COSC 3337 : Data Science I

N. Rizk

College of Natural and Applied Sciences

Department of Computer Science

University of Houston

Speeding up Association rules

Merkle tree

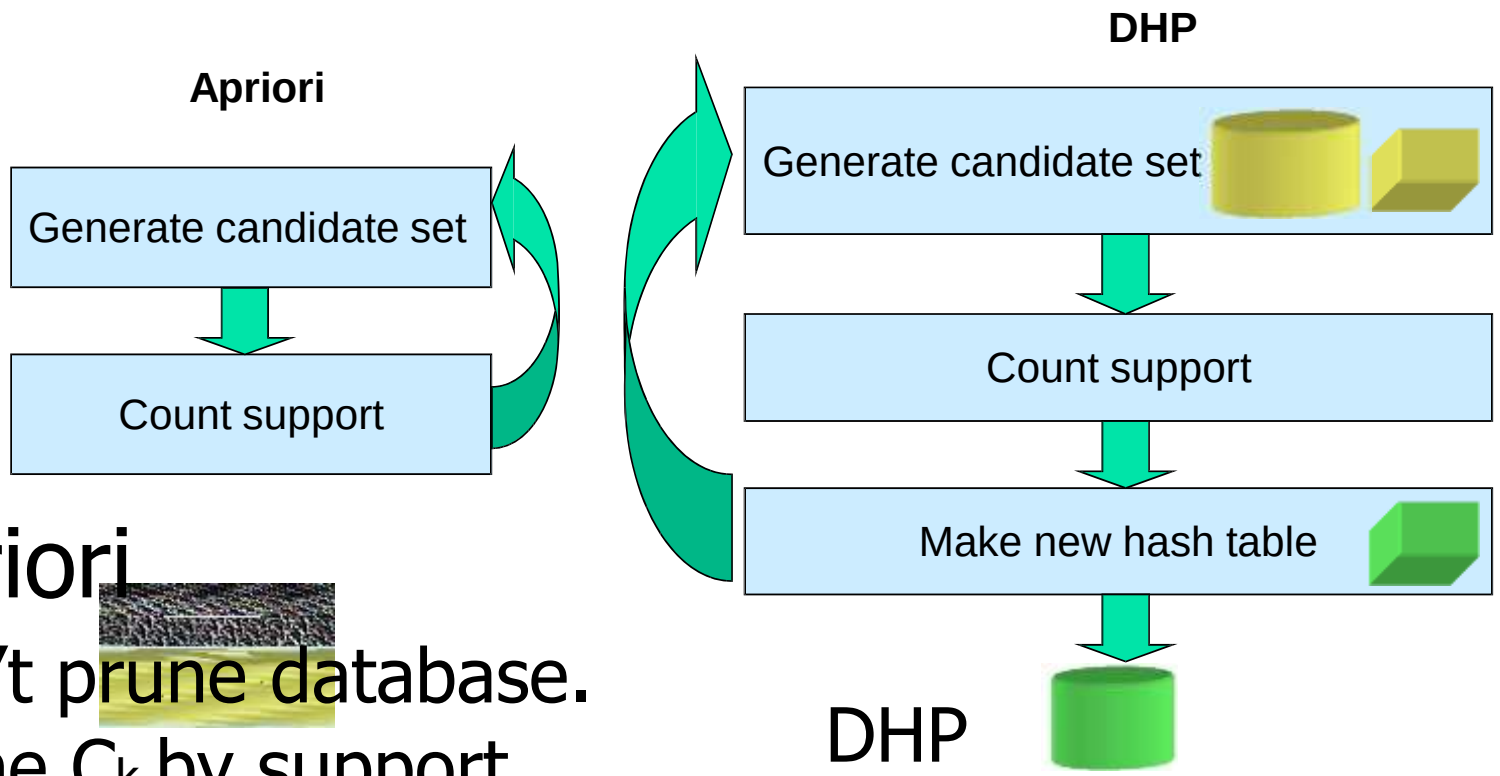
Dynamic Hashing and Pruning technique

DHP: Reduce the Number of Candidates

- A *k*-itemset whose corresponding hashing bucket count is below the threshold cannot be frequent
  - Candidates: a, b, c, d, e
  - Hash entries: {ab, ad, ae} {bd, be, de} …
  - Frequent 1-itemset: a, b, d, e
  - ab is not a candidate 2-itemset if the sum of count of {ab, ad, ae} is below support threshold
- DHP (Park '95): Dynamic Hashing and Pruning

- Candidate large 2-itemsets are huge.
  - DHP: trim them using hashing
- Transaction database is huge that one scan per iteration is costly
  - DHP: prune both number of transactions and number of items in each transaction after each iteration

How does it look like?



Apriori

Don't prune database.

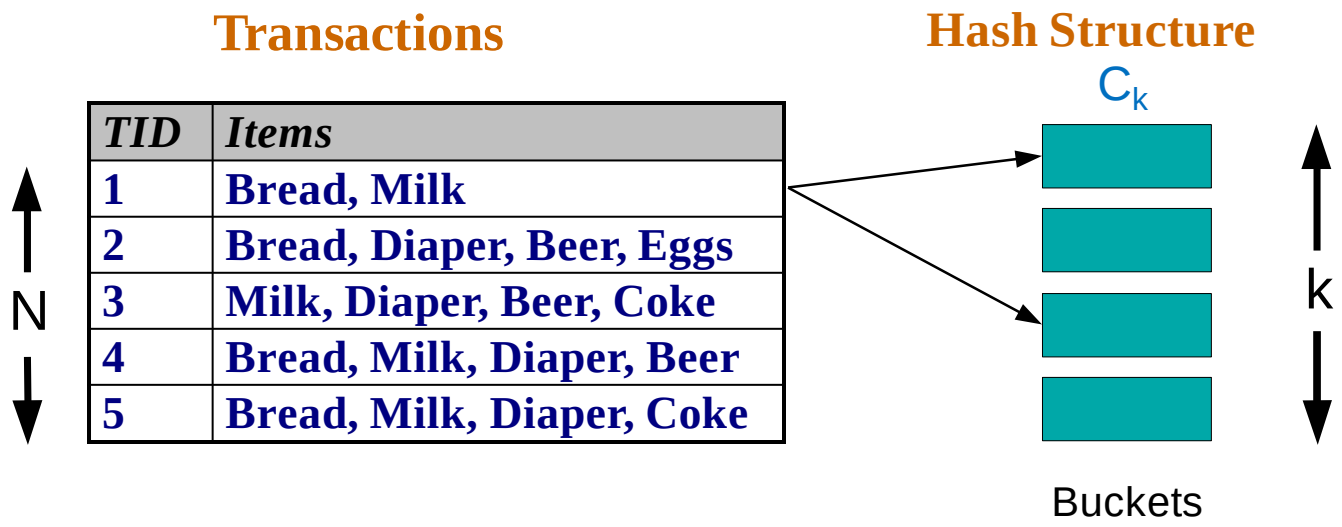
Prune C_k by support counting on the original database.

DHP

More efficient support counting can be achieved on pruned database

Computing Frequent Itemsets

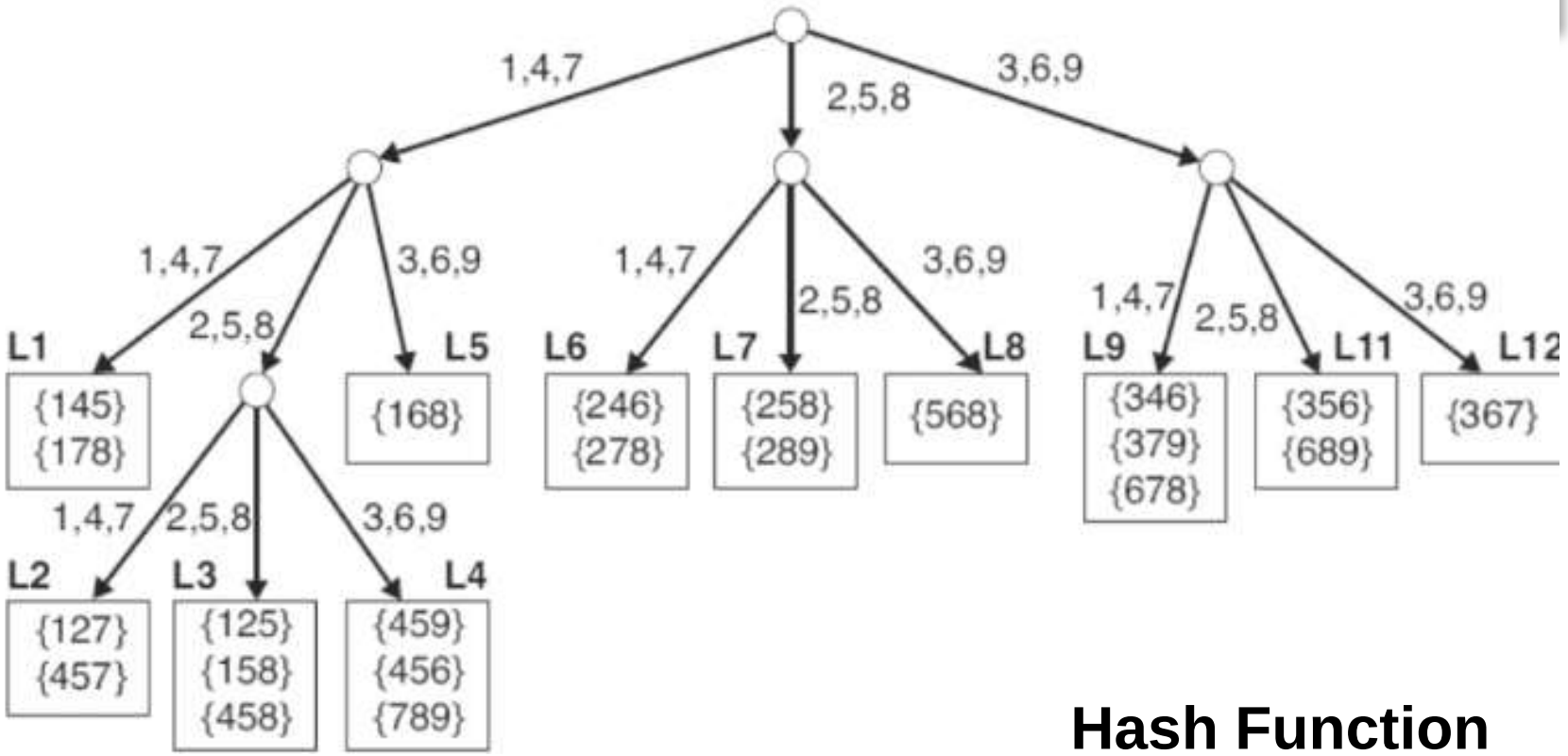
- Given the set of candidate itemsets C_k, we need to compute the support and find the frequent itemsets L_k.
- Scan the data, and use a hash structure to keep a counter for each candidate itemset that appears in the data



How to Count Supports of Candidates?

- Why is counting supports of candidates a problem?
  - The total number of candidates can be very huge
  - One transaction may contain many candidates
- Method:
  - Candidate itemset C_k is stored in a *hash-tree.*
  - *Leaf node* of hash-tree contains a list of itemsets and counts.
  - *Interior node* contains a hash table keyed by items (i.e., an item hashes to a bucket) and each bucket points to a child node at next level.
  - *Subset function*: finds all the candidates contained in a transaction.
  - Increment count per candidate and return frequent ones.

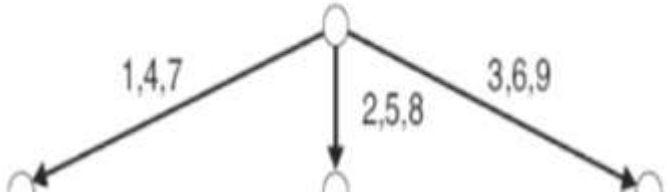
Candidate itemsets are stored in a hash-tree



Leaf node of hash-tree contains a list of itemsets and counts (max 3 rows)

Interior node contains a hash table of candidates

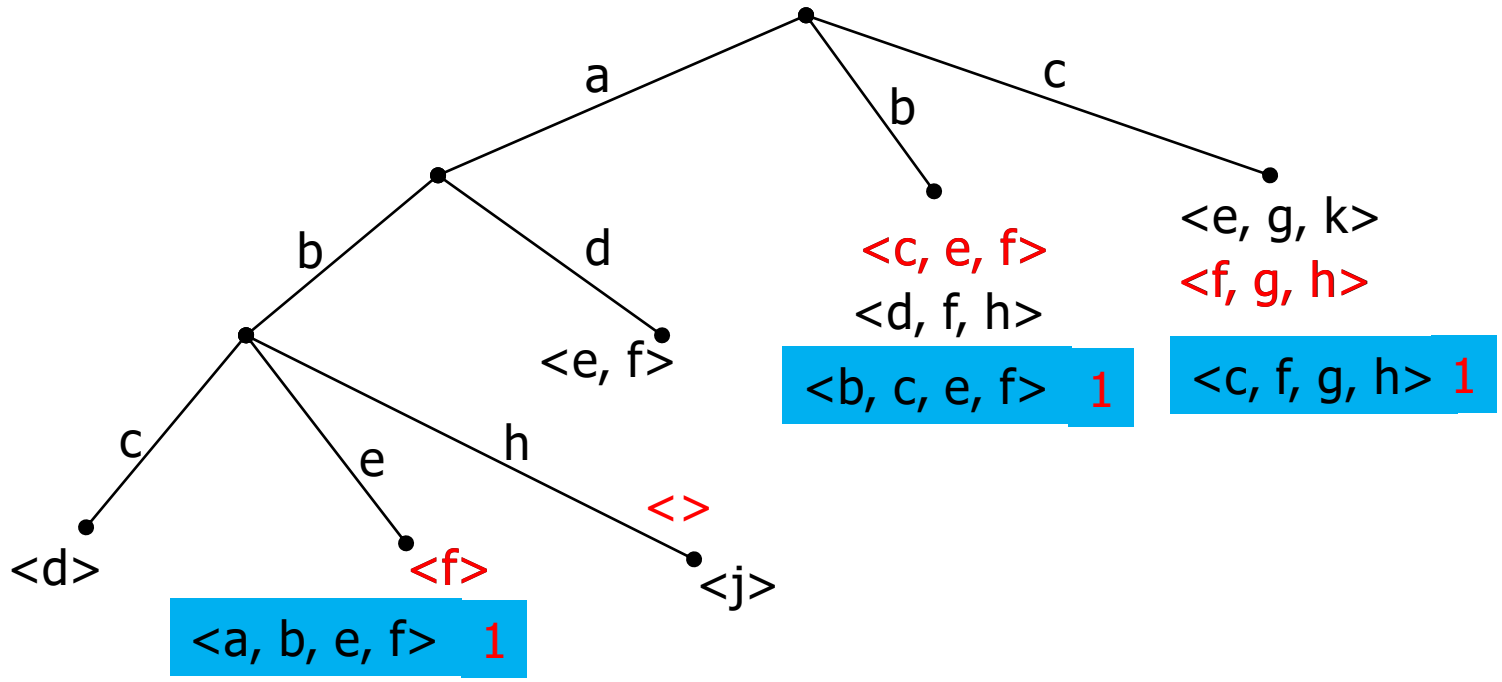
Hash Function



H(x) = 1%3 =1 ➔ 1 left side
H(x) = 5 % 3 =2➔ 2 middle side
H(x) =9%3=0➔0 right side

