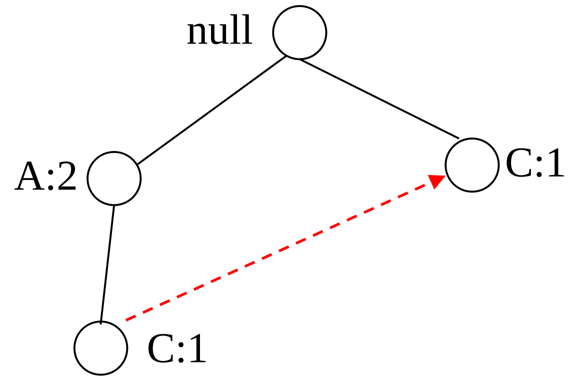
TID	Items
1	{A,B}
2	{B,C,D}
3	{A,C,D,E}
4	{A,D,E}
5	{A,B,C}
6	{A,B,C,D}
7	{B,C}
8	{A,B,C}
9	{A,B,D}
10	{B,C,E}

minsup=2

Conditional FP-Tree for e

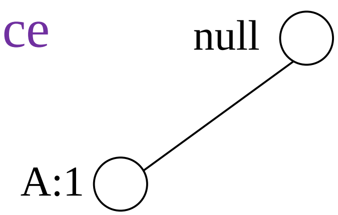


Prefix Paths Ending with ce



ce is frequent (support=2)

Conditional FP-Tree for ce

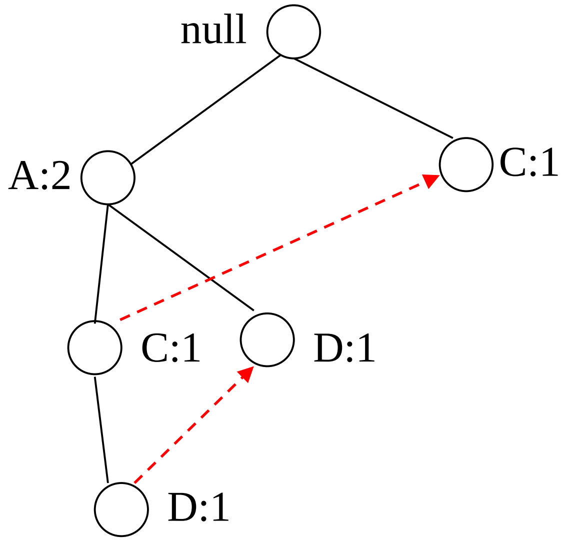


Conditional FP-Tree for ae

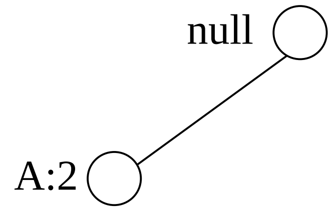
TID	Items
1	{A,B}
2	{B,C,D}
3	{A,C,D,E}
4	{A,D,E}
5	{A,B,C}
6	{A,B,C,D}
7	{B,C}
8	{A,B,C}
9	{A,B,D}
10	{B,C,E}

minsup=2

Conditional FP-Tree for e



Prefix Paths Ending with ae



ae is frequent (support=2)

