Ontario Tech Science

Kourosh Davoudi
kourosh@uoit.ca

Association Rule Mining

CSCI 4150U: Data Mining

Learning Outcomes

- Basic Idea and Concepts of Association Rule Mining
- Apriori algorithm
- Maximal Frequent Itemsets
- Closed Itemsets
- FP-growth algorithm

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

For example: {**Bread**} $\rightarrow$ {**Milk**}

“People who buy bread may also buy milk.”

- *Stores might want to offer specials on bread to get people to buy more milk.*
- *Stores might want to put bread and milk close each other.*

Note that Implication means co-occurrence, not causality!

Definition: Frequent Itemset

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**

- 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 (s)
  - 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

Definition: Association Rule

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**

Association Rule

- An implication expression of the form $X \rightarrow Y$, where X and Y are itemsets
- Example:
  {Milk, Diaper} $\rightarrow$ {Beer}

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:

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

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

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

What is the confidence of the following rule?

A. 0.25
B. O.5
C. 0.67
D. 0.73

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

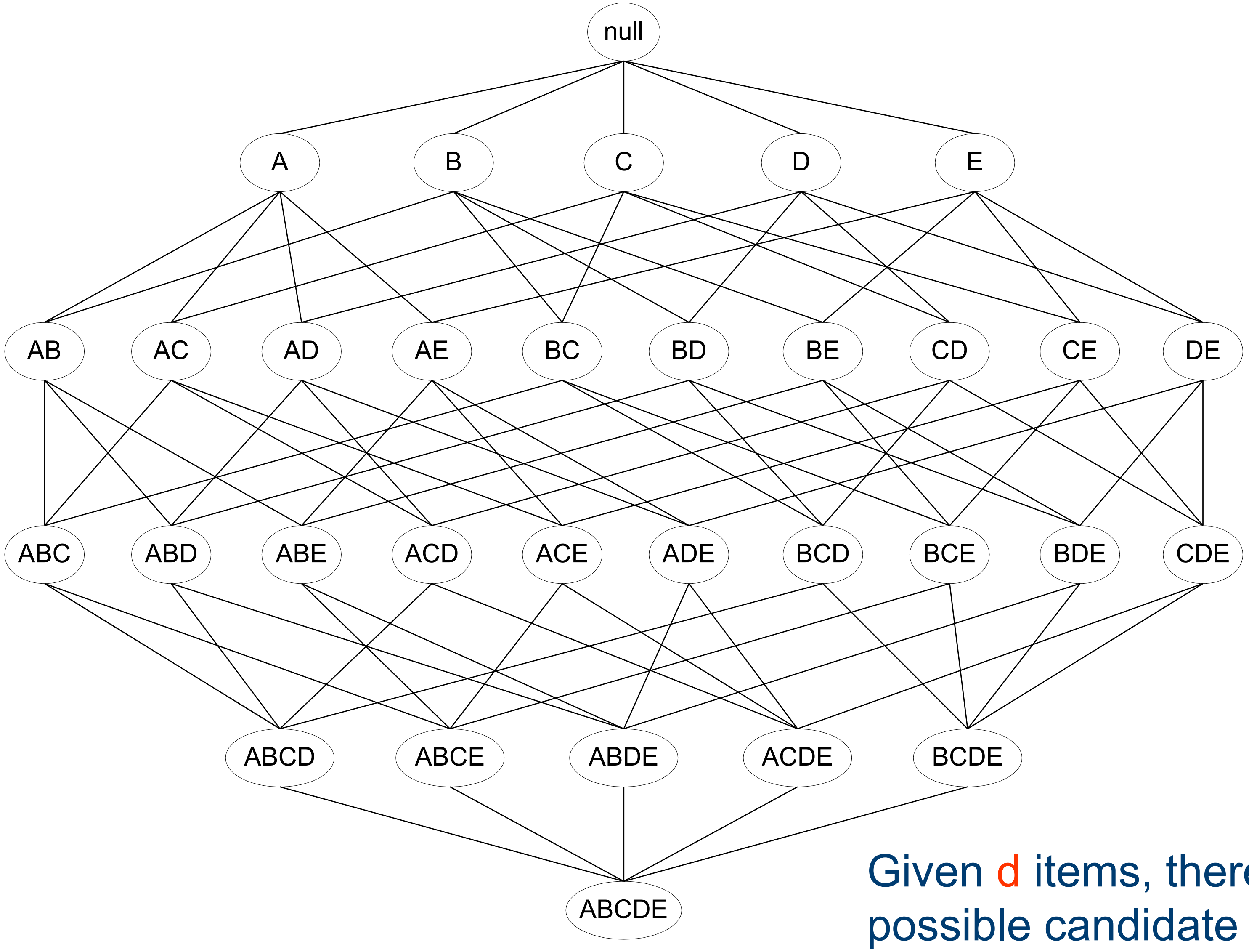
Mining Association Rules

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

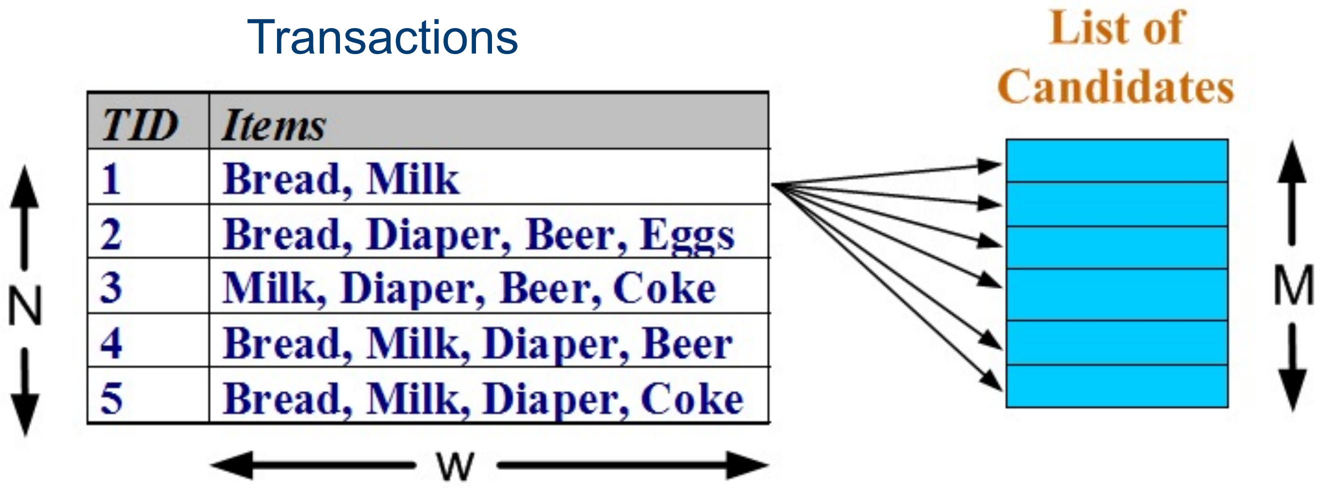
Frequent Itemset Generation: the Itemset Lattice



Given d items, there are $2^d - 1$ 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
  - Output the itemsets with a counter ≥ (min_sup*N)

