Data Mining:

Concepts and Techniques

(3rd ed.)

— Chapter 6 —

Jiawei Han, Micheline Kamber, and Jian Pei

University of Illinois at Urbana-Champaign &

Simon Fraser University

Chapter 6: Mining Frequent Patterns, Association and Correlations: Basic Concepts and Methods

- Basic Concepts
- Frequent Itemset Mining Methods
- Which Patterns Are Interesting?—Pattern Evaluation Methods
- Summary

What Is Frequent Pattern Analysis?

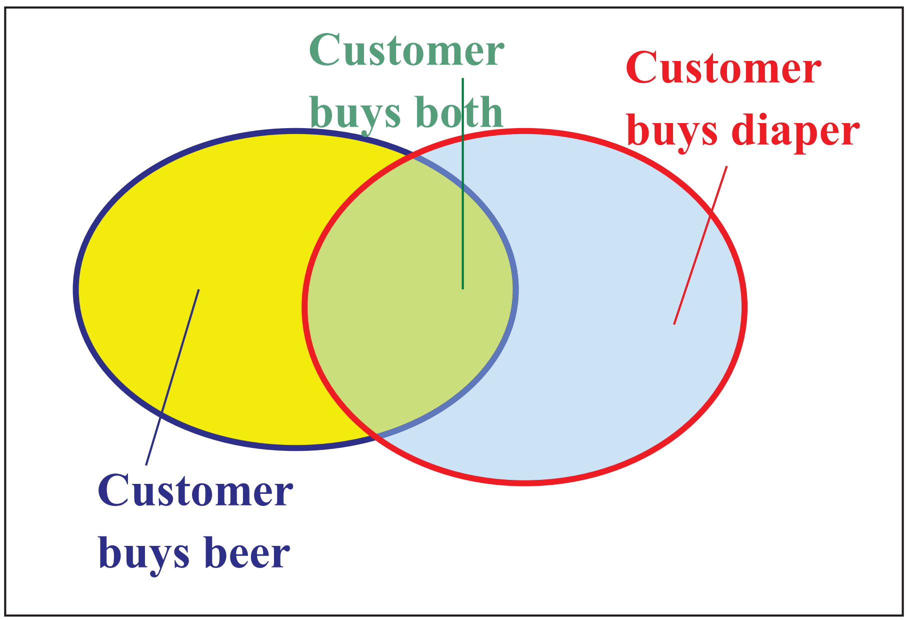
- Frequent pattern: a pattern (a set of items, subsequences, substructures, etc.) that occurs frequently in a data set
- First proposed by Agrawal, Imielinski, and Swami [AIS93] in the context of frequent itemsets and association rule mining
- Motivation: Finding inherent regularities in data
  - What products were often purchased together?— Beer and diapers?!
  - What are the subsequent purchases after buying a PC?
  - What kinds of DNA are sensitive to this new drug?
  - Can we automatically classify web documents?
- Applications
  - Basket data analysis, cross-marketing, catalog design, sale campaign analysis, Web log (click stream) analysis, and DNA sequence analysis.

Why Is Freq. Pattern Mining Important?

- Freq. pattern: An intrinsic and important property of datasets
- Foundation for many essential data mining tasks
  - Association, correlation, and causality analysis
  - Sequential, structural (e.g., sub-graph) patterns
  - Pattern analysis in spatiotemporal, multimedia, time-series, and stream data
  - Classification: discriminative, frequent pattern analysis
  - Cluster analysis: frequent pattern-based clustering
  - Data warehousing: iceberg cube and cube-gradient
  - Semantic data compression: fascicles

Basic Concepts: Frequent Patterns

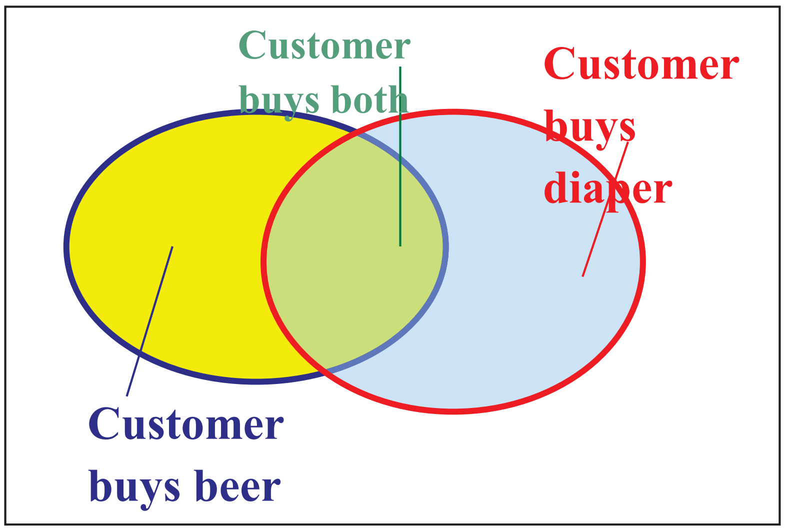
Tid	Items bought
10	Beer, Nuts, Diaper
20	Beer, Coffee, Diaper
30	Beer, Diaper, Eggs
40	Nuts, Eggs, Milk
50	Nuts, Coffee, Diaper, Eggs, Milk



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

Basic Concepts: Association Rules

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



- Find all the rules $X \rightarrow Y$ with minimum support and confidence
  - support, s, probability that a transaction contains $X \cup Y$
  - confidence, c, conditional probability that a transaction having X also contains Y

Let minsup = 50%, minconf = 50%

Freq. Pat.: Beer:3, Nuts:3, Diaper:4, Eggs:3, {Beer, Diaper}:3

- Association rules: (many more!)
  - *Beer* → *Diaper* (60%, 100%)
  - *Diaper* → *Beer* (60%, 75%)

Chapter 5: Mining Frequent Patterns, Association and Correlations: Basic Concepts and Methods

- Basic Concepts
- Frequent Itemset Mining Methods
- Which Patterns Are Interesting?—Pattern Evaluation Methods
- Summary

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} $\rightarrow$ {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)

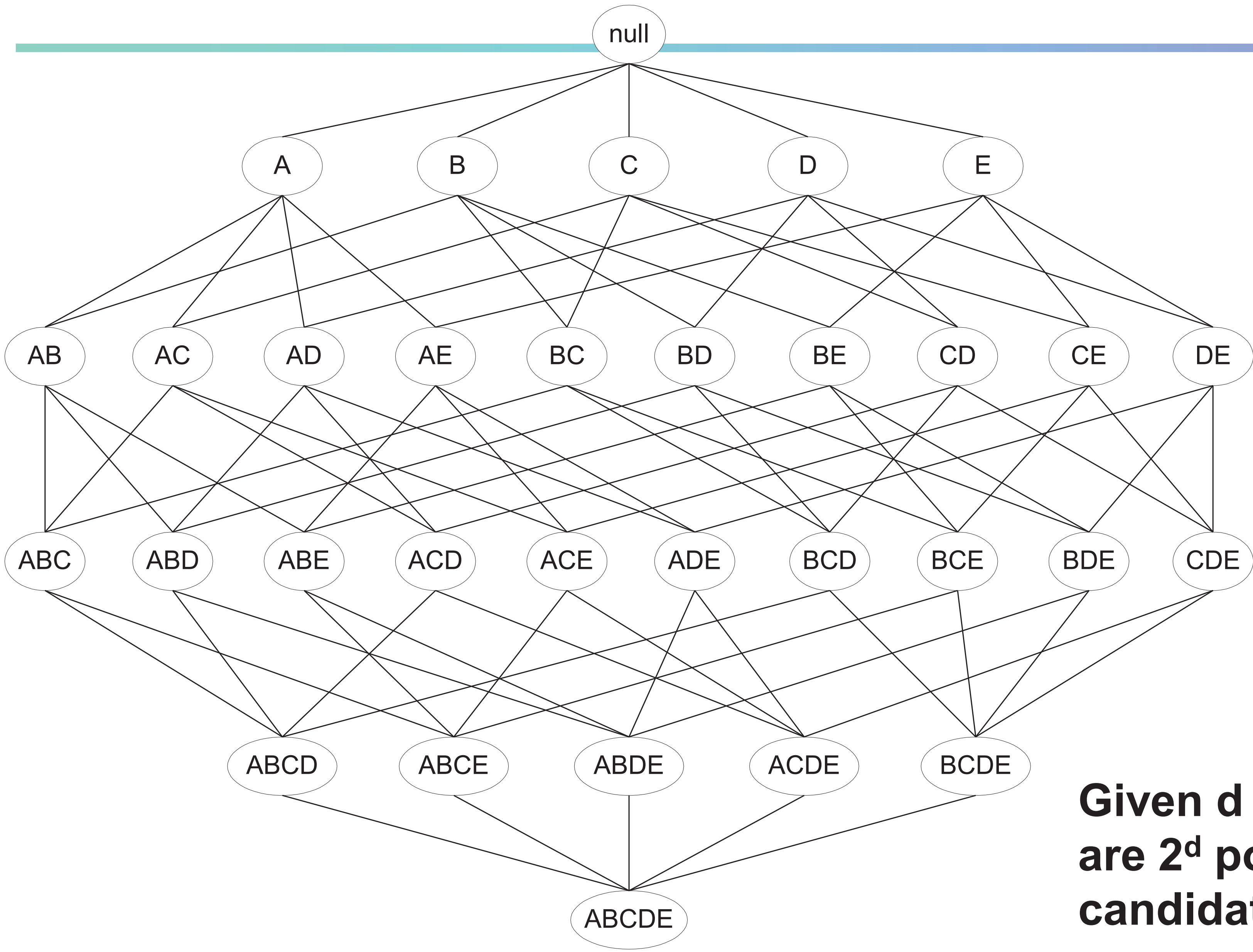
Observations:

- 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

Mining Association Rules

- Two-step approach:
  1. Frequent Itemset Generation
     - Generate all itemsets whose support ≥ *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

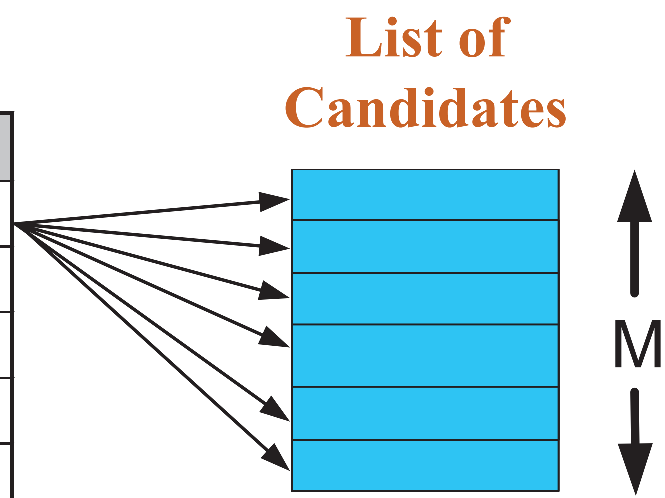
- Brute-force approach:
  - Each itemset in the lattice is a candidate frequent itemset
  - Count the support of each candidate by scanning the database

