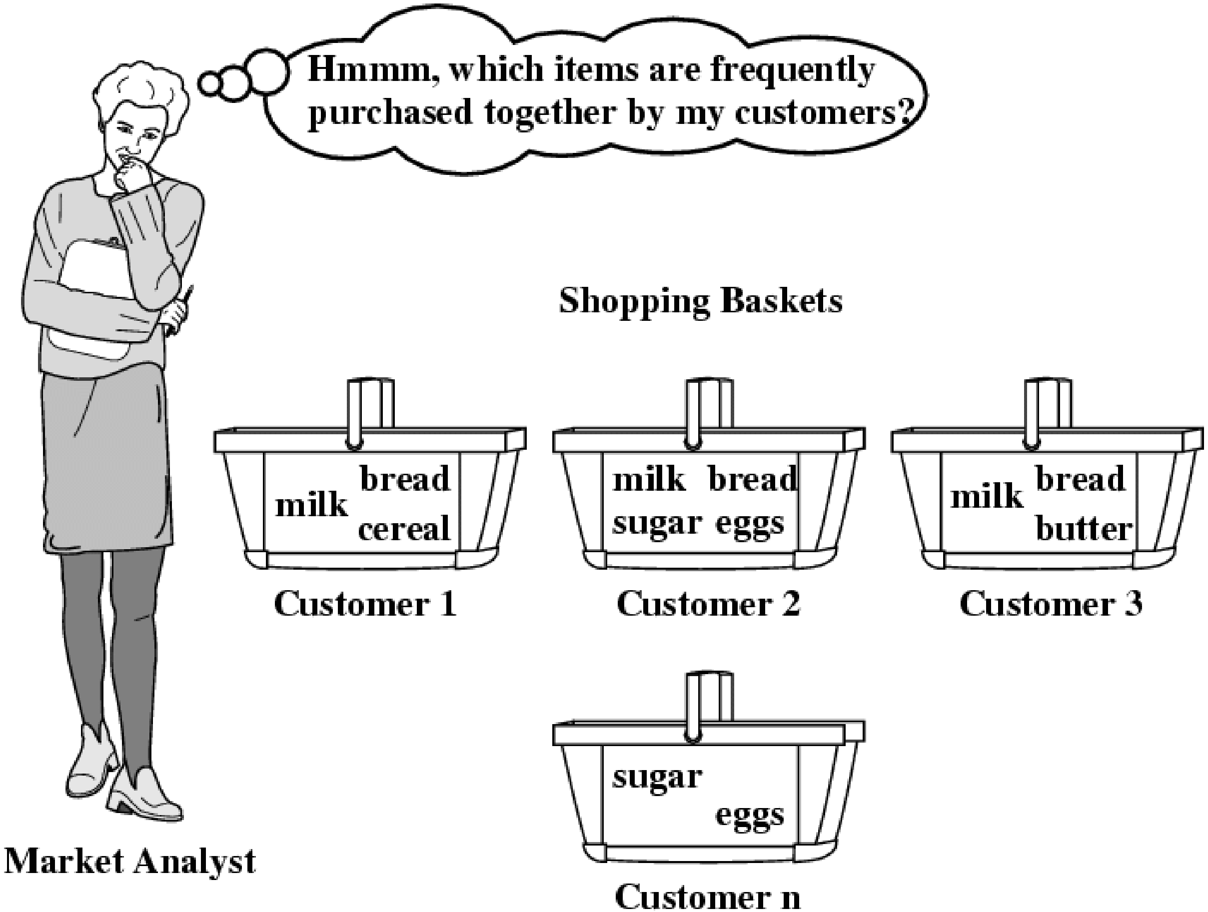
Mining Association Rules

Bassam Kurdy Ph.D

What Is Association Mining?

- Association rule mining:
  - Finding frequent patterns, associations, correlations, or causal structures among sets of items or objects in transaction databases, relational databases, and other information repositories.
- Applications:
  - Basket data analysis, cross-marketing, catalog design, loss-leader analysis, clustering, classification, etc.
- Examples.
  - Rule form: "Body $\rightarrow$ Head [support, confidence]".
  - buys(x, "diapers") $\rightarrow$ buys(x, "beers") [0.5%, 60%]
  - major(x, "CS") ^ takes(x, "DB") $\rightarrow$ grade(x, "A") [1%, 75%]

Market Basket Analysis

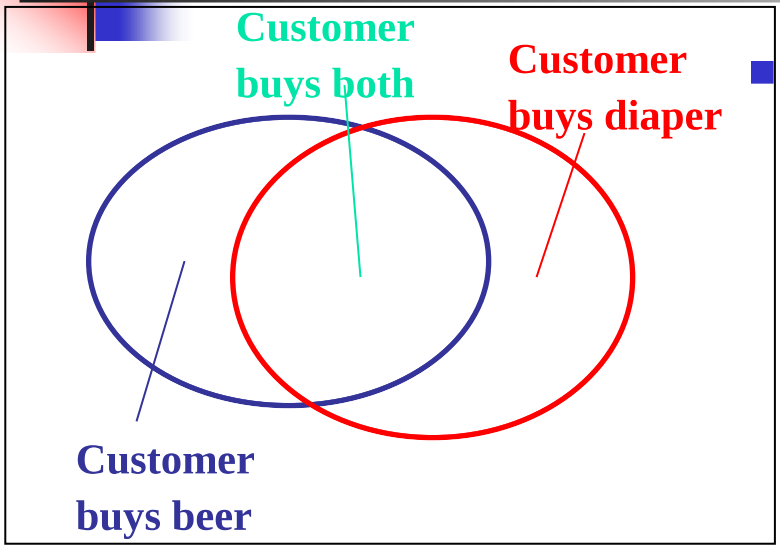


Typically, association rules are considered interesting if they satisfy both a minimum support threshold and a minimum confidence threshold.

Association Rule: Basic Concepts

- Given: (1) database of transactions, (2) each transaction is a list of items (purchased by a customer in a visit)
- Find: all rules that correlate the presence of one set of items with that of another set of items
  - E.g., *98% of people who purchase tires and auto accessories also get automotive services done*
- Applications
  - ** $\Rightarrow$ Maintenance Agreement* (What the store should do to boost Maintenance Agreement sales)
  - *Home Electronics $\Rightarrow$ ** (What other products should the store stocks up?)

Rule Measures: Support and Confidence



Find all the rules $X \& Y \Rightarrow Z$ with minimum confidence and support

- support, *s*, probability that a transaction contains {X & Y & Z}
- confidence, *c*, conditional probability that a transaction having {X & Y} also contains *Z*

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

Let minimum support 50%, and minimum confidence 50%, we have

- $A \Rightarrow C$ (50%, 66.6%)
- $C \Rightarrow A$ (50%, 100%)

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} $\rightarrow$ {Beer},
{Milk, Bread} $\rightarrow$ {Eggs,Coke},
{Beer, Bread} $\rightarrow$ {Milk},

Implication means co-occurrence, not causality!

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: $\{\text{Milk, Diaper}\} \rightarrow \{\text{Beer}\}$

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:

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

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

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

Association Rule Mining Task

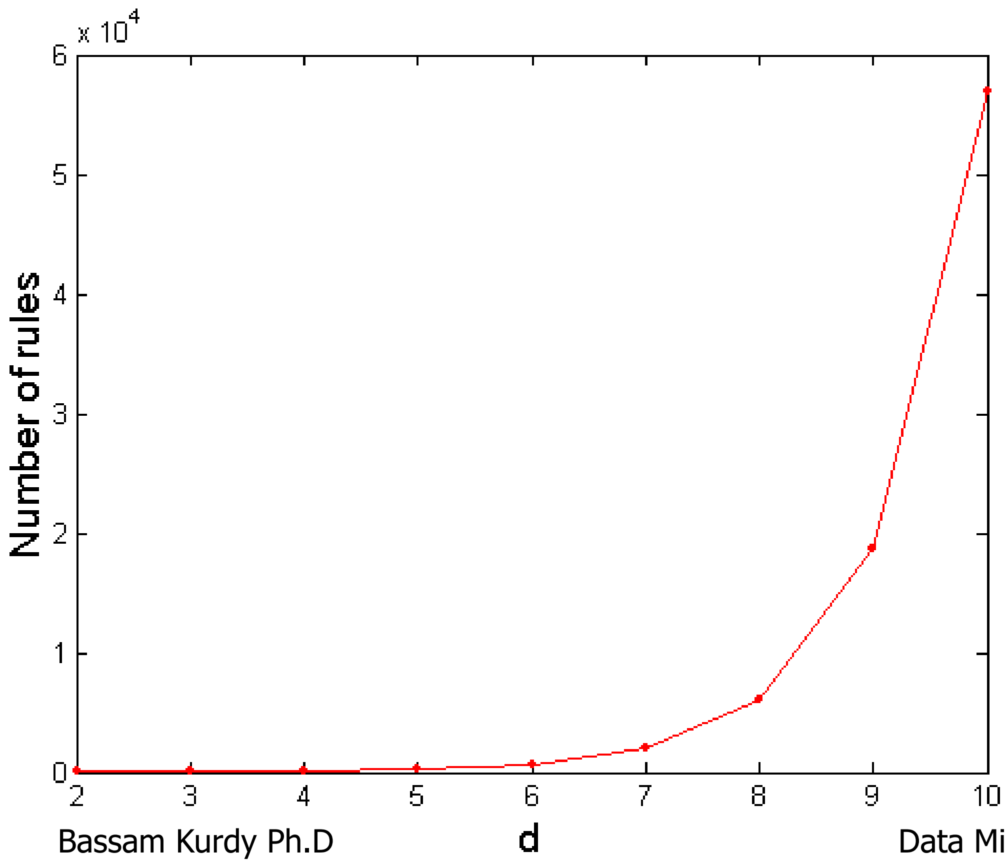
- Given a set of transactions T, the goal of association rule mining is to find all rules having
  - support ≥ *minsup* threshold
  - confidence ≥ *minconf* threshold

- **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 prohibitive!