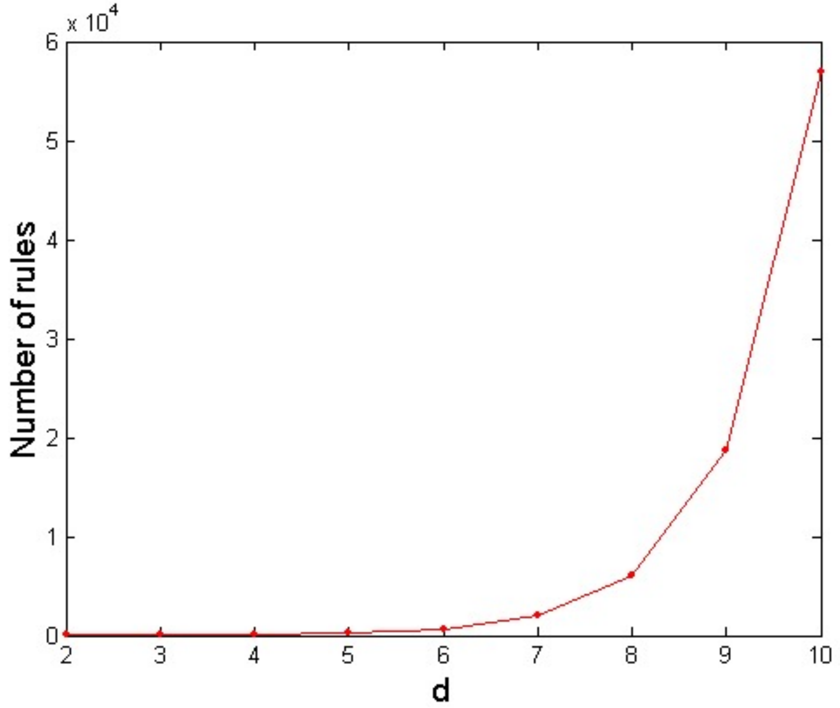


TID	Items
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

  - Match each transaction against every candidate
  - Complexity ~ O(NMw) => Expensive since $M = 2^d-1$!!!

Computational Complexity

- Given d unique items:
  - Total number of itemsets = 2^d -1
  - 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
- 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:

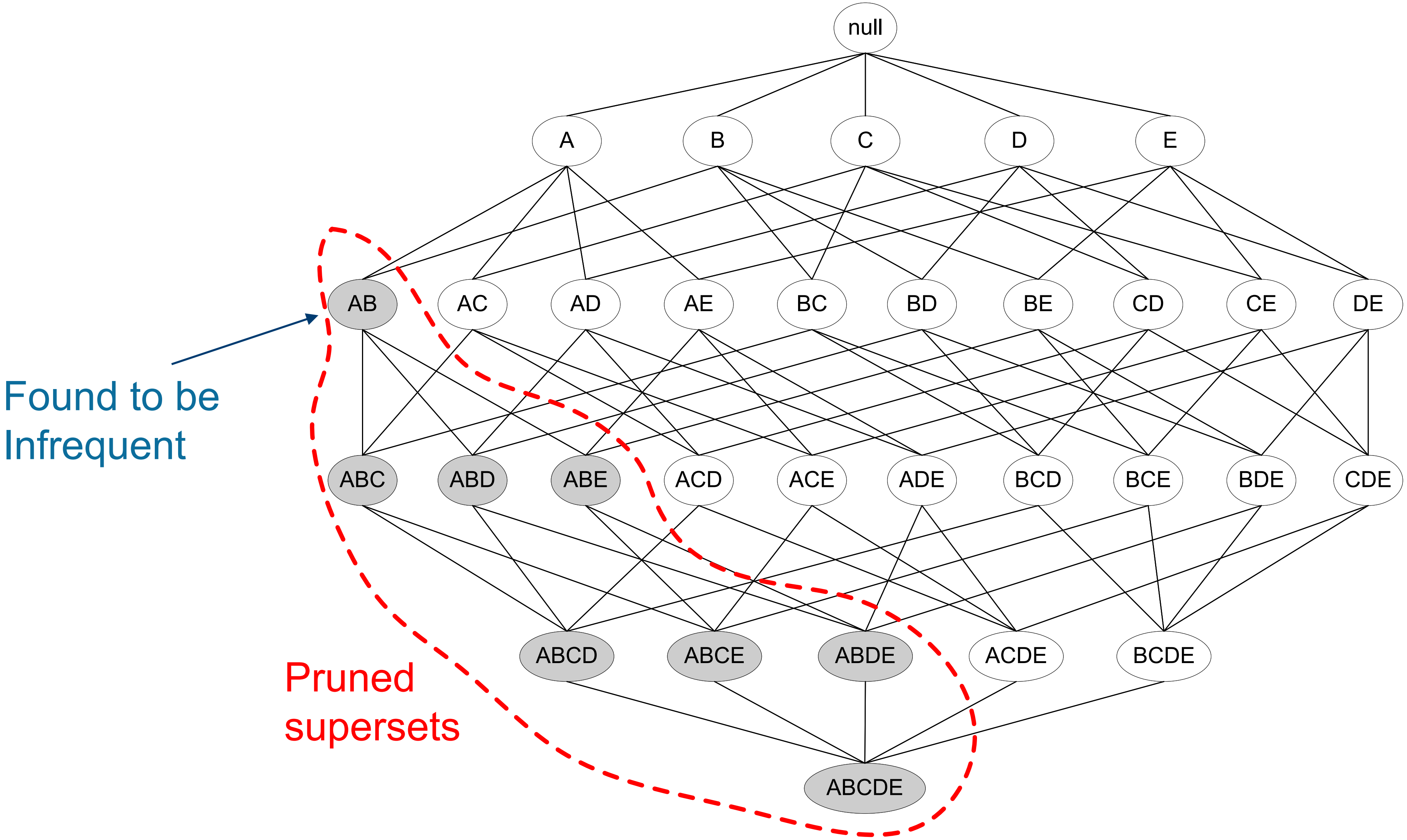
$$\forall X, Y : (X \subseteq Y) \Rightarrow s(X) \geq s(Y)$$