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

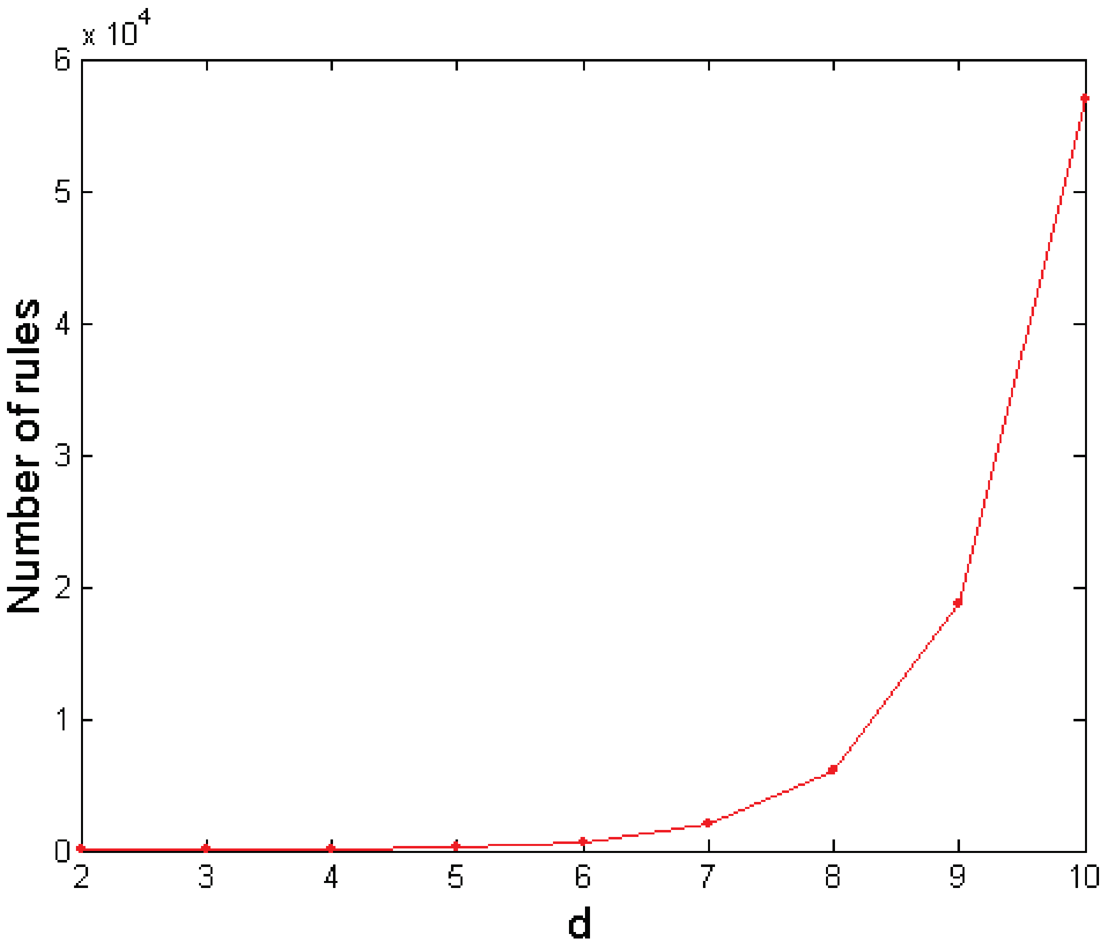
w



  - 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

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

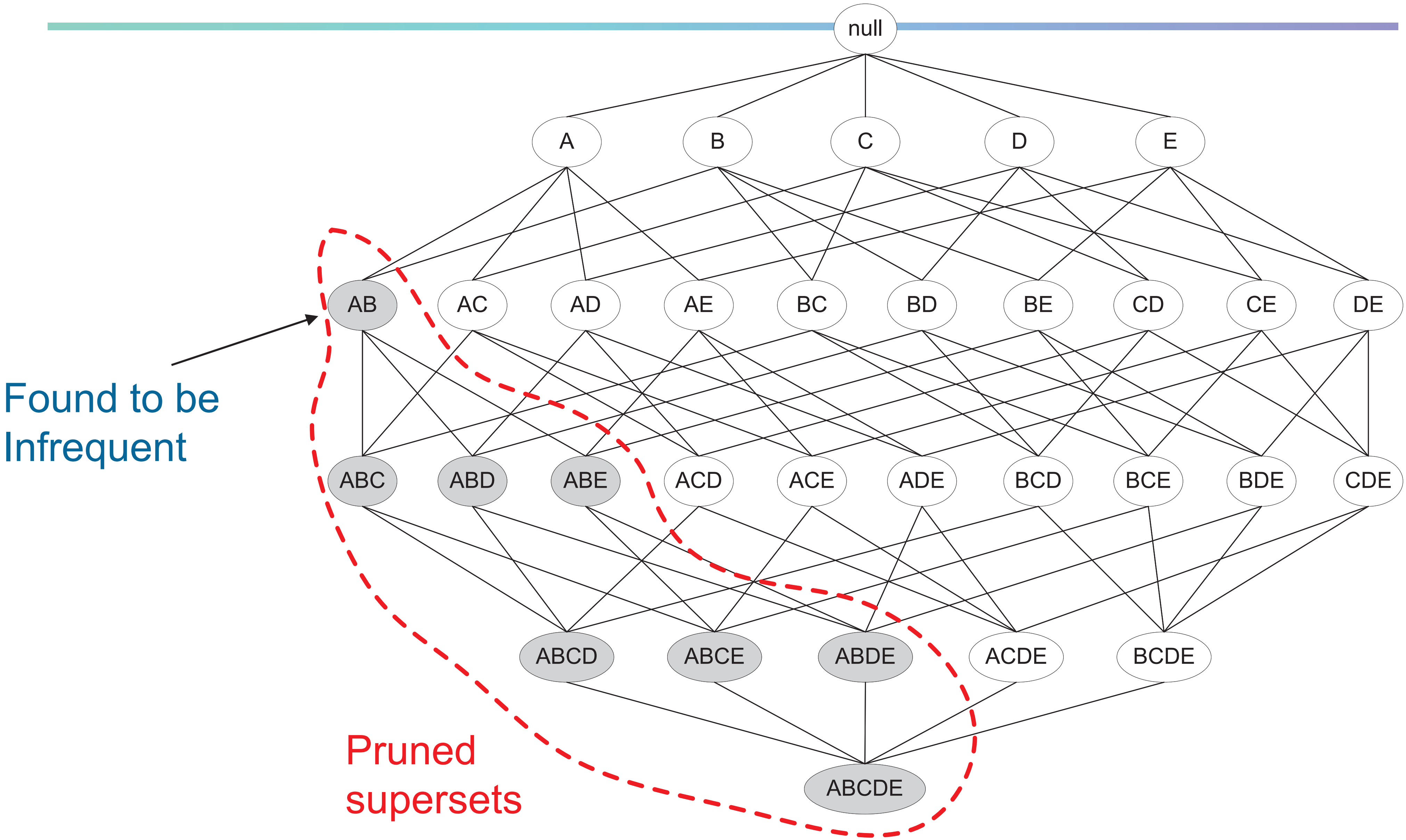
Scalable Frequent Itemset Mining Methods

- Apriori: A Candidate Generation-and-Test Approach
- Improving the Efficiency of Apriori
- FPGrowth: A Frequent Pattern-Growth Approach
- ECLAT: Frequent Pattern Mining with Vertical Data Format

The Downward Closure Property and Scalable Mining Methods

- Scalable mining methods: Three major approaches
  - Apriori (Agrawal & Srikant@VLDB'94)
  - Freq. pattern growth (FPgrowth—Han, Pei & Yin @SIGMOD'00)
  - Vertical data format approach (Charm—Zaki & Hsiao @SDM'02)
- The downward closure property of frequent patterns
  - Any subset of a frequent itemset must be frequent
  - If **{beer, diaper, nuts}** is frequent, so is **{beer, diaper}**
  - i.e., every transaction having {beer, diaper, nuts} also contains {beer, diaper}

Illustrating Apriori Principle



Apriori: A Candidate Generation & Test Approach

- Apriori pruning principle: If there is any itemset which is infrequent, its superset should not be generated/tested! (Agrawal & Srikant @VLDB'94, Mannila, et al. @ KDD' 94)
- Method:
  - Initially, scan DB once to get frequent 1-itemset
  - Generate length (k+1) candidate itemsets from length k frequent itemsets
  - Test the candidates against DB
  - Terminate when no frequent or candidate set can be generated

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

1st 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}

2nd 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}

3rd 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**
    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 *minsup*
    end
return $\cup_k L_k$;

Implementation of Apriori

- How to generate candidates?
  - Step 1: self-joining L_k
  - Step 2: pruning
- Example of Candidate-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
  - $C_4 = \{abcd\}$

Scalable Frequent Itemset Mining Methods

- Apriori: A Candidate Generation-and-Test Approach
- Improving the Efficiency of Apriori
- FPGrowth: A Frequent Pattern-Growth Approach
- ECLAT: Frequent Pattern Mining with Vertical Data Format
- Mining Close Frequent Patterns and Maxpatterns

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

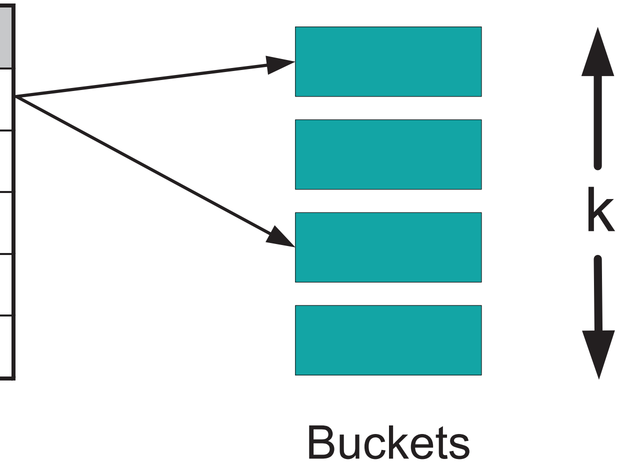
- Candidate counting:
  - Scan the database of transactions to determine the support of each candidate itemset
  - To reduce the number of comparisons, store the candidates in a hash structure
    - Instead of matching each transaction against every candidate, match it against candidates contained in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

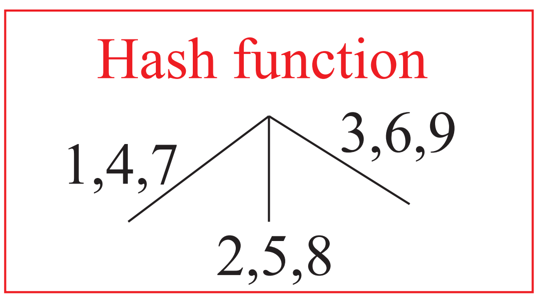
Generate Hash Tree

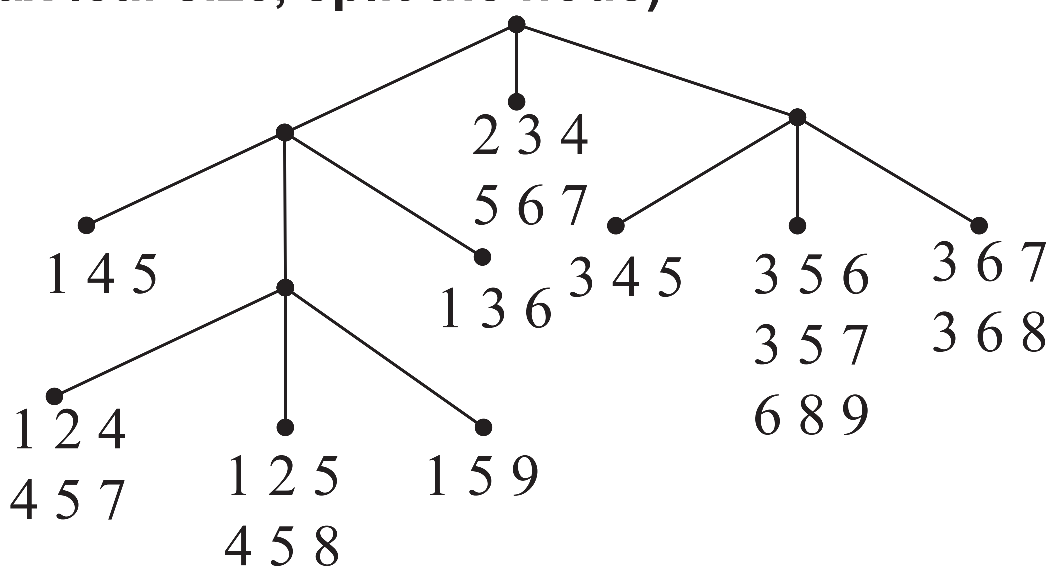
Suppose you have 15 candidate itemsets of length 3:

{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:

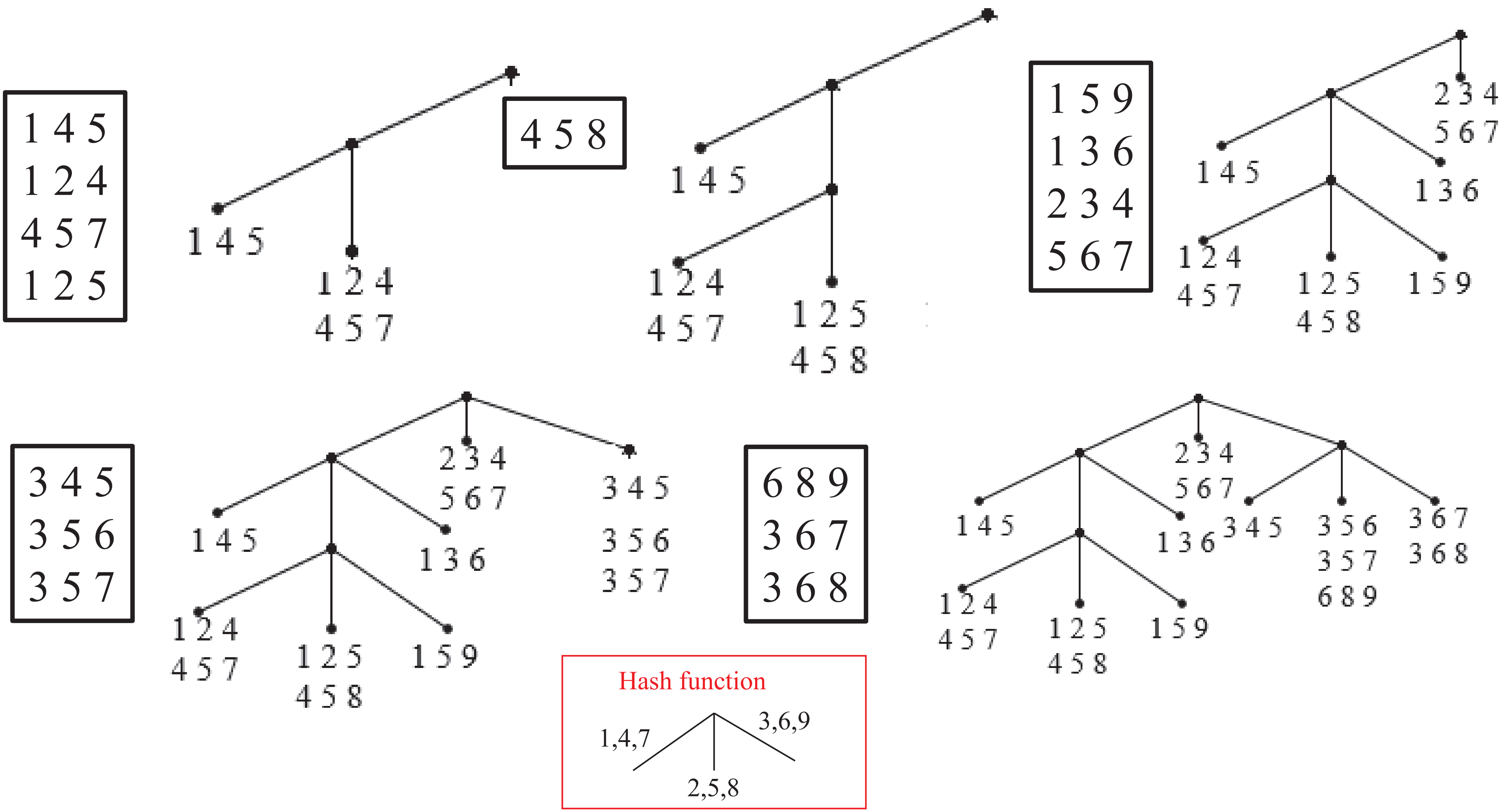
- **Hash function**
- **Max leaf size: max number of itemsets stored in a leaf node (if number of candidate itemsets exceeds max leaf size, split the node)**