How to Use a Hash-Tree for C_k to Count Support

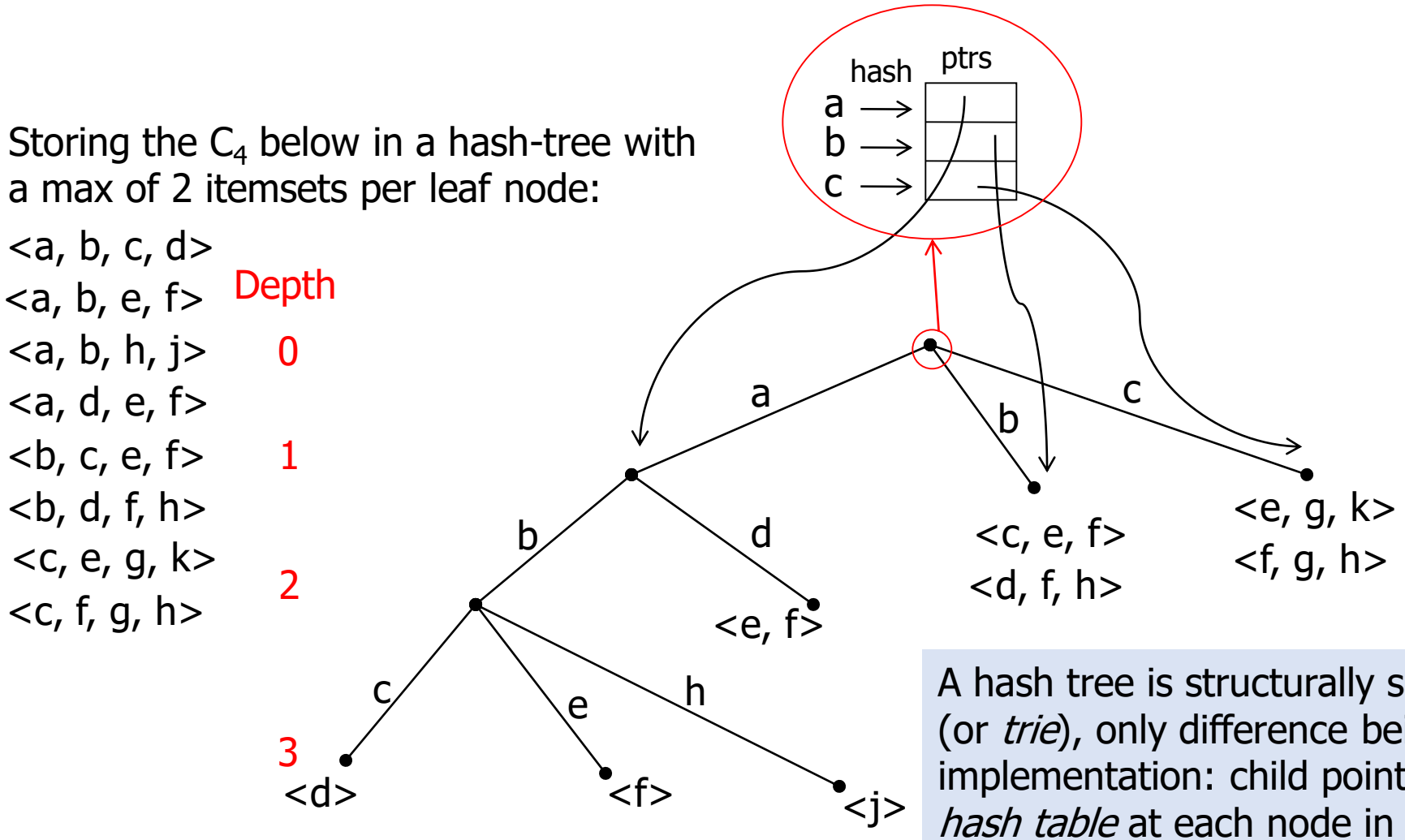
C_4 → L4



Example: Using a Hash-Tree for C_k to Count Support

Storing the C_4 below in a hash-tree with a max of 2 itemsets per leaf node:

<a, b, c, d>
<a, b, e, f>
<a, b, h, j>
<a, d, e, f>
<b, c, e, f>
<b, d, f, h>
<c, e, g, k>
<c, f, g, h>

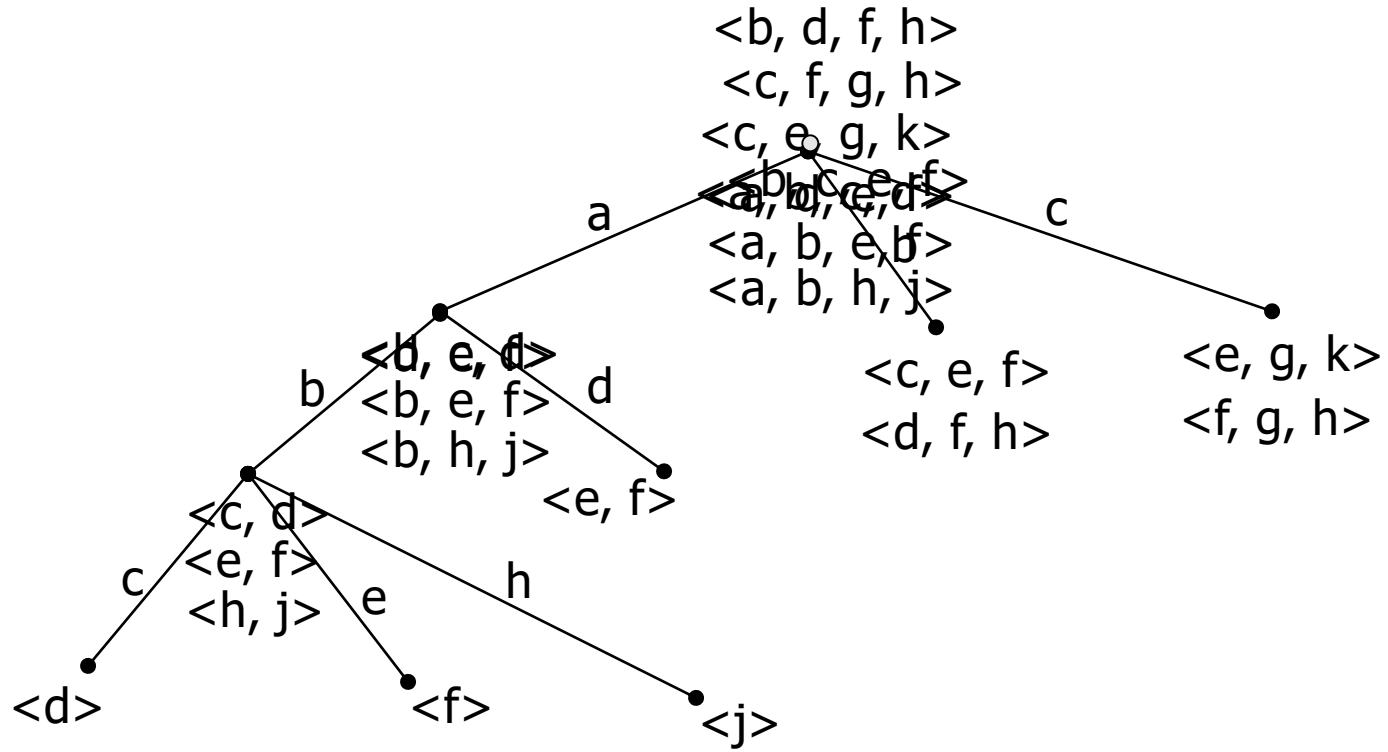


A hash tree is structurally same as a *prefix tree* (or *trie*), only difference being in the implementation: child pointers are stored in a *hash table* at each node in a hash tree vs. a list or array, because of the large and varying numbers of children.

How to Build a Hash Tree on a Candidate Set

Example: Building the hash tree on the candidate set C_4 of the previous slide (*max 2 itemsets per leaf node*)

<a, b, c, d>
<a, b, e, f>
<a, b, h, j>
<a, d, e, f>
<b, c, e, f>
<b, d, f, h>
<c, e, g, k>
<c, f, g, h>



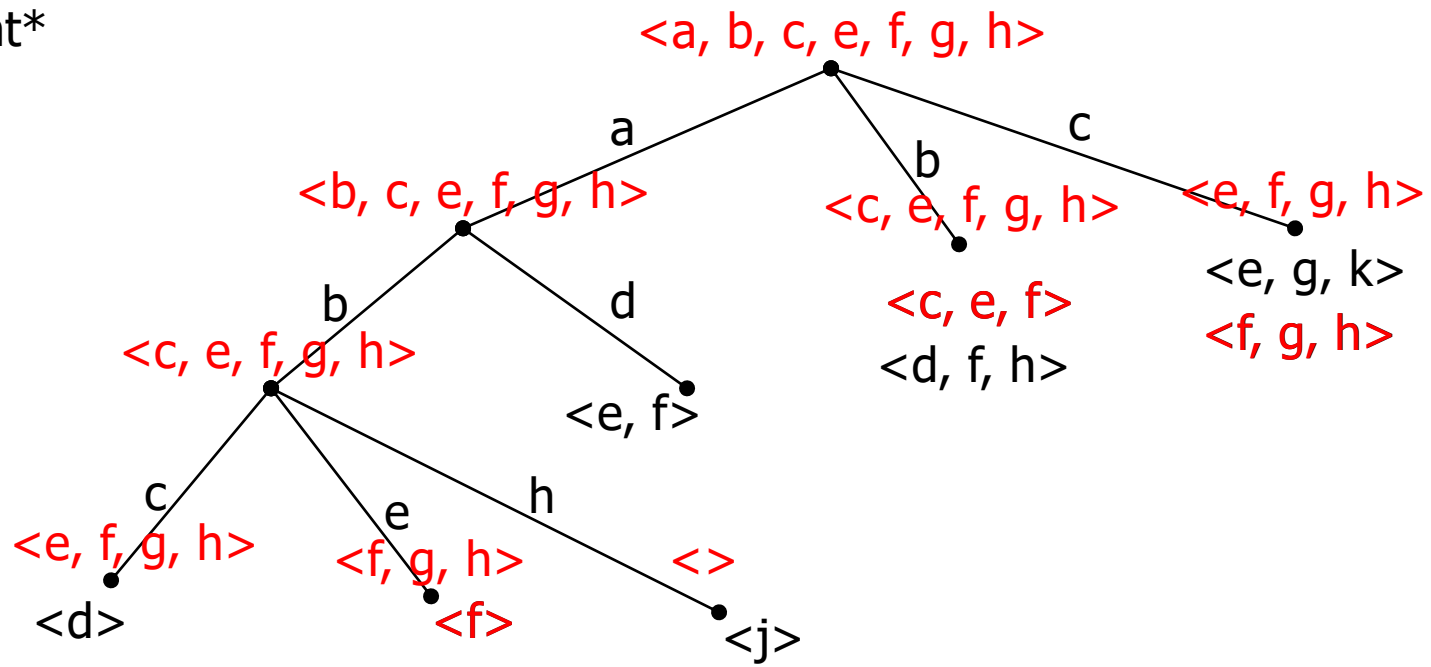
Ex: Find the candidates in C_4 contained in the transaction <a, b, c, e, f, g, h>…

How to Use a Hash-Tree for C_k to Count Support

For each transaction *T*, process *T* through the hash tree to find members of C_k contained in *T* and increment their count. After all transactions are processed, eliminate those candidates with less than min support.

Example: Find candidates in C_4 contained in *T* = <a, b, c, e, f, g, h>

C_4	Count*
<a, b, c, d>	0
<a, b, e, f>	1
<a, b, h, j>	0
<a, d, e, f>	0
<b, c, e, f>	1
<b, d, f, h>	0
<c, e, g, k>	0
<c, f, g, h>	1



Describe a general algorithm find candidates contained in a transaction.

Hint: Recursive…

*Counts are actually stored with the itemsets at the leaves. We show them in a separate table here for convenience.