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

Mining Association Rules: Decoupling

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)

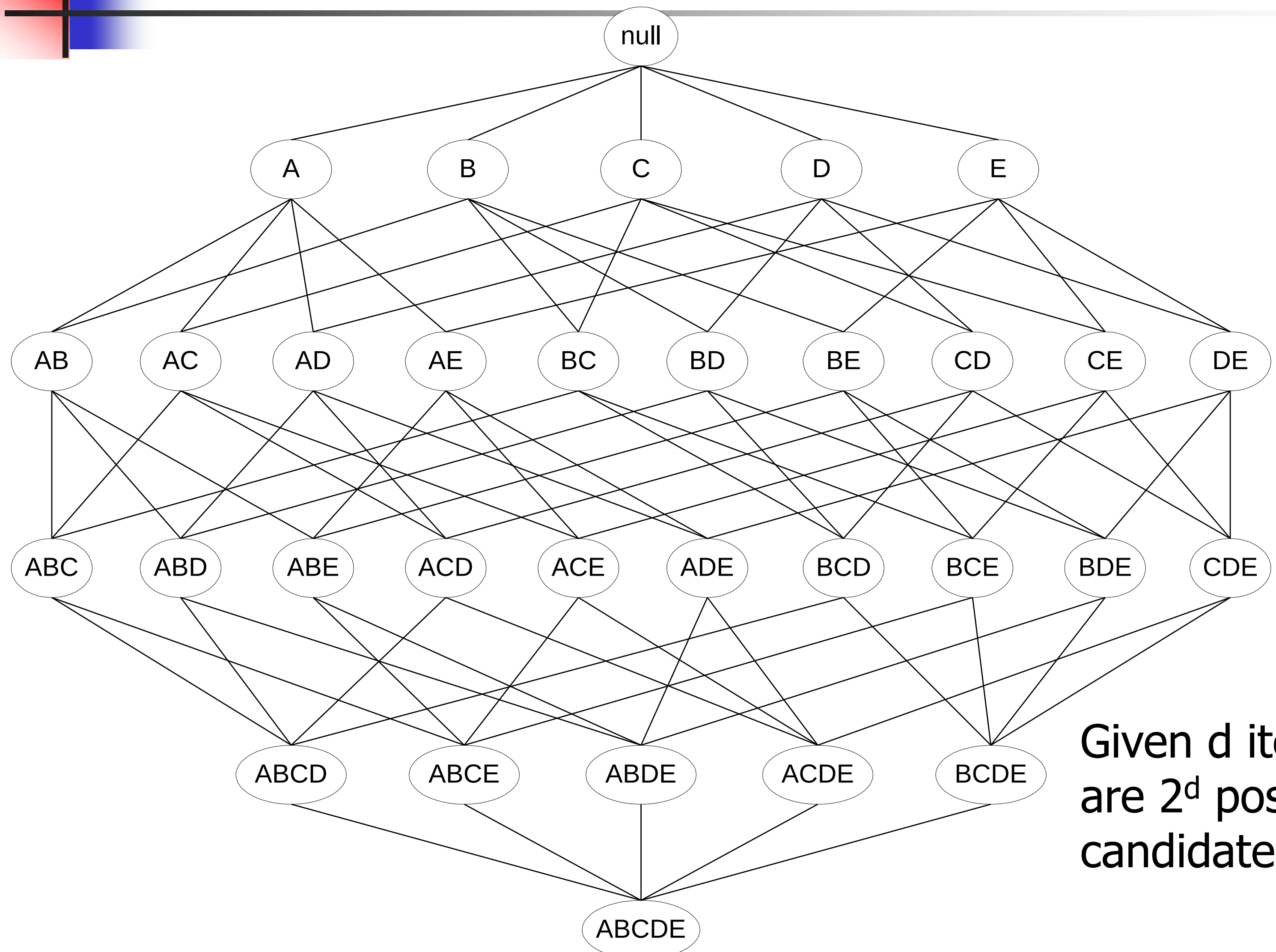
Observations:

- 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

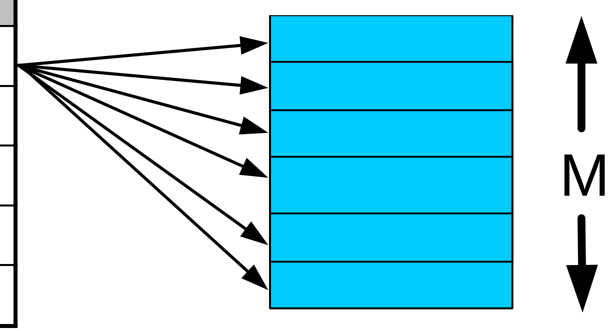
Transactions

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

List of Candidates



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

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
- 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

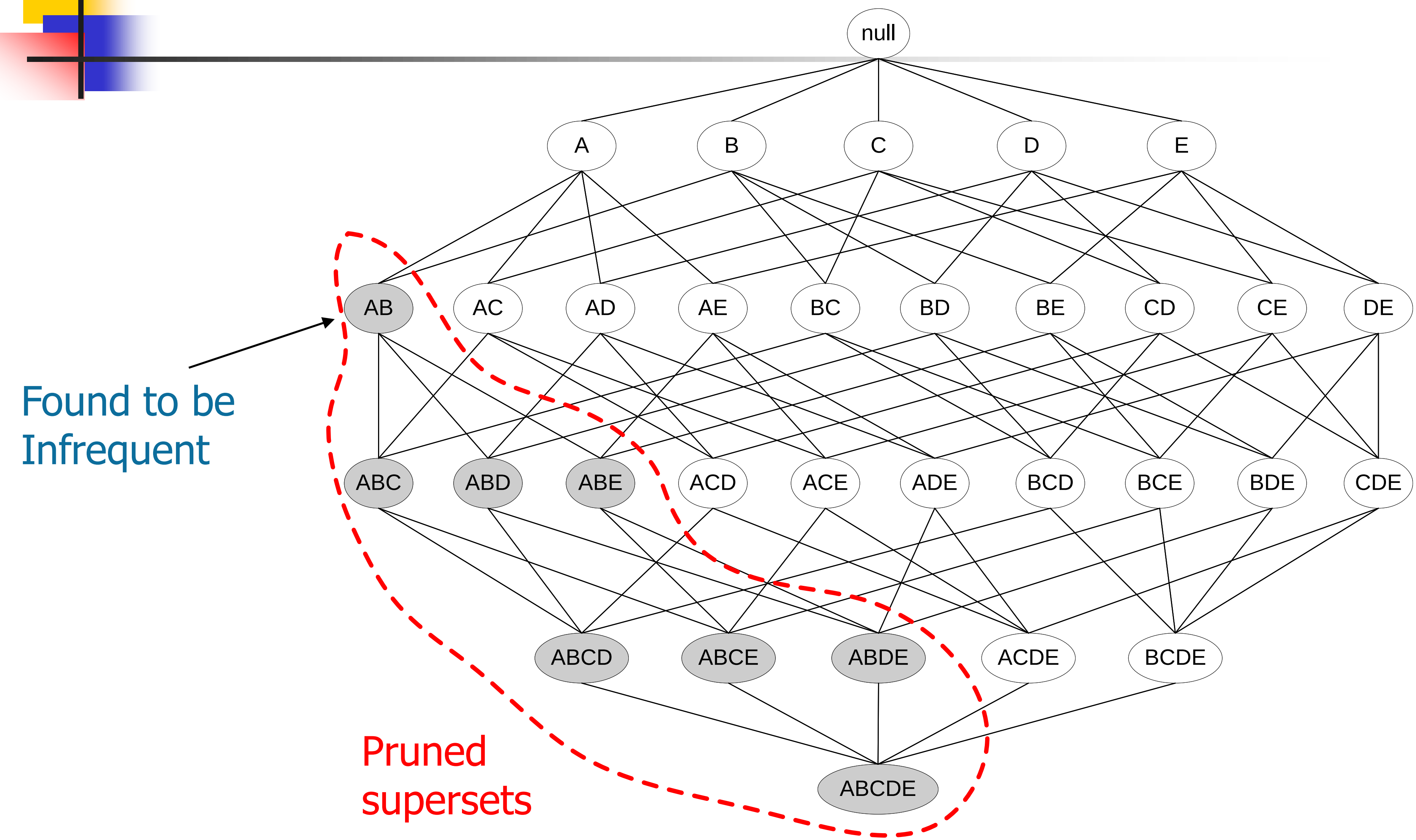
- Apriori principle:
  - If an itemset is frequent, then all of its subsets must also be frequent

- 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	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Items (1-itemsets)

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

Pairs (2-itemsets)

(No need to generate candidates involving Coke or Eggs)