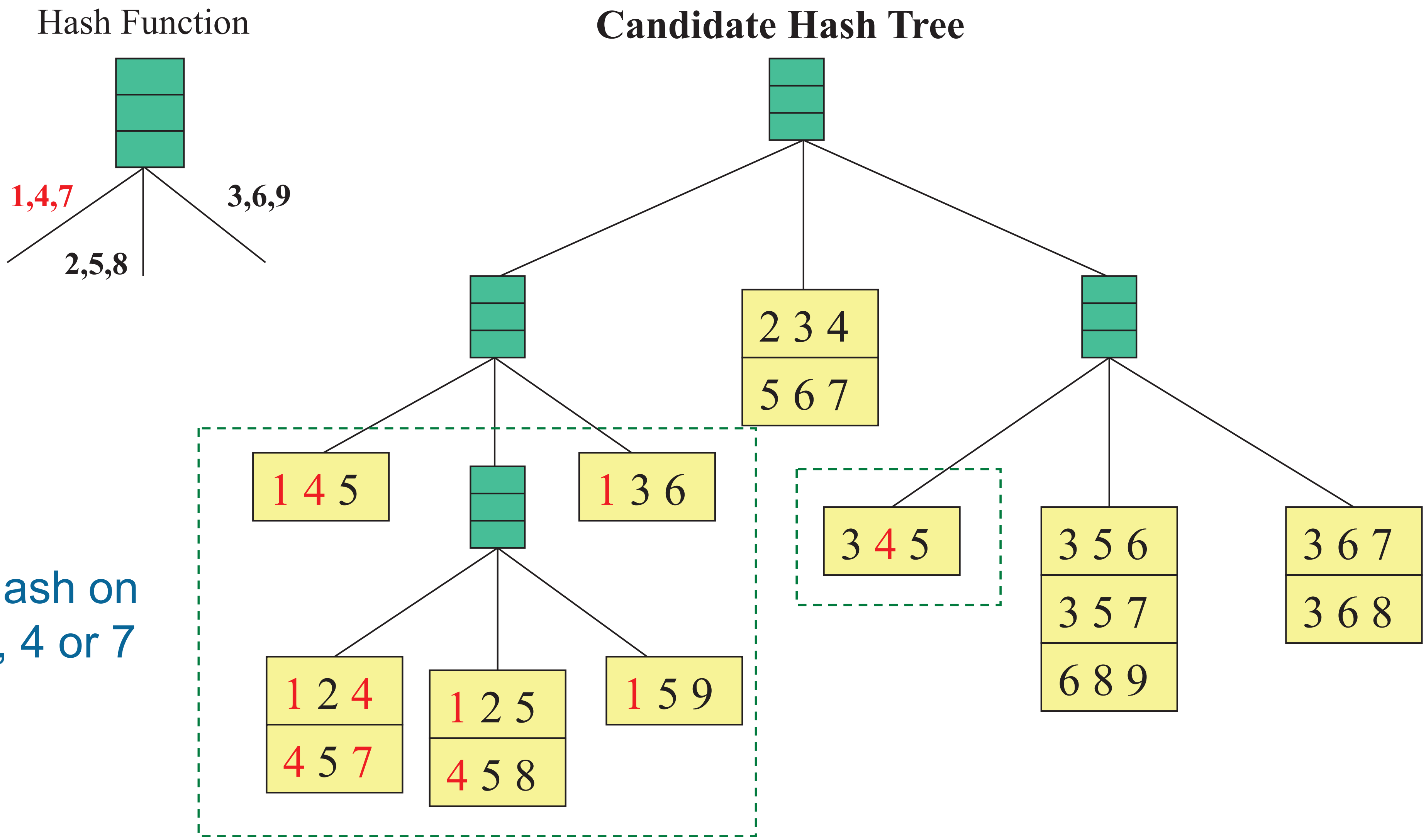


Generate Hash Tree

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

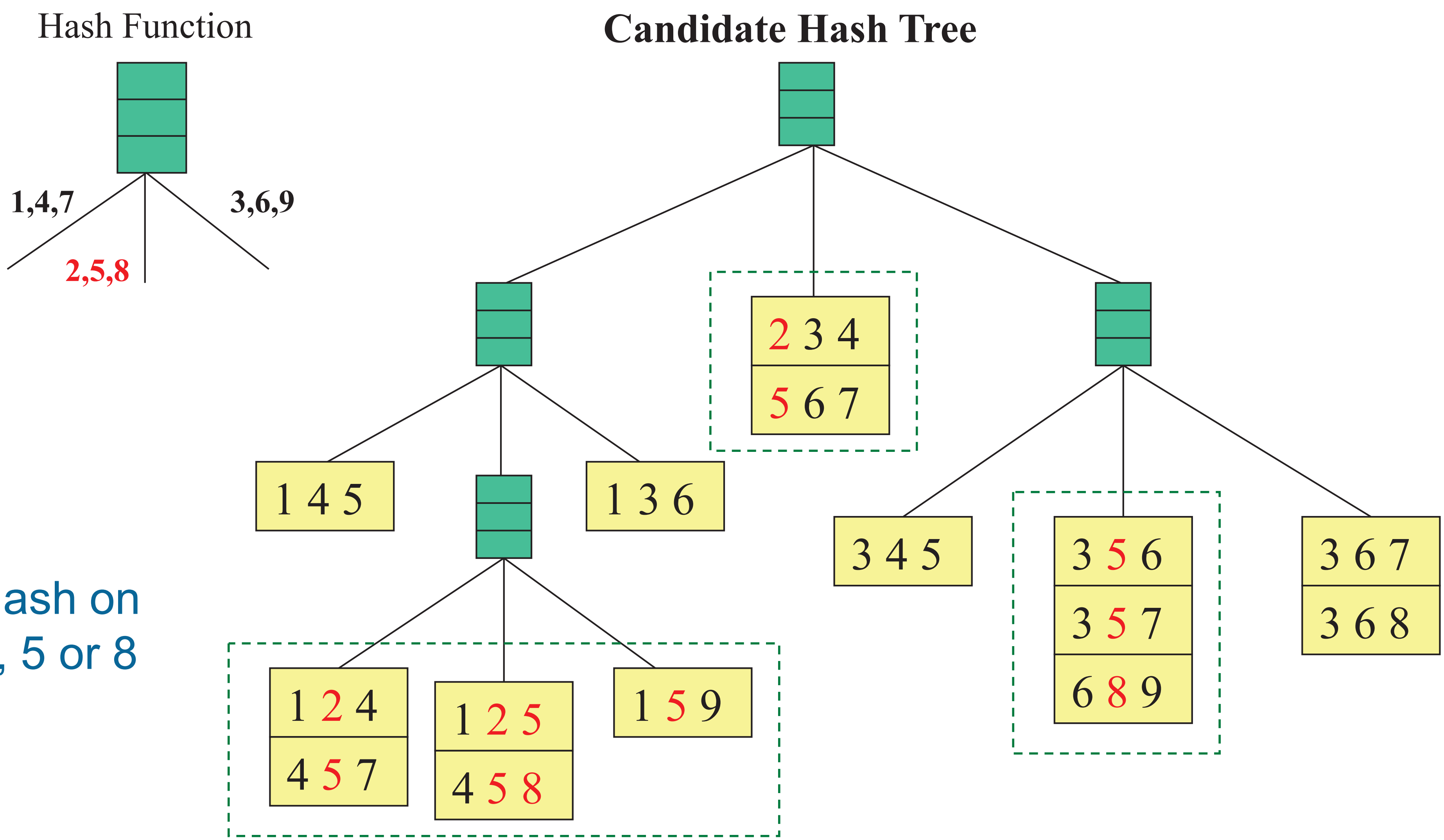


Association Rule Discovery: Hash tree



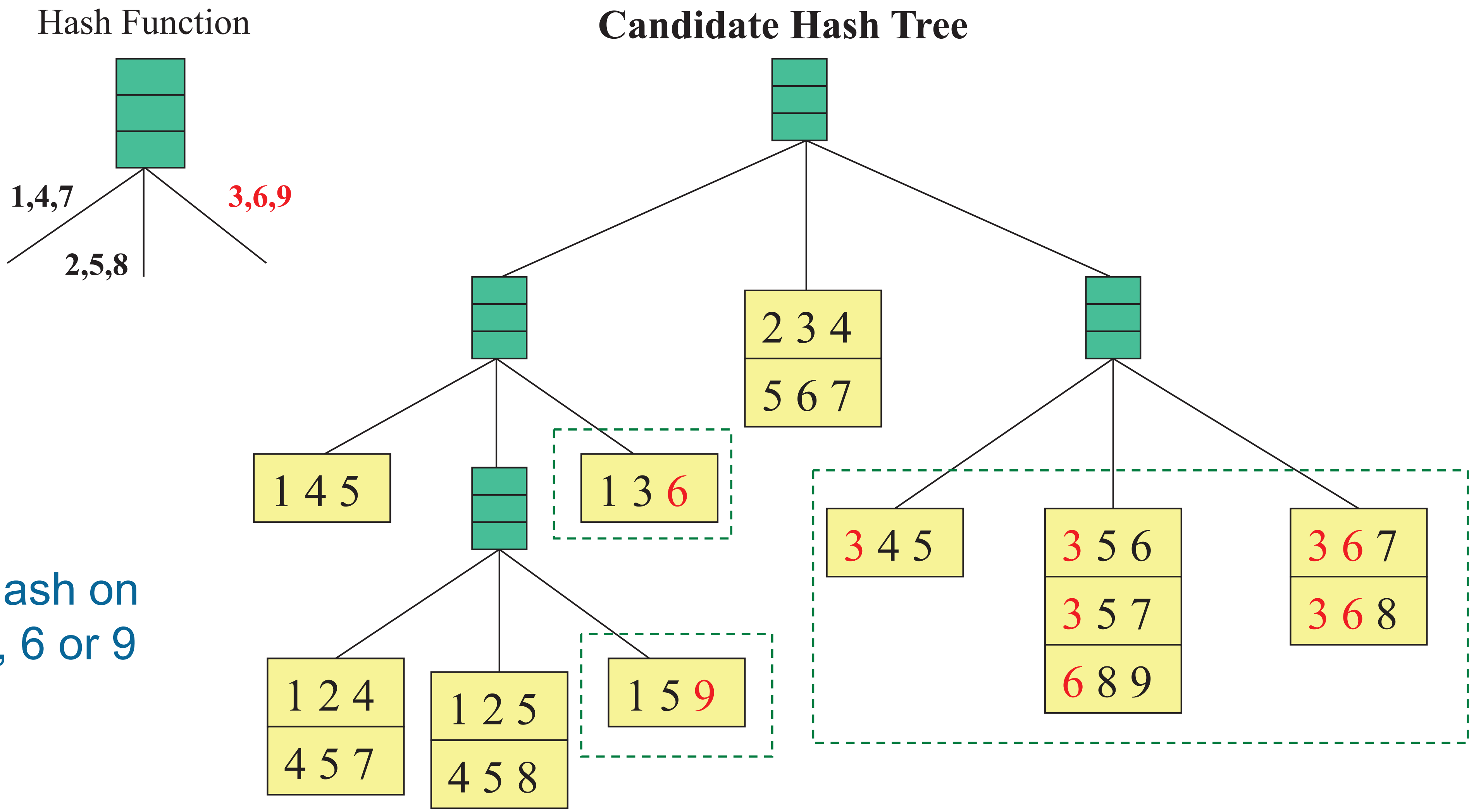
Hash on
1, 4 or 7

Association Rule Discovery: Hash tree



Hash on
2, 5 or 8

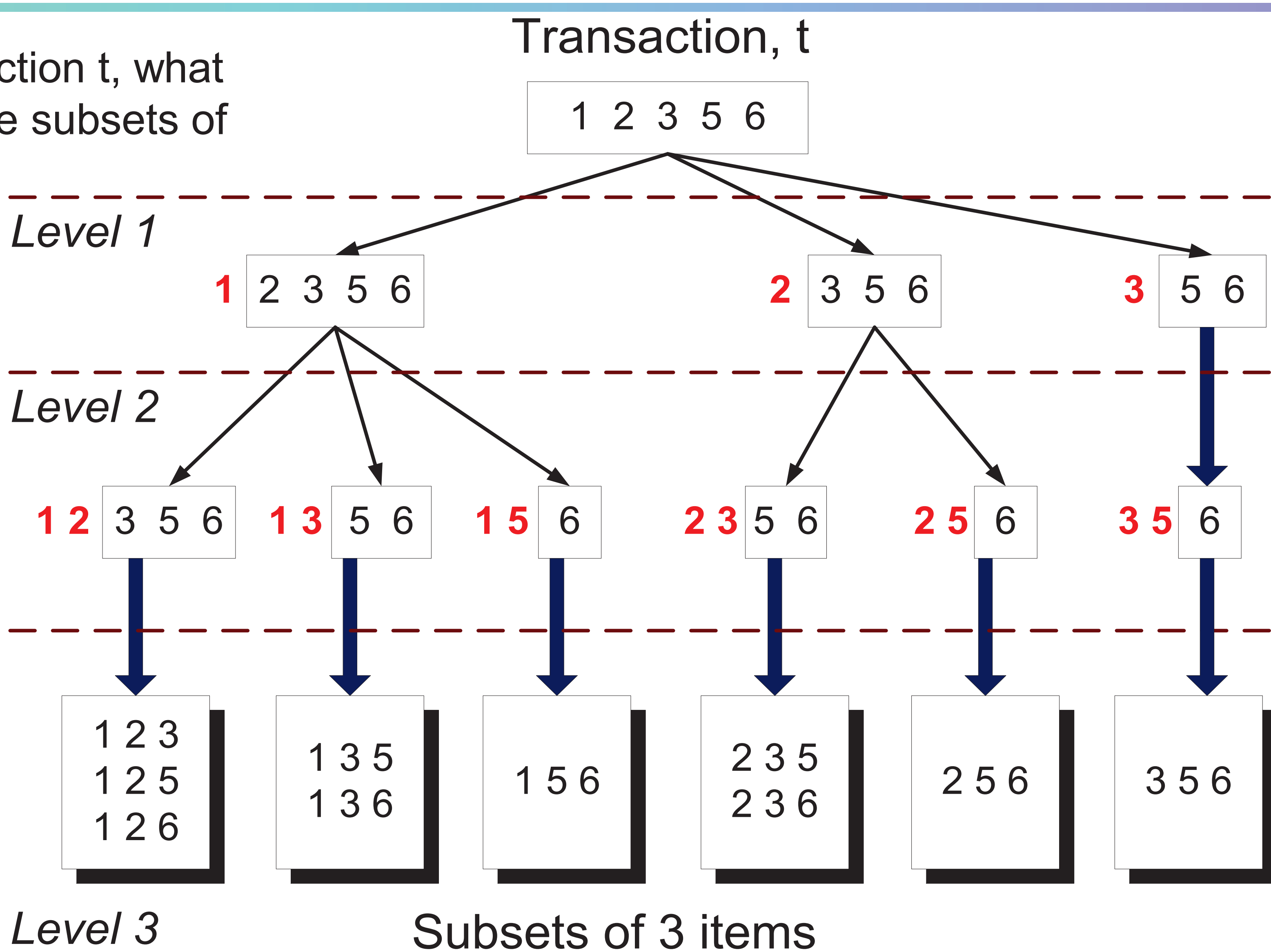
Association Rule Discovery: Hash tree



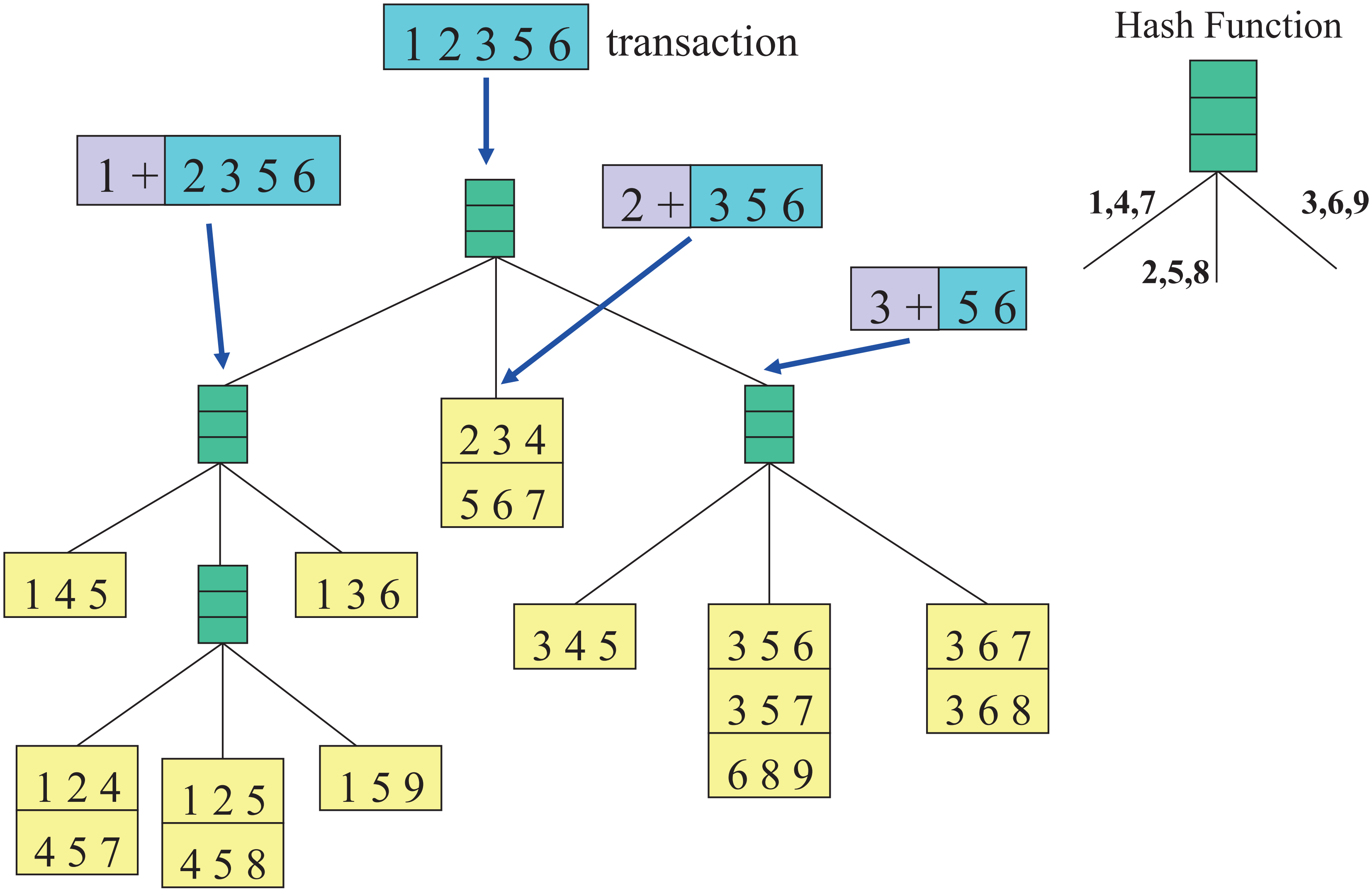
Hash on
3, 6 or 9

Subset Operation

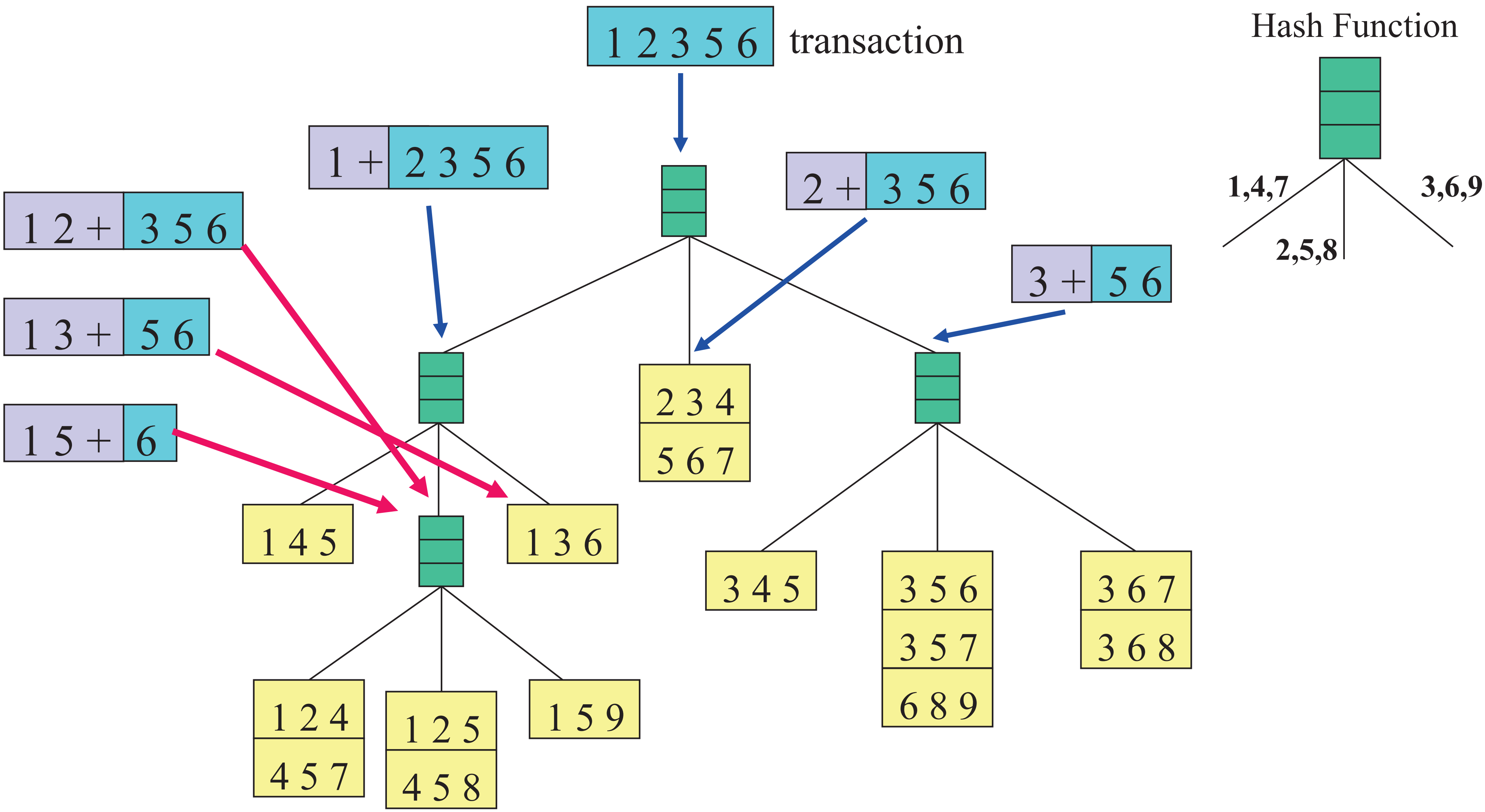
Given a transaction t, what are the possible subsets of size 3?



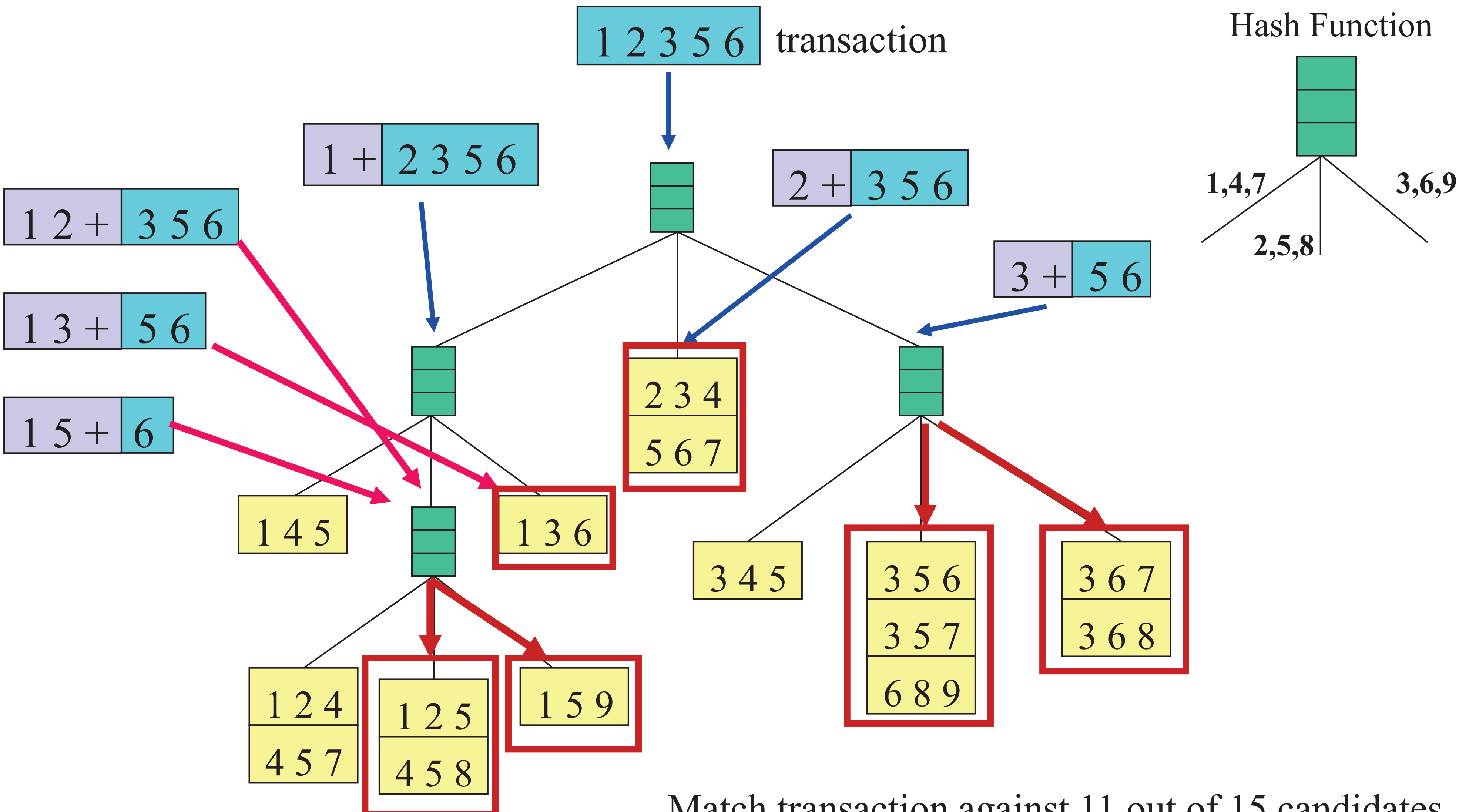
Subset Operation Using Hash Tree