A simple hashstructure

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

- First let us build the HASH TREE

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

The Hash Tree Structure

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

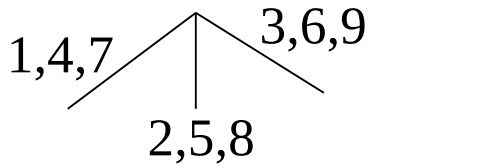
{1 4 5}, {1 2 4}, {4 5 7}, {1 2 5}, {4 5 8}, {1 5 9}, {1 3 6}, {2 3 4},

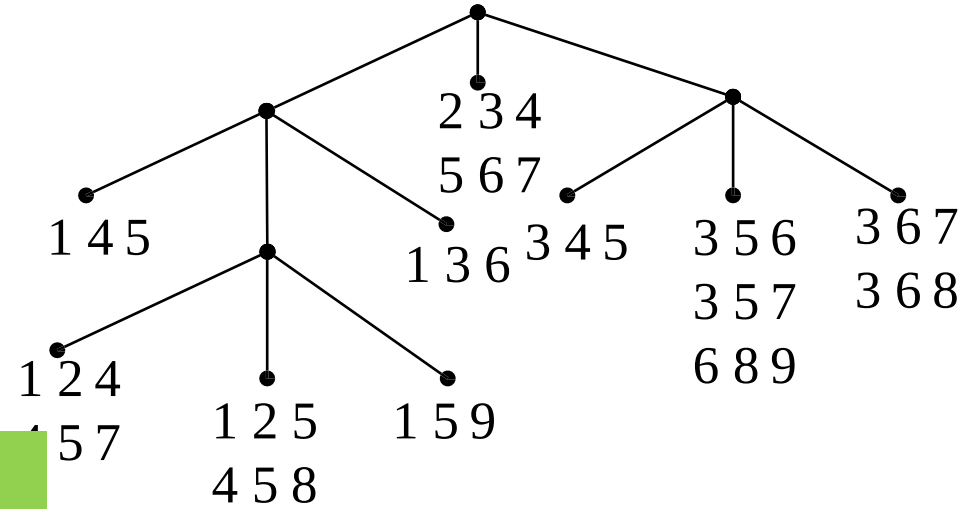
{5 6 7}, {3 4 5}, {3 5 6}, {3 5 7}, {6 8 9}, {3 6 7}, {3 6 8}

You need:

- Hash function
- Leafs: Store the itemsets

Hash function = x mod 3

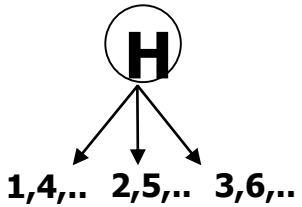




At the i-th level we hash on the i-th item

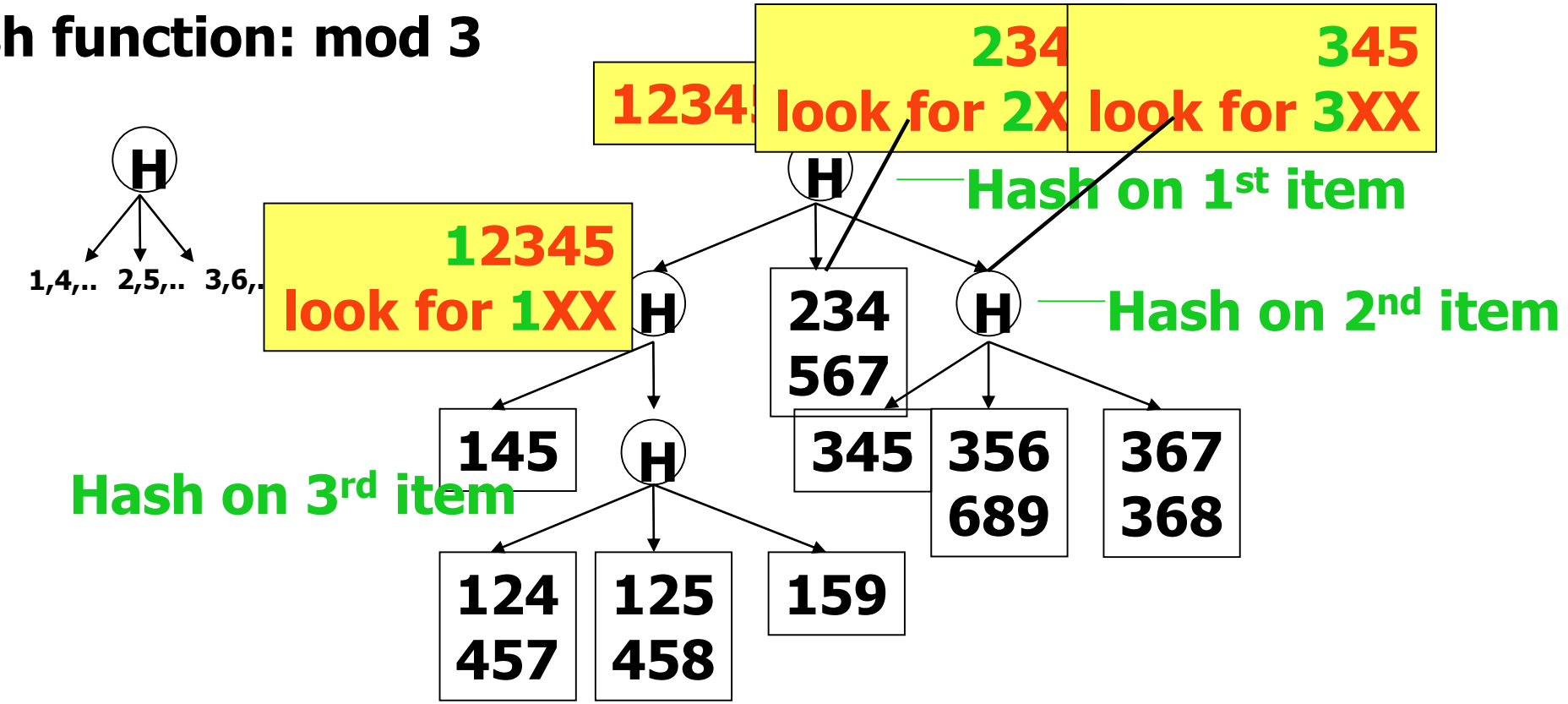
Example of the hash-tree for C_3

Hash function: mod 3

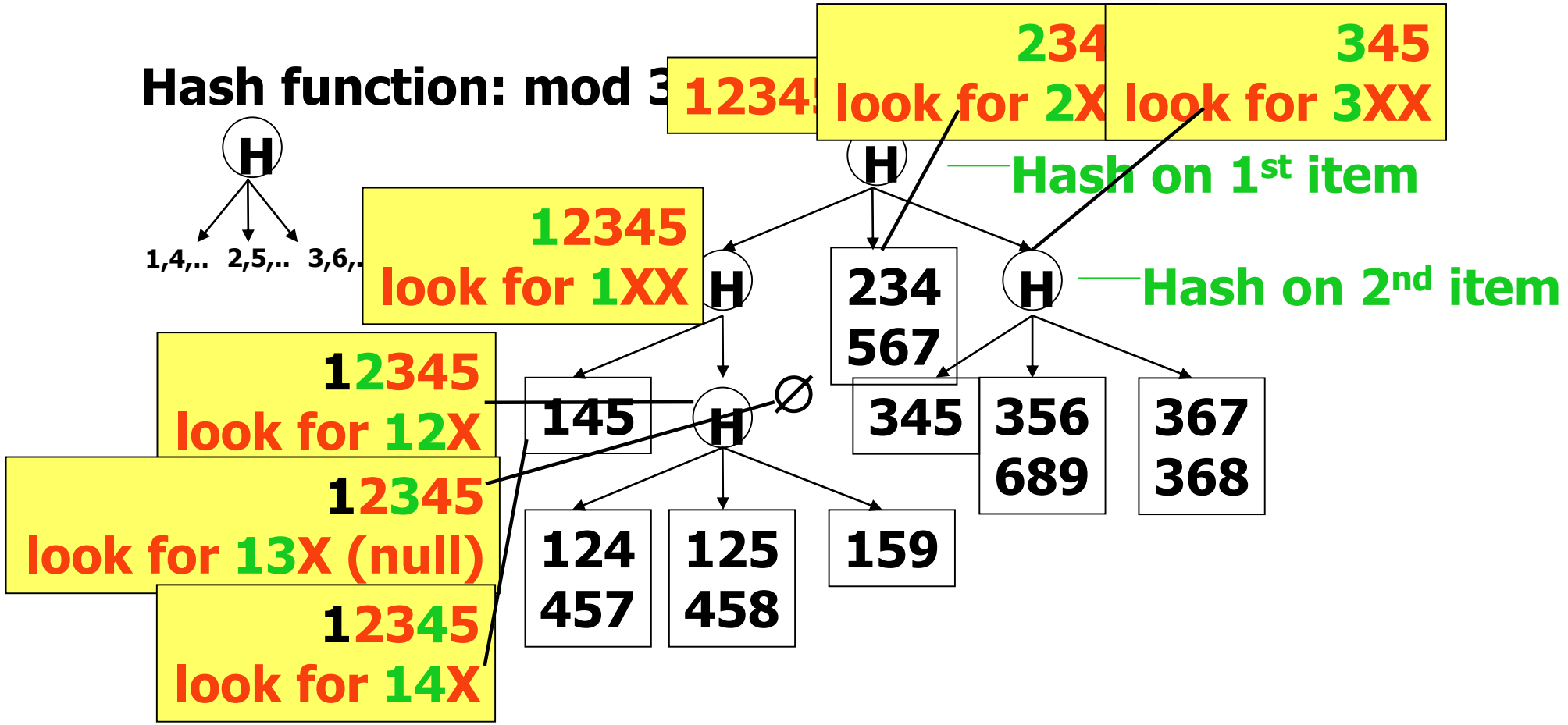


Example of the hash-tree for C_3

Hash function: mod 3



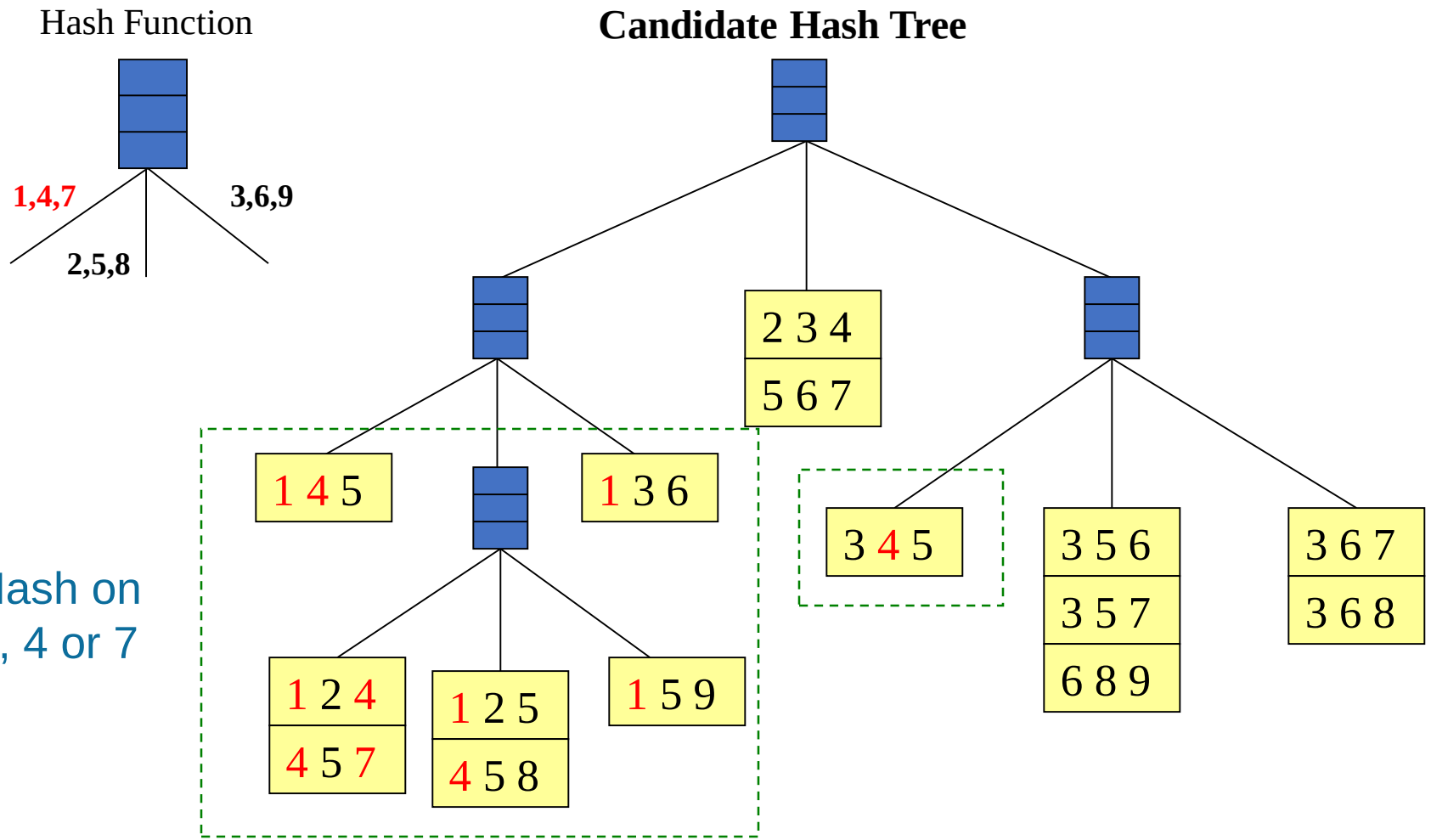
Example of the hash-tree for C_3



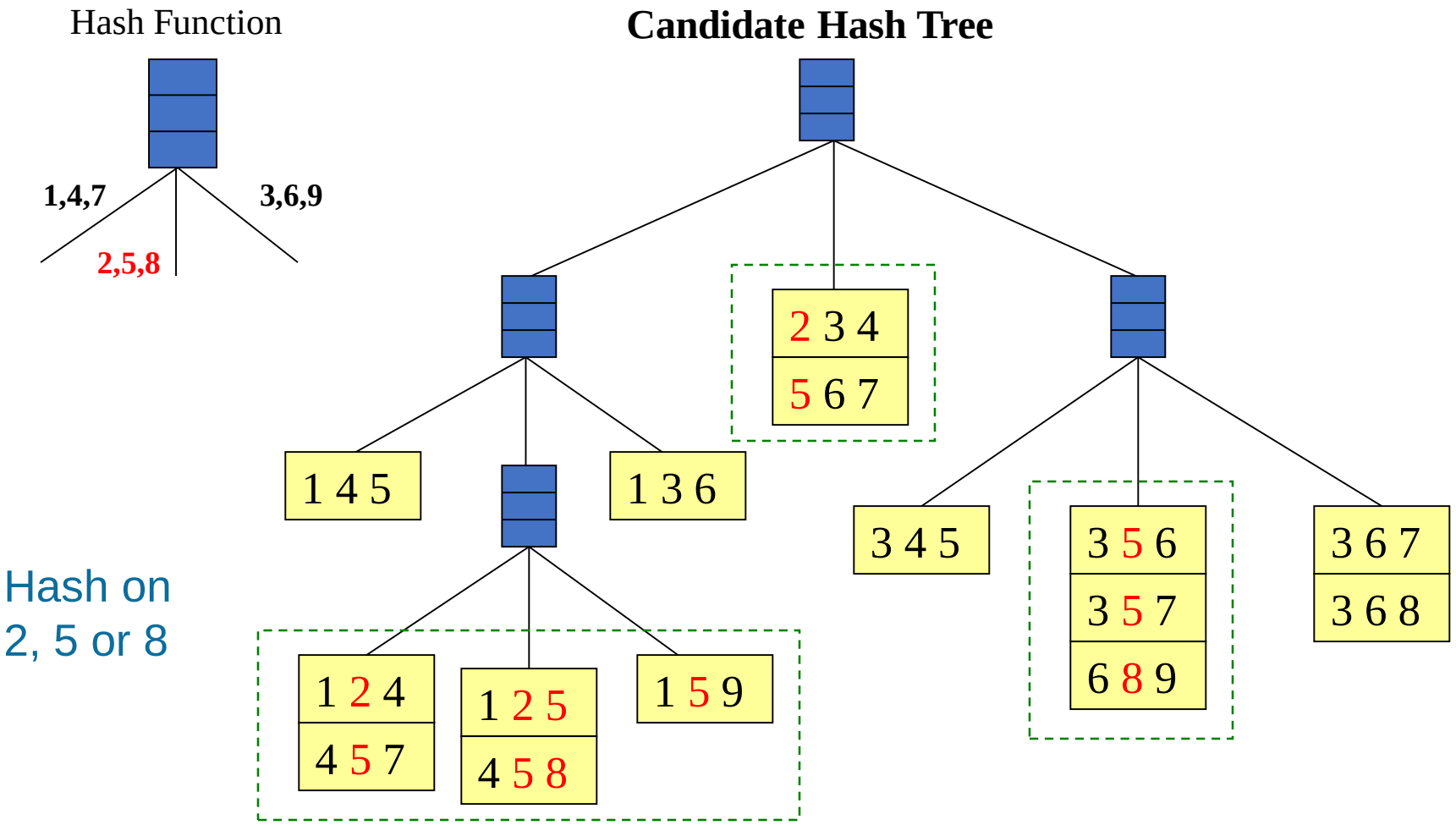
The subset function finds all the candidates contained in a transaction:

- At the root level it hashes on all items in the transaction
- At level i it hashes on all items in the transaction that come after item the i-th item