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

Using **anti-monotone** property, can we say that "if X is infrequent all supersets of X are infrequent" ?

A. True
B. False

Illustrating Apriori Principle



Illustrating Apriori Principle

Minimum Support = 3

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**

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,

$6 + 6 + 1 = 13$

Apriori Algorithm

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

Algorithm

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

Example: Apriori-based Mining

Database D

TID	Items
10	a, c, d
20	b, c, e
30	a, b, c, e
40	b, e

Min_sup=0.5

Scan D

L_1

1-candidates

Itemset	Sup
a	2
b	3
c	3
d	1
e	3

F_1

Freq 1-itemsets

Itemset	Sup
a	2
b	3
c	3
e	3

L_2

2-candidates

Itemset
ab
ac
ae
bc
be
ce

Scan D

Counting

Itemset	Sup
ab	1
ac	2
ae	1
bc	2
be	3
ce	2

Freq 2-itemsets

Itemset	Sup
ac	2
bc	2
be	3
ce	2

F_2

3-candidates

Itemset
bce

L_3

Scan D

Freq 3-itemsets

Itemset	Sup
bce	2

F_3

How to generate candidate set L_k from F_{k-1}

Step 1: Generation

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

Example:

F_3 = {ABC,ABD,ABE,ACD,BCD,BDE,CDE} , what is L_4?

- Merge(<u>AB</u>C, <u>AB</u>D) = <u>AB</u>CD
- Merge(<u>AB</u>C, <u>AB</u>E) = <u>AB</u>CE
- Merge(<u>AB</u>D, <u>AB</u>E) = <u>AB</u>DE

L_4 = {ABCD,ABCE,ABDE}

Can we merge(**A**BD,**A**CD) in F_3?

A. Yes
B. No

How to generate candidate set L_k from F_{k-1}

Step 2: Pruning

- **Rule:** Prune an item from L_k if any of its subsets is not frequent (it is not in F_{k-1})

Example:

- Let F_3 = {ABC,ABD,ABE,ACD,BCD,BDE,CDE} be the set of frequent 3-itemsets
- L_4 = {ABCD,ABCE,ABDE} is the set of candidate 4-itemsets generated (from previous slide)

Candidate pruning examples:

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

After candidate pruning: L_4 = {ABCD}

Example: Apriori-based Mining

Database D

TID	Items
10	a, c, d
20	b, c, e
30	a, b, c, e
40	b, e

Min_sup=0.5

Scan D

L_1

1-candidates

Itemset	Sup
a	2
b	3
c	3
d	1
e	3

F_1

Freq 1-itemsets

Itemset	Sup
a	2
b	3
c	3
e	3

L_2

2-candidates

Itemset
ab
ac
ae
bc
be
ce

Scan D

Counting

Itemset	Sup
ab	1
ac	2
ae	1
bc	2
be	3
ce	2

Freq 2-itemsets

Itemset	Sup
ac	2
bc	2
be	3
ce	2

F_2

3-candidates

Itemset
bce

L_3

Scan D

Freq 3-itemsets

Itemset	Sup
bce	2

F_3

Know you know how to create L_3 from F_2 😄

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

Support Counting of Candidate Itemsets

Scan the database of transactions to determine the support of each candidate itemset

- Must match every candidate itemset against every transaction, which is an expensive operation