Subset Operation Using Hash Tree



Subset Operation Using Hash Tree



Match transaction against 11 out of 15 candidates

Improving the Efficiency of Apriori

- Other Methods (**Projects for Students**)
  - Partition: Scan Database Only Twice
    - A. Savasere, E. Omiecinski and S. Navathe, *VLDB'95*
  - DHP: Reduce the Number of Candidates
    - DHP: Direct Hashing and Pruning
    - J. Park, M. Chen, and P. Yu. An effective hash-based algorithm for mining association rules. *SIGMOD'95*
  - DIC: Reduce Number of Scans
    - DIC: Dynamic itemset counting
    - H. Toivonen. Sampling large databases for association rules. In *VLDB'96*

Rule Generation from Frequent Itemsets

- Strong association rules→ *minsup* and *minconf*
- $Conf.(A \Longrightarrow B) = P(B|A) = \frac{support\ (A \cup B)}{Support\ (A)}$
- Association rules can be generated
  - For each frequent itemset l, generate all nonempty subsets of l
  - For every nonempty subset s, output rule "$s \Longrightarrow (l - s)$ if $\frac{support\ (l)}{Support\ (s)} \geq minconf$
- Example
  - If {A,B,C,D} is a frequent itemset, candidate rules:
  - ABC →D, ABD →C, ACD →B, BCD →A, A →BCD, B →ACD, C →ABD, D →ABC, AB →CD, AC → BD, AD → BC, BC →AD, BD →AC, CD →AB
  - | l | = n → n^2 – 2 candidate association rules (ignoring L → Ø and Ø → L) ?

