BBS654
Data Mining

Pinar Duygulu

Slides are adapted from

Nazli Ikizler

Why?

- Retailers now have massive databases full of transactional history
  - Simply transaction date and list of items
- Is it possible to gain insights from this data?
- How are items in a database associated
  - Association Rules predict members of a set given other members in the set

Why?

- Example Rules:
  - 98% of customers that purchase tires get automotive services done
  - Customers which buy mustard and ketchup also buy burgers
  - Goal: find these rules from just transactional data
- Rules help with: store layout, buying patterns, add-on sales, etc

Association rule mining

- Proposed by Agrawal et al in 1993.
- It is an important data mining model studied extensively by the database and data mining community.
- Assume all data are categorical.
- No good algorithm for numeric data.
- Initially used for Market Basket Analysis to find how items purchased by customers are related.

Bread $\rightarrow$ Milk [sup = 5%, conf = 100%]

The model: data

- $I = \{i_1, i_2, \ldots, i_m\}$: a set of *items*.
- Transaction t :
  - t a set of items, and $t \subseteq I$.
- Transaction Database T: a set of transactions $T = \{t_1, t_2, \ldots, t_n\}$.

Transaction data: supermarket data

- Market basket transactions:

  t1: {bread, cheese, milk}

  t2: {apple, eggs, salt, yogurt}

  … …

  tn: {biscuit, eggs, milk}

- Concepts:
  - An *item*: an item/article in a basket
  - *I*: the set of all items sold in the store
  - A *transaction*: items purchased in a basket; it may have TID (transaction ID)
  - A *transactional dataset*: A set of transactions

Slide from Bing Liu

Transaction data: a set of documents

- **A text document data set. Each document is treated as a “bag” of keywords**

doc1:	Student, Teach, School
doc2:	Student, School
doc3:	Teach, School, City, Game
doc4:	Baseball, Basketball
doc5:	Basketball, Player, Spectator
doc6:	Baseball, Coach, Game, Team
doc7:	Basketball, Team, City, Game

Slide from Bing Liu

Association Rule Mining

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

Market-Basket transactions

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

Example of Association Rules

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

Implication means co-occurrence, not causality!

Applications – (1)

- Items = products; baskets = sets of products someone bought in one trip to the store.
- Example application: given that many people buy beer and diapers together:
  - Run a sale on diapers; raise price of beer.
- Only useful if many buy diapers & beer.

Applications – (2)

- Baskets = sentences; items = documents containing those sentences.
- Items that appear together too often could represent plagiarism.

Applications – (3)

- Baskets = Web pages; items = words.
- Unusual words appearing together in a large number of documents, e.g., "Brad" and "Angelina," may indicate an interesting relationship.

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 → Y, where X and Y are itemsets
  - Example:
    {Milk, Diaper} → {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:

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

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

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

Support and Confidence

- Support is important because
  - A rule that has a low support may occur simply by chance
  - A low support rule also is likely to be uninteresting from a business perspective because it may not be profitable
- Confidence measures the reliability 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

  $\Rightarrow$ Computationally prohibitive!

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)

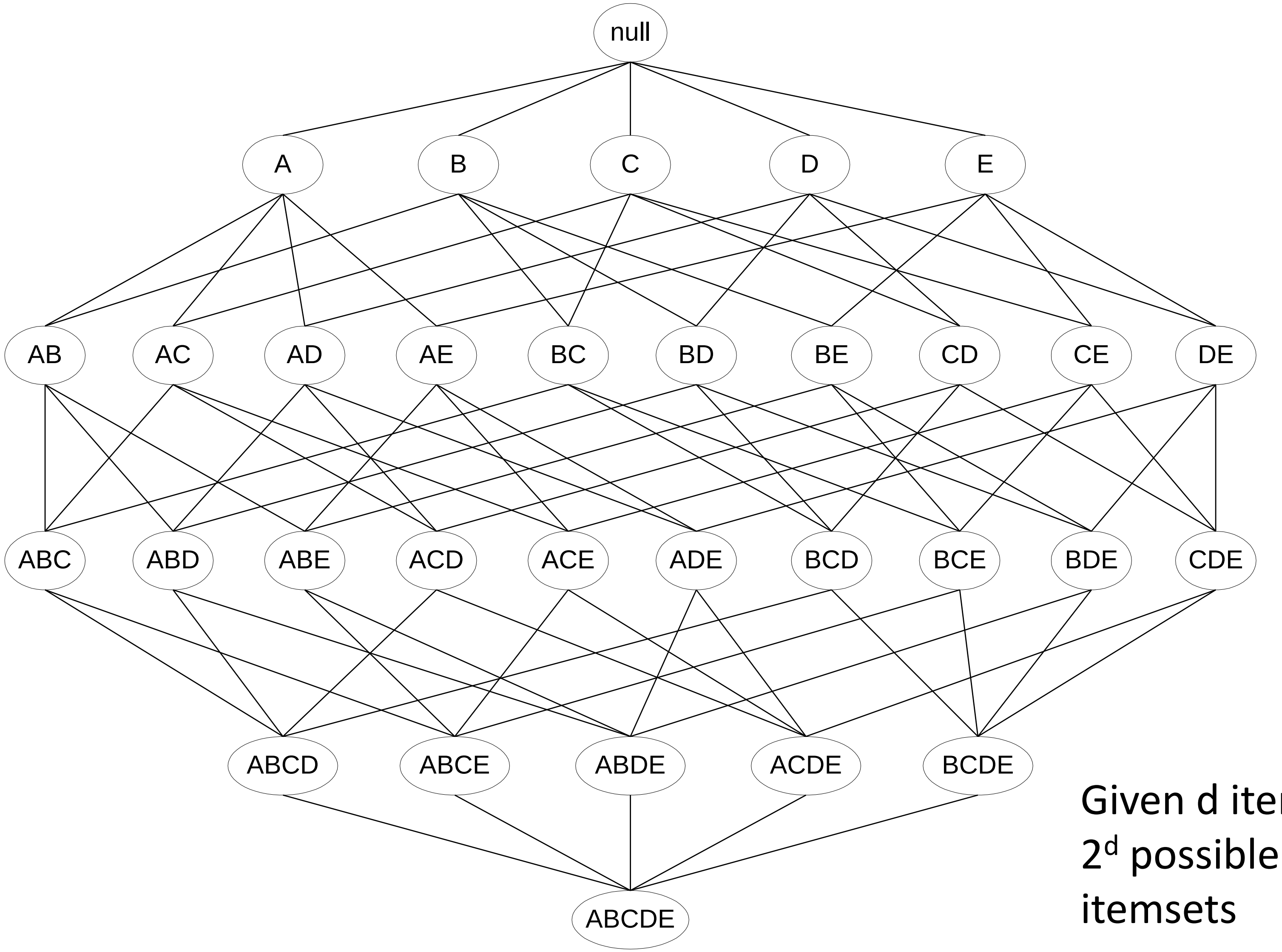
Observations:

- 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

- Two-step approach:
  1. Frequent Itemset Generation
     – Generate all itemsets whose support $\geq$ minsup
  2. 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



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

Frequent Itemset Generation

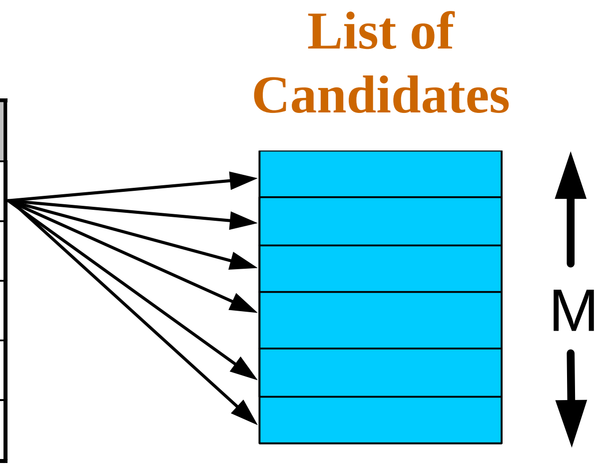
- Brute-force approach:
  - Each itemset in the lattice is a candidate frequent itemset
  - Count the support of each candidate by scanning the database

Transactions

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

N

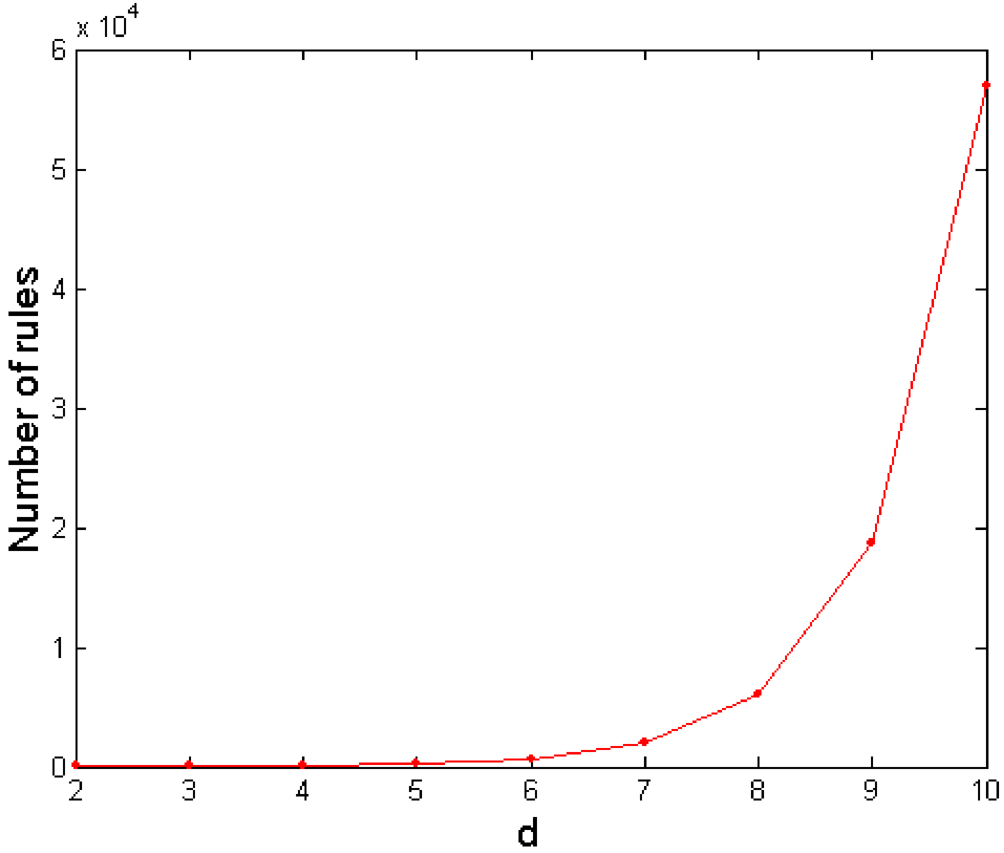
w



  - 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

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
- Reduce the number of transactions (N)
  - Reduce size of N as the size of itemset increases
  - Used by DHP and vertical-based mining algorithms
- Reduce the number of comparisons (NM)
  - Use efficient data structures to store the candidates or transactions
  - No need to match every candidate against every transaction

Reducing Number of Candidates

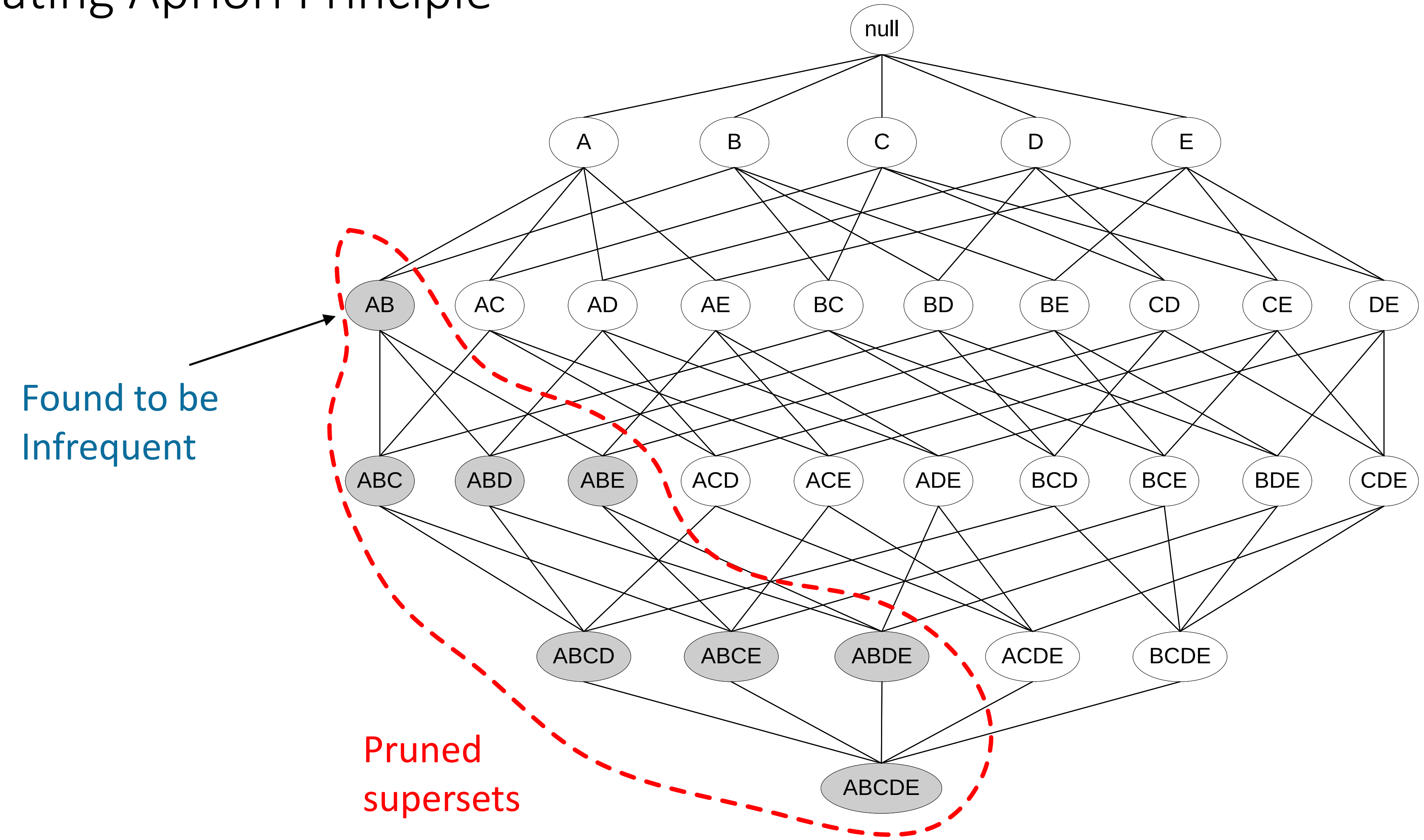
- Apriori principle:
  - If an itemset is frequent, then all of its subsets must also be frequent
  - In other words, if an itemset is infrequent, all of its supersets must also be infrequent

- 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

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
{Bread,Milk,Diaper}	**3**

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

68% decrease in processed subsets

Apriori Algorithm

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

The Apriori Algorithm (Pseudo-Code)

C_k: Candidate itemset of size k

L_k : frequent itemset of size k

L_1 = {frequent items};

for (k = 1; L_k !=∅; k++) **do begin**

  C_{k+1} = candidates generated from L_k;

  for each transaction t in database do

    increment the count of all candidates in C_{k+1} that are contained in t

  L_{k+1} = candidates in C_{k+1} with min_support

  end

return $\cup_k L_k$;

The 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

The Apriori Algorithm (Pseudo-Code)

C_k: Candidate itemset of size k

L_k : frequent itemset of size k

L_1 = {frequent items};

for (k = 1; L_k !=$\emptyset$; k++) **do begin**

$\boxed{C_{k+1} \text{ = candidates generated from } L_k;}$

for each transaction t in database do

increment the count of all candidates in C_{k+1} that are contained in t

L_{k+1} = candidates in C_{k+1} with min_support

end

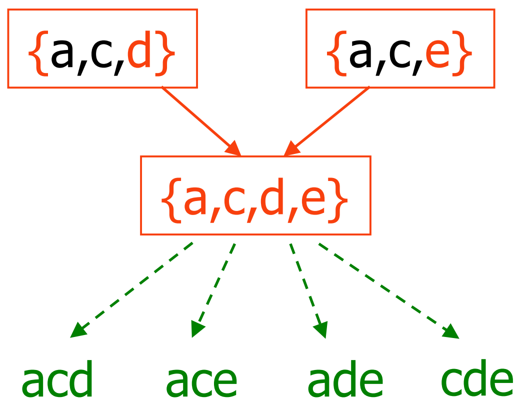
return $\cup_k L_k$;