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

Minimum Support = 3

Triplets (3-itemsets)

Itemset	Count
{Bread,Milk,Diaper}	2

If every subset is considered,

$$^6C_1 + {}^6C_2 + {}^6C_3 = 41$$

With support-based pruning,

$$6 + 6 + 1 = 13$$

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

The Apriori Algorithm

- **Join Step**: C_k is generated by joining $L_{k-1}$with itself
- **Prune Step**: Any (k-1)-itemset that is not frequent cannot be a subset of a frequent k-itemset
- <u>Pseudo-code</u>:

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**
  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 — Example

Database D

TID	Items
100	1 3 4
200	2 3 5
300	1 2 3 5
400	2 5

Scan D →

C_1

itemset	sup.
{1}	2
{2}	3
{3}	3
{4}	1
{5}	3

→ L_1

itemset	sup.
{1}	2
{2}	3
{3}	3
{5}	3

C_2

itemset
{1 2}
{1 3}
{1 5}
{2 3}
{2 5}
{3 5}

Scan D →

C_2

itemset	sup
{1 2}	1
{1 3}	2
{1 5}	1
{2 3}	2
{2 5}	3
{3 5}	2

→ L_2

itemset	sup
{1 3}	2
{2 3}	2
{2 5}	3
{3 5}	2

C_3

itemset
{2 3 5}

Scan D →

L_3

itemset	sup
{2 3 5}	2

How to Generate Candidates?

- Suppose the items in L_{k-1} are listed in an order
- Step 1: self-joining L_{k-1}

insert into $\boldsymbol{C_k}$

select $\boldsymbol{p.item_1, p.item_2, \ldots, p.item_{k-1}, q.item_{k-1}}$

from $\boldsymbol{L_{k-1}\ p, L_{k-1}\ q}$

where $\boldsymbol{p.item_1=q.item_1, \ldots, p.item_{k-2}=q.item_{k-2}, p.item_{k-1} < q.item_{k-1}}$

- Step 2: pruning

forall ***itemsets c in C_k*** do

forall ***(k-1)-subsets s of c*** do

if *(s is not in L_{k-1})* **then delete** *c* **from** C_k

Example of Generating Candidates

- L_3={*abc, abd, acd, ace, bcd*}

Example of Generating Candidates

- L_3={*abc, abd, acd, ace, bcd*}
- Self-joining: L_3*L_3
  - *abcd* from *abc* and *abd*
  - *acde* from *acd* and *ace*

Example of Generating Candidates

- 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*}

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

Reducing Number of Comparisons

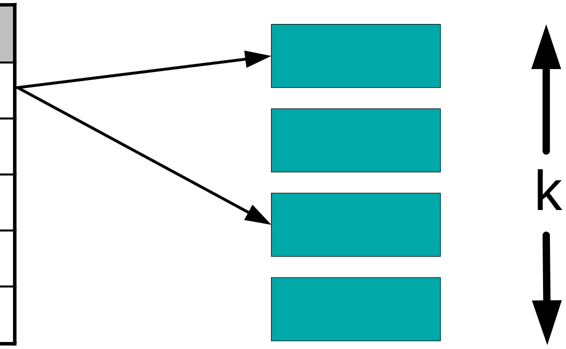
- Candidate counting:
  - Scan the database of transactions to determine the support of each candidate itemset
  - To reduce the number of comparisons, store the candidates in a hash structure
    - Instead of matching each transaction against every candidate, match it against candidates contained in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



Buckets

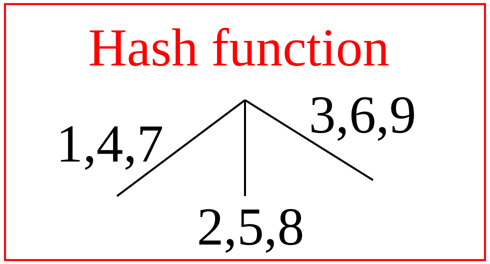
Generate Hash Tree

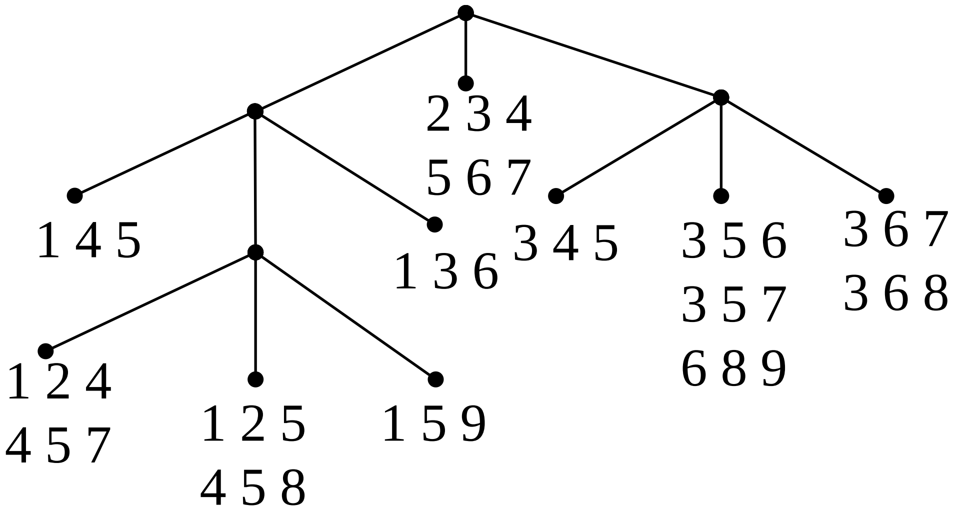
Suppose you have 15 candidate itemsets of length 3:

{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}

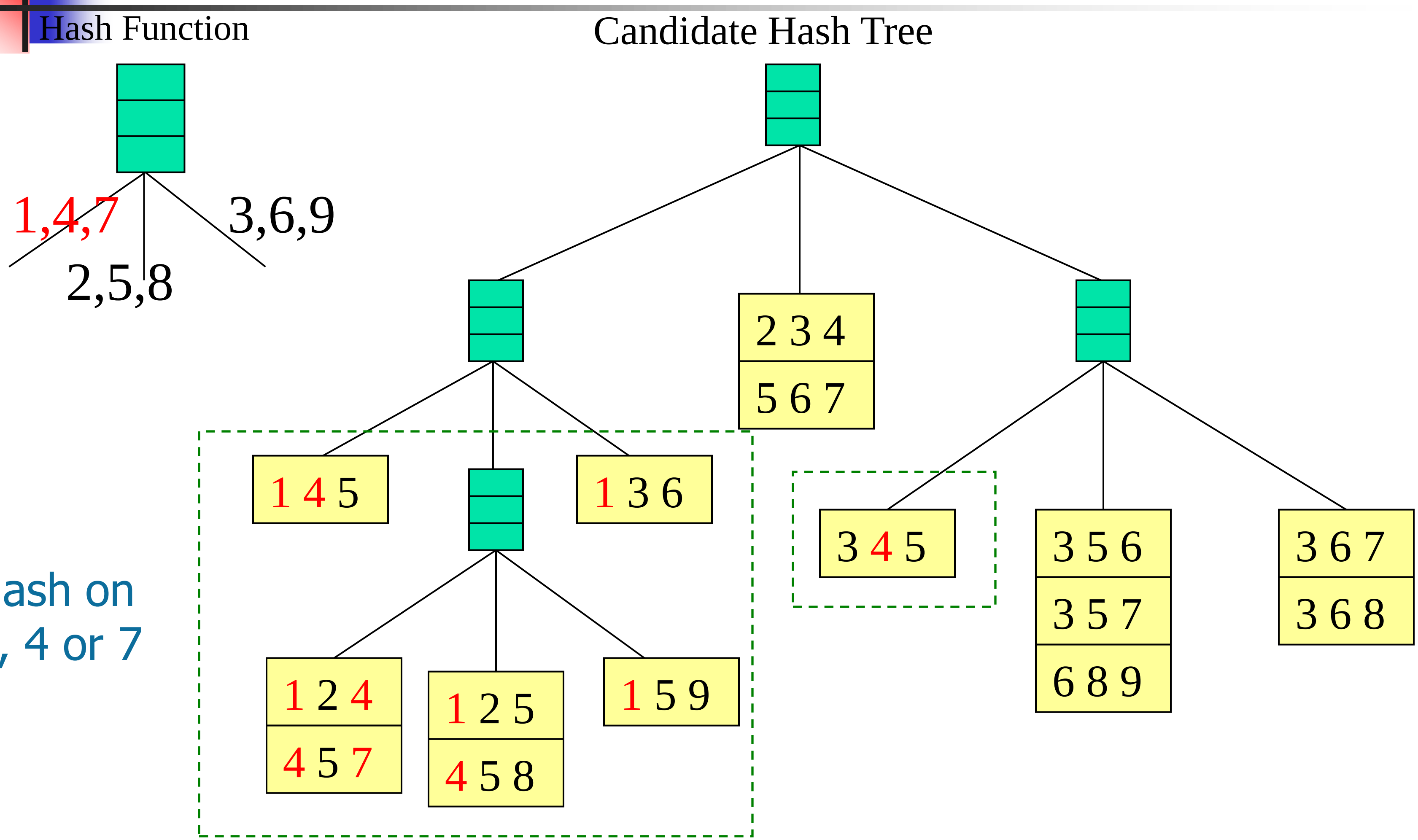
You need:

- 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)



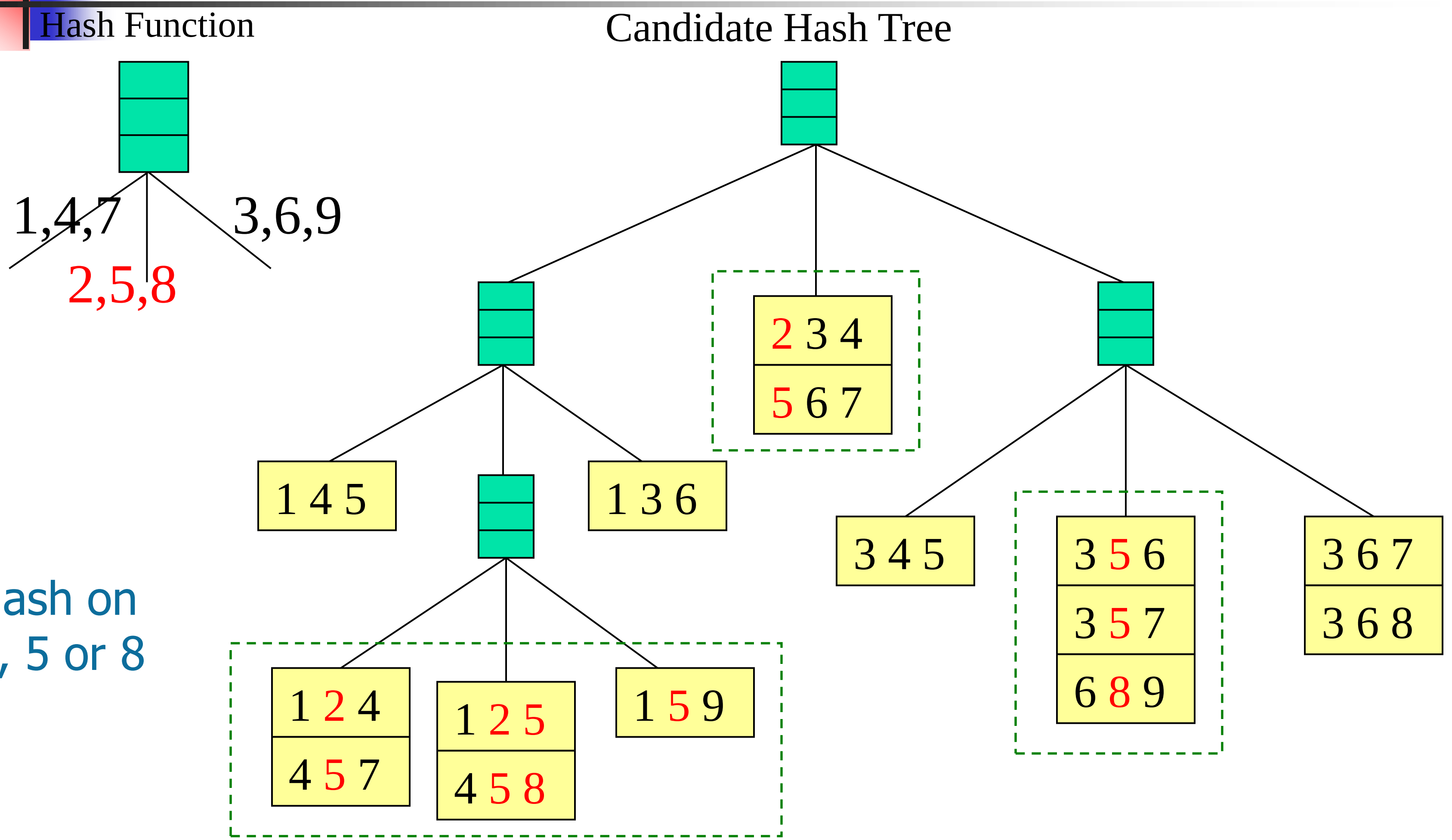


Association Rule Discovery: Hash tree



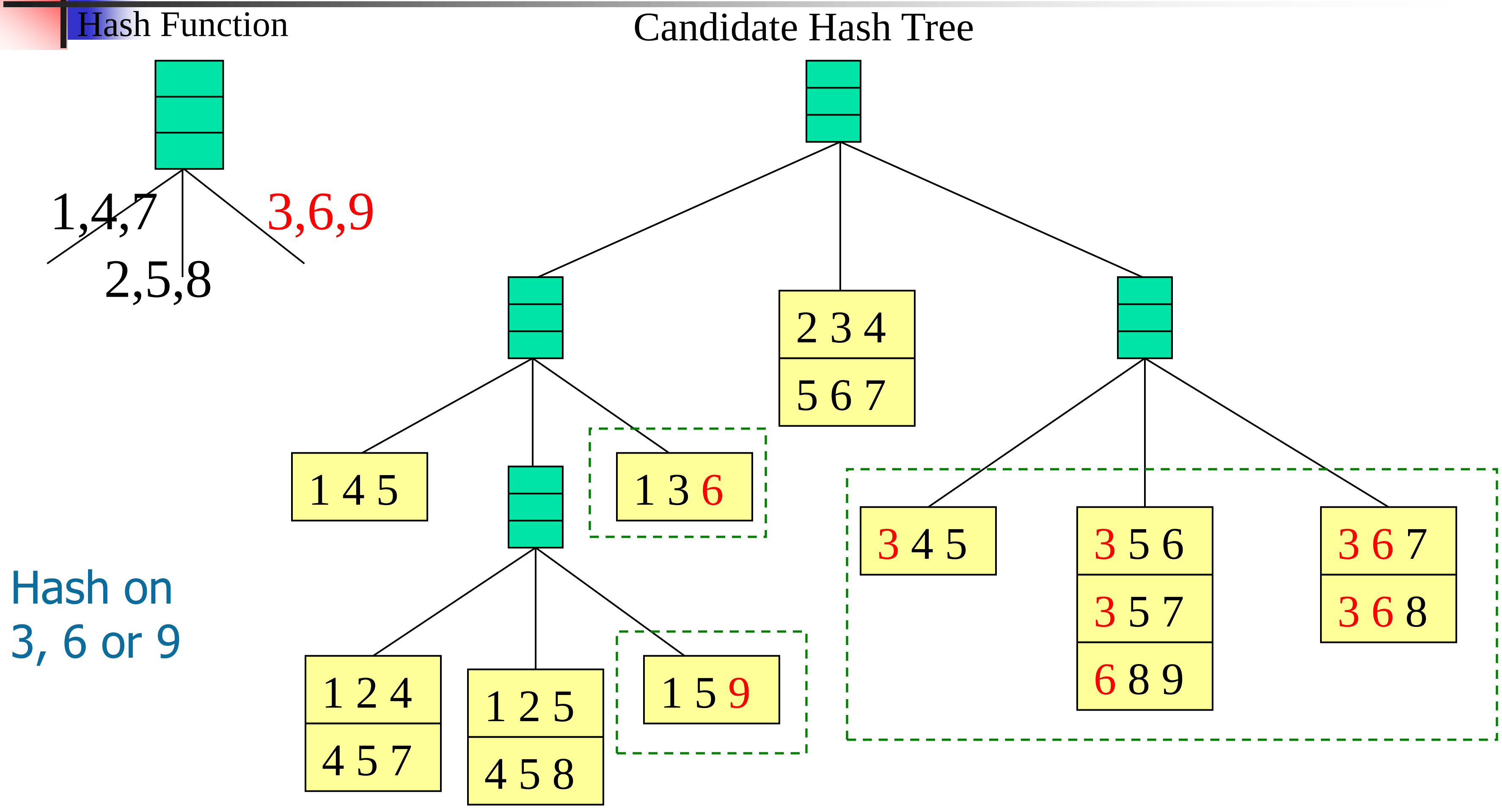
Hash on
1, 4 or 7

Association Rule Discovery: Hash tree