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

Candidates

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

Support Counting of Candidate Itemsets

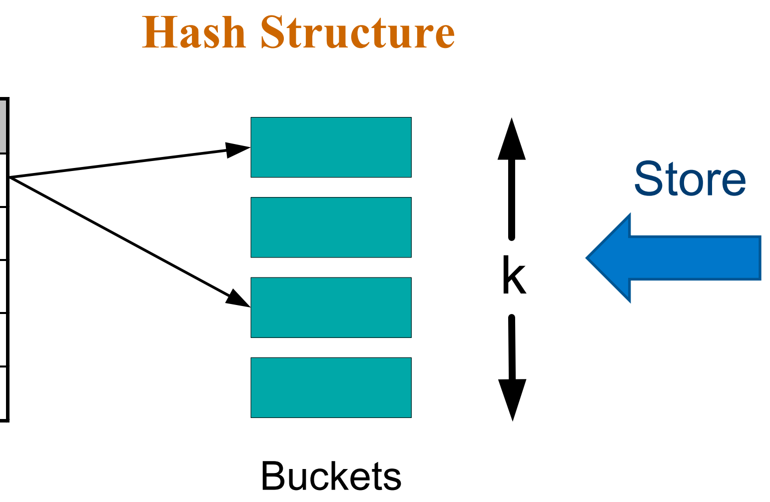
To reduce number of comparisons, store the candidate itemsets in a hash structure

- 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



Candidates

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

Support Counting Using a Hash Tree

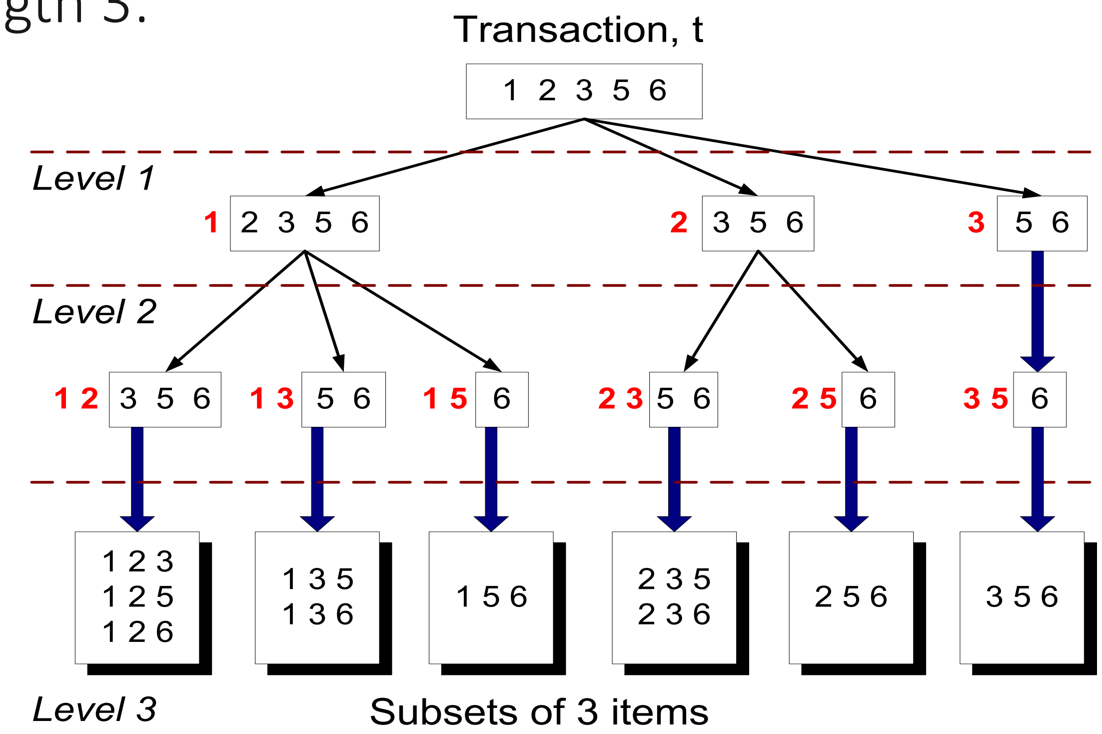
Suppose you have 15 candidate itemsets of length 3:

{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}

If the transaction is (1,2,3,5,6), how many of these itemsets should be checked for possible count increment?