Implementation of Apriori

- How to generate candidates?
  - Step 1: self-joining L_k
  - Step 2: pruning

Example of Candidates Generation

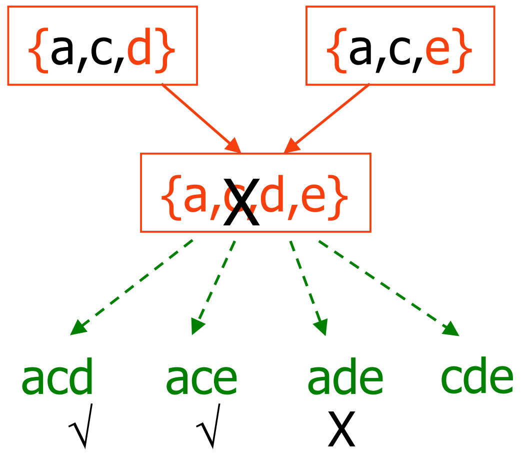
- *Assume the items in L_k are listed in an order (e.g., alphabetical)*
- L_3={*abc, abd, acd, ace, bcd*}
- *Self-joining*: $L_3 * L_3$
  - *abcd* from *abc* and *abd*
  - *acde* from *acd* and *ace*



Slide from Evimaria Terzi

Example of Candidates Generation

- L_3={*abc, abd, acd, ace, bcd*}
- *Self-joining*: $L_3 * L_3$
  - *abcd* from *abc* and *abd*
  - *acde* from *acd* and *ace*
- *Pruning:*
  - *acde* is removed because *ade* is not in L_3
- C_4={*abcd*}



Brute-force method for generating candidates

Items

Item
Beer
Bread
Cola
Diapers
Milk
Eggs

→

Candidate Generation

Itemset
{Beer, Bread, Cola}
{Beer, Bread, Diapers}
{Beer, Bread, Milk}
{Beer, Bread, Eggs}
{Beer, Cola, Diapers}
{Beer, Cola, Milk}
{Beer, Cola, Eggs}
{Beer, Diapers, Milk}
{Beer, Diapers, Eggs}
{Beer, Milk, Eggs}
{Bread, Cola, Diapers}
{Bread, Cola, Milk}
{Bread, Cola, Eggs}
{Bread, Diapers, Milk}
{Bread, Diapers, Eggs}
{Bread, Milk, Eggs}
{Cola, Diapers, Milk}
{Cola, Diapers, Eggs}
{Cola, Milk, Eggs}
{Diapers, Milk, Eggs}

→

Candidate Pruning

Itemset
{Bread, Diapers, Milk}

Figure 6.6. A brute-force method for generating candidate 3-itemsets.

F(k-1)xF(1)

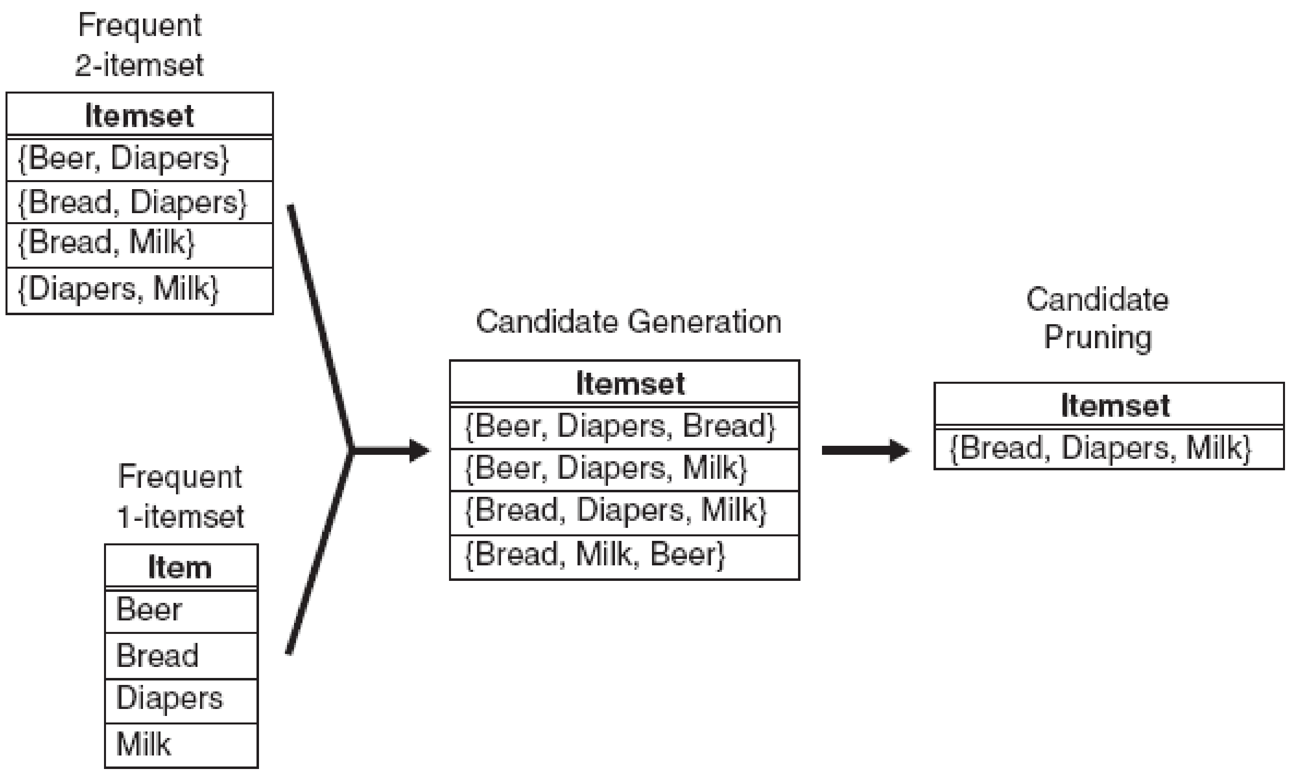


Figure 6.7. Generating and pruning candidate k-itemsets by merging a frequent $(k-1)$-itemset with a frequent item. Note that some of the candidates are unnecessary because their subsets are infrequent.

F(k-1)xF(k-1)

Frequent 2-itemset

Itemset
{Beer, Diapers}
{Bread, Diapers}
{Bread, Milk}
{Diapers, Milk}

Frequent 2-itemset

Itemset
{Beer, Diapers}
{Bread, Diapers}
{Bread, Milk}
{Diapers, Milk}

Candidate Generation

Itemset
{Bread, Diapers, Milk}

Candidate Pruning

Itemset
{Bread, Diapers, Milk}

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

Further Improvement of the Apriori Method

- Major computational challenges
  - Multiple scans of transaction database
  - Huge number of candidates
  - Tedious workload of support counting for candidates
- Improving Apriori: general ideas
  - Reduce passes of transaction database scans
  - Shrink number of candidates
  - Facilitate support counting of candidates

Reducing Number of Comparisons

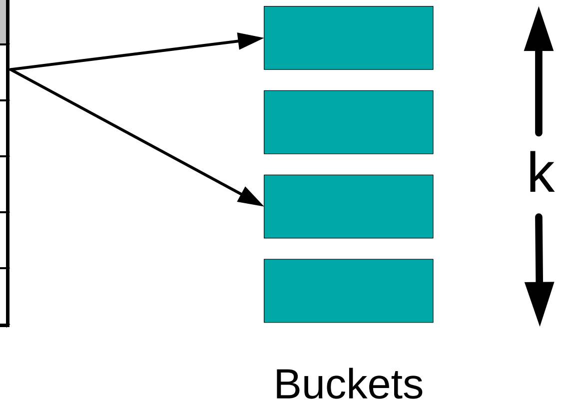
- Candidate counting:
  - Scan the database of transactions to determine the support of each candidate itemset
  - To reduce the number of comparisons, store the candidates in a hash structure
    - Instead of matching each transaction against every candidate, match it against candidates contained in 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

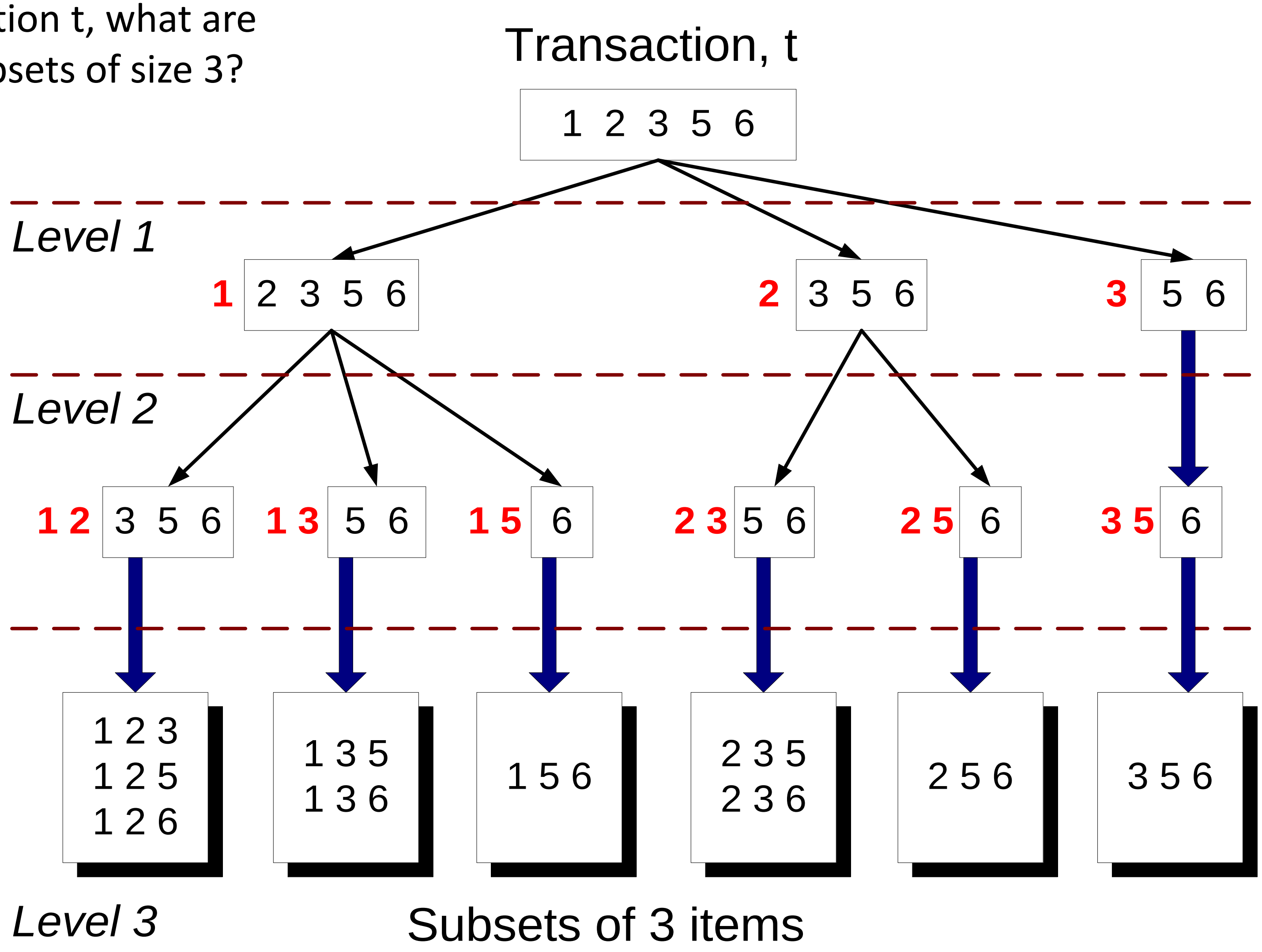


How to Count Supports of Candidates?

- Why counting supports of candidates a problem?
  - The total number of candidates can be very huge
  - One transaction may contain many candidates
- 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

Subset Operation – Support Counting

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



Generate Hash Tree

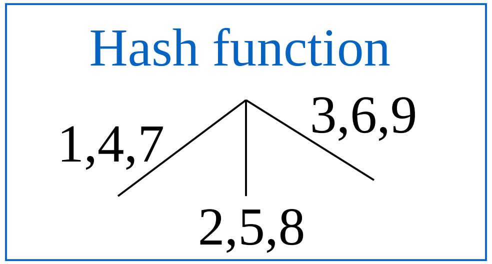
Suppose you have 15 candidate itemsets of length 3:

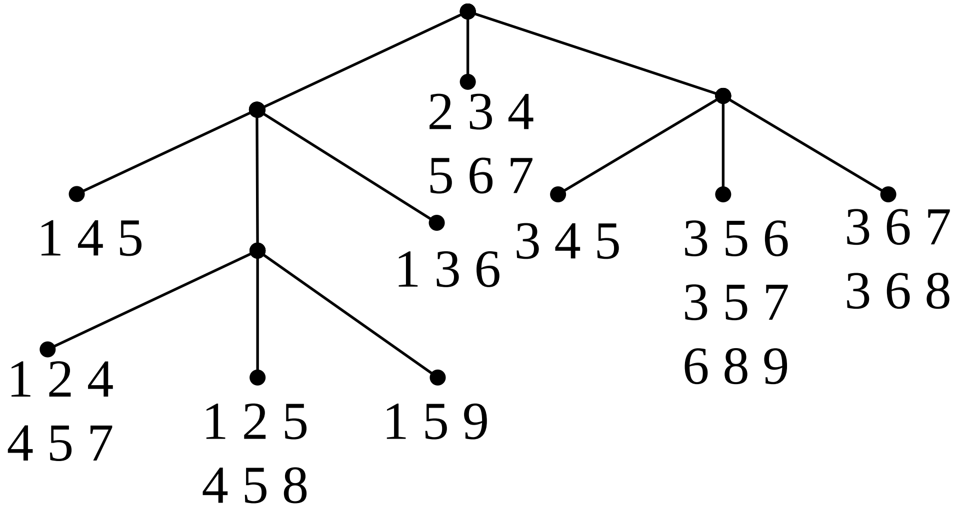
{1 4 5}, {1 2 4}, {4 5 7}, {1 2 5}, {4 5 8}, {1 5 9}, {1 3 6}, {2 3 4}, {5 6 7}, {3 4 5}, {3 5 6}, {3 5 7}, {6 8 9}, {3 6 7}, {3 6 8}

You need:

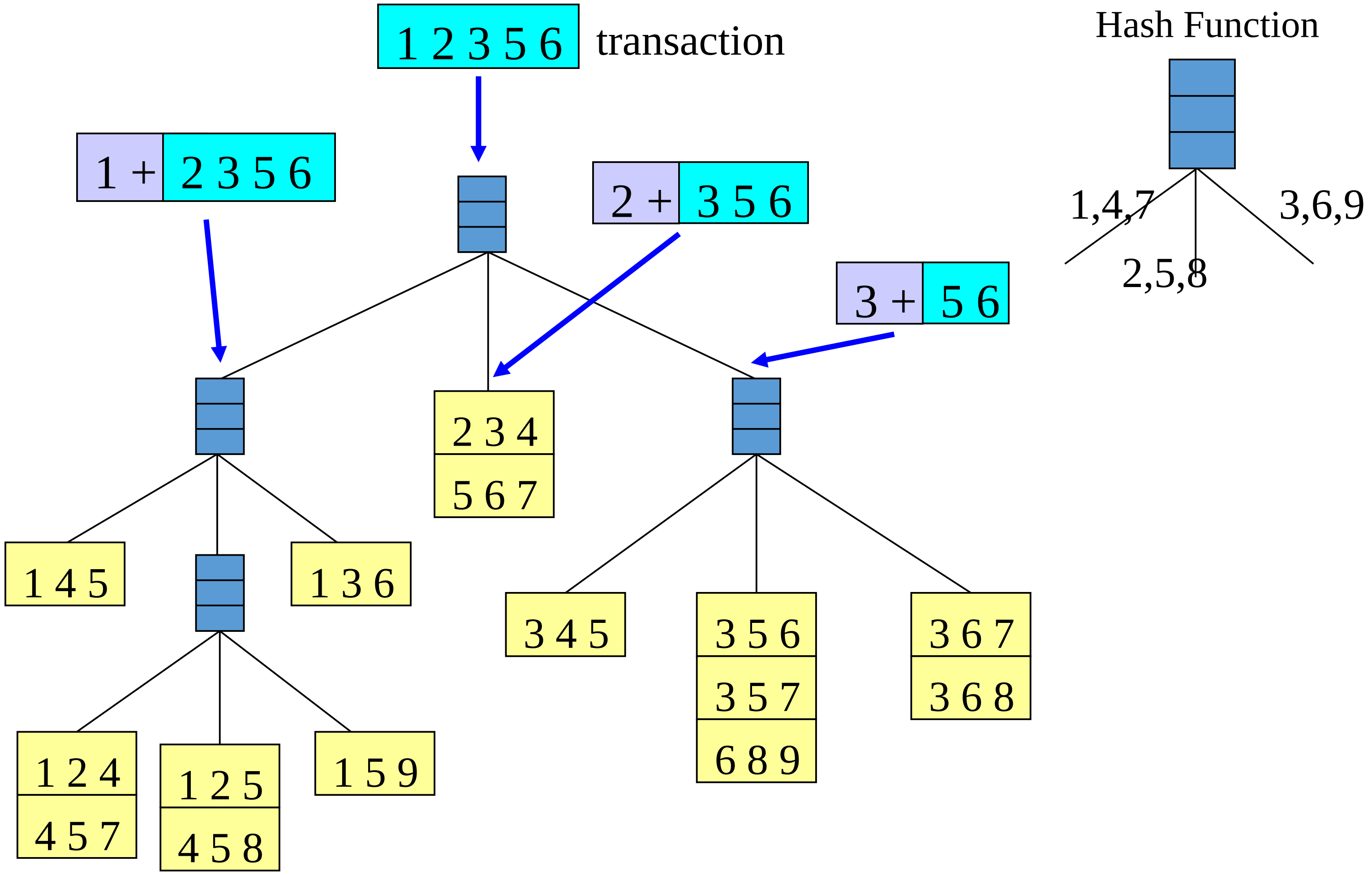
• Hash function

• Max leaf size: max number of itemsets stored in a leaf node (if number of candidate itemsets exceeds max leaf size, split the node)

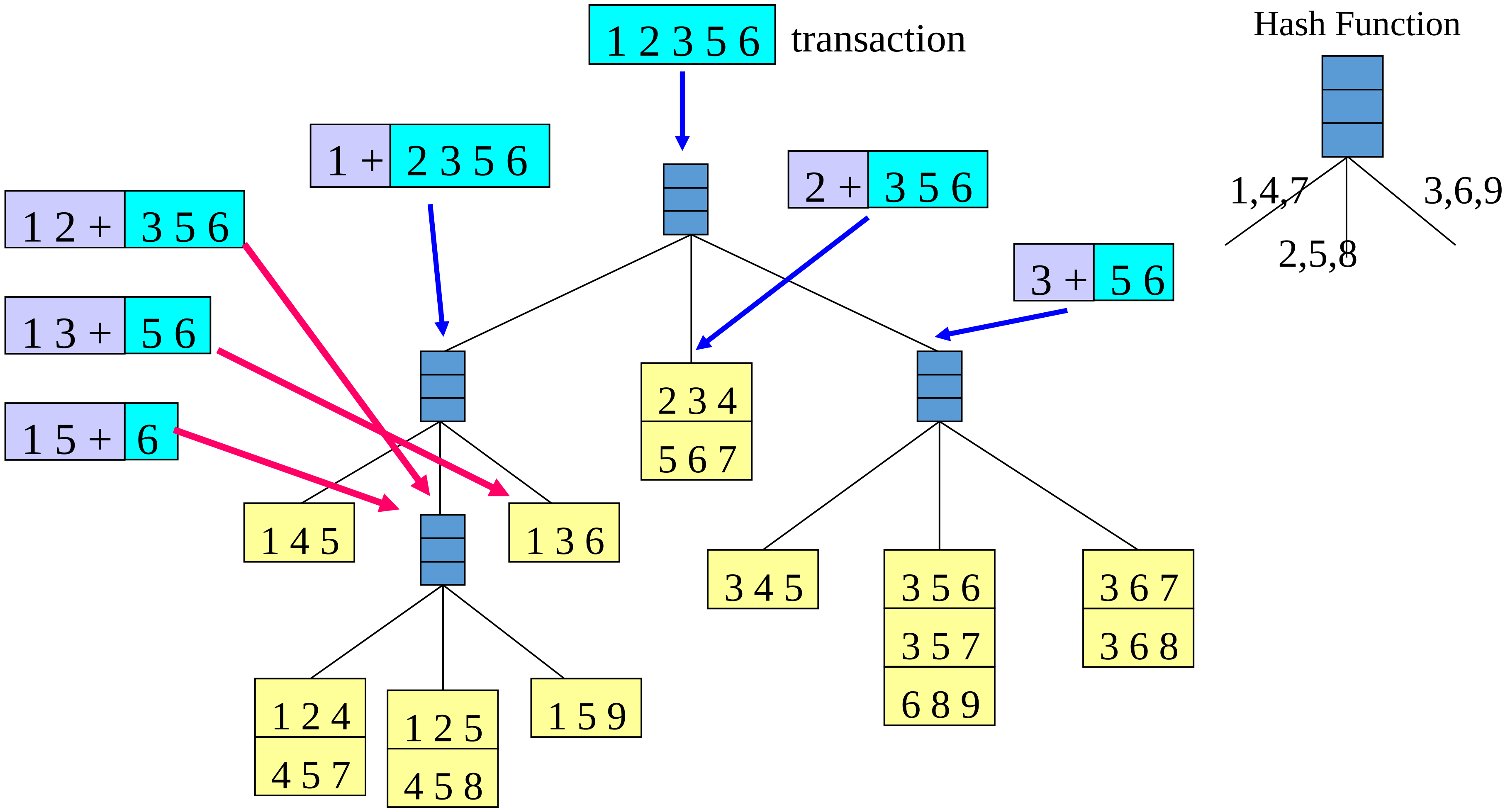




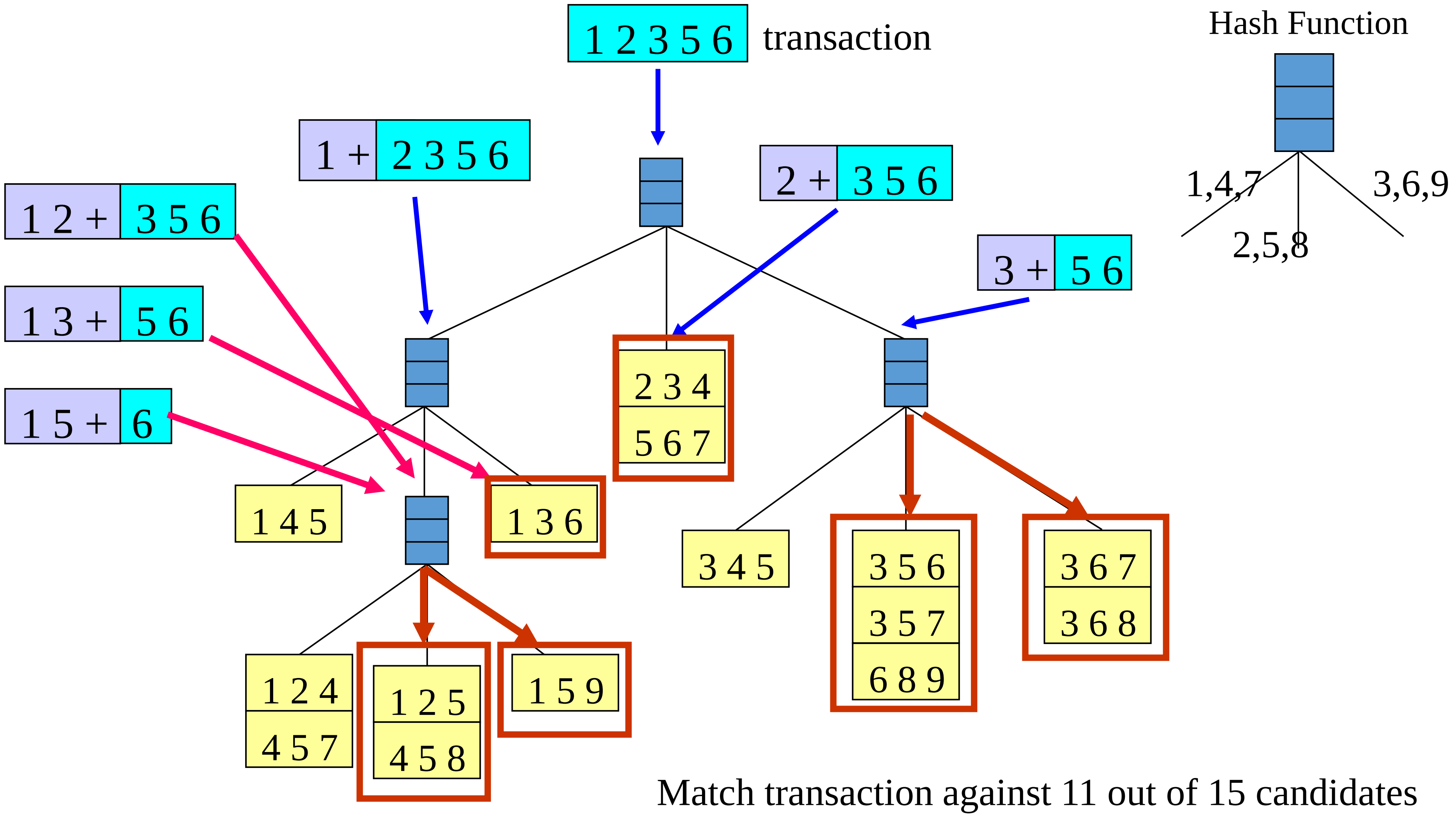
Subset Operation Using Hash Tree



Subset Operation Using Hash Tree



Subset Operation Using Hash Tree



Match transaction against 11 out of 15 candidates

Factors Affecting Complexity

- 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)

Compact Representation of Frequent Itemsets

- Some itemsets are redundant because they have identical support as their supersets

$$= 3 \times \sum_{k=1}^{10} \binom{10}{k}$$

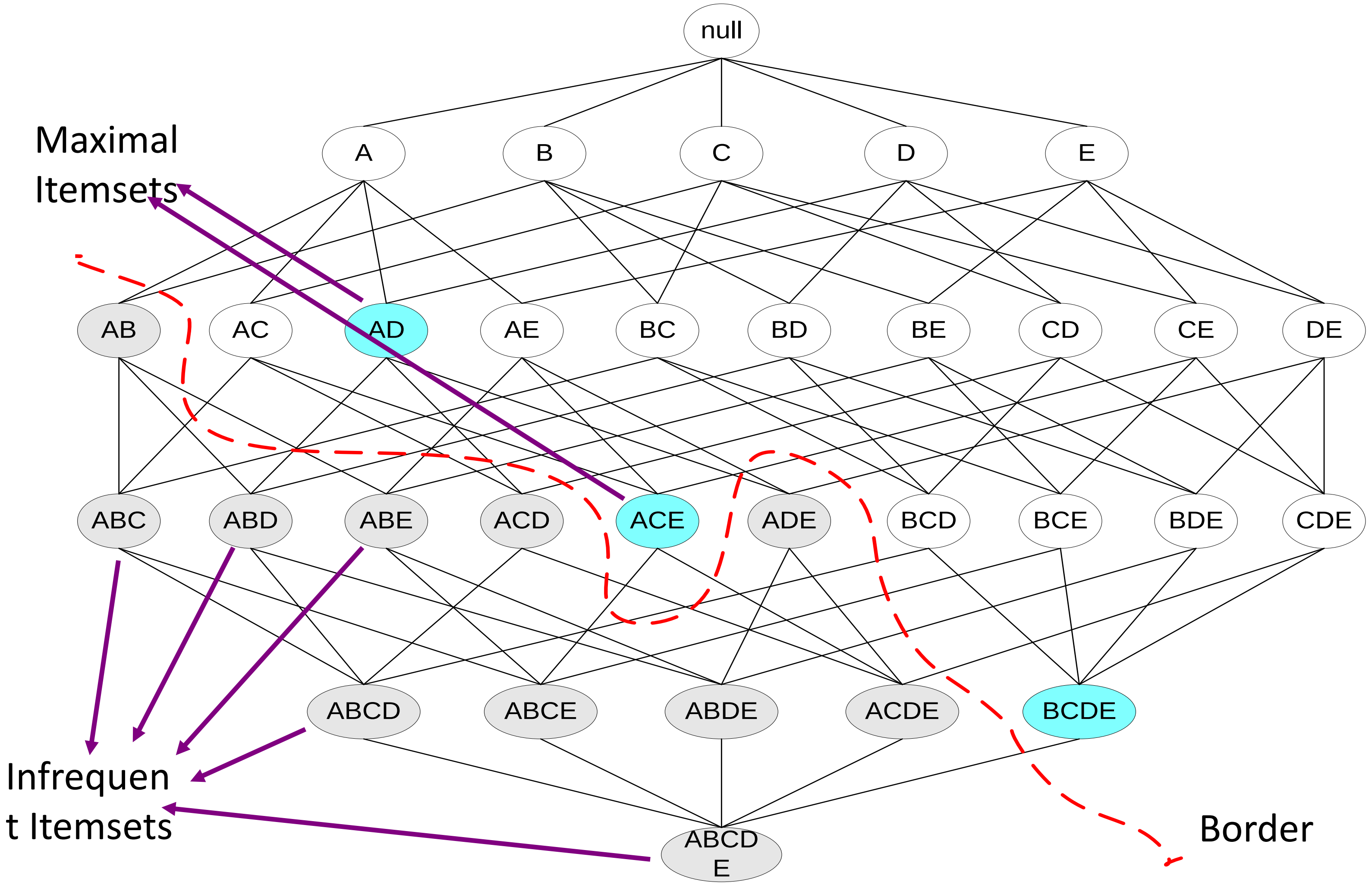
- Number of frequent itemsets

- 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 Itemset

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



Maximal Frequent Itemsets

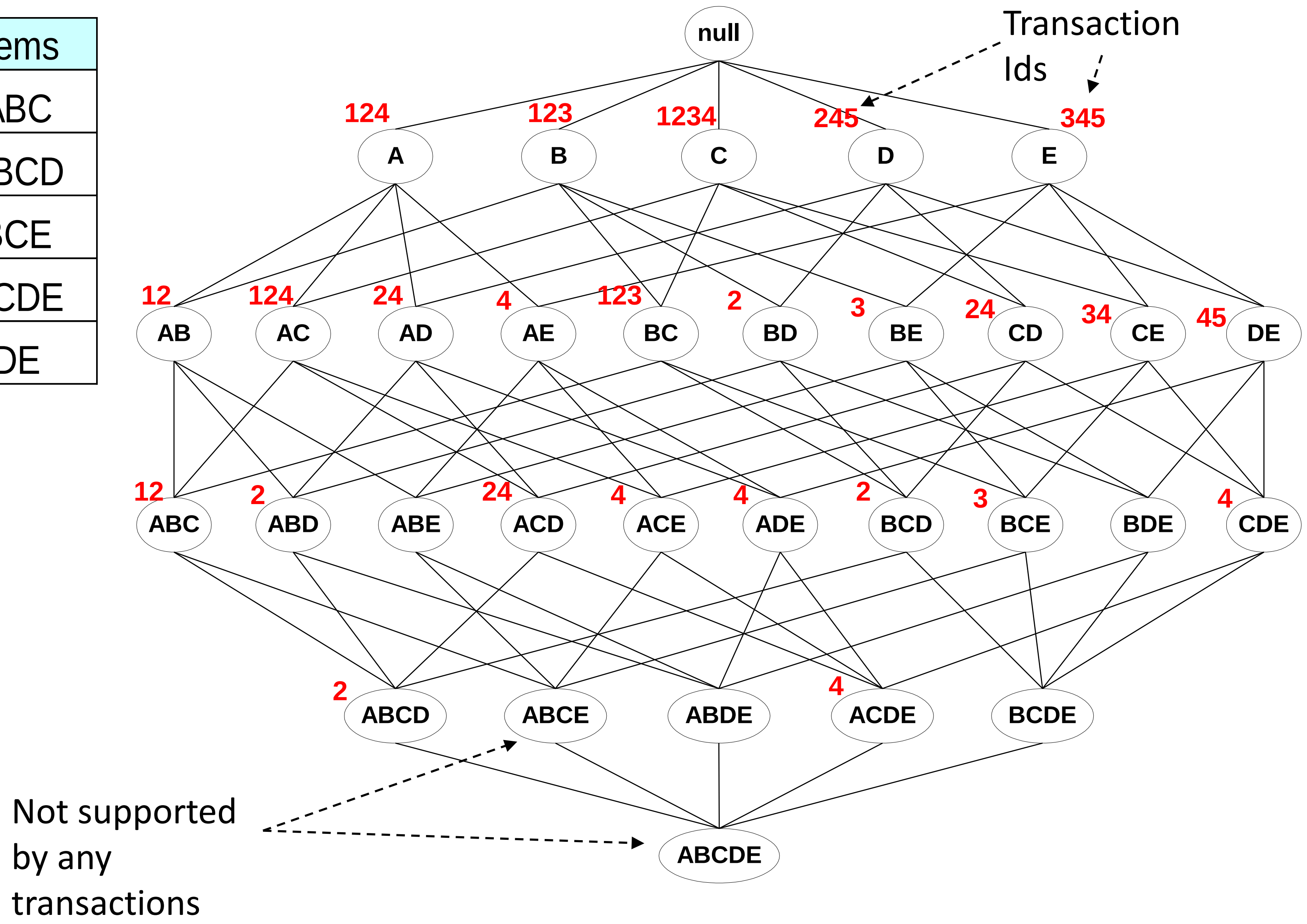
- They form the smallest set of itemsets from which all frequent itemsets can be derived
- Practical if an efficient algorithm exists to explicitly find the maximal frequent itemsets without having to enumerate all their subsets
- They don’t include the support information