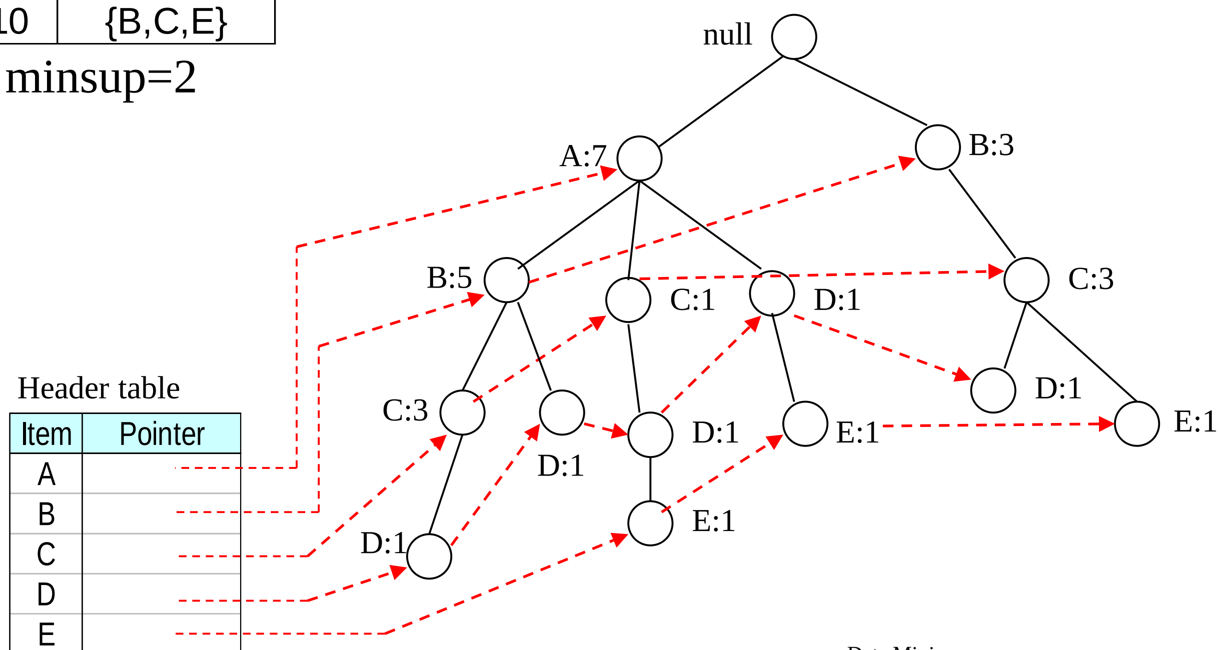
Conditional FP-Tree for ae

Frequent Itemsets Ordered by Suffixes

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	{B,C}
8	{A,B,C}
9	{A,B,D}
10	{B,C,E}

minsup=2

Suffix	Frequent Itemsets
E	{E}, {D,E}, {A,D,E}, {C,E}, {A, E},
D	{D}, {C,D}, {B,C,D}, {A,C,D}, {B,D}, {A,B,D}, {A,D}
C	{C}, {B,C}, {A,B,C}, {A,C}
B	{B}, {A,B}
A	{A}



FP Tree

- **Frequent Itemsets, Association Rules**
- **Apriori Algorithm**
- **Compact Representation of Frequent Itemsets**
- **FP-Growth Algorithm: An Alternative Frequent Itemset Generation Algorithm**
- **Evaluation of Association Patterns**

Evaluation of Association Patterns

- Association rule algorithms tend to produce too many rules
  - many of them are ***uninteresting*** or ***redundant***
  - {A,B} $\rightarrow$ {D} is ***Redundant*** if {A,B,C} $\rightarrow$ {D} and {A,B} $\rightarrow$ {D} have same support & confidence
  - An association rule X −→ Y is **redundant** if there exists another rule X'→ Y', where X is a subset of X' and Y is a subset of Y', such that the support and confidence for both rules are identical.

- ***Interestingness measure*** can be used to prune/rank the derived patterns

- In the original formulation of association rules, support & confidence are the only measures used

Computing Interestingness Measure

- Given a rule X → Y, information needed to compute rule interestingness can be obtained from a contingency table

Contingency table for $X \rightarrow Y$

	Y	$\overline{Y}$	
X	f_{11}	f_{10}	f_{1+}
$\overline{X}$	f_{01}	f_{00}	f_{0+}
	f_{+1}	f_{+0}	$\|T\|$

f_{11}: support of X and Y
f_{10}: support of X and $\overline{Y}$
f_{01}: support of $\overline{X}$ and Y
f_{00}: support of $\overline{X}$ and $\overline{Y}$

Drawback of Confidence

	Coffee	$\overline{\text{Coffee}}$	
Tea	15	5	20
$\overline{\text{Tea}}$	75	5	80
	90	10	100

Association Rule: Tea $\rightarrow$ Coffee

Confidence= P(Coffee|Tea) = 0.75 = support({Tea,Coffee}) / support({Tea})

but P(Coffee) = 0.9

$\Rightarrow$ Although confidence is high, rule is misleading

$\Rightarrow$ $P(\text{Coffee}|\overline{\text{Tea}}) = 0.9375$

Measure for Association Rules

- So, what kind of rules do we really want?
  - Confidence($X \rightarrow Y$) should be sufficiently high
    - To ensure that people who buy X will more likely buy Y than not buy Y
  - Confidence($X \rightarrow Y$) > support(Y)
    - Otherwise, rule will be misleading because having item X actually reduces the chance of having item Y in the same transaction
    - Is there any measure that capture this constraint?
      - Answer: Yes. There are many of them.