Can we do the better job?

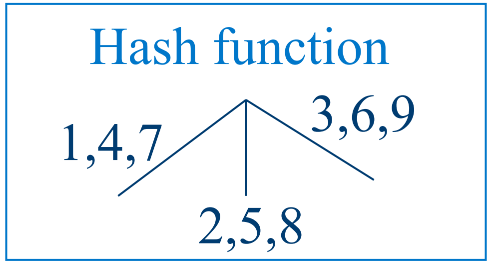
Support Counting Using a Hash Tree

Suppose you have 15 candidate itemsets of length 3:

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

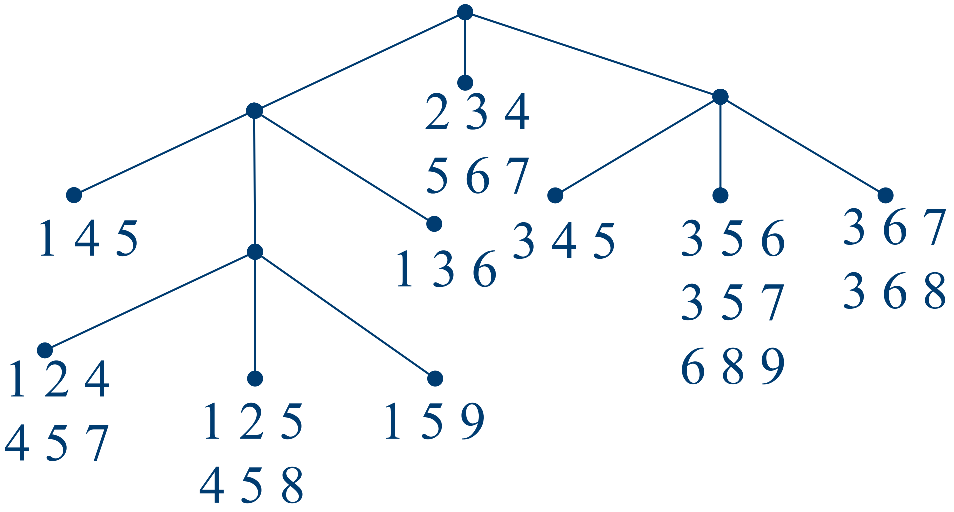
{1 5 9}, {1 3 6}, {2 3 4}, {5 6 7}, {3 4 5},

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



You need:

- Hash function (e.g., h(p) = p mod 3, and max leaf size = 3)
- Max leaf size: max number of itemsets stored in a leaf node



If number of candidate itemsets exceeds max leaf size, split the node)

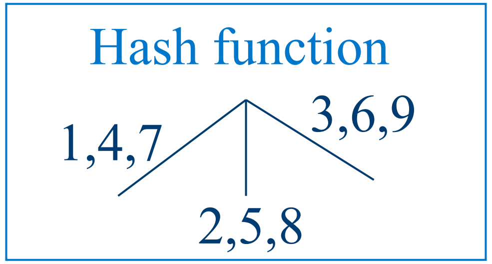
Support Counting Using a Hash Tree

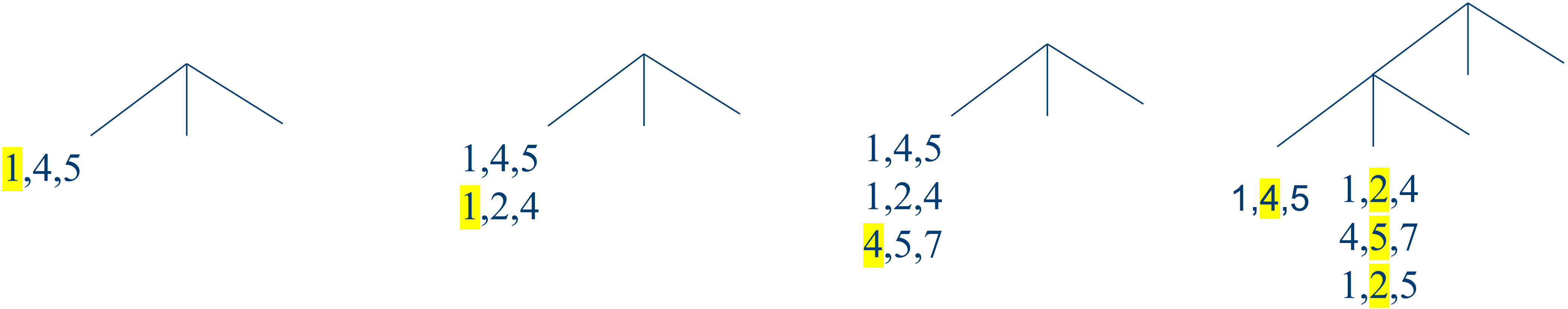
Suppose you have 15 candidate itemsets of length 3:

{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}





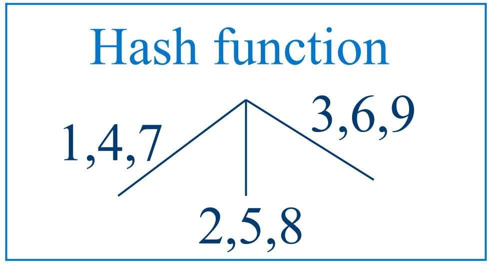
Support Counting Using a Hash Tree

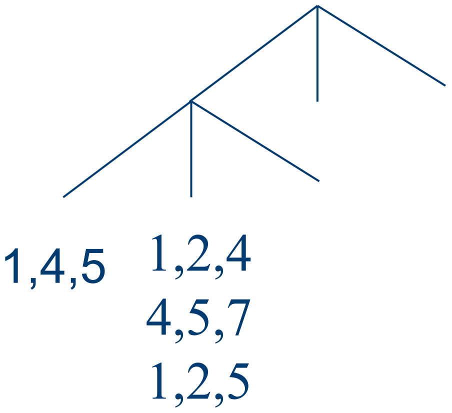
Suppose you have 15 candidate itemsets of length 3:

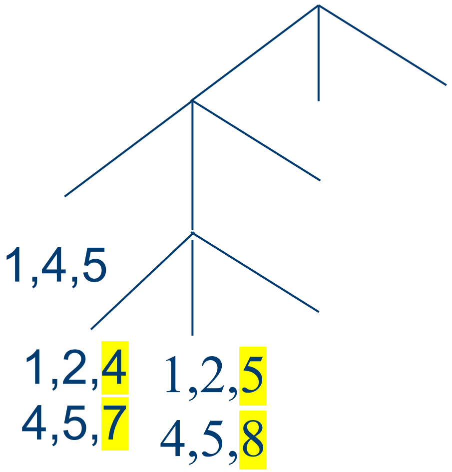
{1 4 5}, {1 2 4}, {4 5 7}, {1 2 5}, {4 5 8},

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

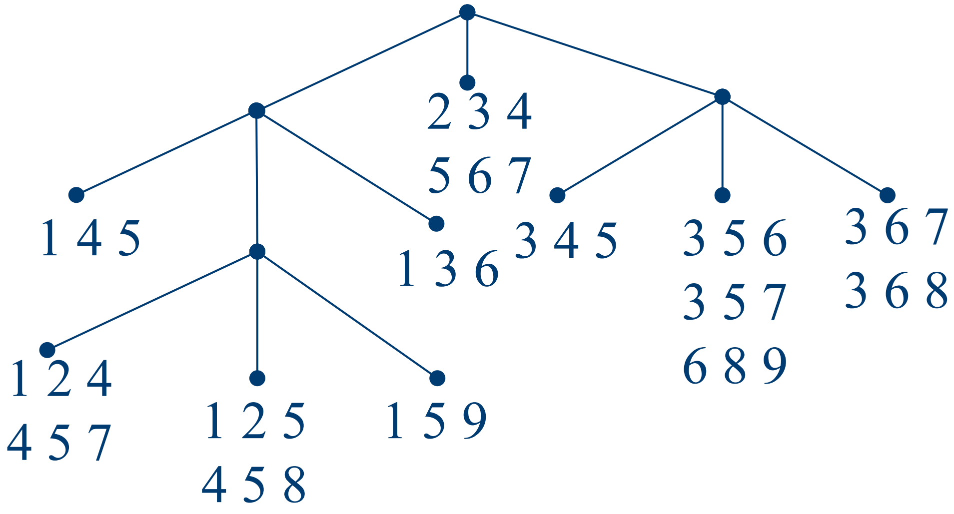
{3 5 6}, {3 5 7}, {6 8 9}, {3 6 7}, {3 6 8}