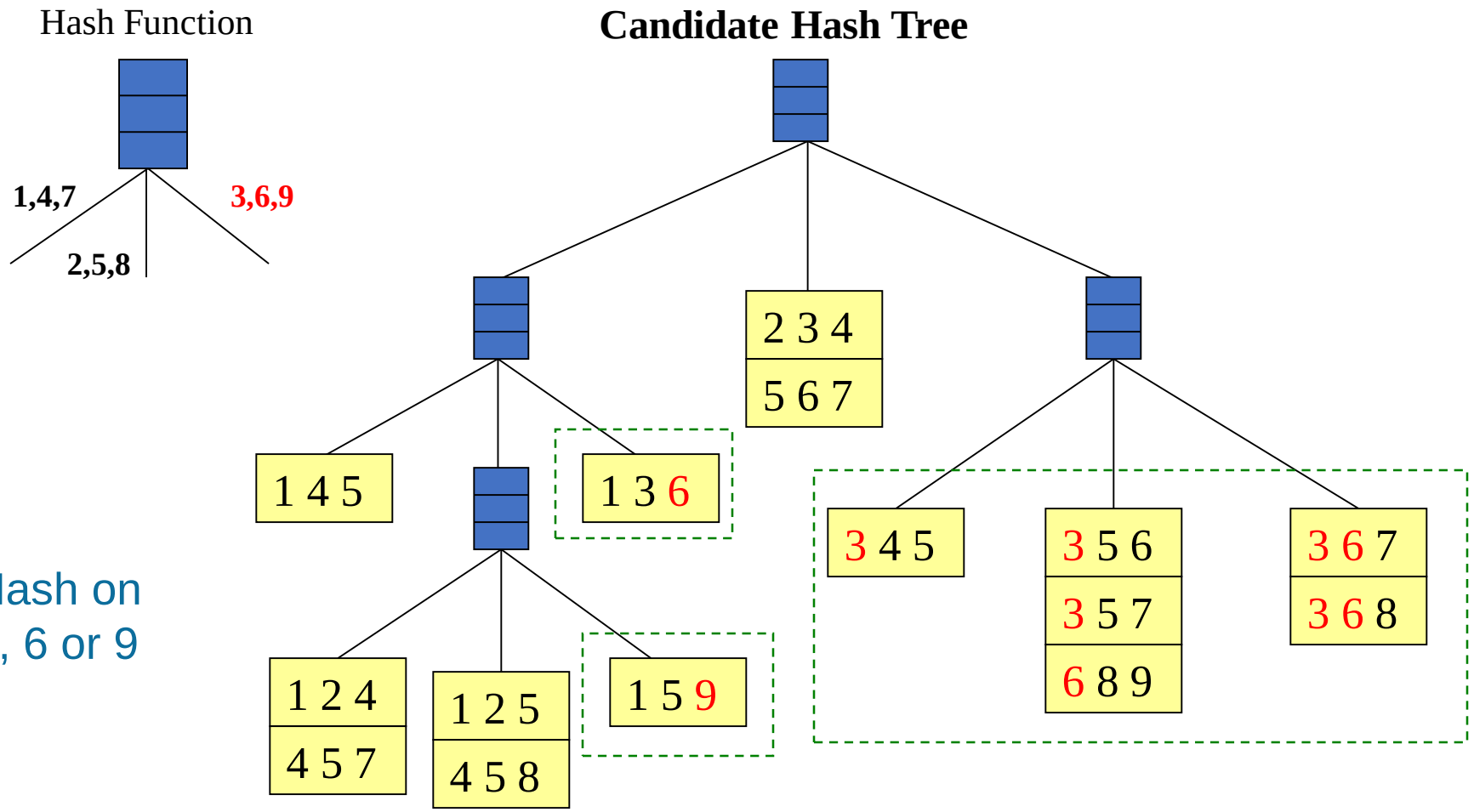
Association Rule Discovery: Hash tree



Association Rule Discovery: Hash tree

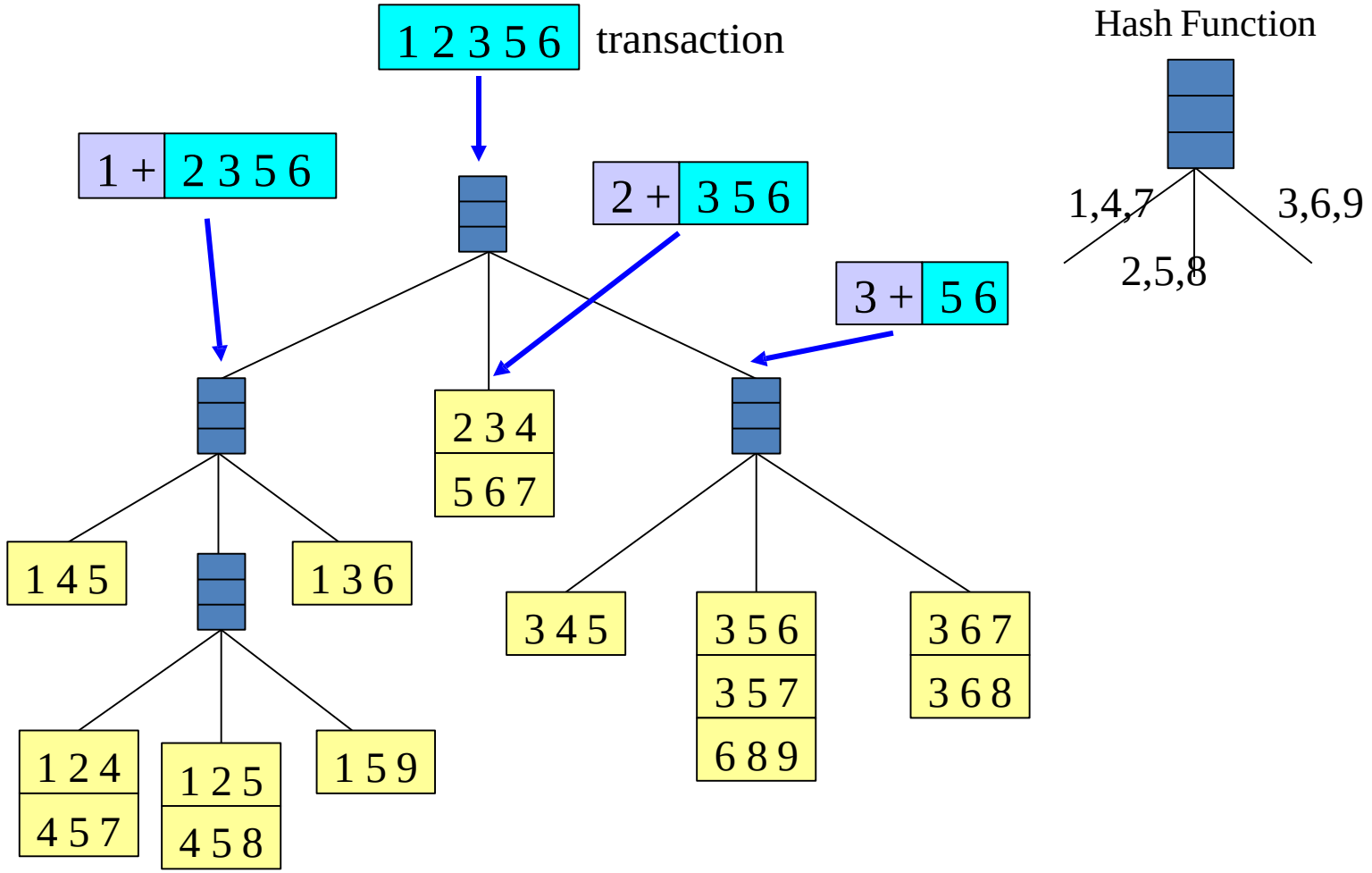


Association Rule Discovery: Hash tree

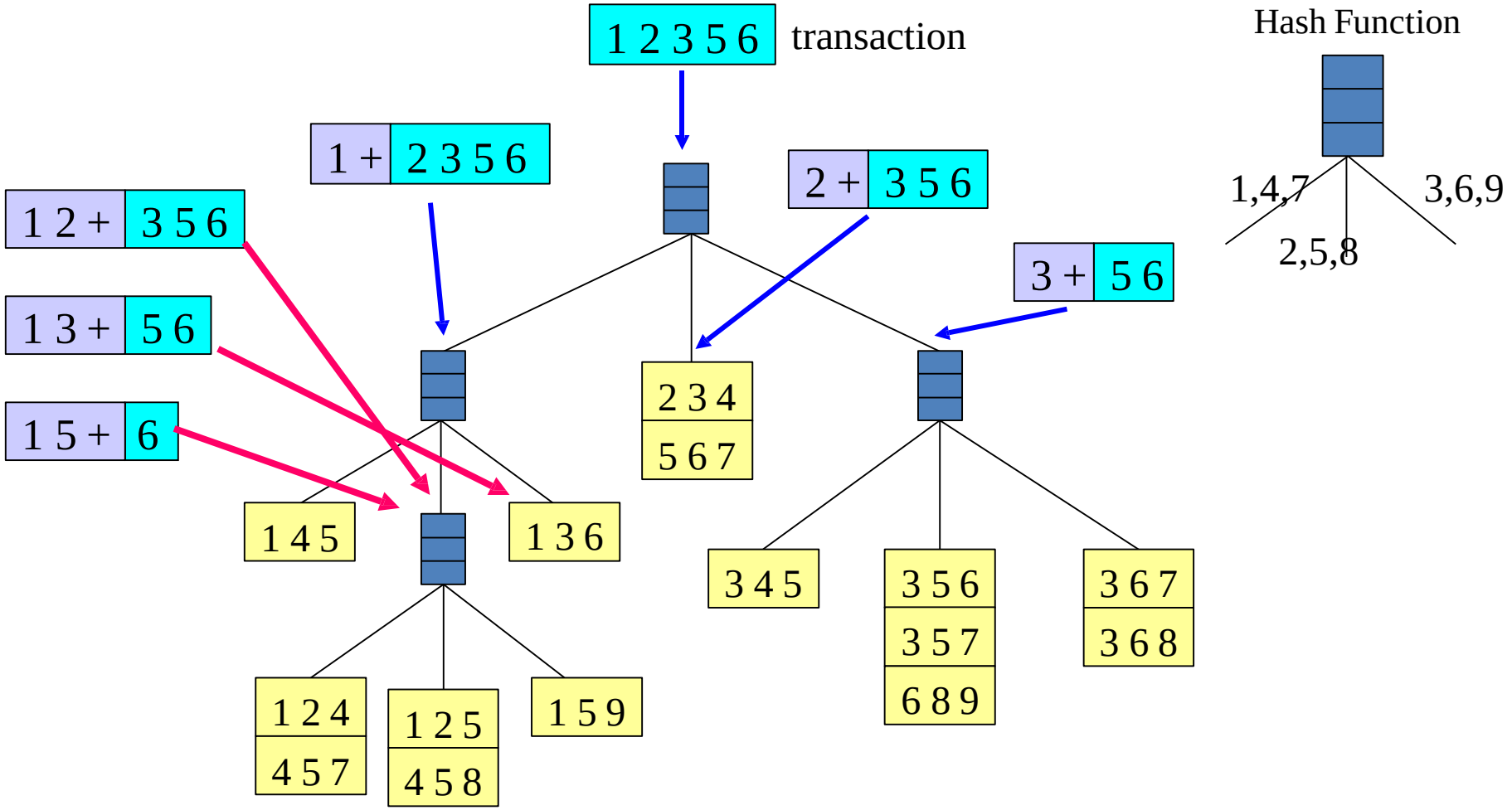


Hash on
3, 6 or 9

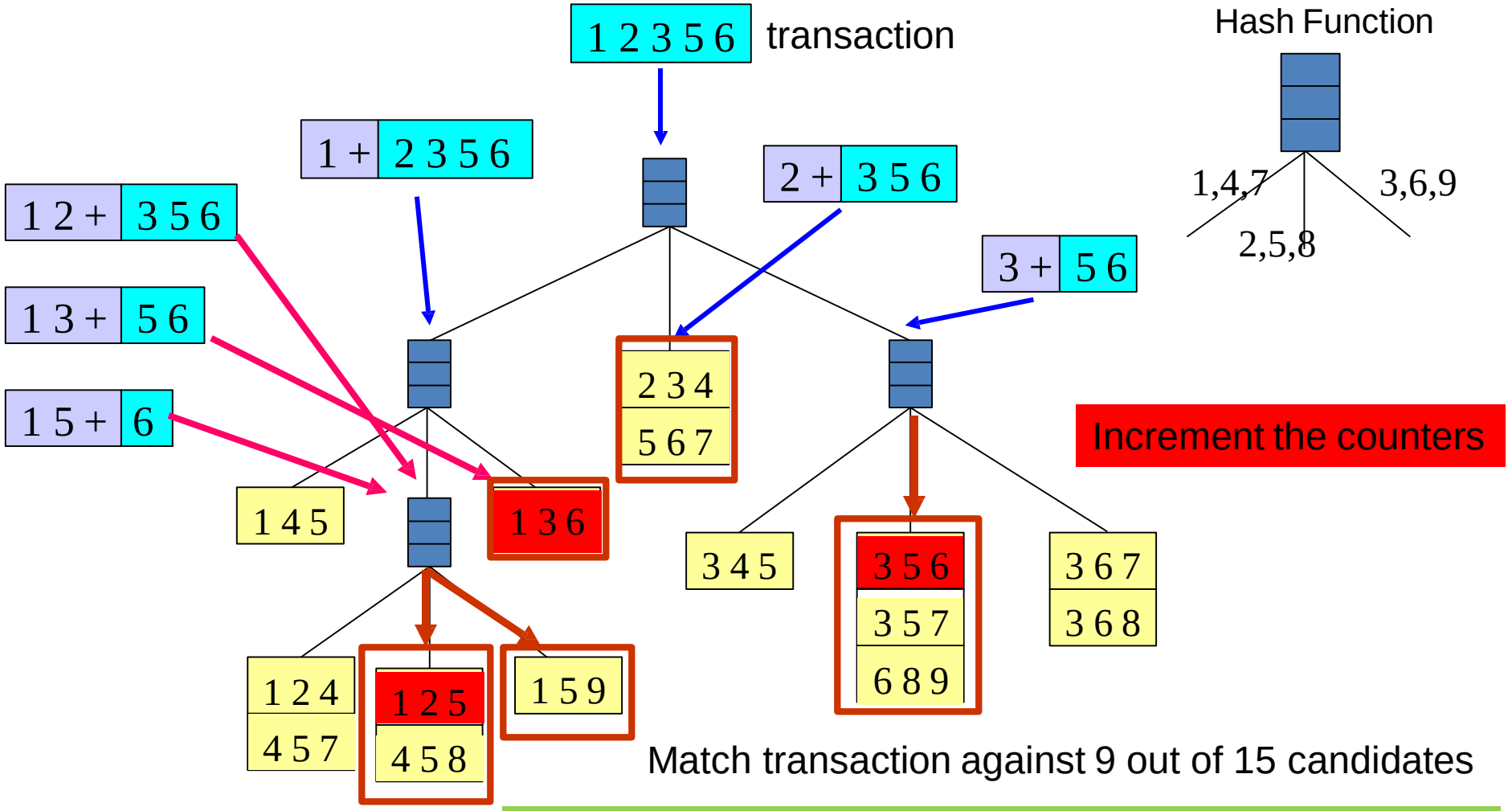
Subset Operation Using HashTree



Subset Operation Using Hash Tree



Subset Operation Using HashTree



Hash-tree enables to enumerate itemsets in transaction and match them against candidates

Example

Suppose you have 15 candidate itemsets of length 3:

{1 4 5}, {1 2 4}, {4 5 7}, {1 2 5}, {4 5 8},

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

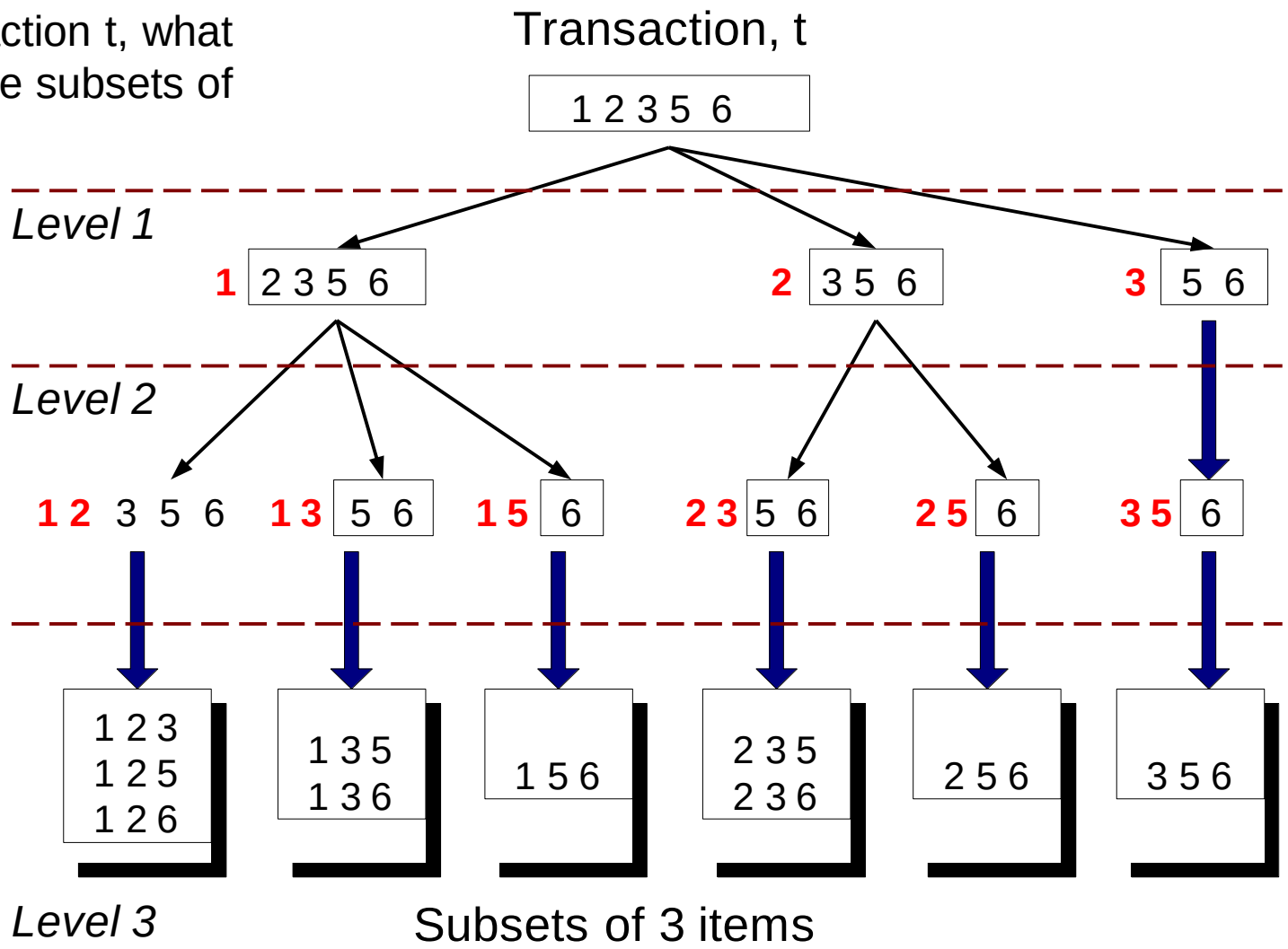
{3 5 6}, {3 5 7}, {6 8 9}, {3 6 7}, {3 6 8}

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

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

Subset Generation

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



Recursion!

Example

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

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

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

Increment the counters for the itemsets in the dictionary

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

Example

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

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

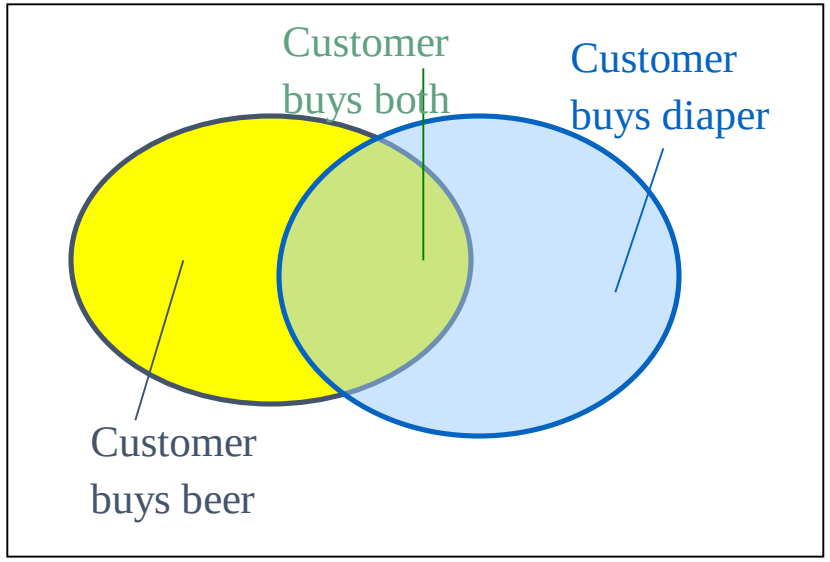
{1 5 6}, {2 3 5}, {2 3 6}, {3 5 6},

Increment the counters for the itemsets in the dictionary

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

Basic Concepts: Frequent Patterns and Association Rules

Transaction-id	Items bought
10	A, B, D
20	A, C, D
30	A, D, E
40	B, E, F
50	B, C, D, E, F



- Itemset X = $\{x_1, \ldots, x_k\}$
- 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 $sup_{min} = 50\%$, $conf_{min} = 50\%$

Freq. Pat.: {A:3, B:3, D:4, E:3, AD:3}

Association rules:

$A \rightarrow D$ (60%, 100%)

$D \rightarrow A$ (60%, 75%)

How to trim candidate itemsets

- In k-iteration, hash all "appearing" k+1 itemsets in a hashtable, count all the occurrences of an itemset in the correspondent bucket.

- In k+1 iteration, examine each of the candidate itemset to see if its correspondent bucket value is above the support (necessary condition)

Hash Table Construction

- Consider two items sets, all items are numbered as i1, i_2, …i_n.
- For any pair (x, y), has according to
  - **Hash function bucket #=h({x y}) = ((order of x)*10+(order of y)) % 7**

- Example:
  - Items = A, B, C, D, E,
  - Order = 1, 2,3,4, 5,
  - **H({C, E})= (3*10 + 5) % 7 = 0**
  - Thus, {C, E} belong to bucket 0.

Example

TID	Items
100	A C D
200	B C E
300	A B C E
400	B E

Generation of C1 & L1(1st iteration)

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

C1

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

L1

Hash Table Construction

- Find all 2-itemset of each transaction

TID	2-itemset
100	{A C} {A D} {C D}
200	{B C} {B E} {C E}
300	{A B} {A C} {A E} {B C} {B E} {C E}
400	{B E}

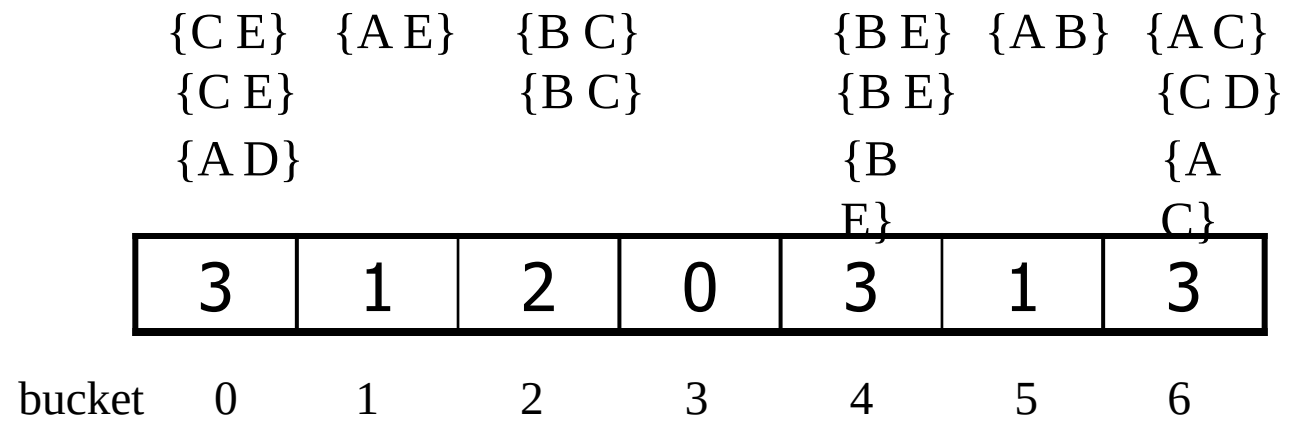
Hash Table Construction (2)

- Hash function

  h({x y}) = ((order of x)*10+(order of y)) % 7

  - Items = A, B, C, D, E,
  - Order = 1, 2,3,4, 5,

- Hash table



C2 Generation (2nd iteration)

L1*L1	# in the bucket
{A B}	1
{A C}	3
{A E}	1
{B C}	2
{B E}	3
{C E}	3

Resulted C2
{A C}
{B C}
{B E}
{C E}

C2 of Apriori
{A B}
{A C}
{A E}
{B C}
{B E}
{C E}

Improving Apriori – 1

DHP: Direct Hashing and Pruning, by

J. Park, M. Chen, and P. Yu. An effective hash-based algorithm for mining association rules. In *SIGMOD'95*:

Three Main ideas:

a. Candidates are restricted to be subsets of transactions.