Statistical Independence

- Population of 1000 students
  - 600 students know how to swim (S)
  - 700 students know how to bike (B)
  - 420 students know how to swim and bike (S,B)

  - $P(S \wedge B) = 420/1000 = 0.42$
  - $P(S) \times P(B) = 0.6 \times 0.7 = 0.42$

  - $P(S \wedge B) = P(S) \times P(B)$ => Statistical independence
  - $P(S \wedge B) > P(S) \times P(B)$ => Positively correlated
  - $P(S \wedge B) < P(S) \times P(B)$ => Negatively correlated

Statistical-Based Measures for Interestingness

- Statistical-Based Measures use statistical dependence information.
- Two of them are **Lift** and **Interest** (they are equal).

$$\text{Lift} = P(Y|X) \,/\, P(Y)$$

$$\text{Interest} = P(X,Y) \,/\, P(X)\,P(Y)$$

$$\text{Lift}(A,B) = \text{conf}(A \rightarrow B) \,/\, \text{support}(B)$$

$$= \text{support}(A \cup B) \,/\, \text{support}(A)\ \text{support}(B)$$

$$\text{Interest}(A,B) = \text{support}(A \cup B) \,/\, \text{support}(A)\ \text{support}(B)$$

$$\text{Interest}(A,B) \begin{cases} = 1 & \mathit{if\ A\ and\ B\ are\ independent} \\ > 1 & \mathit{if\ A\ and\ B\ are\ positively\ correlated} \\ < 1 & \mathit{if\ A\ and\ B\ are\ negatively\ correlated} \end{cases}$$

Example: Lift/Interest

	Coffee	$\overline{\text{Coffee}}$	
Tea	15	5	20
$\overline{\text{Tea}}$	75	5	80
	90	10	100

Association Rule: Tea $\rightarrow$ Coffee

Confidence= P(Coffee|Tea) = 0.75 = support({Tea,Coffee}) / support({Tea})

but P(Coffee) = 0.9

➔ Lift = 0.75/0.9 = 0.8333 (< 1, therefore is negatively correlated)

Example: Lift/Interest

- *play basketball* $\Rightarrow$ *eat cereal* [40%, 66.7%] is misleading
  - The overall % of students eating cereal is 75% > 66.7%.
- *play basketball* $\Rightarrow$ *not eat cereal* [20%, 33.3%] is more accurate, although with lower support and confidence

	Basketball	Not basketball	Sum (row)
Cereal	2000	1750	3750
Not cereal	1000	250	1250
Sum(col.)	3000	2000	5000

$$lift(B,C) = \frac{2000/5000}{3000/5000*3750/5000} = 0.89$$

$$lift(B,\neg C) = \frac{1000/5000}{3000/5000*1250/5000} = 1.33$$

Limitations of Interest Factor

- We expect the words *data* and *mining* to appear together more frequently than the words *compiler* and *mining* in a collection of computer science articles.

	p	$\overline{p}$	
q	880	50	930
$\overline{q}$	50	20	70
	930	70	1000

	r	$\overline{r}$	
s	20	50	70
$\overline{s}$	50	880	930
	70	930	1000

Contingency tables for word pairs { p,q} and { r,s}.

- The interest factor for {p,q} is 1.02 and for {r, s} is 4.08.
  - Although p and q appear together in 88% of the documents, their interest factor is close to 1, which is the value when p and q are statistically independent.
  - On the other hand, the interest factor for {r, s} is higher than {p, q} even though r and s seldom appear together in the same document.
  - Confidence is perhaps the better choice in this situation because it considers the association between p and q (94.6%) to be much stronger than that between r and s (28.6%).

Different Measures

- There are lots of measures proposed in the literature
- Some measures are good for certain applications, but not for others
- What criteria should we use to determine whether a measure is good or bad?