Closed Itemset

- Provide a minimal representation without losing their support information
- An itemset is closed if none of its immediate supersets has the same support as the itemset

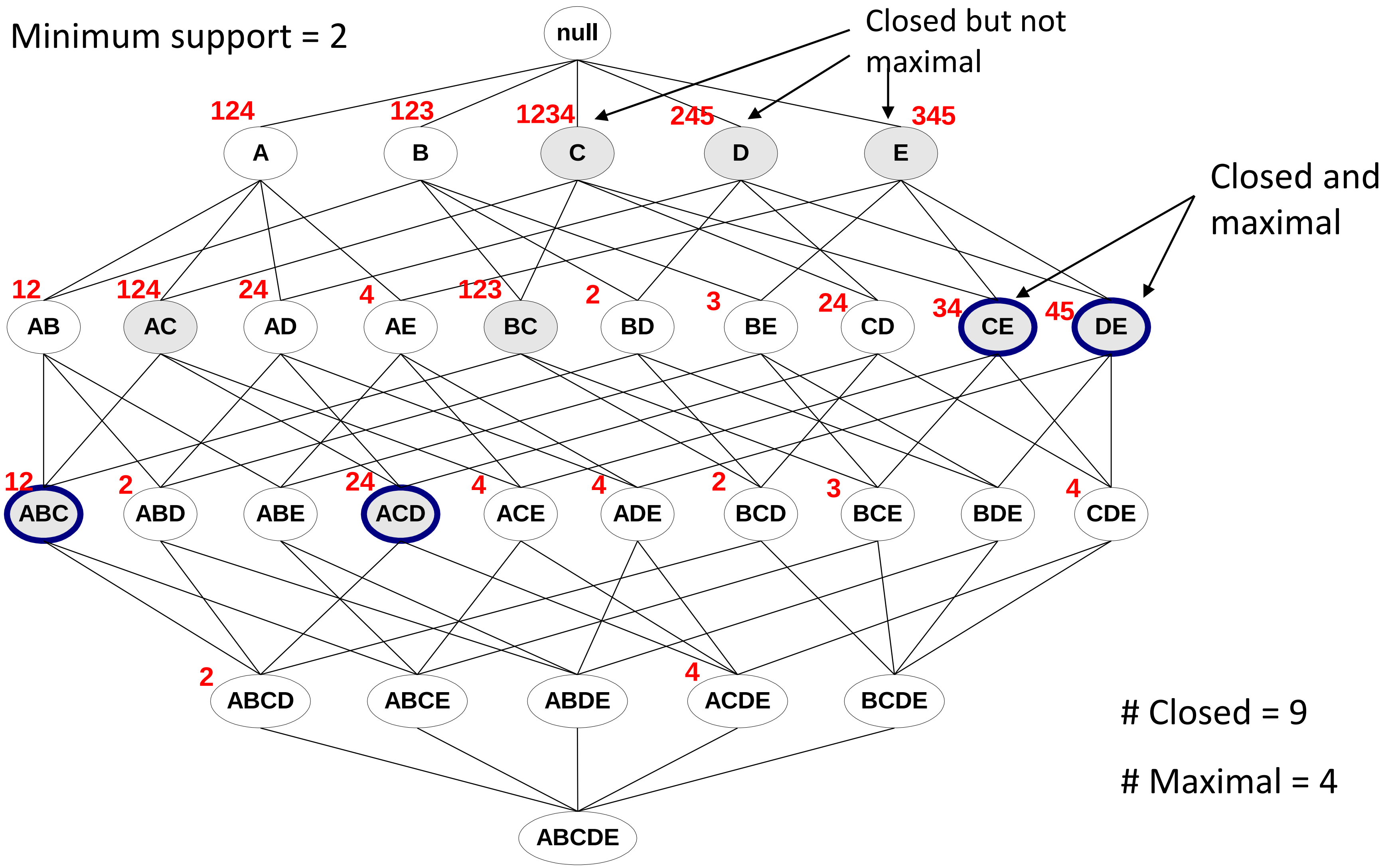
Maximal vs Closed Itemsets

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



Maximal vs Closed Frequent Itemsets

Minimum support = 2



Closed = 9

Maximal = 4

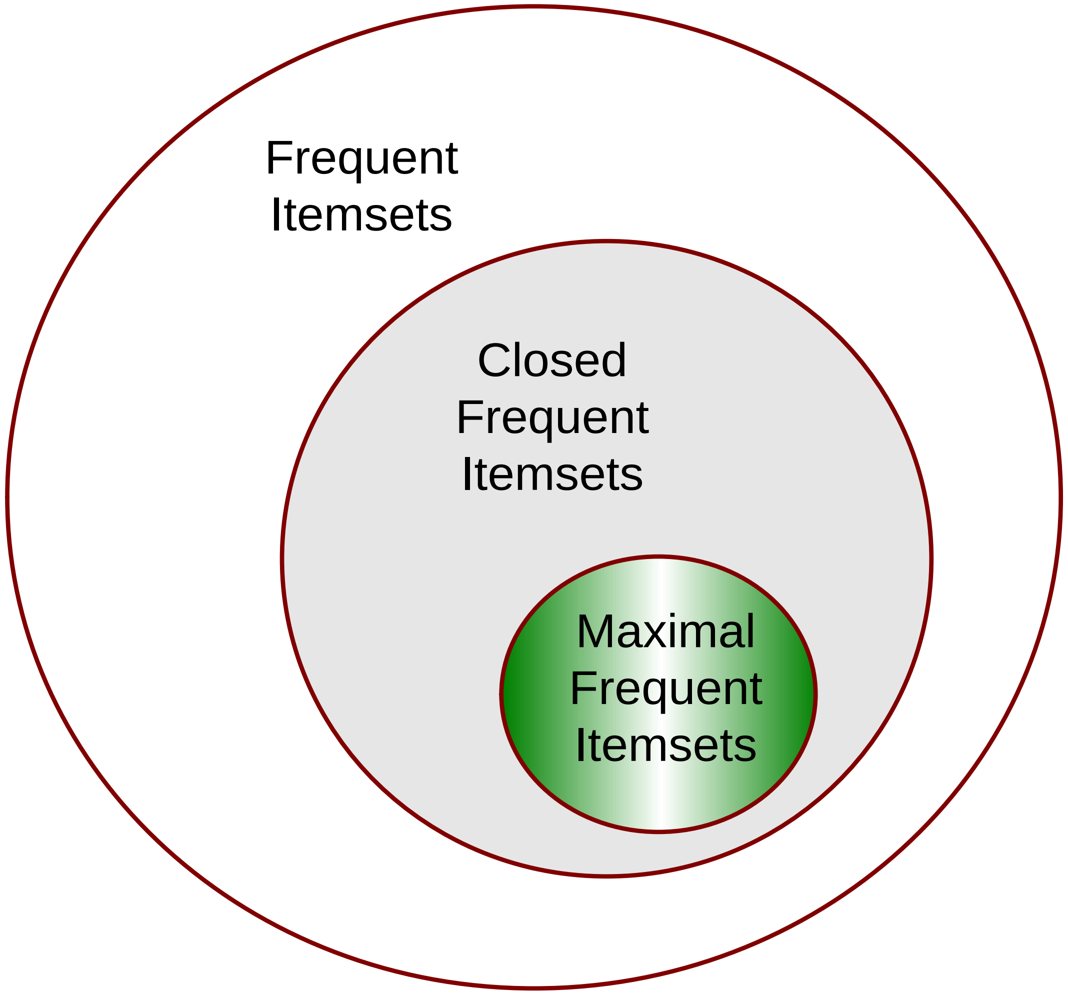
Why are closed patterns interesting?

- Closed patterns and their frequencies alone are sufficient representation for all the frequencies of all frequent patterns

- **Proof:** Assume a frequent itemset **X**:
  - **X** is closed → **s(X)** is known
  - **X** is not closed →
    s(X) = max {s(Y) | Y is closed and X subset of Y}

Slide from EviMaria Terzi

Maximal vs Closed Itemsets



Alternative Algorithm – FP growth

FP-Growth: Frequent Pattern-Growth

- FP-tree is a compressed representation of the input data
- Adopts a divide and conquer strategy
- Compress the database representing frequent items into a frequent –pattern tree or FP-tree
  - Retains the itemset association information
- If FP-tree is small enough to fit the memory, this will allow to extract frequent itemsets directly in memory

Example: FP-Growth

- The first scan of data is the same as Apriori
- Derive the set of frequent 1-itemsets
- Let min-sup=2
- Generate a set of ordered items

Item ID	Support count
I2	7
I1	6
I3	6
I4	2
I5	2

Transactional Database

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

Construct the FP-Tree

Transactional Database

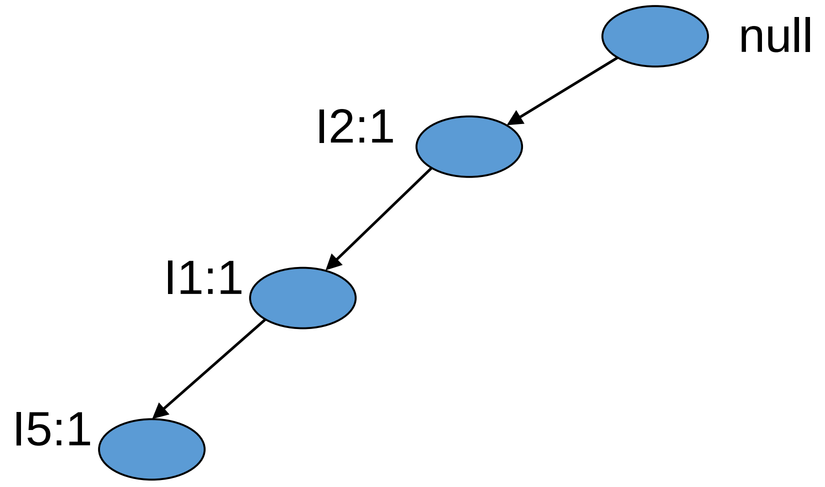
TID	Items	TID	Items	TID	Items
T100	I1,I2,I5	T400	I1,I2,I4	T700	I1,I3
T200	I2,I4	T500	I1,I3	T800	I1,I2,I3,I5
T300	I2,I3	T600	I2,I3	T900	I1,I2,I3

- Create a branch for each transaction
- Items in each transaction are processed in order

1- Order the items T100: {I2,I1,I5}

2- Construct the first branch: <I2:1>, <I1:1>,<I5:1>

Item ID	Support count
I2	7
I1	6
I3	6
I4	2
I5	2



Construct the FP-Tree

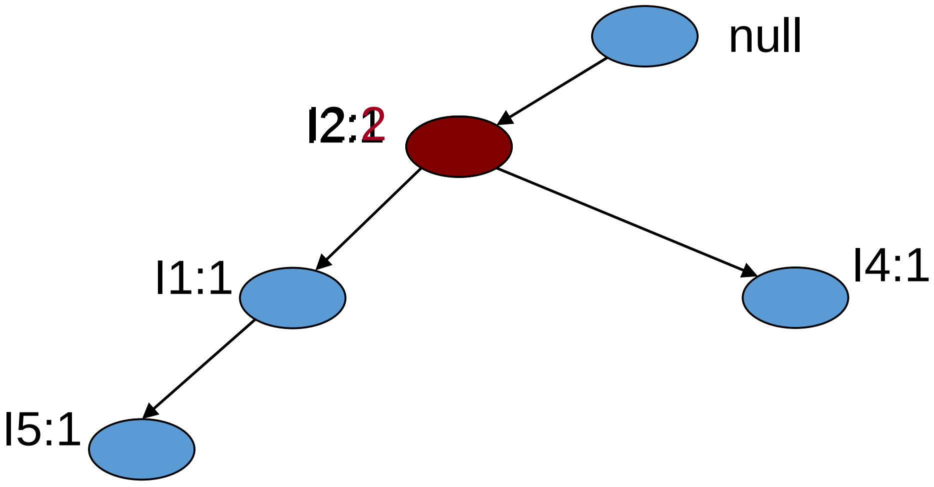
Transactional Database

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

- Create a branch for each transaction
- Items in each transaction are processed in order

1- Order the items T200: {I2,I4}
2- Construct the second branch: <I2:1>, <I4:1>

Item ID	Support count
I2	7
I1	6
I3	6
I4	2
I5	2



Construct the FP-Tree

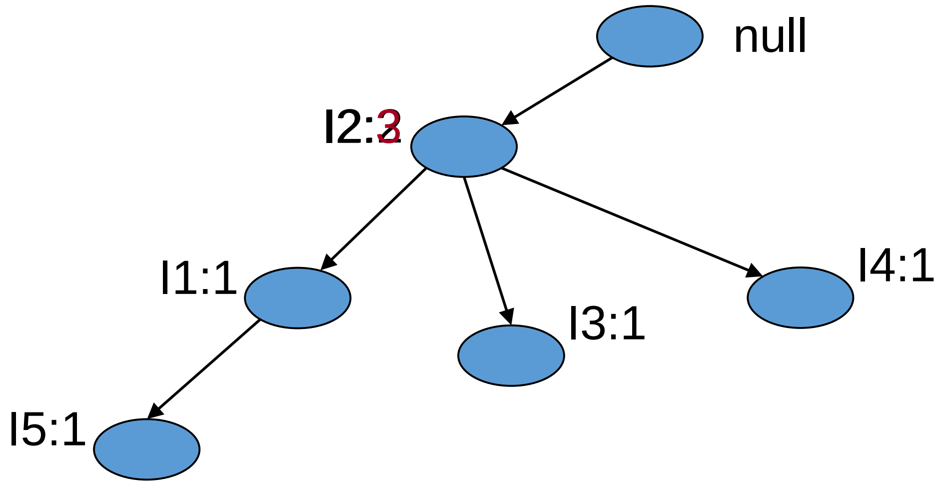
Transactional Database

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

- Create a branch for each transaction
- Items in each transaction are processed in order