E.g., D = {*a*, *b*, *c*}, {*d*, *e*, *f*}, {*b*}, {*c*}, {*d*} and min_sup = 2.

Then 3 1-itemsets *b*, *c*, *d* are frequent. Then, Apriori considers ${}^3C_2 = 3$ candidate 2-itemsets, viz., *bc*, *bd*, *cd*. However, DHP considers only 1 candidate 2-itemset, viz., *bc* (because *bd*, *cd* are not in any transaction).

Possible downside: Have to visit transactions in the database (on disk)!

Ideas behind DHP

b. A hash table is used to count support of itemsets of small size.

E.g., hash table created using hash fn.

h(Ix, Iy) = (10x +y) mod 7

from Table 5.1 considering every pair of items in every transaction:

Bucket address	0	1	2	3	4	5	6
Bucket count	2	2	4	2	2	4	4
Bucket contents	{I1, I4}	{I1, I5}	{I2, I3}	{I2, I4}	{I2, I5}	{I1, I2}	{I1, I3}
	{I3, I5}	{I1, I5}	{I2, I3}	{I2, I4}	{I2, I5}	{I1, I2}	{I1, I3}
			{I2, I3}			{I1, I2}	{I1, I3}
			{I2, I3}			{I1, I2}	{I1, I3}

If min_sup = 3, the itemsets in buckets 0, 1, 3, 4, are infrequent and pruned.

Ideas behind DHP

c. Database itself is pruned by removing transactions based on the logic that a transaction can contain a frequent (*k+1*)-itemset only if contains at least *k+1* different frequent *k*-itemsets. So, a transaction that **doesn't** contain *k+1* frequent *k*-itemsets can be pruned.

E.g., say a transaction is {*a*, *b*, *c*, *d*, *e*, *f* }. Now, if it contains a frequent 3-itemset, say *aef*, then it contains the 3 frequent 2-itemsets *ae*, *af*, *ef*.

So, at the time that L_k, the frequent *k*-itemsets are determined, one can check transactions according to the condition above for possible pruning before the next stage.

Say, we have determined L_2 = {*ac*, *bd*, *eg*, *fg* }. Then, we can drop the transaction {*a*, *b*, *c*, *d*, *e*, *f* } from the database for the next step. Why?

Improving Apriori – 2

Partition: Scanning the Database only Twice, by

Savasere, E. Omiecinski, and S. Navathe. An efficient algorithm for mining association in large databases. In VLDB'95: Main Idea:

Partition the database (first scan) into *n* parts so that each fits in main. Observe that an itemset frequent in the whole DB (globally frequent) must be frequent in **at least one** partition (locally frequent). Therefore, collection of all locally frequent itemsets forms the global candidate set. Second scan is required to find the frequent itemsets from the global candidates.

Improving Apriori – 3

Sampling: Mining a Subset of the Database, by

H. Toivonen. Sampling large databases for association rules. In *VLDB'96:*

Main idea:

Choose a sufficiently small random sample S of the database D as to fit in main. Find all frequent itemsets in S (locally frequent) using a lower min_sup value (e.g., 1.5% instead of 2%) to lessen the probability of missing globally frequent itemsets. With high prob: locally frequent $\supseteq$ globally frequent.

Test each locally frequent if globally frequent!

Improving Apriori – 4

S. Brin, R. Motwani, J. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data. In *SIGMOD'97*

Practice

And Review

ID	apples,	beer,	cheese,	dates,	eggs,	fish,	glue,	honey,	ice-cream
1	1	1		1			1	1	
2			1	1	1				
3		1	1			1			
4		1				1			1
5					1		1		
6						1			1
7	1			1				1	
8						1			1
9			1		1				
10		1					1		
11					1		1		
12	1								
13			1			1			
14			1			1			
15								1	1
16				1					
17	1					1			
18	1	1	1	1				1	
19	1	1		1			1	1	
20					1				

Find rules in two stages

1. *Find all itemsets with a specified minimal support* (*coverage*). An itemset is just a specific set of items, e.g. {apples, cheese}. The *Apriori* algorithm can *efficiently* find all itemsets whose coverage is above a given minimum.
2. *Use these itemsets to help generate interersting rules.* Having done stage 1, we have considerably narrowed down the possibilities, and can do reasonably fast processing of the large itemsets to generate candidate rules.

Termniology

k-itemset : a set of *k* items. E.g.

{beer, cheese, eggs} is a 3-itemset

{cheese} is a 1-itemset

{honey, ice-cream} is a 2-itemset

support: an itemset has support *s*% if *s*% of the records in the DB contain that itemset.

minimum support: the Apriori algorithm starts with the specification of a minimum level of support, and will focus on itemsets with this level or above.

Terminology

large itemset: doesn't mean an itemset with many items. It means one whose support is at least minimum support.

L_k : the set of all large *k*-itemsets in the DB.

C_k : a set of *candidate* large *k*-itemsets. In the algorithm we will look at, it generates this set, which contains all the *k*-itemsets that might be large, and then eventually generates the set above.

ID	a,	b,	c,	d,	e,	f,	g,	h,	i
1	1	1		1			1	1	
2			1	1	1				
3		1	1			1			
4		1				1			1
5					1		1		
6						1			1
7	1			1				1	
8						1			1
9			1		1				
10		1					1		
11					1		1		
12	1								
13			1			1			
14			1			1			
15								1	1
16				1					
17	1					1			
18	1	1	1	1				1	
19	1	1		1			1	1	
20					1				

E.g.
3-itemset {a,b,h}
has support 15%

2-itemset {a, i}
has support 0%

4-itemset {b, c, d, h}
has support 5%

If minimum support is 10%, then {b} is a large
itemset, but {b, c, d, h}
Is a *small* itemset!

Explaining the Apriori Algorithm …

1: Find all large 1-itemsets

To start off, we simply find all of the large 1-itemsets. This is done by a basic scan of the DB. We take each item in turn, and count the number of times that item appears in a basket. In our running example, suppose minimum support was 60%, then the only large 1-itemsets would be: {a}, {b}, {c}, {d} and {f}. So we get

$$L_1 = \{ \{a\}, \{b\}, \{c\}, \{d\}, \{f\}\}$$

Explaining the Apriori Algorithm …

1: Find all large 1-itemsets

2: For (k = 2 ; while L_{k-1} is non-empty; k++)

We already have L_1. This next bit just means that the remainder of the algorithm generates L_2, L_3, and so on until we get to an L_k that's empty.

How these are generated is like this:

Explaining the Apriori Algorithm …

1: Find all large 1-itemsets

2: For (k = 2 ; while L_{k-1} is non-empty; k++)

3 {C_k = `apriori-gen`(L_{k-1})

Given the large k-1-itemsets, this step generates some candidate k-itemsets that *might* be large. Because of how `apriori-gen` works, the set C_k is guaranteed to contain all the large k-itemsets, but also contains some that will turn out not to be `large'.

Explaining the Apriori Algorithm …

1: Find all large 1-itemsets
2: For (k = 2 ; while L_{k-1} is non-empty; k++)
3 {C_k = `apriori-gen`(L_{k-1})
4 For each c in C_k, initialise $c.count$ to zero
5 For all records r in the DB
6 {C_r = `subset`(C_k, r); For each c in C_r, $c.count$++ }
7 Set L_k := all c in C_k whose $count$ >= $minsup$

Here, we are simply scanning through the DB to count the support for each of our candidates in C_k , throwing out the ones without enough support, and the rest become our set of 'large' k-itemsets, L_k

Explaining the Apriori Algorithm …

1: Find all large 1-itemsets
2: For (k = 2 ; while L_{k-1} is non-empty; k++)
3 {C_k = `apriori-gen`(L_{k-1})
4 For each c in C_k, initialise $c.count$ to zero
5 For all records r in the DB
6 {C_r = `subset`(C_k, r); For each c in C_r , $c.count$++ }
7 Set L_k := all c in C_k whose $count$ >= $minsup$
8 } /* end -- return all of the L_k sets.

We finish at the point where we get an empty L_k.

The algorithm returns all of the (non-empty) L_k sets, from L_1 through to whichever was the last non-empty one. This gives us an excellent start in finding interesting rules (although the large itemsets themselves will usually be very interesting and useful).

Test yourself: Understanding rules

Suppose itemset *A* = {beer, cheese, eggs} has 30% support in the DB
{beer, cheese} has 40%, {beer, eggs} has 30%, {cheese, eggs} has 50%,
and each of beer, cheese, and eggs alone has 50% support..

What is the confidence of:
IF basket contains Beer and Cheese, THEN basket also contains Eggs ?

The confidence of a rule if *A* then *B* is simply:
support(*A* + *B*) / *support*(*A*).

What is the confidence of:
IF basket contains Beer, THEN basket also contains Cheese and Eggs ?

The answers are in the above boxes in white font colour

Test yourself: Understanding rules

If the following rule has confidence *c*: *If* A *then* B

and if support(A) = 2 * support(B), what can be said about the confidence of: *If* B *then* A

E.g. *A* might be milk and *B* might be newspapers

The answers are in the above box in white font colour

Mining Association Rules

- Two-step approach:
  - Frequent Itemset Generation
    - Generate all itemsets whose support ≥ minsup
  - Rule Generation
    - Generate high confidence rules from each frequent itemset, where each rule is a binary partition of a frequent itemset

Rule Generation – Naive algorithm

- Given a frequent itemset **X**, find all non-empty subsets **$y \subset X$** such that **$y \rightarrow X - y$** satisfies the minimum confidence requirement

  - 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,

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

Efficient rule generation

- How to efficiently generate rules from frequent itemsets?
  - In general, confidence does not have an anti-monotone property
    c(ABC →D) can be larger or smaller than **c(AB →D)**
  - ***But confidence of rules generated from the same itemset has an anti-monotone property***
  - Example: X = {A,B,C,D}:

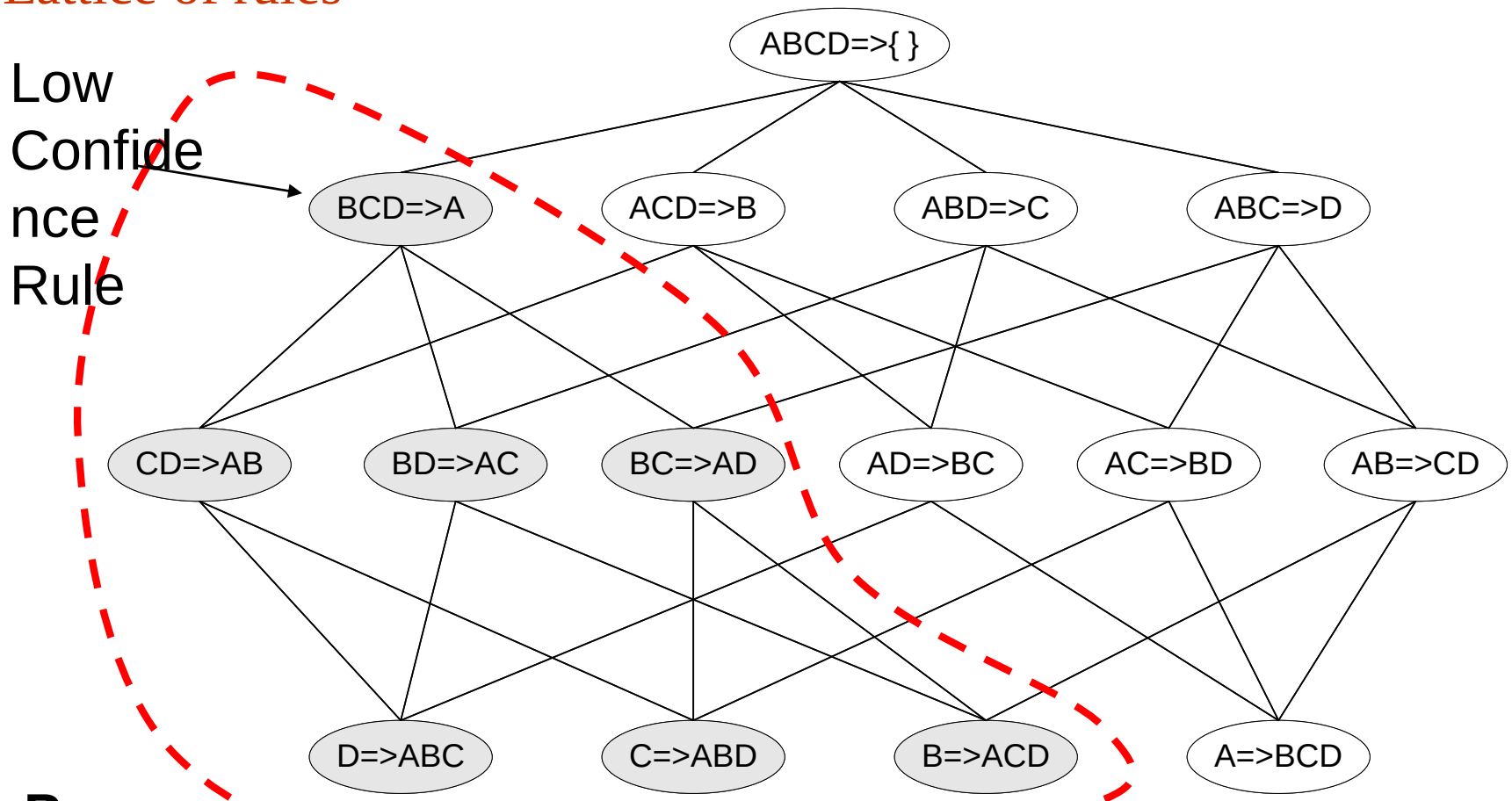
$$c(ABC \rightarrow D) \geq c(AB \rightarrow CD) \geq c(A \rightarrow BCD)$$