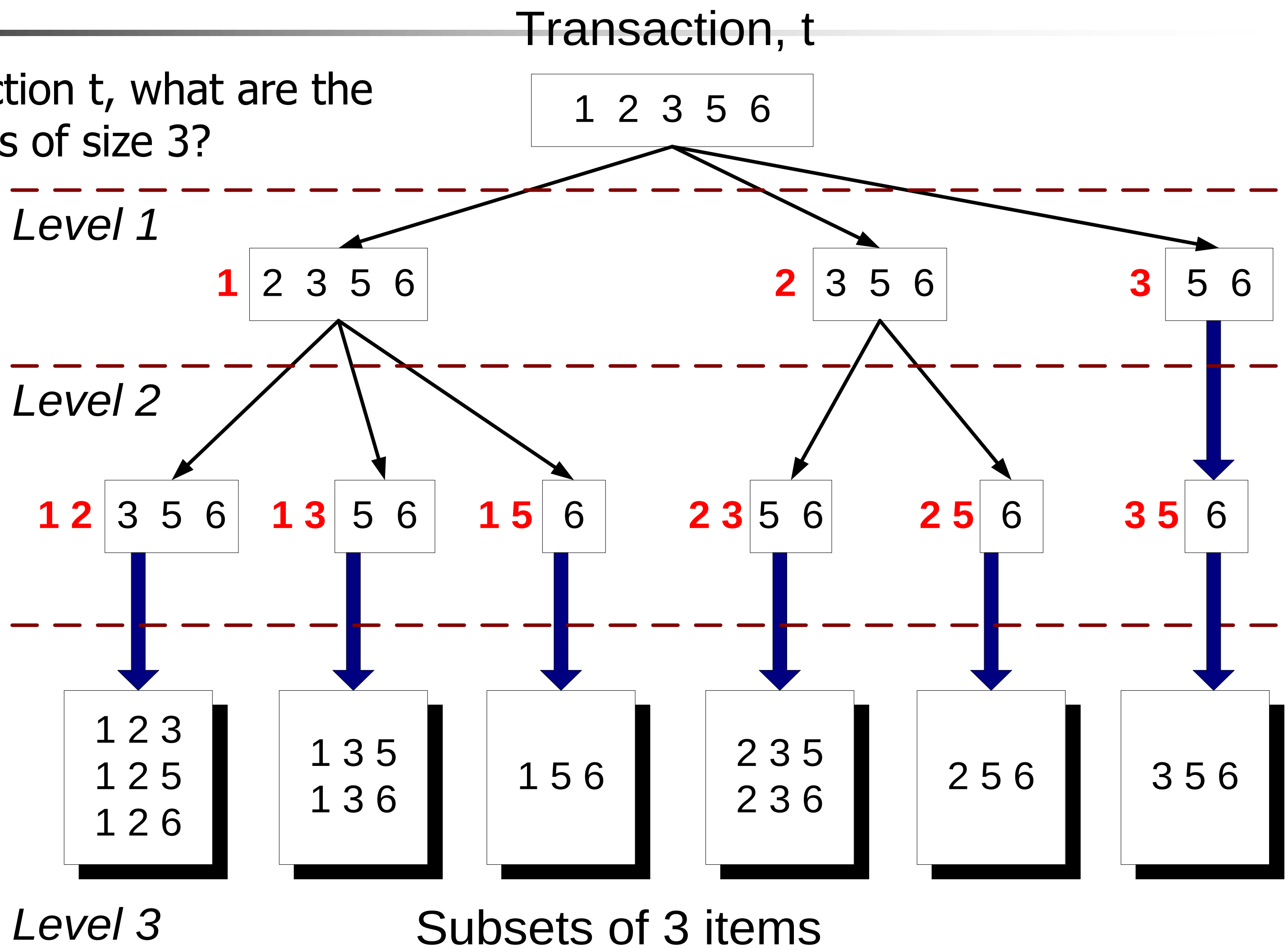
Hash on
2, 5 or 8

Association Rule Discovery: Hash tree

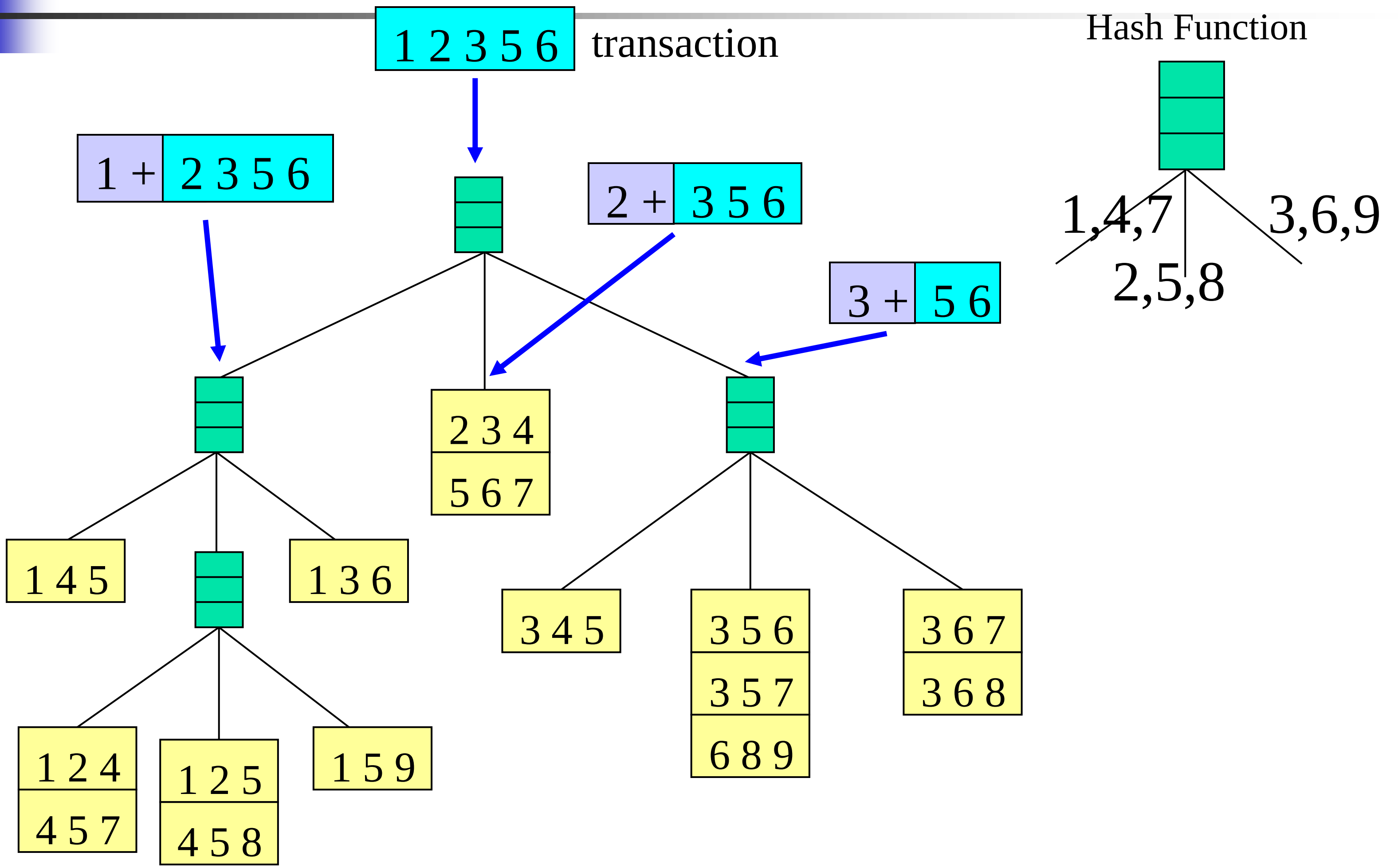


Subset Operation

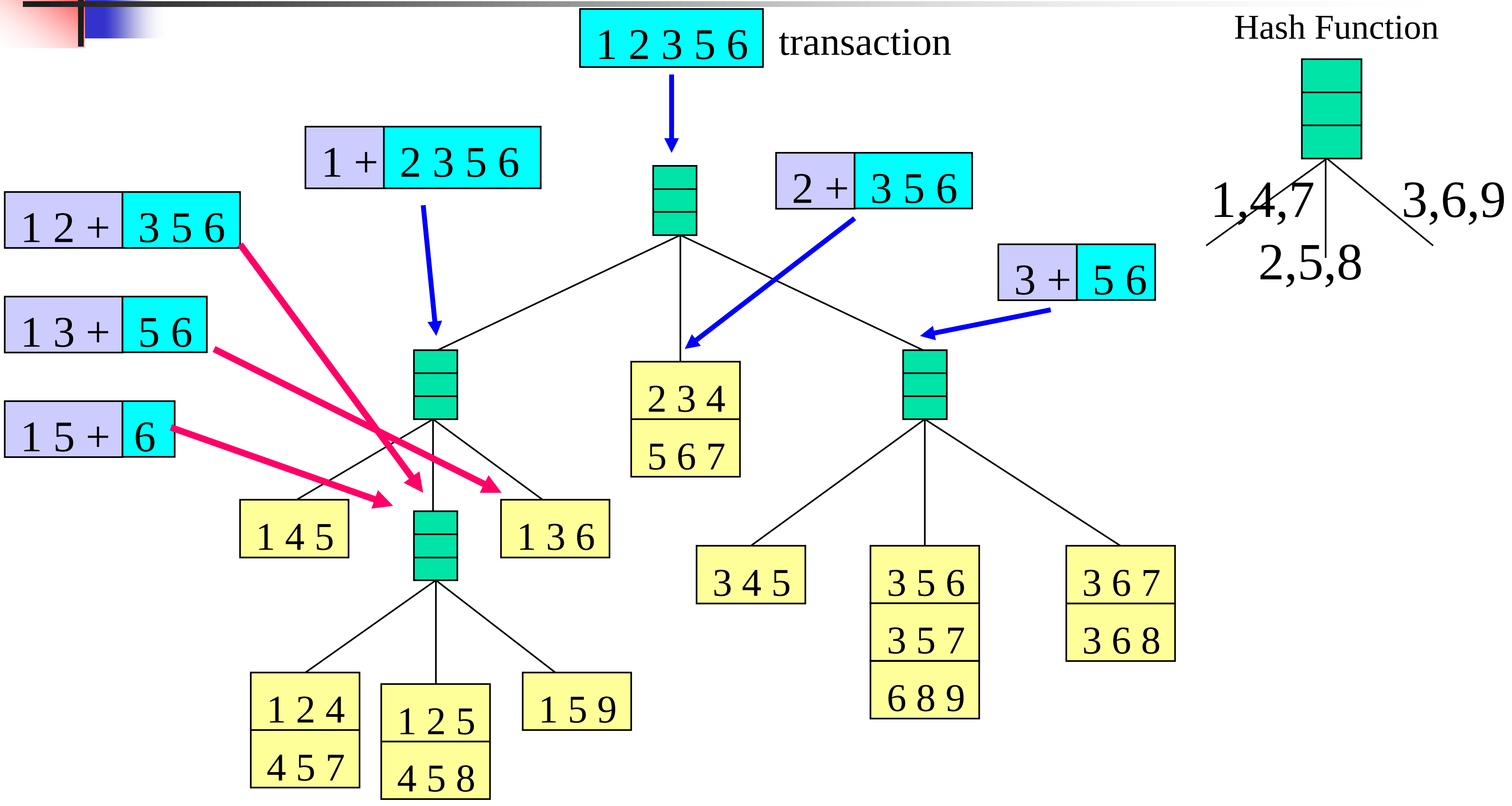
Given a transaction t, what are the possible subsets of size 3?