1- Order the items T300: {I2,I3}
2- Construct the third branch: <I2:2>, <I3:1>

Item ID	Support count
I2	7
I1	6
I3	6
I4	2
I5	2



Construct the FP-Tree

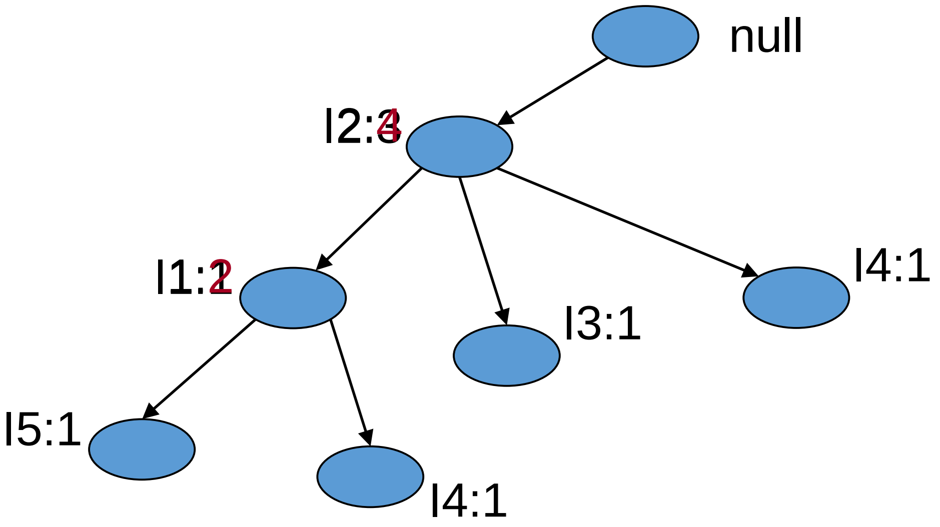
Transactional Database

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

- Create a branch for each transaction
- Items in each transaction are processed in order

1- Order the items T400: {I2,I1,I4}
2- Construct the fourth branch: <I2:3>, <I1:1>,<I4:1>

Item ID	Support count
I2	7
I1	6
I3	6
I4	2
I5	2



Construct the FP-Tree

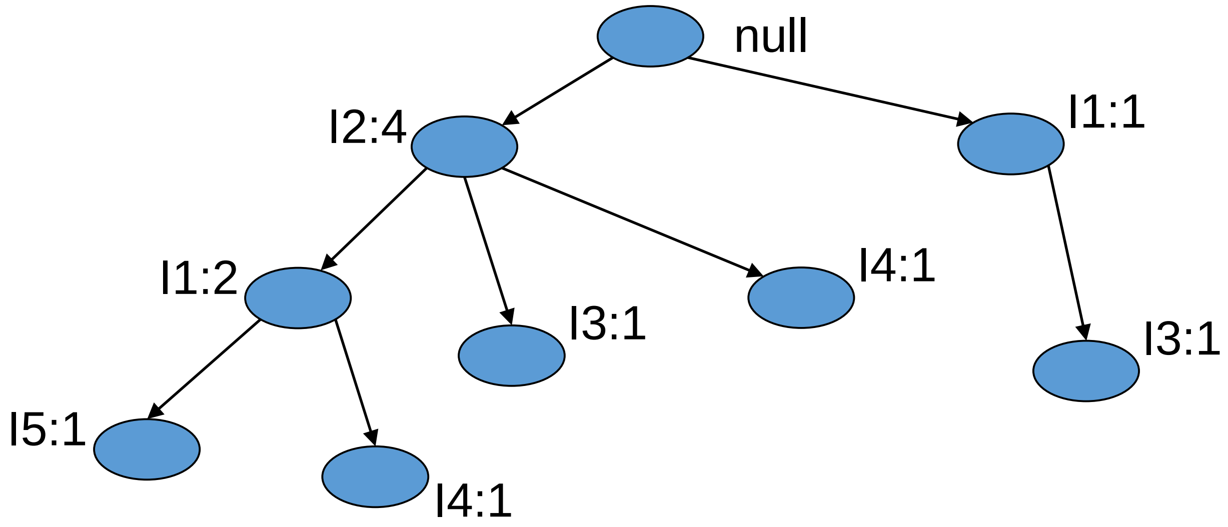
Transactional Database

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

- Create a branch for each transaction
- Items in each transaction are processed in order

1- Order the items T400: {I1,I3}
2- Construct the fifth branch: <I1:1>, <I3:1>

Item ID	Support count
I2	7
I1	6
I3	6
I4	2
I5	2

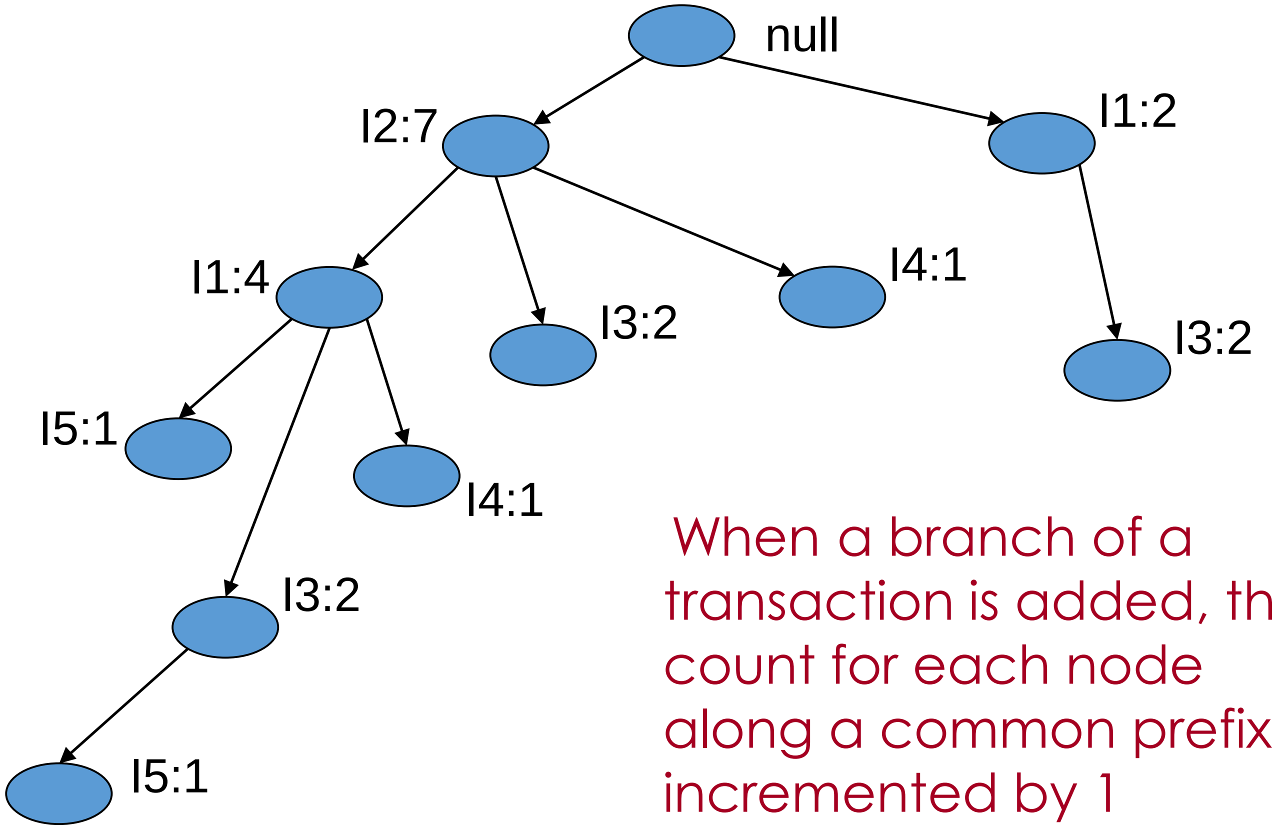


Construct the FP-Tree

Transactional Database

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

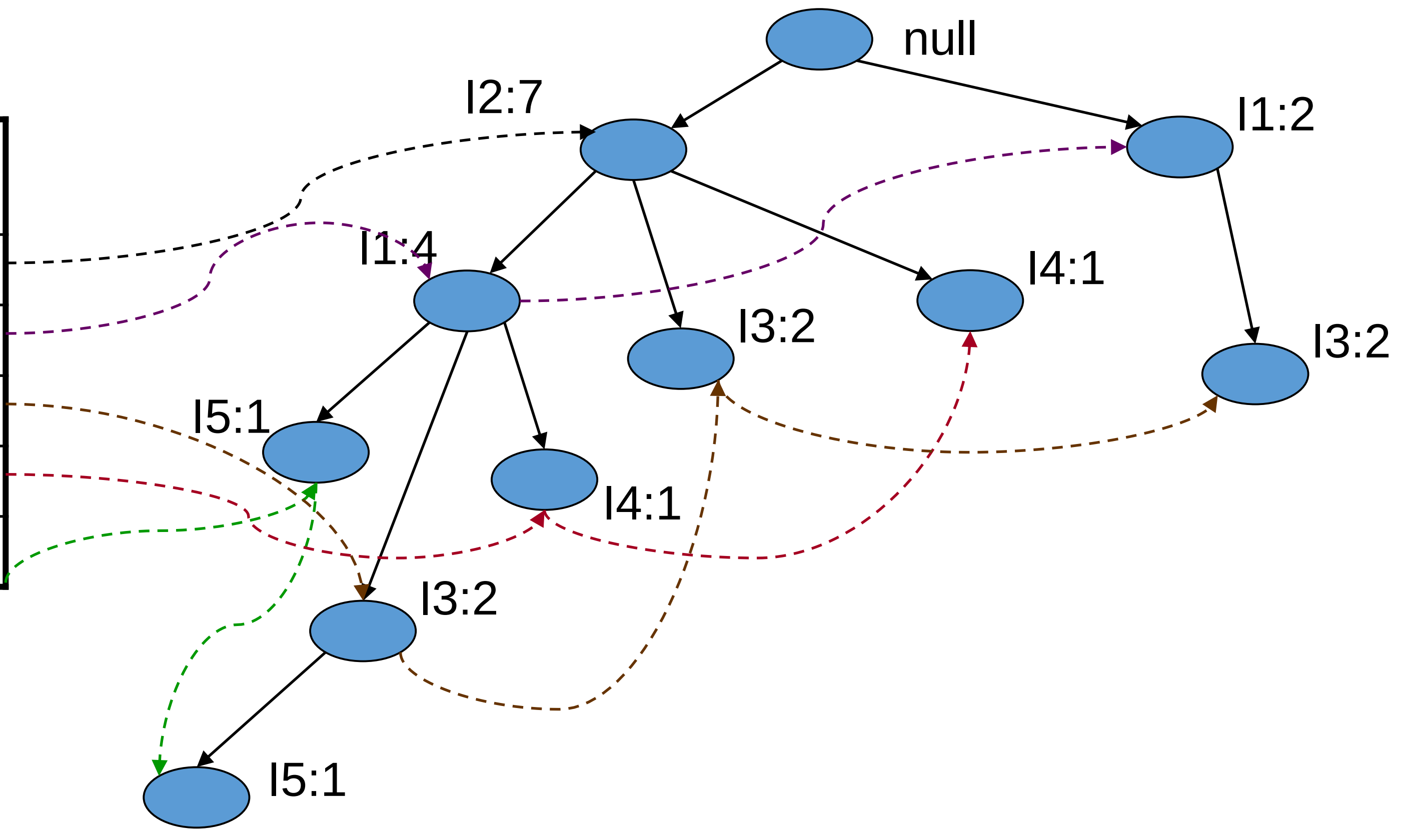
Item ID	Support count
I2	7
I1	6
I3	6
I4	2
I5	2



When a branch of a transaction is added, the count for each node along a common prefix is incremented by 1

Construct the FP-Tree

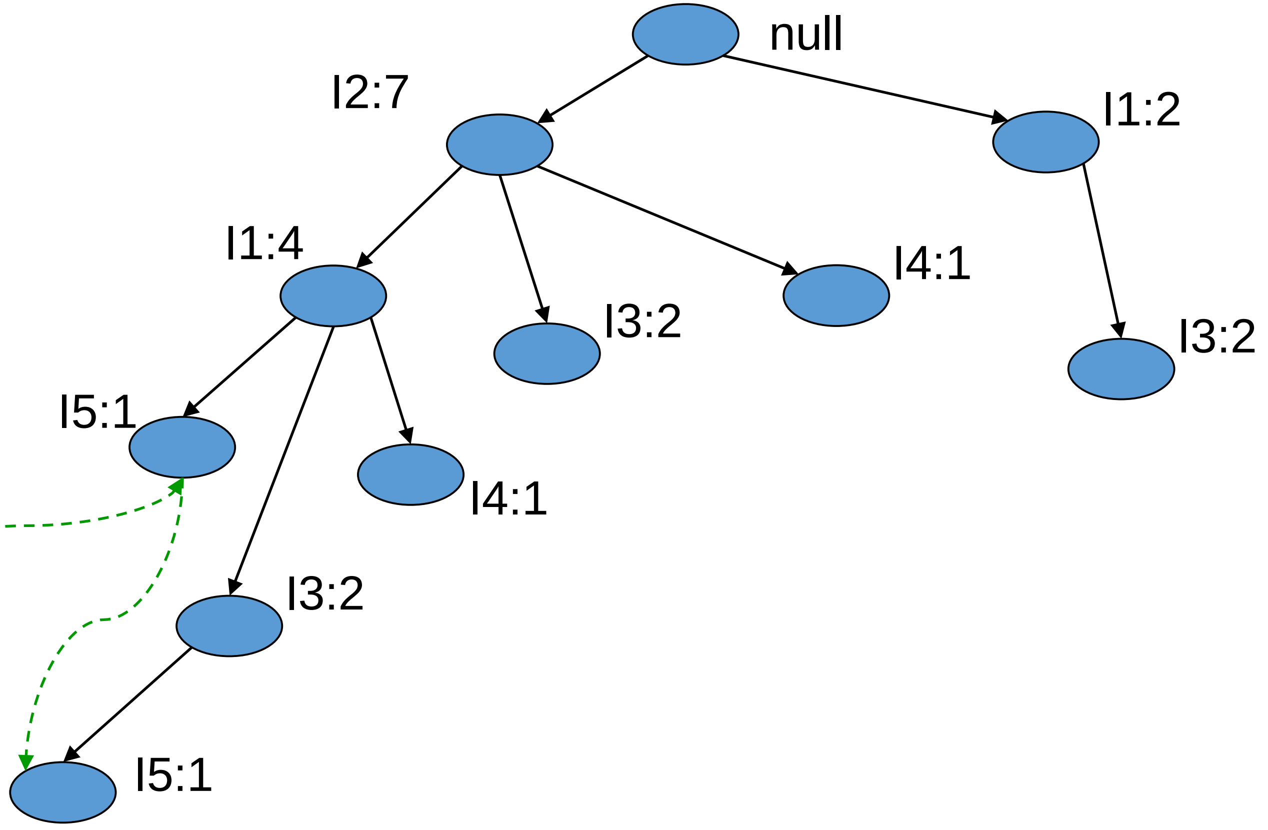
Item ID	Support count
I2	7
I1	6
I3	6
I4	2
I5	2



The problem of mining frequent patterns in databases is transformed to that of mining the FP-tree

Construct the FP-Tree

Item ID	Support count
I2	7
I1	6
I3	6
I4	2
I5	2



-Occurrences of I5: <I2,I1,I5> and <I2,I1,I3,I5>

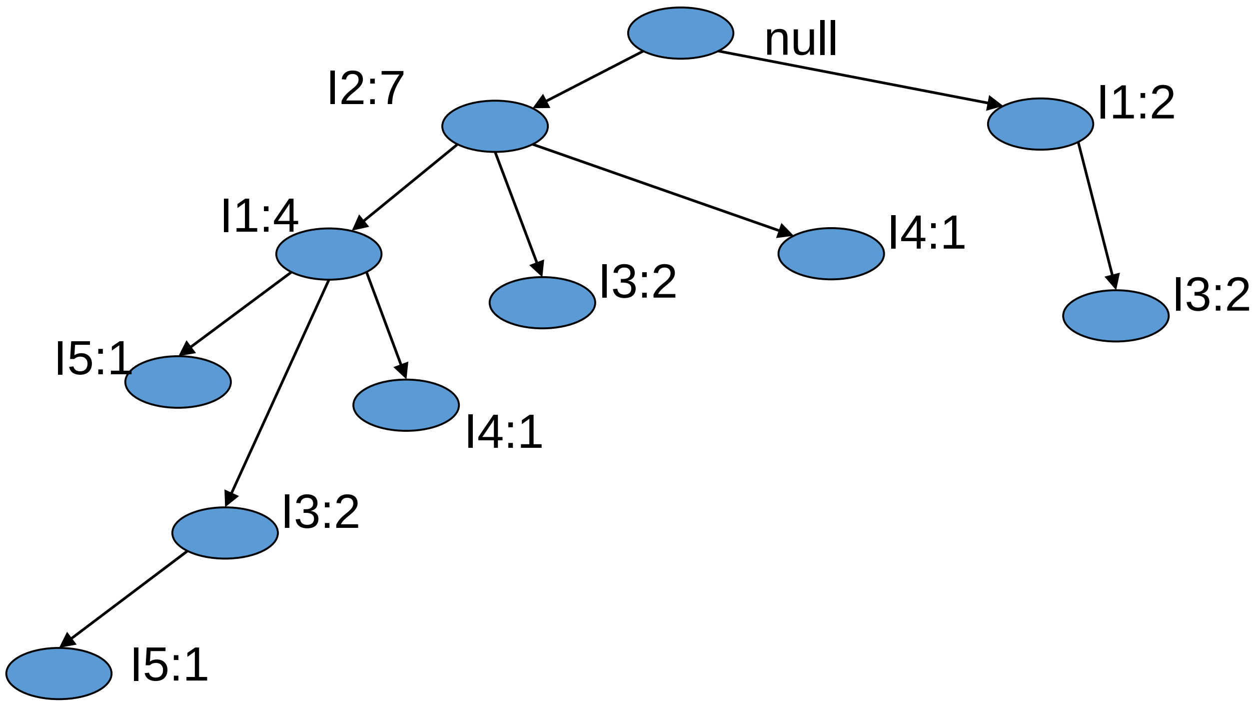
-Two prefix Paths <I2, I1: 1> and <I2,I1,I3: 1>

-Conditional FP tree contains only <I2: 2, I1: 2>, I3 is not considered because its support count of 1 is less than the minimum support count.