  - **Why?**

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

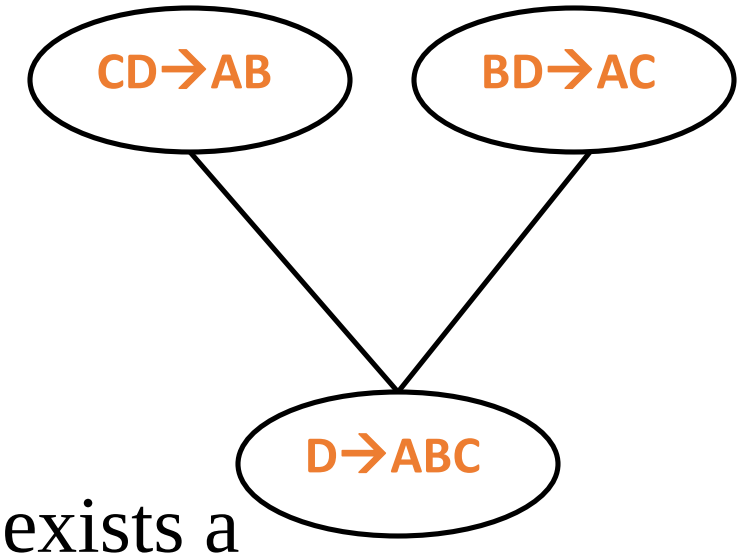
Rule Generation for Apriori Algorithm

Lattice of rules



Apriori algorithm for rule generation

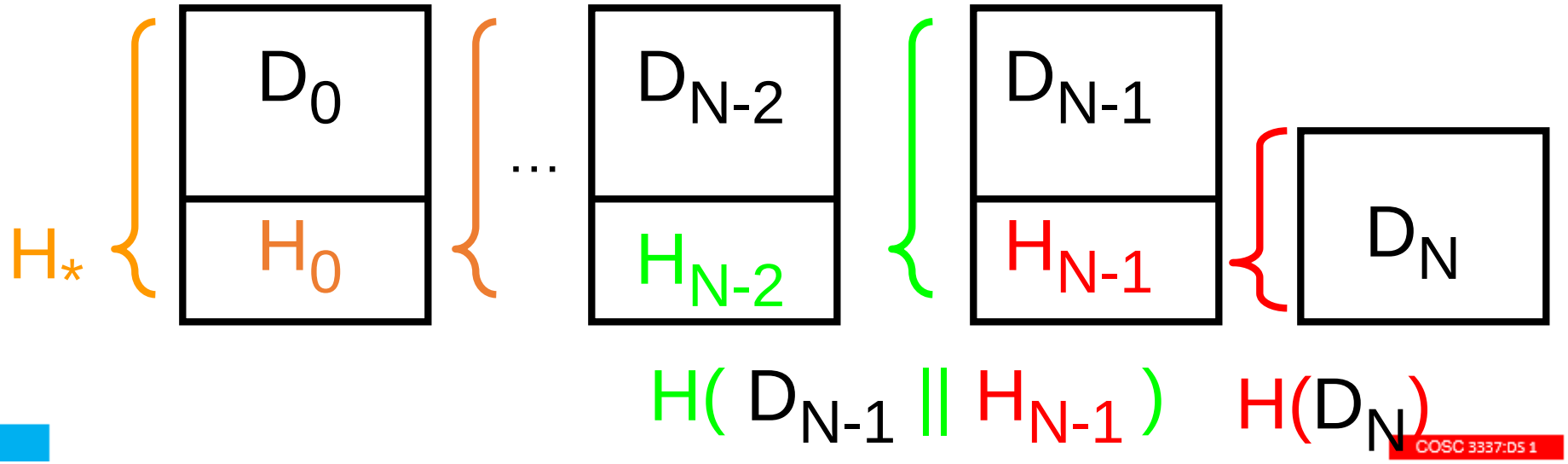
- Candidate rule is generated by merging two rules that share the same prefix in the rule consequent



- **join(CD→AB,BD—>AC)** would produce the candidate rule **D →ABC**
- **Prune** rule **D→ABC** if there exists a subset (e.g., **AD→BC**) that does not have high confidence

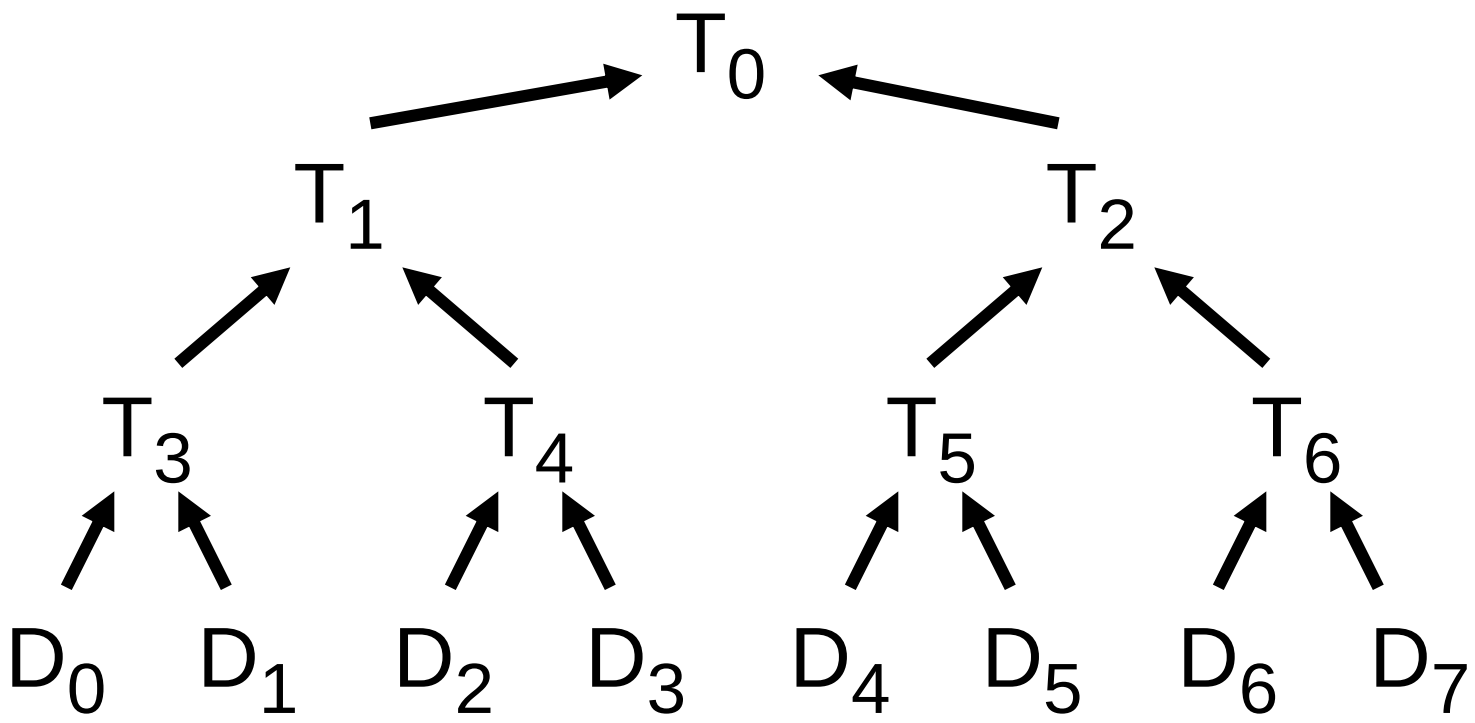
Chained Hashes

- More general construction than one-way hash chains
- Useful for authenticating a sequence of data values D_0 , D_1 , …, D_N
- H_* authenticates entire chain



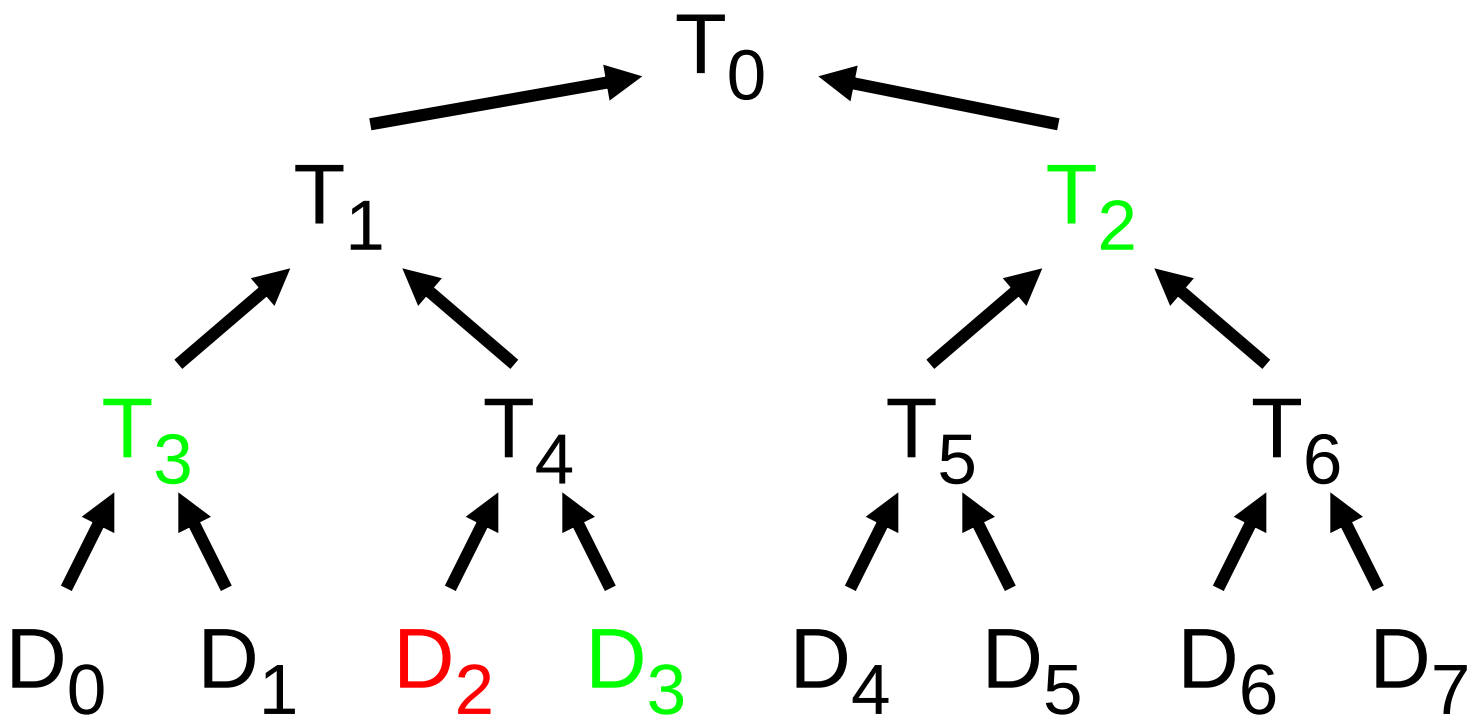
Merkle Hash Trees

- Authenticate a sequence of data values D_0 , D_1 , …, D_N
- Construct binary tree over data values