#	Measure	Formula
1	ϕ-coefficient	$\frac{P(A,B)-P(A)P(B)}{\sqrt{P(A)P(B)(1-P(A))(1-P(B))}}$
2	Goodman-Kruskal's (λ)	$\frac{\sum_j \max_k P(A_j,B_k)+\sum_k \max_j P(A_j,B_k)-\max_j P(A_j)-\max_k P(B_k)}{2-\max_j P(A_j)-\max_k P(B_k)}$
3	Odds ratio (α)	$\frac{P(A,B)P(\overline{A},\overline{B})}{P(A,\overline{B})P(\overline{A},B)}$
4	Yule's Q	$\frac{P(A,B)P(\overline{AB})-P(A,\overline{B})P(\overline{A},B)}{P(A,B)P(\overline{AB})+P(A,\overline{B})P(\overline{A},B)} = \frac{\alpha-1}{\alpha+1}$
5	Yule's Y	$\frac{\sqrt{P(A,B)P(\overline{AB})}-\sqrt{P(A,\overline{B})P(\overline{A},B)}}{\sqrt{P(A,B)P(\overline{AB})}+\sqrt{P(A,\overline{B})P(\overline{A},B)}} = \frac{\sqrt{\alpha}-1}{\sqrt{\alpha}+1}$
6	Kappa (κ)	$\frac{P(A,B)+P(\overline{A},\overline{B})-P(A)P(B)-P(\overline{A})P(\overline{B})}{1-P(A)P(B)-P(\overline{A})P(\overline{B})}$
7	Mutual Information (M)	$\frac{\sum_i \sum_j P(A_i,B_j) \log \frac{P(A_i,B_j)}{P(A_i)P(B_j)}}{\min(-\sum_i P(A_i) \log P(A_i), -\sum_j P(B_j) \log P(B_j))}$
8	J-Measure (J)	$\max\Big(P(A,B)\log(\frac{P(B\mid A)}{P(B)}) + P(A\overline{B})\log(\frac{P(\overline{B}\mid A)}{P(\overline{B})}),$ $P(A,B)\log(\frac{P(A\mid B)}{P(A)}) + P(\overline{A}B)\log(\frac{P(\overline{A}\mid B)}{P(\overline{A})})\Big)$
9	Gini index (G)	$\max\Big(P(A)[P(B\mid A)^2 + P(\overline{B}\mid A)^2] + P(\overline{A})[P(B\mid \overline{A})^2 + P(\overline{B}\mid \overline{A})^2]$ $-P(B)^2 - P(\overline{B})^2,$ $P(B)[P(A\mid B)^2 + P(\overline{A}\mid B)^2] + P(\overline{B})[P(A\mid \overline{B})^2 + P(\overline{A}\mid \overline{B})^2]$ $-P(A)^2 - P(\overline{A})^2\Big)$
10	Support (s)	$P(A,B)$
11	Confidence (c)	$\max(P(B\mid A), P(A\mid B))$
12	Laplace (L)	$\max\Big(\frac{NP(A,B)+1}{NP(A)+2}, \frac{NP(A,B)+1}{NP(B)+2}\Big)$
13	Conviction (V)	$\max\Big(\frac{P(A)P(\overline{B})}{P(A\overline{B})}, \frac{P(B)P(\overline{A})}{P(B\overline{A})}\Big)$
14	Interest (I)	$\frac{P(A,B)}{P(A)P(B)}$
15	cosine (IS)	$\frac{P(A,B)}{\sqrt{P(A)P(B)}}$
16	Piatetsky-Shapiro's (PS)	$P(A,B) - P(A)P(B)$
17	Certainty factor (F)	$\max\Big(\frac{P(B\mid A)-P(B)}{1-P(B)}, \frac{P(A\mid B)-P(A)}{1-P(A)}\Big)$
18	Added Value (AV)	$\max(P(B\mid A) - P(B), P(A\mid B) - P(A))$
19	Collective strength (S)	$\frac{P(A,B)+P(\overline{AB})}{P(A)P(B)+P(\overline{A})P(\overline{B})} \times \frac{1-P(A)P(B)-P(\overline{A})P(\overline{B})}{1-P(A,B)-P(\overline{AB})}$
20	Jaccard (ζ)	$\frac{P(A,B)}{P(A)+P(B)-P(A,B)}$
21	Klosgen (K)	$\sqrt{P(A,B)}\max(P(B\mid A) - P(B), P(A\mid B) - P(A))$

Properties of A Good Measure

3 properties a good measure M must satisfy:

- $M(A,B) = 0$ if A and B are statistically independent
- $M(A,B)$ increase monotonically with $P(A,B)$ when $P(A)$ and $P(B)$ remain unchanged
- $M(A,B)$ decreases monotonically with $P(A)$ [or $P(B)$] when $P(A,B)$ and $P(B)$ [or $P(A)$] remain unchanged