-Frequent patterns {I2,I5:2}, {I1,I5:2},{I2,I1,I5:2}

Construct the FP-Tree

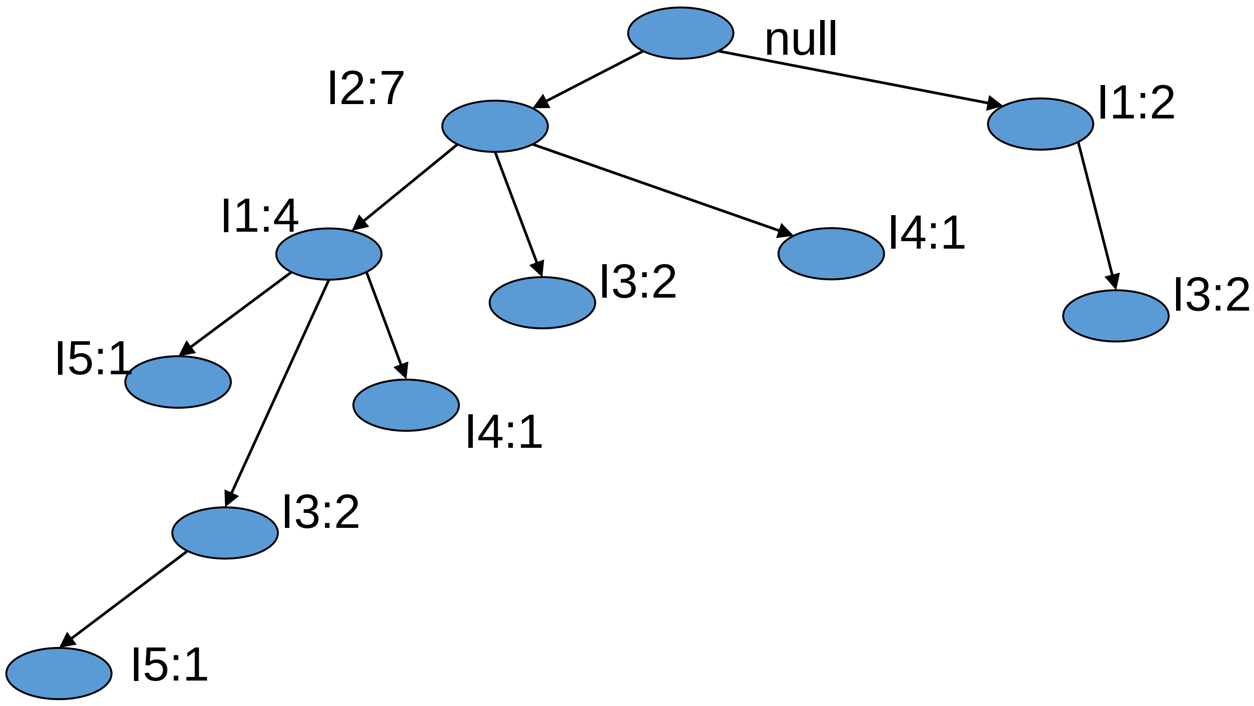
Item ID	Support count
I2	7
I1	6
I3	6
I4	2
I5	2



TID	Conditional Pattern Base	Conditional FP-tree
I5	{{I2,I1:1},{I2,I1,I3:1}}	<I2:2,I1:2>
I4	{{I2,I1:1},{I2,1}}	<I2:2>
I3	{{I2,I1:2},{I2:2}, {I1:2}}	<I2:4,I1:2>,<I1:2>
I1	{I2,4}	<I2:4>

Construct the FP-Tree

Item ID	Support count
I2	7
I1	6
I3	6
I4	2
I5	2



TID	Conditional FP-tree	**Frequent Patterns Generated**
I5	<I2:2,I1:2>	{I2,I5:2}, {I1,I5:2},{I2,I1,I5:2}
I4	<I2:2>	{I2,I4:2}
I3	<I2:4,I1:2>,<I1:2>	{I2,I3:4},{I1,I3:4},{I2,I1,I3:2}
I1	<I2:4>	{I2,I1:4}

FP-growth properties

- FP-growth transforms the problem of finding long frequent patterns to searching for shorter once recursively and the concatenating the suffix
- It uses the least frequent suffix offering a good selectivity
- It reduces the search cost
- If the tree does not fit into main memory, partition the database
- Efficient and scalable for mining both long and short frequent patterns

Mining Association Rules

- Two-step approach:
  1. Frequent Itemset Generation
     – Generate all itemsets whose support $\geq$ minsup

  2. **Rule Generation**
     – Generate high confidence rules from each frequent itemset, where each rule is a binary partitioning of a frequent itemset

Re-Definition: Association Rule

Let D be database of transactions

– e.g.:

Transaction ID	Items
2000	A, B, C
1000	A, C
4000	A, D
5000	B, E, F

- Let I be the set of items that appear in the database, e.g., $I=\{A,B,C,D,E,F\}$
- A rule is defined by $X \rightarrow Y$, where $X \subset I$, $Y \subset I$, and $X \cap Y = \emptyset$
  - e.g.: $\{B,C\} \rightarrow \{A\}$ is a rule

Generating Association Rules

- Once the frequent itemsets have been found, it is straightforward to generate strong association rules that satisfy:
  - → minimum Support
  - → minimum confidence

- Relation between support and confidence:

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

  - → Support_count(A∪B) is the number of transactions containing the itemsets A ∪ B
  - → Support_count(A) is the number of transactions containing the itemset A.

Generating Association Rules

- For each frequent itemset *L*, generate all non empty subsets of *L*
- *For every no empty subset S of L, output the rule:*

$$S \Rightarrow (L\text{-}S)$$

If (support_count(L)/support_count(S)) >= min_conf

Example

- Suppose the frequent Itemset L={I1,I2,I5}
- Subsets of L are: {I1,I2}, {I1,I5},{I2,I5},{I1},{I2},{I5}
- Association rules :

$I1 \wedge I2 \Rightarrow I5$ confidence = 2/4= 50%

$I1 \wedge I5 \Rightarrow I2$ confidence=2/2=100%

$I2 \wedge I5 \Rightarrow I1$ confidence=2/2=100%

$I1 \Rightarrow I2 \wedge I5$ confidence=2/6=33%

$I2 \Rightarrow I1 \wedge I5$ confidence=2/7=29%

$I5 \Rightarrow I2 \wedge I2$ confidence=2/2=100%

If the minimum confidence =70%

Transactional Database

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

Rule Generation

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

$ABC \rightarrow D$,	$ABD \rightarrow C$,	$ACD \rightarrow B$,	$BCD \rightarrow A$,
$A \rightarrow BCD$,	$B \rightarrow ACD$,	$C \rightarrow ABD$,	$D \rightarrow ABC$
$AB \rightarrow CD$,	$AC \rightarrow BD$,	$AD \rightarrow BC$,	$BC \rightarrow AD$,
$BD \rightarrow AC$,	$CD \rightarrow AB$,		

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

Rule Generation

- 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 $c(AB \rightarrow D)$

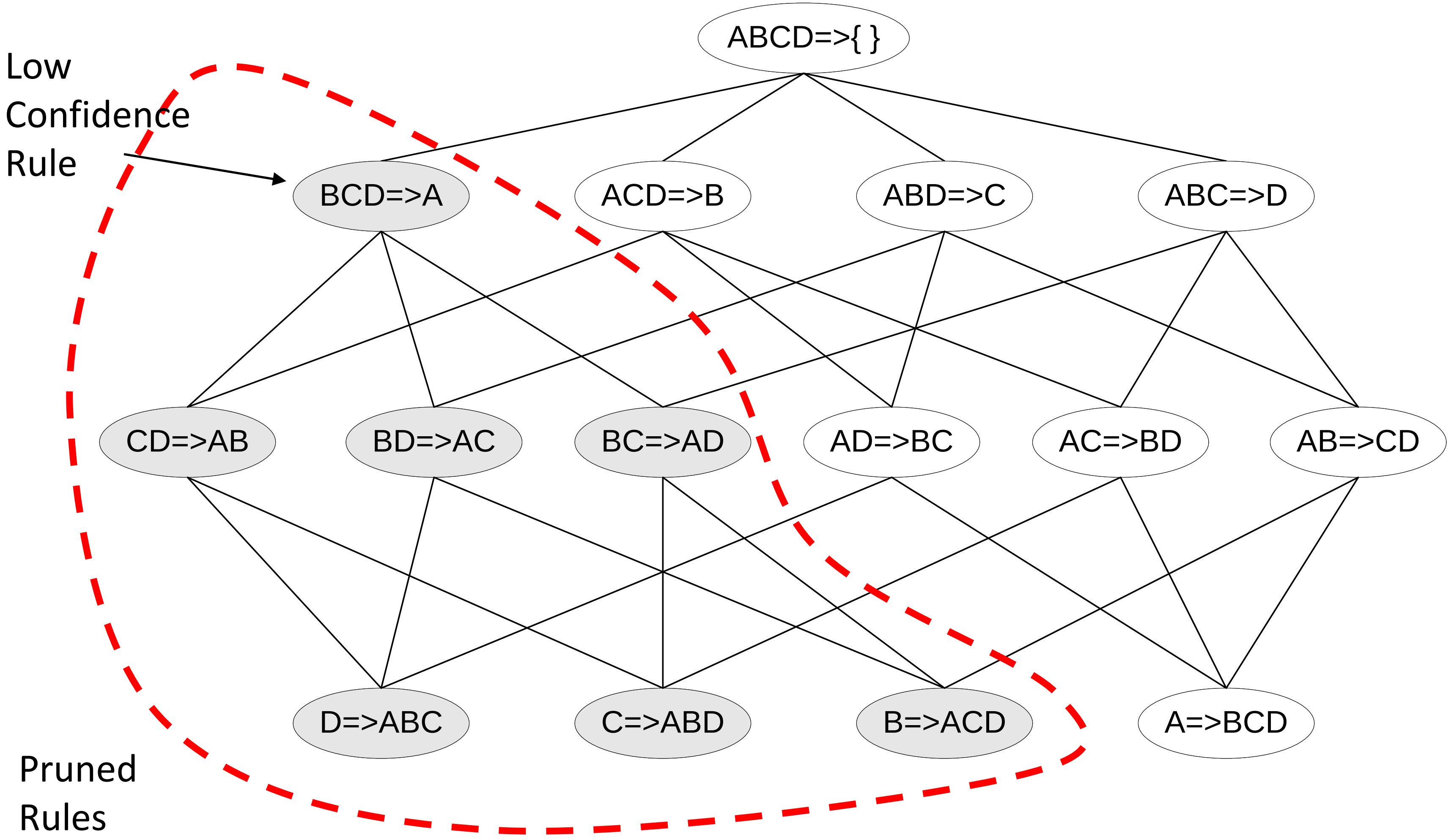
  - But confidence of rules generated from the same itemset has an anti-monotone property
  - e.g., L = {A,B,C,D}:

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

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

Rule Generation for Apriori Algorithm

Lattice of rules

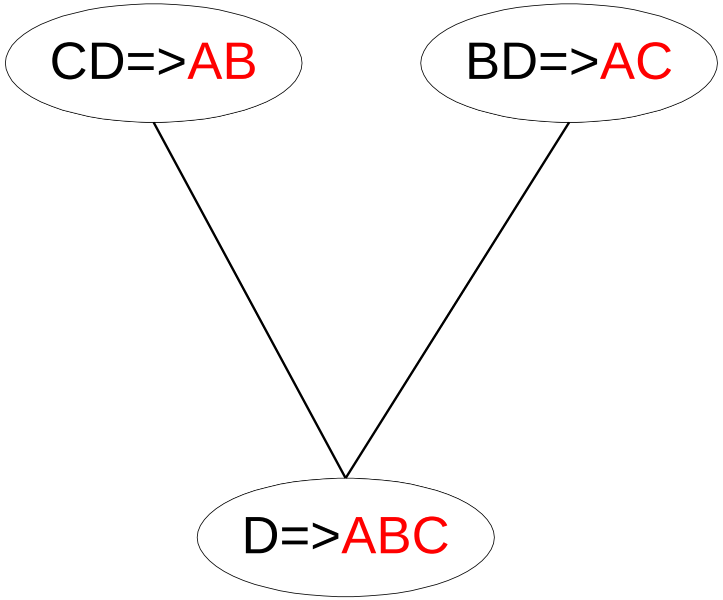


Rule Generation for Apriori Algorithm

- Candidate rule is generated by merging two rules that share the same prefix
  in the rule consequent

- join(CD=>AB,BD=>AC)
  would produce the candidate
  rule D => ABC

- Prune rule D=>ABC if its
  subset AD=>BC does not have
  high confidence



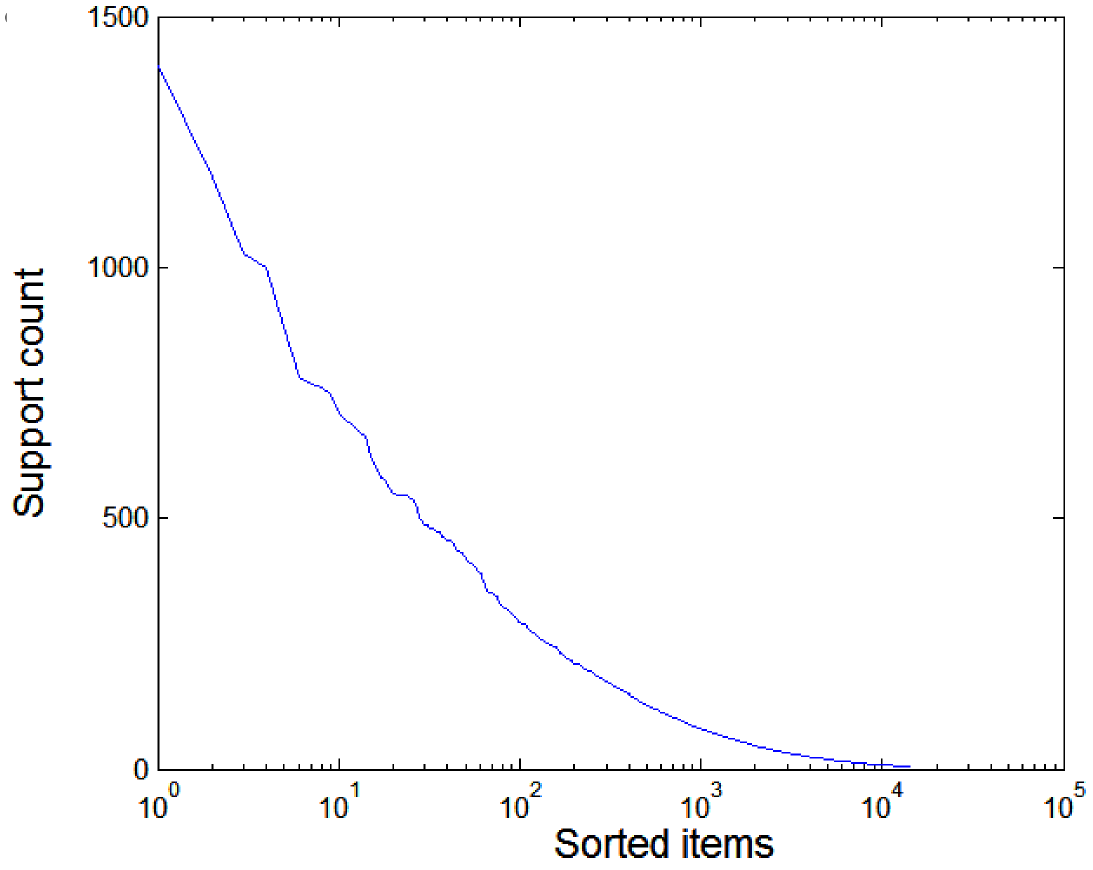
Problems with the association mining

- Single minsup: It assumes that all items in the data are of the same nature and/or have similar frequencies.
- Not true: In many applications, some items appear very frequently in the data, while others rarely appear.