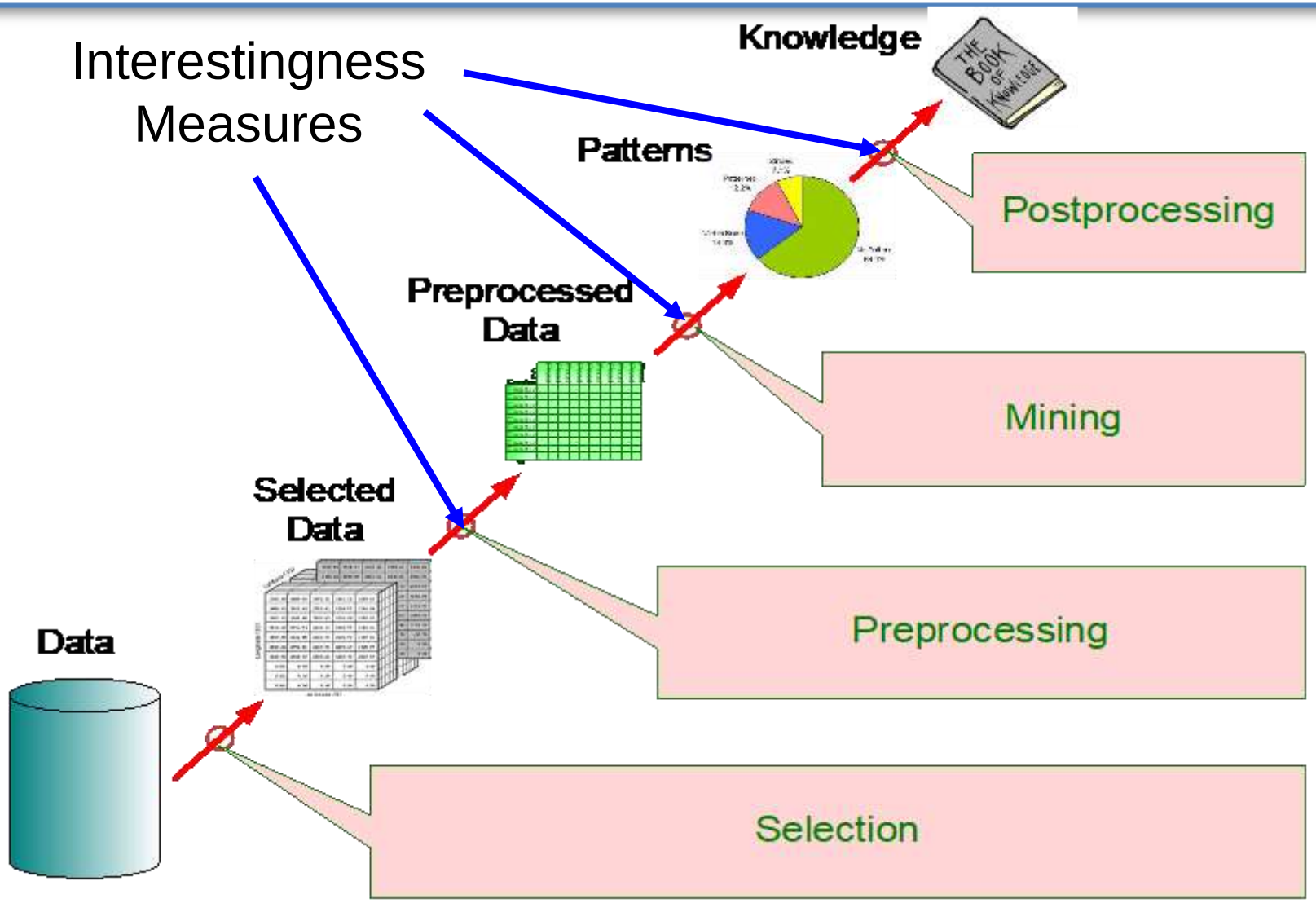


Merkle Hash Trees II

- Verifier knows T_0
- How can verifier authenticate leaf D_i ?
- Solution: recompute T_0 using D_i
- Example authenticate D_2 , send D_3 T_3 T_2
- Verify $T_0 = H(H(T_3 || H(D_2 || D_3)) || T_2)$



Application of Interestingness Measure



Support

Confidence

Lift interestingness Measure

Chi square

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

Used to define various measures

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

Lift :Measure of dependent /correlated

- lift(B,C)= c(B->C)/ s(C)

  = s(B U C)/ s(B)* s(C)

If lift(B,C)= 1 B and C are independent

>1 Positively correlated

< Negatively correlated

Example

Contingency table for $B \rightarrow C$

	B	!B	TOT
C	400	350	750
!C	200	50	250
TOT	600	400	1000

$$lift(B,C) = \frac{400/1000}{600/1000 * 750/1000} = 0.89$$

$$lift(B,!C) = \frac{200/1000}{600/1000 * 250/1000} = 1.33$$

B and C are negatively correlated 0.89 <1

B and !C are positively correlated 1.33>1

Another Measure X^2

Contingency table for $B \rightarrow C$

	B	!B	TOT
C	400(450)	350 (300)	750
!C	200 (150)	50(100)	250
TOT	600	400	1000

$$X^2 = \Sigma \frac{(Observed - Expected)^2}{Expected}$$

General rules:

$X^2 = 0$ ➔ *independent*

$X^2 > 0$ ➔ *correlated either positively or negatively*

$$X^2 = \frac{(400-450)^2}{400} + \frac{(350-300)^2}{350} + \frac{(200-150)^2}{200} + \frac{(50-100)^2}{50}$$

= 55.89 positive ➔ correlated

Negatively correlated because 400 (observed)< 450 expected

	B	!B	TOT
C	100	1000	1100
!C	1000	100000	101000
TOT	1100	101000	102100

	B	!B	TOT
C	100 (11.85)	1000	1100
!C	1000 (988.15)	100000	101000
TOT	1100	101000	102100

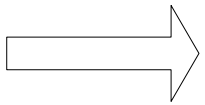
- lift(B,C)= c(B->C)/ s(C)?

X^2 ?

Are they good measures?

Property under Null Addition

	B	$\overline{B}$
A	p	q
$\overline{A}$	r	s



	B	$\overline{B}$
A	p	q
$\overline{A}$	r	s + k

Invariant measures:

- support, cosine, Jaccard, etc

Non-invariant measures:

- correlation, Gini, mutual information, odds ratio, etc

- Why is null invariance crucial for the analysis of massive transaction data?
  - Many transactions may contain neither milk nor coffee!

milk vs. coffee contingency table

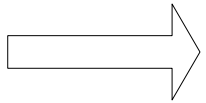
	$milk$	$\neg milk$	Σ_{row}
$coffee$	mc	$\neg mc$	c
$\neg coffee$	$m\neg c$	$\neg m\neg c$	$\neg c$
Σ_{col}	m	$\neg m$	Σ

- Lift and χ^2 are not null-invariant: not good to evaluate data that contain too many or too few null transactions!
- Many measures are not null-invariant!

Data set	mc	$\neg mc$	$m\neg c$	$\neg m\neg c$	χ^2	$Lift$
D_1	10,000	1,000	1,000	100,000	90557	9.26
D_2	10,000	1,000	1,000	100	0	1
D_3	100	1,000	1,000	100,000	670	8.44
D_4	1,000	1,000	1,000	100,000	24740	25.75
D_5	1,000	100	10,000	100,000	8173	9.18
D_6	1,000	10	100,000	100,000	965	1.97

Property under Variable Permutation

	B	$\overline{B}$
A	p	q
$\overline{A}$	r	s



	A	$\overline{A}$
B	p	r
$\overline{B}$	q	s

Does M(A,B) = M(B,A)?

Symmetric measures:

- support, lift, collective strength, cosine, Jaccard, etc

Asymmetric measures:

- confidence, conviction, Laplace, J-measure, etc

Property under Row/Column Scaling

Grade-Gender Example (Mosteller, 1968):

	Male	Female	
High	2	3	5
Low	1	4	5
	3	7	10

	Male	Female	
High	4	30	34
Low	2	40	42
	6	70	76

2x 10x

Mosteller:

Underlying association should be independent of the relative number of male and female students in the samples

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)

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

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

- Measures that take into account statistical dependence

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

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

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

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

Another Example of Drawback of Lift & Interest

	Y	$\bar{Y}$	
X	10	0	10
$\bar{X}$	0	90	90
	10	90	100

$$Lift = \frac{0.1}{(0.1)(0.1)} = 10$$

	Y	$\bar{Y}$	
X	90	0	90
$\bar{X}$	0	10	10
	90	10	100

$$Lift = \frac{0.9}{(0.9)(0.9)} = 1.11$$

Statistical independence:

If P(X,Y)=P(X)P(Y) => Lift = 1

Example: ϕ-Coefficient

- ϕ-coefficient is analogous to correlation coefficient for continuous variables

	Y	$\overline{Y}$	
X	60	10	70
$\overline{X}$	10	20	30
	70	30	100

	Y	$\overline{Y}$	
X	20	10	30
$\overline{X}$	10	60	70
	30	70	100

$$\phi = \frac{0.6 - 0.7 \times 0.7}{\sqrt{0.7 \times 0.3 \times 0.7 \times 0.3}} = 0.5238$$

$$\phi = \frac{0.2 - 0.3 \times 0.3}{\sqrt{0.7 \times 0.3 \times 0.7 \times 0.3}} = 0.5238$$

ϕ Coefficient is the same for both tables

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?

What about Apriori-style support based pruning? How does it affect these measures?

#	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)	$\max\left(P(A,B)\log(\frac{P(B\|A)}{P(B)}) + P(A\overline{B})\log(\frac{P(\overline{B}\|A)}{P(\overline{B})}),\right.$ $\left. P(A,B)\log(\frac{P(A\|B)}{P(A)}) + P(\overline{A}B)\log(\frac{P(\overline{A}\|B)}{P(\overline{A})})\right)$
9	Gini index (G)	$\max\Big(P(A)[P(B\|A)^2+P(\overline{B}\|A)^2]+P(\overline{A})[P(B\|\overline{A})^2+P(\overline{B}\|\overline{A})^2]$ $-P(B)^2-P(\overline{B})^2,$ $P(B)[P(A\|B)^2+P(\overline{A}\|B)^2]+P(\overline{B})[P(A\|\overline{B})^2+P(\overline{A}\|\overline{B})^2]$ $-P(A)^2-P(\overline{A})^2\Big)$
10	Support (s)	$P(A,B)$
11	Confidence (c)	$\max(P(B\|A), P(A\|B))$
12	Laplace (L)	$\max\left(\frac{NP(A,B)+1}{NP(A)+2}, \frac{NP(A,B)+1}{NP(B)+2}\right)$
13	Conviction (V)	$\max\left(\frac{P(A)P(\overline{B})}{P(A\overline{B})}, \frac{P(B)P(\overline{A})}{P(B\overline{A})}\right)$
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)	$\max\left(\frac{P(B\|A)-P(B)}{1-P(B)}, \frac{P(A\|B)-P(A)}{1-P(A)}\right)$
18	Added Value (AV)	$\max(P(B\|A)-P(B), P(A\|B)-P(A))$
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)	$\sqrt{P(A,B)}\max(P(B\|A)-P(B), P(A\|B)-P(A))$

Comparing Different Measures

10 examples of contingency tables:

Example	f_{11}	f_{10}	f_{01}	f_{00}
E1	8123	83	424	1370
E2	8330	2	622	1046
E3	9481	94	127	298
E4	3954	3080	5	2961
E5	2886	1363	1320	4431
E6	1500	2000	500	6000
E7	4000	2000	1000	3000
E8	4000	2000	2000	2000
E9	1720	7121	5	1154
E10	61	2483	4	7452

Rankings of contingency tables using various measures:

#	ϕ	λ	α	Q	Y	κ	M	J	G	s	c	L	V	I	IS	PS	F	AV	S	ζ	K
E1	1	1	3	3	3	1	2	2	1	3	5	5	4	6	2	2	4	6	1	2	5
E2	2	2	1	1	1	2	1	3	2	2	1	1	1	8	3	5	1	8	2	3	6
E3	3	3	4	4	4	3	3	8	7	1	4	4	6	10	1	8	6	10	3	1	10
E4	4	7	2	2	2	5	4	1	3	6	2	2	2	4	4	1	2	3	4	5	1
E5	5	4	8	8	8	4	7	5	4	7	9	9	9	3	6	3	9	4	5	6	3
E6	6	6	7	7	7	7	6	4	6	9	8	8	7	2	8	6	7	2	7	8	2
E7	7	5	9	9	9	6	8	6	5	4	7	7	8	5	5	4	8	5	6	4	4
E8	8	9	10	10	10	8	10	10	8	4	10	10	10	9	7	7	10	9	8	7	9
E9	9	9	5	5	5	9	9	7	9	8	3	3	3	7	9	9	3	7	9	9	8
E10	10	8	6	6	6	10	5	9	10	10	6	6	5	1	10	10	5	1	10	10	7

Different Measures have Different Properties

Symbol	Measure	Range	P1	P2	P3	O1	O2	O3	O3'	O4
Φ	Correlation	-1 … 0 … 1	Yes	Yes	Yes	Yes	No	Yes	Yes	No
λ	Lambda	0 … 1	Yes	No	No	Yes	No	No*	Yes	No
α	Odds ratio	0 … 1 … ∞	Yes*	Yes	Yes	Yes	Yes	Yes*	Yes	No
Q	Yule's Q	-1 … 0 … 1	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No
Y	Yule's Y	-1 … 0 … 1	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No
κ	Cohen's	-1 … 0 … 1	Yes	Yes	Yes	Yes	No	No	Yes	No
M	Mutual Information	0 … 1	Yes	Yes	Yes	Yes	No	No*	Yes	No
J	J-Measure	0 … 1	Yes	No	No	No	No	No	No	No
G	Gini Index	0 … 1	Yes	No	No	No	No	No*	Yes	No
s	Support	0 … 1	No	Yes	No	Yes	No	No	No	No
c	Confidence	0 … 1	No	Yes	No	Yes	No	No	No	Yes
L	Laplace	0 … 1	No	Yes	No	Yes	No	No	No	No
V	Conviction	0.5 … 1 … ∞	No	Yes	No	Yes**	No	No	Yes	No
I	Interest	0 … 1 … ∞	Yes*	Yes	Yes	Yes	No	No	No	No
IS	IS (cosine)	0 .. 1	No	Yes	Yes	Yes	No	No	No	Yes
PS	Piatetsky-Shapiro's	-0.25 … 0 … 0.25	Yes	Yes	Yes	Yes	No	Yes	Yes	No
F	Certainty factor	-1 … 0 … 1	Yes	Yes	Yes	No	No	No	Yes	No
AV	Added value	0.5 … 1 … 1	Yes	Yes	Yes	No	No	No	No	No
S	Collective strength	0 … 1 … ∞	No	Yes	Yes	Yes	No	Yes*	Yes	No
ζ	Jaccard	0 .. 1	No	Yes	Yes	Yes	No	No	No	Yes
K	Klosgen's	$\left(\sqrt{\frac{2}{\sqrt{3}}-1}\right)\left(2-\sqrt{3}-\frac{1}{\sqrt{3}}\right)\ldots 0 \ldots \frac{2}{3\sqrt{3}}$	Yes	Yes	Yes	No	No	No	No	No