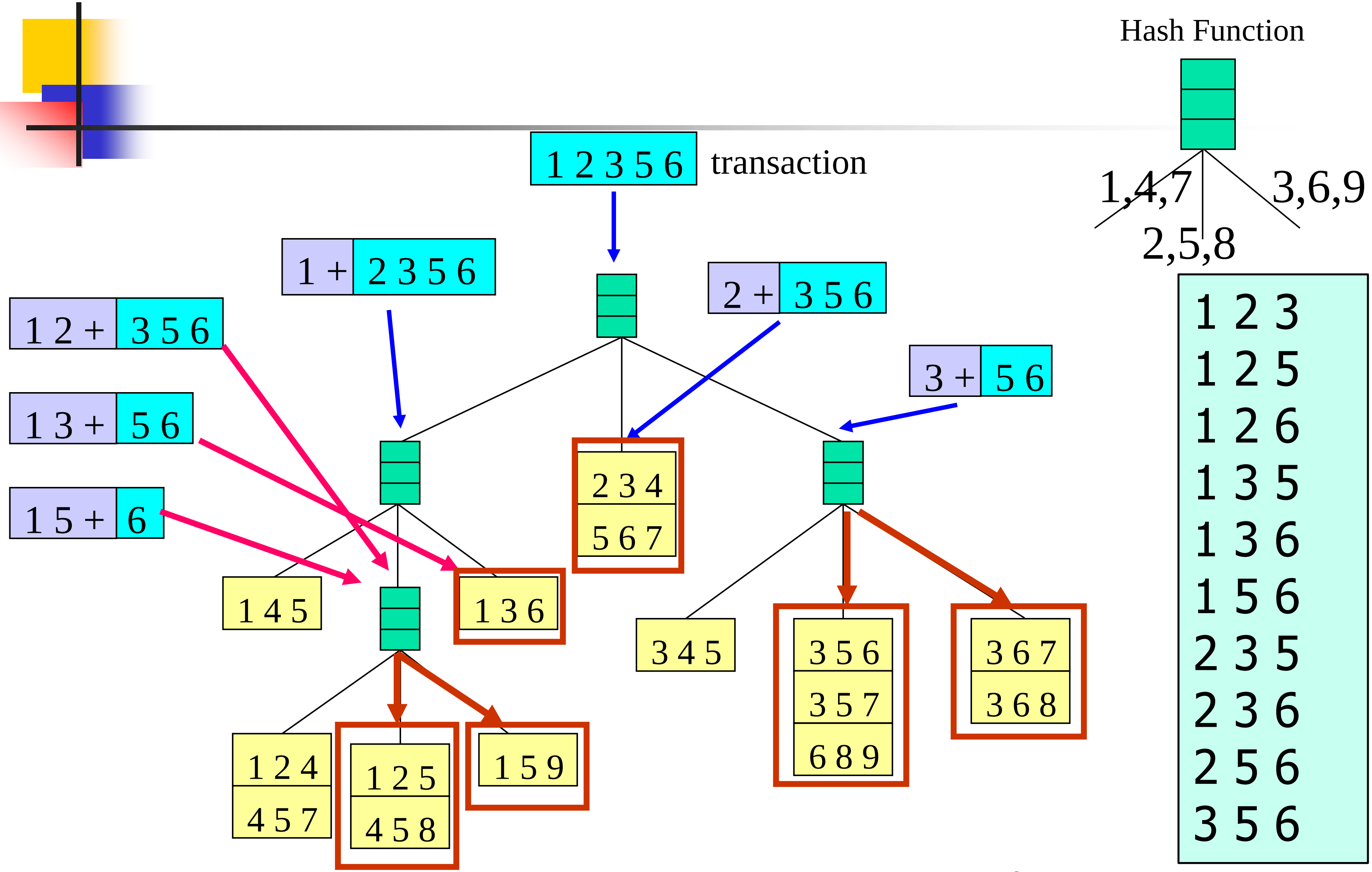
Subset Operation Using Hash Tree



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)

Methods to Improve Apriori's Efficiency

- Hash-based itemset counting: A *k*-itemset whose corresponding hashing bucket count is below the threshold cannot be frequent
- Transaction reduction: A transaction that does not contain any frequent k-itemset is useless in subsequent scans
- Partitioning: Any itemset that is potentially frequent in DB must be frequent in at least one of the partitions of DB
- Sampling: mining on a subset of given data, lower support threshold + a method to determine the completeness
- Dynamic itemset counting: add new candidate itemsets only when all of their subsets are estimated to be frequent

Is Apriori Fast Enough? — Performance Bottlenecks

- The core of the Apriori algorithm:
  - Use frequent (k – 1)-itemsets to generate candidate frequent k-itemsets
  - Use database scan and pattern matching to collect counts for the candidate itemsets
- The bottleneck of *Apriori*: candidate generation
  - Huge candidate sets:
    - 10^4 frequent 1-itemset will generate 10^7 candidate 2-itemsets
    - To discover a frequent pattern of size 100, e.g., $\{a_1, a_2, \ldots, a_{100}\}$, one needs to generate $2^{100} \approx 10^{30}$ candidates.
  - Multiple scans of database:
    - Needs ($n + 1$) scans, n is the length of the longest pattern

Mining Frequent Patterns Without Candidate Generation

- Compress a large database into a compact, Frequent-Pattern tree (FP-tree) structure
  - highly condensed, but complete for frequent pattern mining
  - avoid costly database scans
- Develop an efficient, FP-tree-based frequent pattern mining method
  - A divide-and-conquer methodology: decompose mining tasks into smaller ones
  - Avoid candidate generation: sub-database test only!

Construct FP-tree from a Transaction DB

TID	Items bought	(ordered) frequent items
100	{f, a, c, d, g, i, m, p}	{f, c, a, m, p}
200	{a, b, c, f, l, m, o}	{f, c, a, b, m}
300	{b, f, h, j, o}	{f, b}
400	{b, c, k, s, p}	{c, b, p}
500	{a, f, c, e, l, p, m, n}	{f, c, a, m, p}

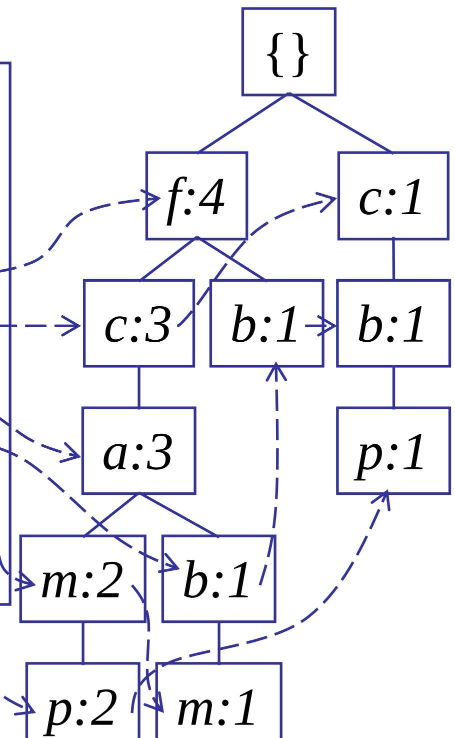
Min-support=3

Steps:

1. Scan DB once, find frequent 1-itemset (single item pattern)
2. Order frequent items in frequency descending order
3. Scan DB again, construct FP-tree