  E.g., in a supermarket, people buy *food processor* and *cooking pan* much less frequently than they buy *bread* and *milk*.

Effect of Support Distribution

- Many real data sets h



Support distribution of a retail data set

Rare Item Problem

- If the frequencies of items vary a great deal, we will encounter two problems
  - If minsup is set too high, those rules that involve rare items will not be found.
  - To find rules that involve both frequent and rare items, minsup has to be set very low. This may cause combinatorial explosion because those frequent items will be associated with one another in all possible ways.
- Using a single minimum support threshold may not be effective

Multiple minsups model

- The minimum support of a rule is expressed in terms of *minimum item supports* (MIS) of the items that appear in the rule.
- Each item can have a minimum item support.
- By providing different MIS values for different items, the user effectively expresses different support requirements for different rules.

Minsup of a rule

- Let MIS(i) be the MIS value of item i. The *minsup* of a rule R is the lowest MIS value of the items in the rule.
- I.e., a rule R: $a_1, a_2, \ldots, a_k \rightarrow a_{k+1}, \ldots, a_r$ satisfies its minimum support if its actual support is $\geq$

$$\min(\text{MIS}(a_1), \text{MIS}(a_2), \ldots, \text{MIS}(a_r)).$$

An Example

- Consider the following items:

    bread, shoes, clothes

    The user-specified MIS values are as follows:

    MIS(*bread*) = 2%   MIS(*shoes*) = 0.1%

    MIS(*clothes*) = 0.2%

    The following rule doesn’t satisfy its minsup:

    clothes $\rightarrow$ *bread* [sup=0.15%,conf =70%]

    The following rule satisfies its minsup:

    clothes $\rightarrow$ *shoes* [sup=0.15%,conf =70%]

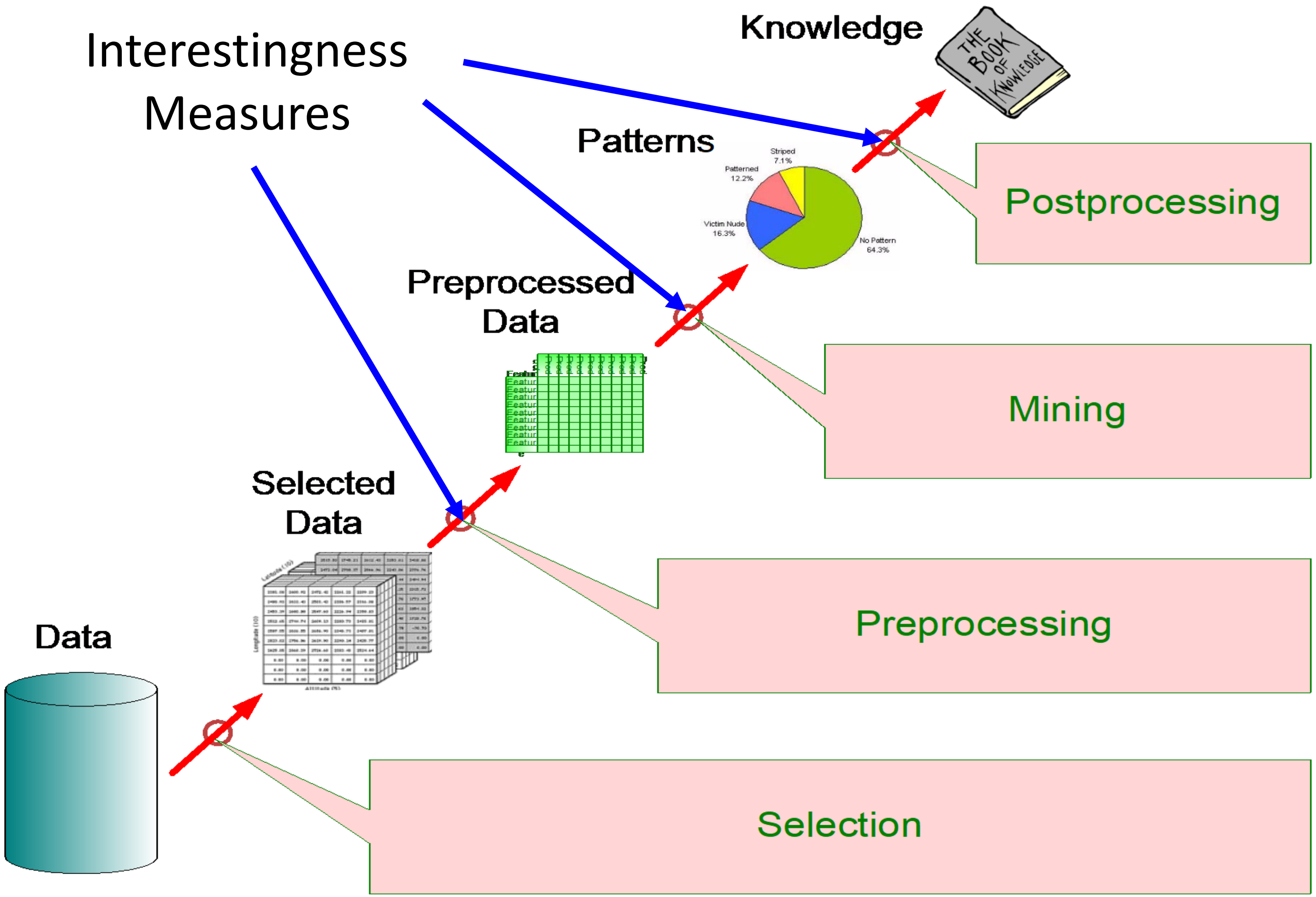
Pattern Evaluation

- Association rule algorithms tend to produce too many rules
  - many of them are uninteresting or redundant
  - Redundant if $\{A,B,C\} \rightarrow \{D\}$ and $\{A,B\} \rightarrow \{D\}$ have same support & confidence

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

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

Application of Interestingness Measure



Computing Interestingness Measure

- Given a rule $X \rightarrow Y$, 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_{o+}
	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 $\overline{Y}$

Used to define various measures

- support, confidence, lift, Gini, J-measure, etc.

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

but P(Coffee) = 0.9

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

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

Statistical-based Measures

- Measures that take into account statistical dependence

$$Lift = \frac{P(Y \mid X)}{P(Y)}$$

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

$$PS = P(X,Y) - P(X)P(Y)$$

$$\phi - coefficient = \frac{P(X,Y) - P(X)P(Y)}{\sqrt{P(X)[1-P(X)]P(Y)[1-P(Y)]}}$$

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

but P(Coffee) = 0.9

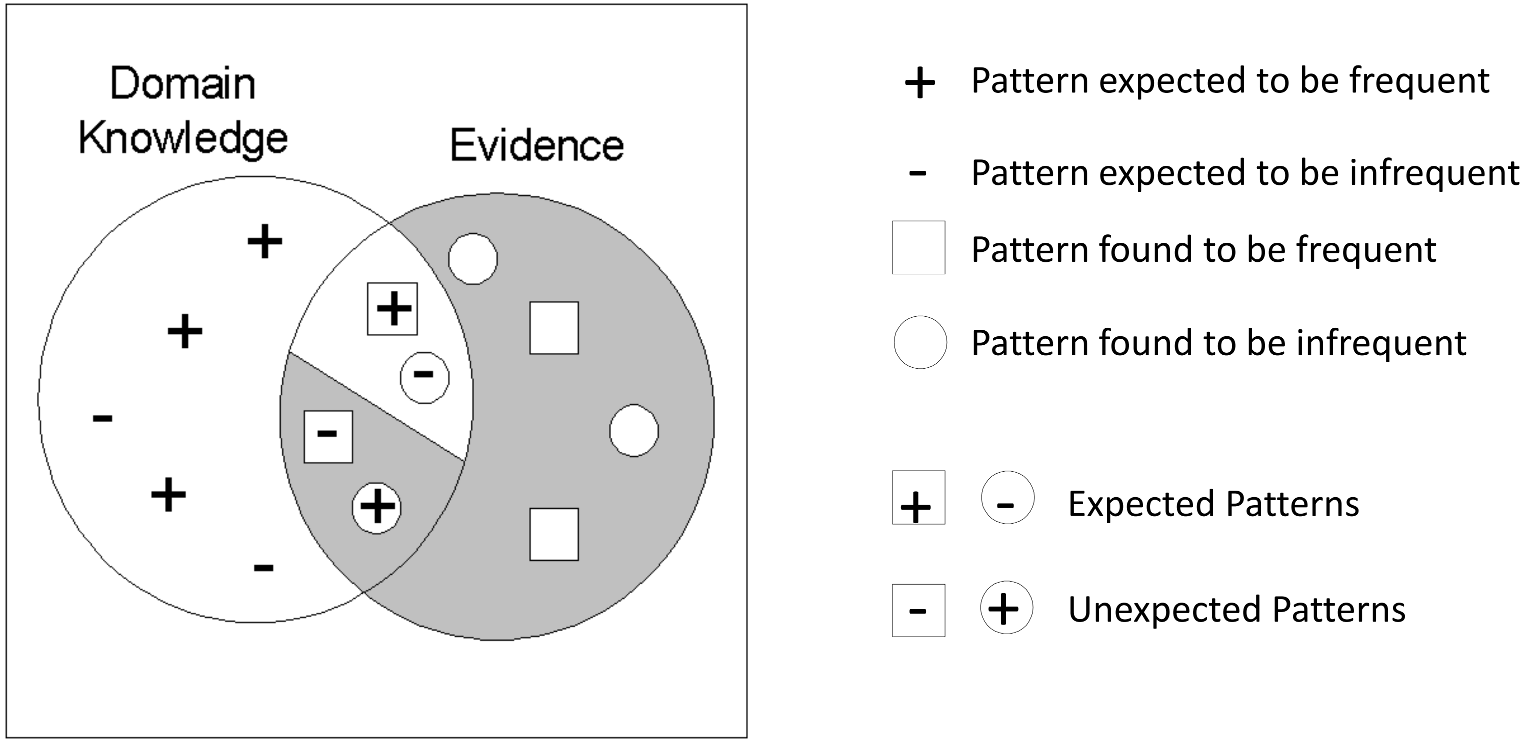
$\Rightarrow$ Lift = 0.75/0.9= 0.8333 (< 1, therefore is negatively associated)

Subjective Interestingness Measure

- Objective measure:
  - Rank patterns based on statistics computed from data
  - e.g., 21 measures of association (support, confidence, Laplace, Gini, mutual information, Jaccard, etc).
- Subjective measure:
  - Rank patterns according to user’s interpretation
    - A pattern is subjectively interesting if it contradicts the expectation of a user (Silberschatz & Tuzhilin)
    - A pattern is subjectively interesting if it is actionable (Silberschatz & Tuzhilin)

Interestingness via Unexpectedness

- Need to model expectation of users (domain knowledge)



- Need to combine expectation of users with evidence from data (i.e., extracted patterns)