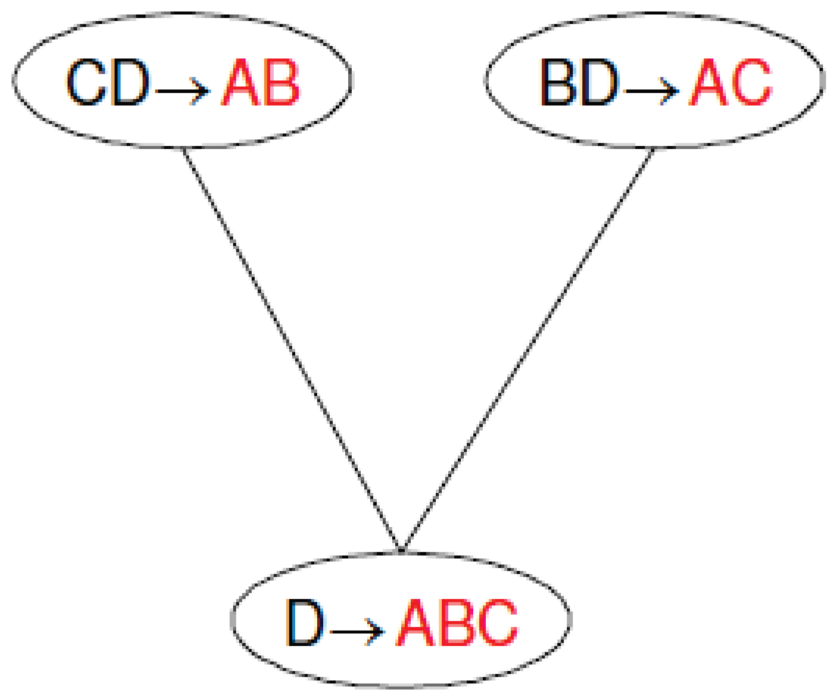
Rule Generation from Frequent Itemsets

- How to efficiently generate rules from frequent itemsets?
  - In general, confidence does not have an antimonotone property
    - *conf*(ABC $\rightarrow$D) can be larger or smaller than *conf*(AB $\rightarrow$D)
  - But confidence of rules generated from the same itemset has an anti-monotone property
    - e.g., L = {A,B,C,D}:
    - *conf*(ABC $\rightarrow$ D) $\geq$ *conf*(AB $\rightarrow$ CD) $\geq$ *conf*(A $\rightarrow$ BCD)

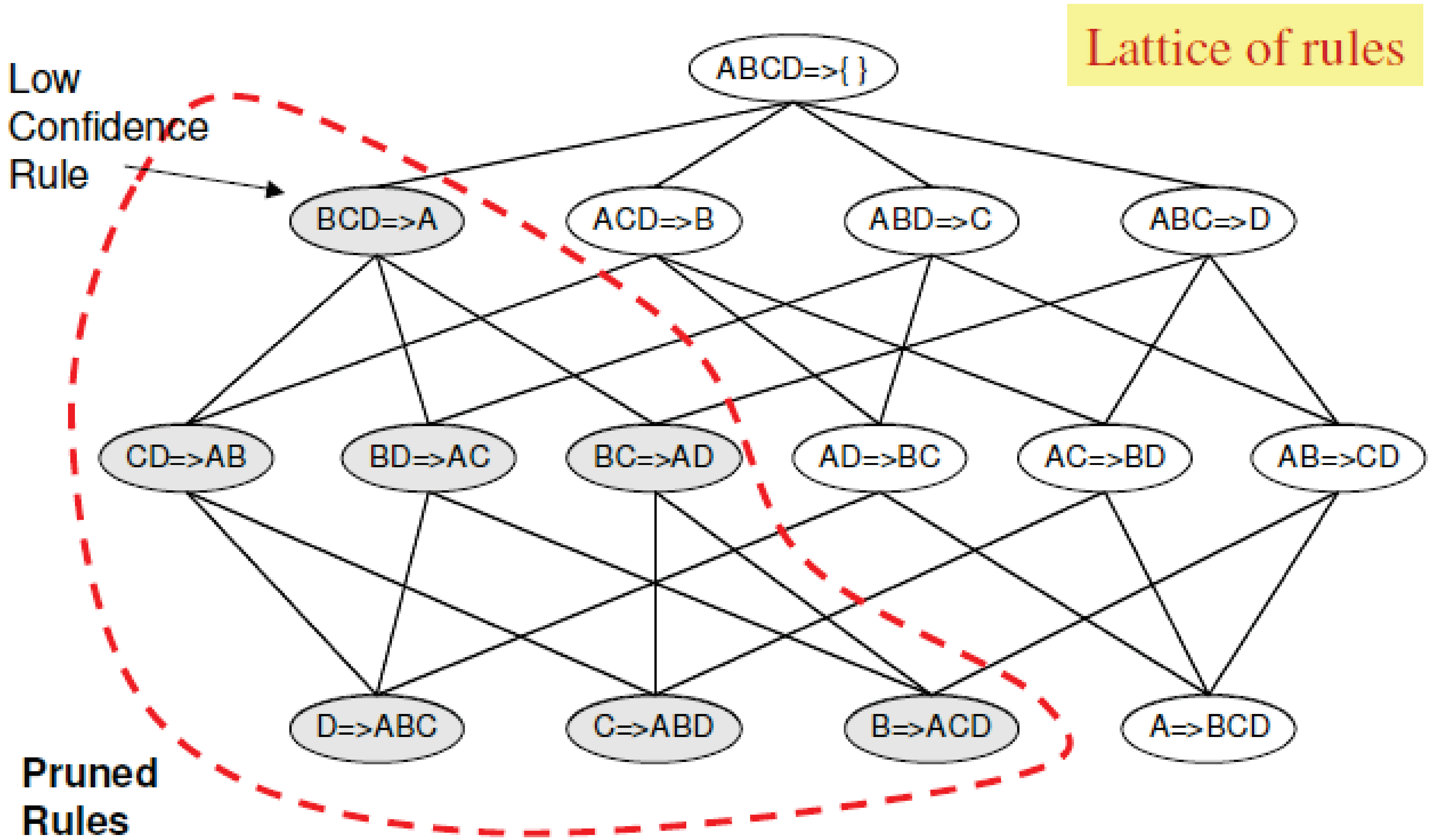
Rule Generation

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

- 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



Rule Pruning



Rule Generation Algorithm

Moving items from the antecedent to the consequent never changes support, and never increases confidence

Homework #1

Dead time: 96/2/09

Email: Vahidipour@kashanu.ac.ir

Scalable Frequent Itemset Mining Methods

- Projects for Students
  - FPGrowth: A Frequent Pattern-Growth Approach
  - ECLAT: Frequent Pattern Mining with Vertical Data Format
  - Mining Close Frequent Patterns and Maxpatterns