Header Table

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



Benefits of the FP-tree Structure

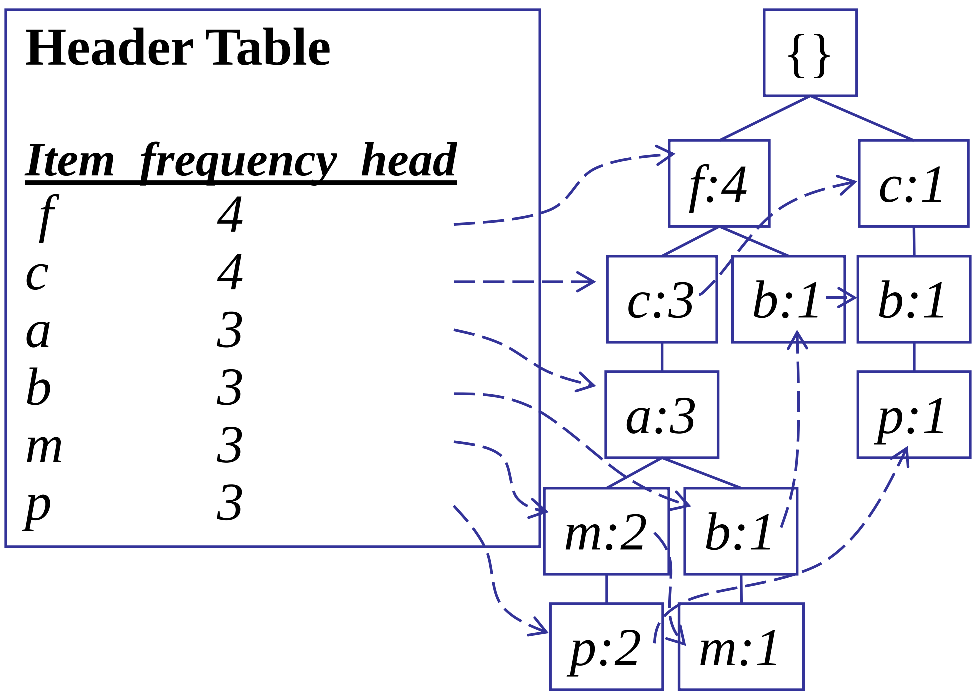
- Completeness:
  - never breaks a long pattern of any transaction
  - preserves complete information for frequent pattern mining
- Compactness
  - reduce irrelevant information—infrequent items are gone
  - frequency descending ordering: more frequent items are more likely to be shared
  - never be larger than the original database (if not count node-links and counts)

Mining Frequent Patterns Using FP-tree

- General idea (divide-and-conquer)
  - Recursively grow frequent pattern path using the FP-tree
- Method
  - For each item, construct its conditional pattern-base, and then its conditional FP-tree
  - Repeat the process on each newly created conditional FP-tree
  - Until the resulting FP-tree is empty, or it contains only one path (single path will generate all the combinations of its sub-paths, each of which is a frequent pattern)

Step 1: From FP-tree to Conditional Pattern Base

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



***Conditional* pattern bases**

item	***cond. pattern base***
c	***f:3***
a	***fc:3***
b	***fca:1, f:1, c:1***
m	***fca:2, fcab:1***
p	***fcam:2, cb:1***

Properties of FP-tree for Conditional Pattern Base Construction

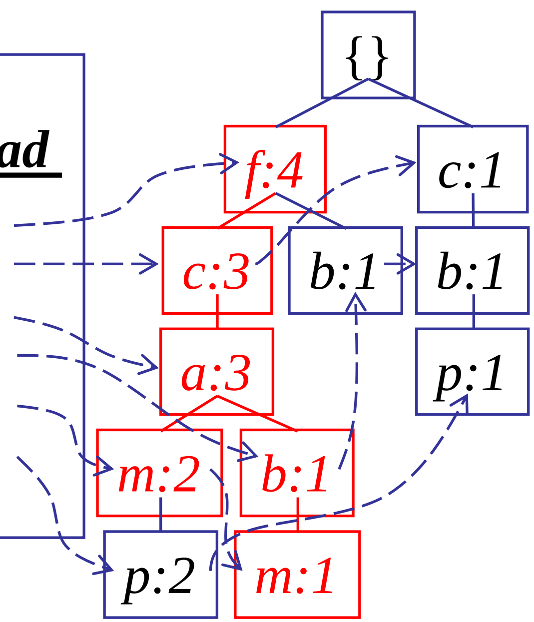
- Node-link property
  - For any frequent item a_i all the possible frequent patterns that contain a_i can be obtained by following a_i's node-links, starting from a_i's head in the FP-tree header
- Prefix path property
  - To calculate the frequent patterns for a node a_i in a path P, only the prefix sub-path of a_i in P need to be accumulated, and its frequency count should carry the same count as node a_i.

Step 2: Construct Conditional FP-tree

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

Header Table

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



m-conditional **pattern base:**

fca:2, fcab:1

→

{}
|
f:3
|
c:3
|
a:3

m-conditional **FP-tree**

→

All frequent patterns concerning *m*

m,

fm, cm, am,

fcm, fam, cam,

fcam

Mining Frequent Patterns by Creating Conditional Pattern-Bases

Item	Conditional pattern-base	Conditional FP-tree
p	{(fcam:2), (cb:1)}	{(c:3)}|p
m	{(fca:2), (fcab:1)}	{(f:3, c:3, a:3)}|m
b	{(fca:1), (f:1), (c:1)}	Empty
a	{(fc:3)}	{(f:3, c:3)}|a
c	{(f:3)}	{(f:3)}|c
f	Empty	Empty

Step 3: Recursively mine the conditional FP-tree

{}
|
f:3
|
c:3
|
a:3

***m-conditional* FP-tree**

Cond. pattern base of "am": (fc:3)

{}
|
f:3
|
c:3

***am-conditional* FP-tree**

Cond. pattern base of "cm": (f:3)

{}
|
f:3

***cm-conditional* FP-tree**

Cond. pattern base of "cam": (f:3)

{}
|
f:3

***cam-conditional* FP-tree**

Single FP-tree Path Generation

- Suppose an FP-tree T has a single path P
- The complete set of frequent pattern of T can be generated by enumeration of all the combinations of the sub-paths of P

{}
|
f:3
|
c:3
|
a:3

m-conditional FP-tree

→

All frequent patterns concerning _m_

***m*,**

***fm*, *cm*, *am*,**

***fcm*, *fam*, *cam*,**

fcam

Principles of Frequent Pattern Growth

- Pattern growth property
  - Let α be a frequent itemset in DB, B be α's conditional pattern base, and β be an itemset in B. Then $\alpha \cup \beta$ is a frequent itemset in DB iff β is frequent in B.
- "*abcdef*" is a frequent pattern, if and only if
  - "*abcde*" is a frequent pattern, and
  - "*f*" is frequent in the set of transactions containing "*abcde*"

Why Is Frequent Pattern Growth Fast?

- Our performance study shows
  - FP-growth is an order of magnitude faster than Apriori
- Reasoning
  - No candidate generation, no candidate test
  - Use compact data structure
  - Eliminate repeated database scan
  - Basic operation is counting and FP-tree building

Association Rules

- Association rule *R* : *Itemset1* => *Itemset2*
  - *Itemset1, 2* are disjoint and *Itemset2* is non-empty
  - meaning: if transaction includes *Itemset1* then it also has *Itemset2*
- Examples
  - A,B => E,C
  - A => B,C

From Frequent Itemsets to Association Rules

- *Q: Given frequent set {A,B,E}, what are possible association rules?*
  - A => B, E
  - A, B => E
  - A, E => B
  - B => A, E
  - B, E => A
  - E => A, B
  - __ => A,B,E (empty rule), or true => A,B,E

Rule Support and Confidence

- Suppose $R : I \Rightarrow J$ is an association rule
  - sup (R) = sup (I ∪ J) is the *support count*
    - support of itemset I ∪ J
  - conf (R) = sup(R) / sup(I) is the *confidence* of R
    - fraction of transactions with I ∪ J that have I
- Association rules with minimum support and count are sometimes called "***strong***" rules

Association Rules Example

- ***Q: Given frequent set {A,B,E}, what association rules have minsup = 2 and minconf= 50% ?***

  A, B => E : conf=2/4 = 50%

TID	List of items
1	**A, B, E**
2	B, D
3	B, C
4	A, B, D
5	A, C
6	B, C
7	A, C
8	**A, B, C, E**
9	A, B, C

Association Rules Example

- ***Q: Given frequent set {A,B,E}, what association rules have minsup = 2 and minconf= 50% ?***

  A, B => E : conf=2/4 = 50%

  A, E => B : conf=2/2 = 100%

  B, E => A : conf=2/2 = 100%

  E => A, B : conf=2/2 = 100%

Don't qualify

A =>B, E : conf=2/6 =33%< 50%

B => A, E : conf=2/7 = 28% < 50%

__ => A,B,E : conf: 2/9 = 22% < 50%

TID	List of items
1	**A, B, E**
2	B, D
3	B, C
4	A, B, D
5	A, C
6	B, C
7	A, C
8	**A, B, C, E**
9	A, B, C

Find Strong Association Rules

- A rule has the parameters *minsup* and *minconf*:
  - sup(R) >= *minsup* and conf (R) >= *minconf*
- Problem:
  - Find all association rules with given *minsup* and *minconf*
- First, find all frequent itemsets

Generating Association Rules

- Two stage process:
  - Determine frequent itemsets e.g. with the Apriori algorithm.
  - For each frequent item set I
    - for each subset J of I
      - determine all association rules of the form: $I\text{-}J \Rightarrow J$
- Main idea used in both stages : subset property

Weather Data: Play or not Play?