Extra

Illustration

Candidate
1-Itemsets

Item	Count
Beer	3
Bread	4
Cola	2
Diapers	4
Milk	4
Eggs	1

Minimum support count = 3

Candidate
2-Itemsets

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

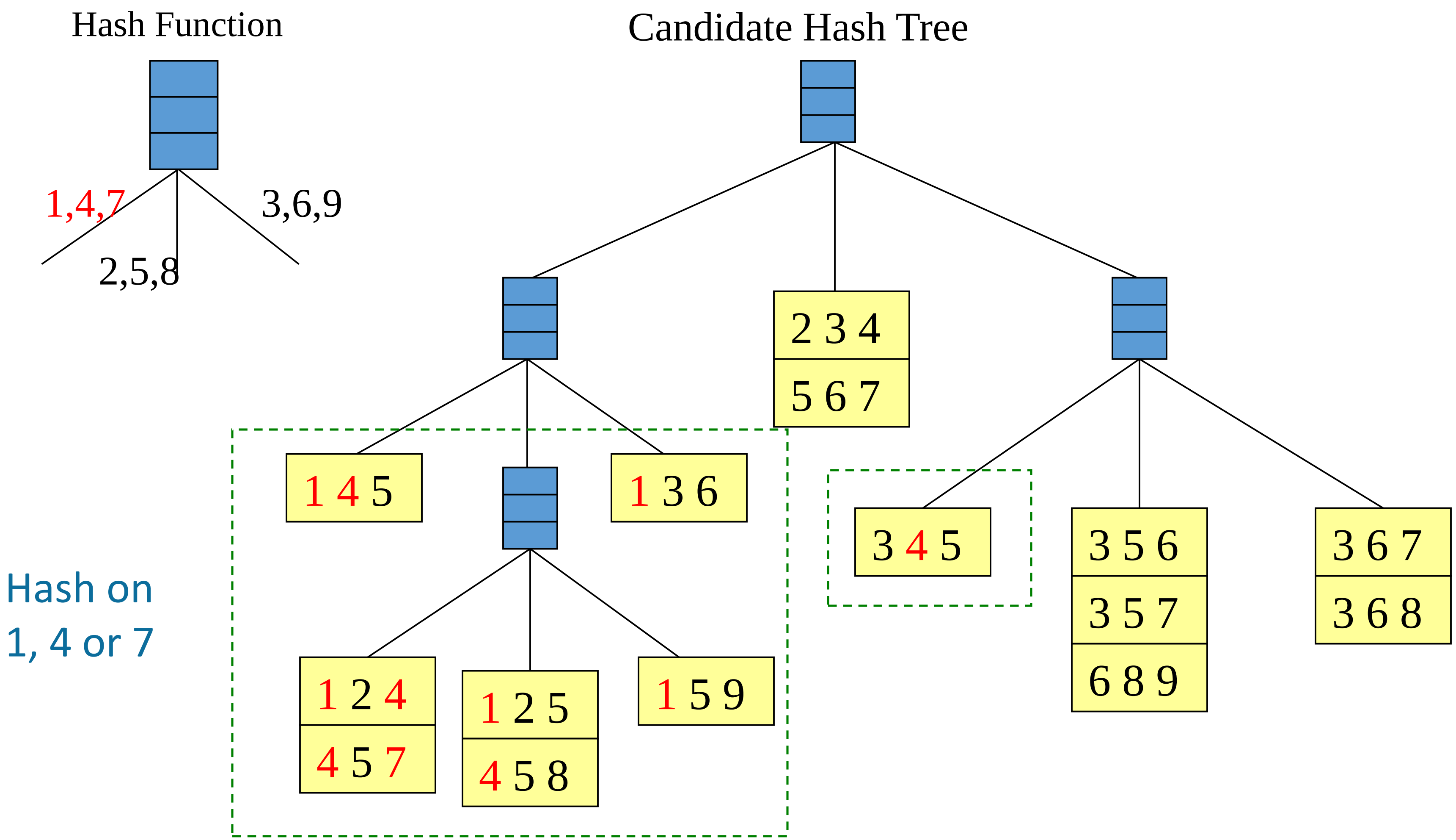
Itemsets removed
because of low
support

Candidate
3-Itemsets

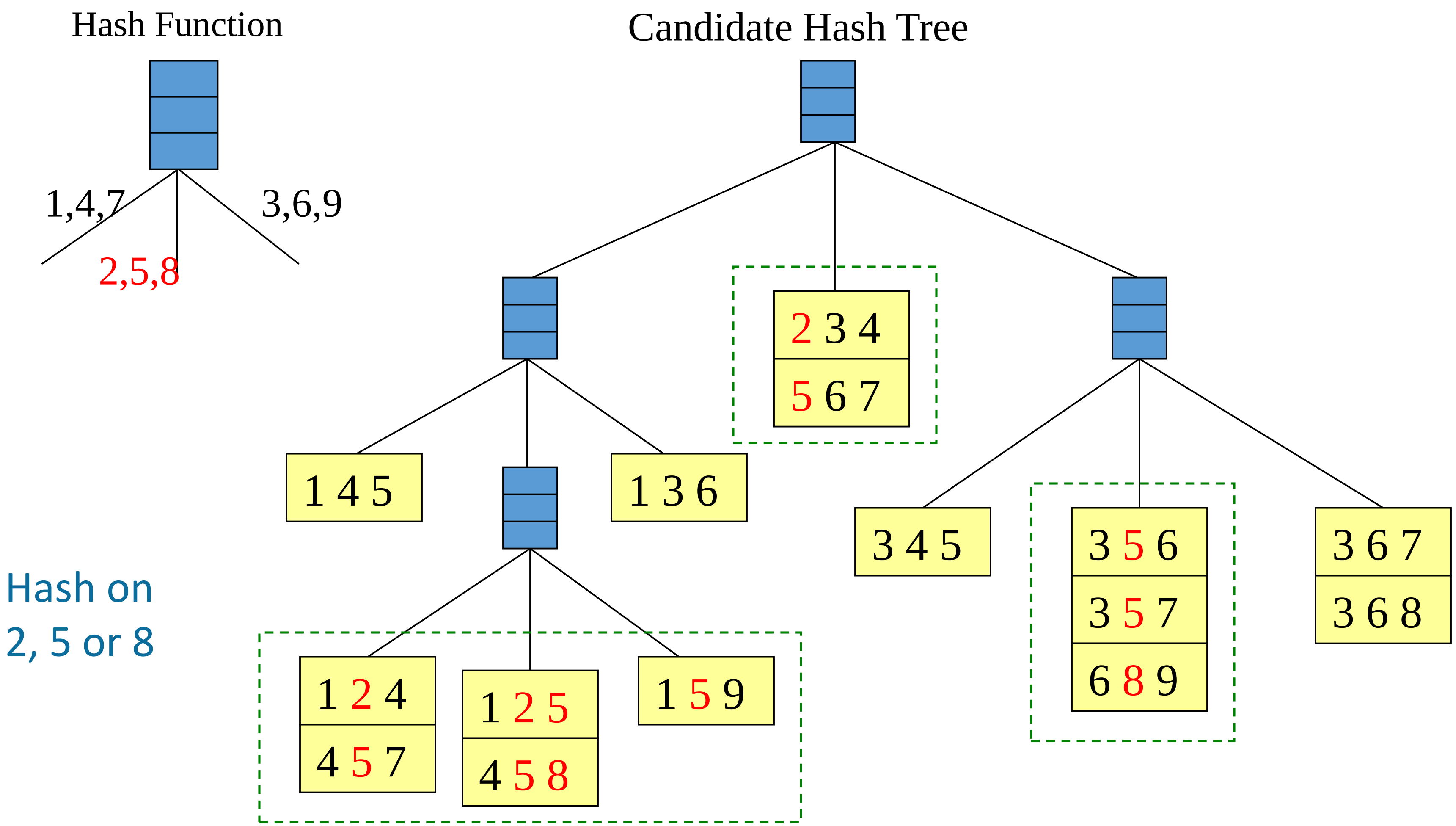
Itemset	Count
{Bread, Diapers, Milk}	3

Figure 6.5. Illustration of frequent itemset generation using the *Apriori* algorithm.

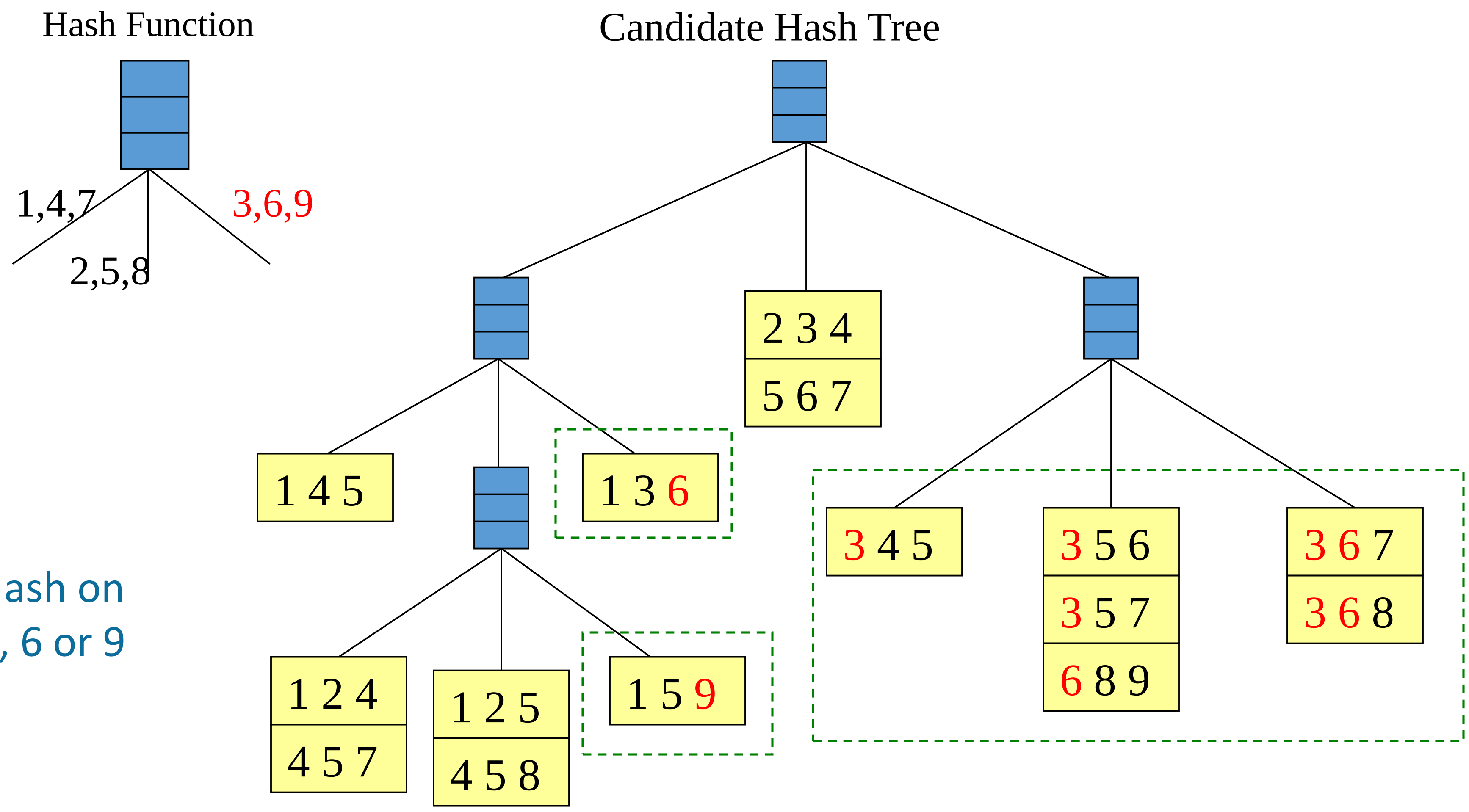
Association Rule Discovery: Hash tree



Association Rule Discovery: Hash tree



Association Rule Discovery: Hash tree



Hash on
3, 6 or 9

FP-growth Algorithm

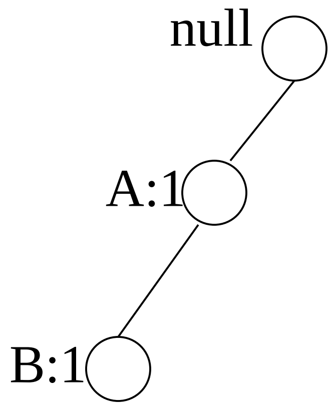
- Use a compressed representation of the database using an FP-tree

- Once an FP-tree has been constructed, it uses a recursive divide-and-conquer approach to mine the frequent itemsets

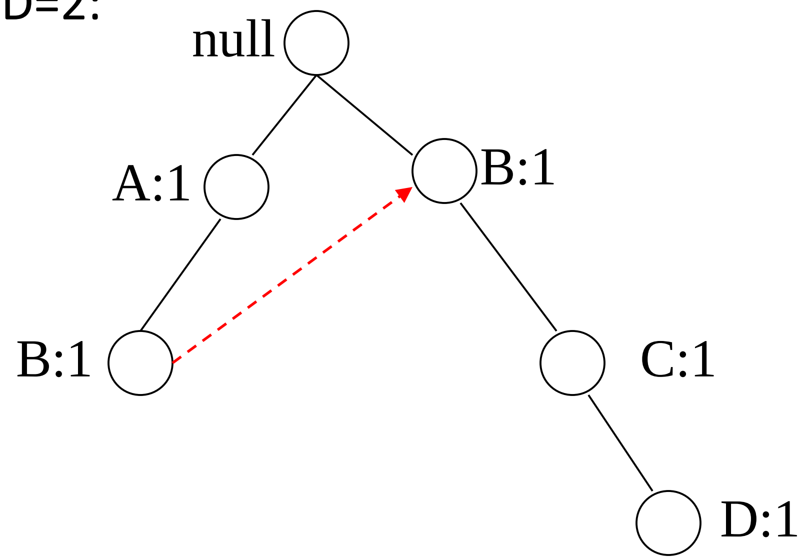
FP-tree construction

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=2:



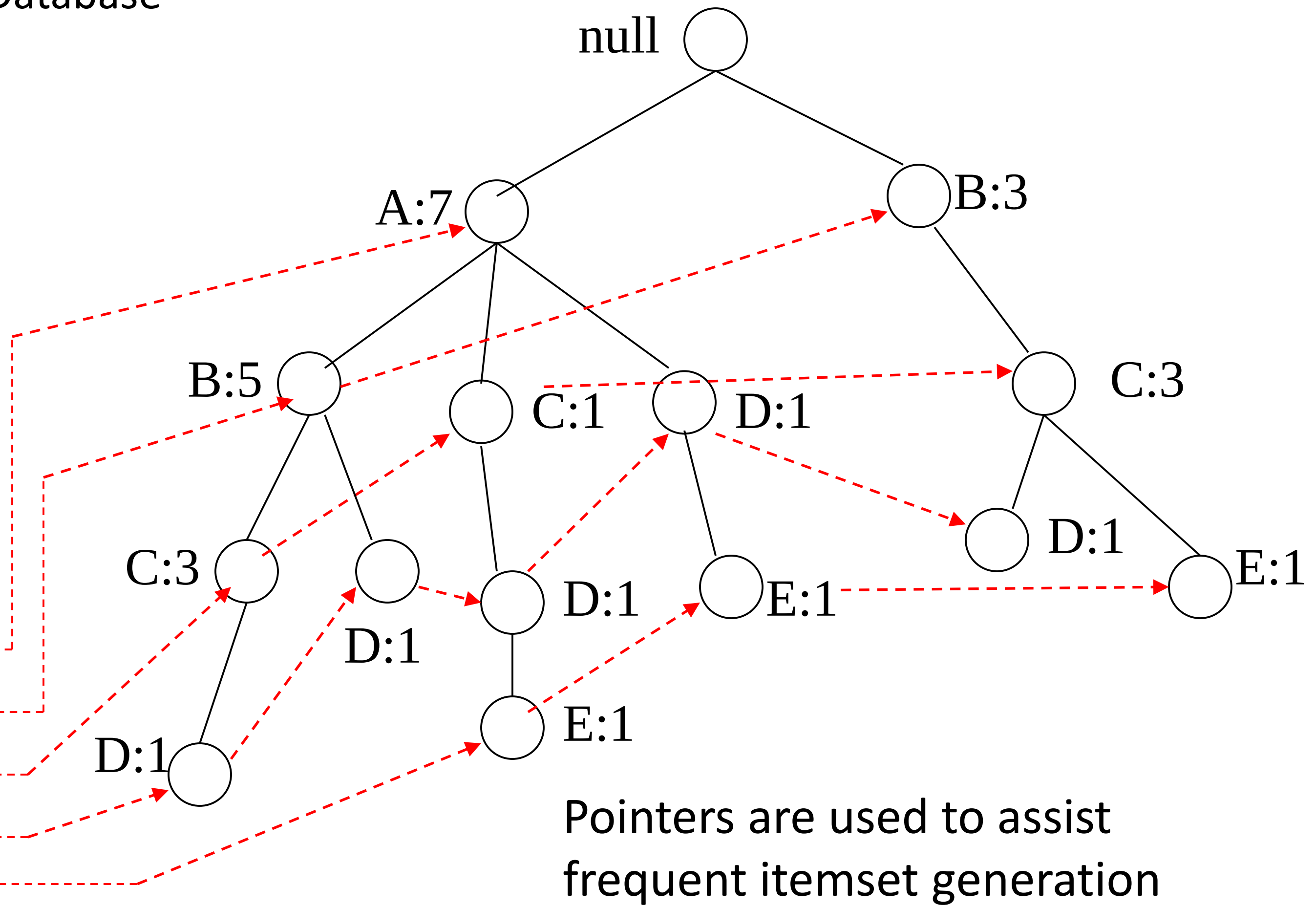
FP-Tree Construction

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

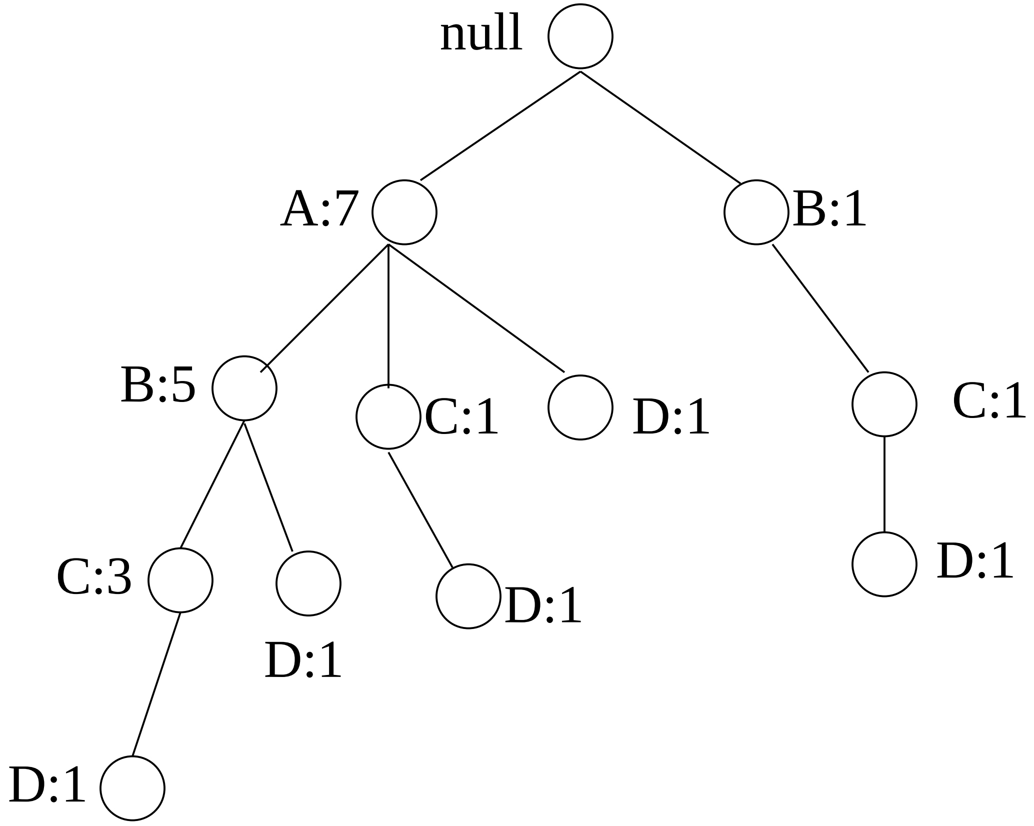
Header table

Item	Pointer
A	
B	
C	
D	
E	



Pointers are used to assist frequent itemset generation

FP-growth



Conditional Pattern base for D:

P = {(A:1,B:1,C:1),
(A:1,B:1),
(A:1,C:1),
(A:1),
(B:1,C:1)}

Recursively apply FP-growth on P

Frequent Itemsets found (with sup > 1):
AD, BD, CD, ACD, BCD