Outlook	Temperature	Humidity	Windy	Play?
sunny	hot	high	false	No
sunny	hot	high	true	No
overcast	hot	high	false	Yes
rain	mild	high	false	Yes
rain	cool	normal	false	Yes
rain	cool	normal	true	No
overcast	cool	normal	true	Yes
sunny	mild	high	false	No
sunny	cool	normal	false	Yes
rain	mild	normal	false	Yes
sunny	mild	normal	true	Yes
overcast	mild	high	true	Yes
overcast	hot	normal	false	Yes
rain	mild	high	true	No

Example: Generating Rules from an Itemset

- Frequent itemset from golf data:

```
Humidity = Normal, Windy = False, Play = Yes (4)
```

- Seven potential rules:

```
If Humidity = Normal and Windy = False then Play = Yes          4/4
If Humidity = Normal and Play = Yes then Windy = False          4/6
If Windy = False and Play = Yes then Humidity = Normal          4/6
If Humidity = Normal then Windy = False and Play = Yes          4/7
If Windy = False then Humidity = Normal and Play = Yes          4/8
If Play = Yes then Humidity = Normal and Windy = False          4/9
If True then Humidity = Normal and Windy = False and Play = Yes 4/14
```

Rules for the weather data

- Rules with support > 1 and confidence = 100%:

	Association rule		Sup.	Conf.
1	Humidity=Normal Windy=False	⇒Play=Yes	4	100%
2	Temperature=Cool	⇒Humidity=Normal	4	100%
3	Outlook=Overcast	⇒Play=Yes	4	100%
4	Temperature=Cold Play=Yes	⇒Humidity=Normal	3	100%
...	...	...	...	...
58	Outlook=Sunny Temperature=Hot	⇒Humidity=High	2	100%

- In total: 3 rules with support four, 5 with support three, and 50 with support two

Filtering Association Rules

- Problem: any large dataset can lead to very large number of association rules, even with reasonable Min Confidence and Support
- Confidence by itself is not sufficient
  - e.g. if all transactions include Z, then
  - any rule I => Z will have confidence 100%.
- Other measures to filter rules

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 $L \rightarrow \varnothing$ and $\varnothing \rightarrow L$)

Rule Generation

- How to efficiently generate rules from frequent itemsets?
  - In general, confidence does not have an anti-monotone property

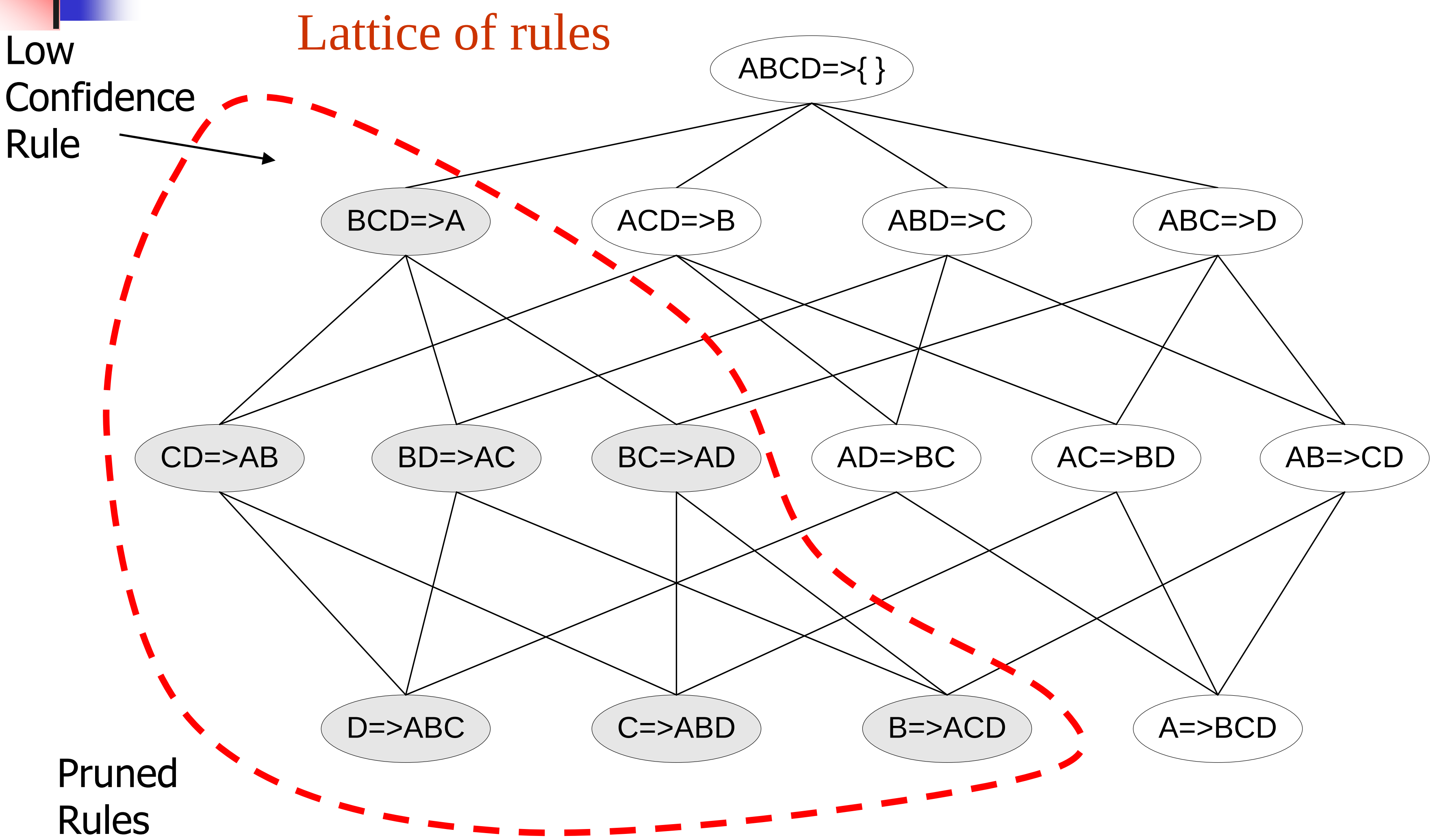
    $c(ABC \rightarrow D)$ can be larger or smaller than $c(AB \rightarrow D)$

  - 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 (num. is fix den. increases).

Rule Generation for Apriori Algorithm

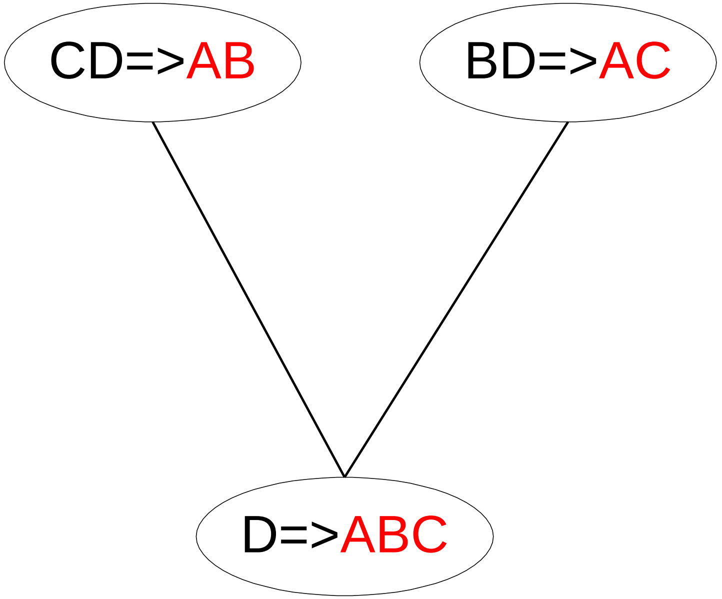


Rule Generation for Apriori Algorithm

- 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



Exercise

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

min-sup = 20%
min-conf = 80%

Exercise

min-sup = 20%
min-conf = 80%

major	*status*	*age*	*nationality*	*gpa*	*count*
French	M.A	over_30	Canada	2.8 . . . 3.2	3
cs	junior	16 . . . 20	Europe	3.2 . . . 3.6	29
physics	M.S	26 . . . 30	Latin_America	3.2 . . . 3.6	18
engineering	Ph.D	26 . . . 30	Asia	3.6 . . . 4.0	78
philosophy	Ph.D	26 . . . 30	Europe	3.2 . . . 3.6	5
French	senior	16 . . . 20	Canada	3.2 . . . 3.6	40
chemistry	junior	21 . . . 25	USA	3.6 . . . 4.0	25
cs	senior	16 . . . 20	Canada	3.2 . . . 3.6	70
philosophy	M.S	over_30	Canada	3.6 . . . 4.0	15
French	junior	16 . . . 20	USA	2.8 . . . 3.2	8
philosophy	junior	26 . . . 30	Canada	2.8 . . . 3.2	9
philosophy	M.S	26 . . . 30	Asia	3.2 . . . 3.6	9
French	junior	16 . . . 20	Canada	3.2 . . . 3.6	52
math	senior	16 . . . 20	USA	3.6 . . . 4.0	32
cs	junior	16 . . . 20	Canada	3.2 . . . 3.6	76
philosophy	Ph.D	26 . . . 30	Canada	3.6 . . . 4.0	14
philosophy	senior	26 . . . 30	Canada	2.8 . . . 3.2	19
French	Ph.D	over_30	Canada	2.8 . . . 3.2	1
engineering	junior	21 . . . 25	Europe	3.2 . . . 3.6	71
math	Ph.D	26 . . . 30	Latin_America	3.2 . . . 3.6	7
chemistry	junior	16 . . . 20	USA	3.6 . . . 4.0	46
engineering	junior	21 . . . 25	Canada	3.2 . . . 3.6	96
French	M.S	over_30	Latin_America	3.2 . . . 3.6	4
philosophy	junior	21 . . . 25	USA	2.8 . . . 3.2	8
math	junior	16 . . . 20	Canada	3.6 . . . 4.0	59