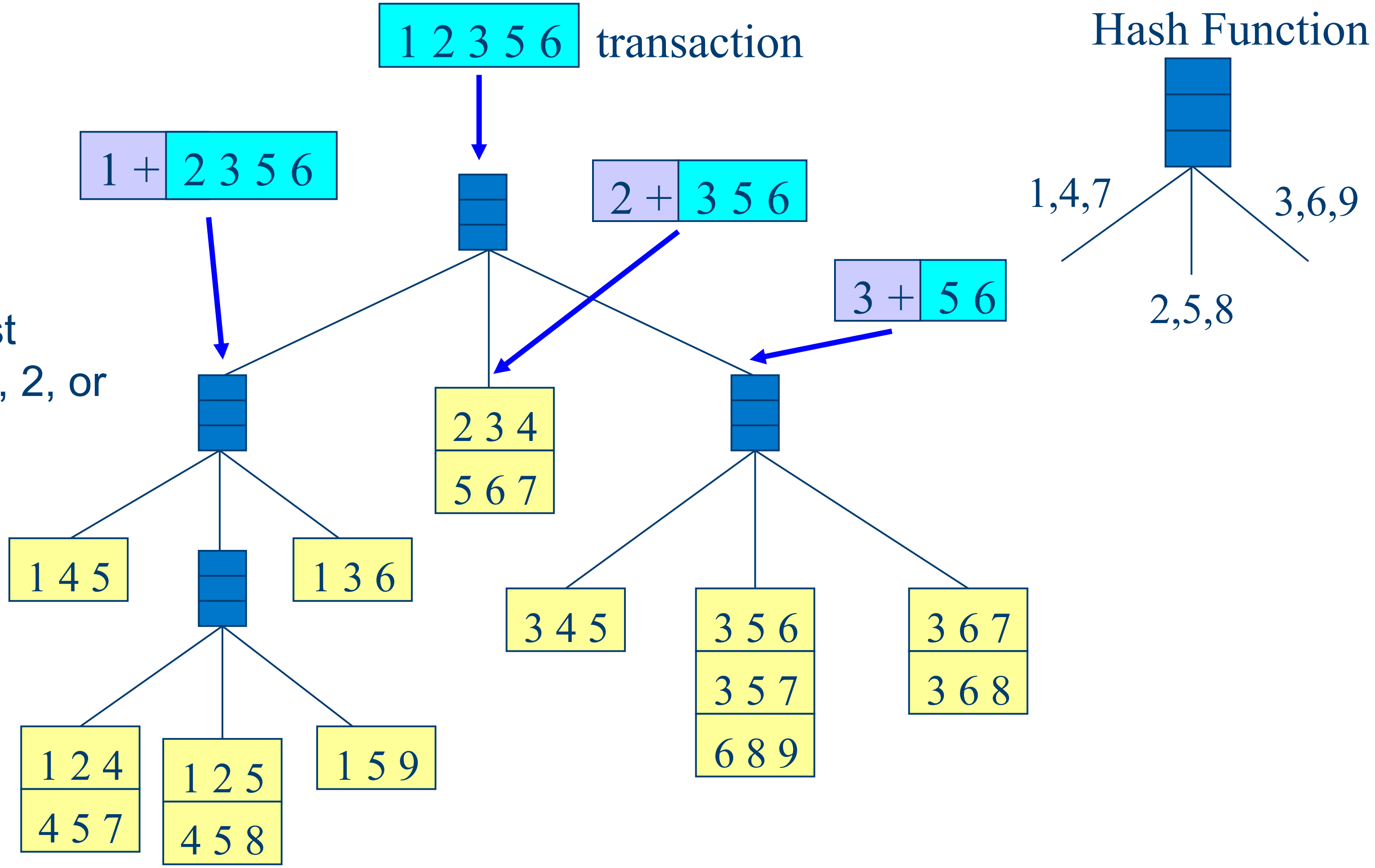
…



Subset Operation Using Hash Tree

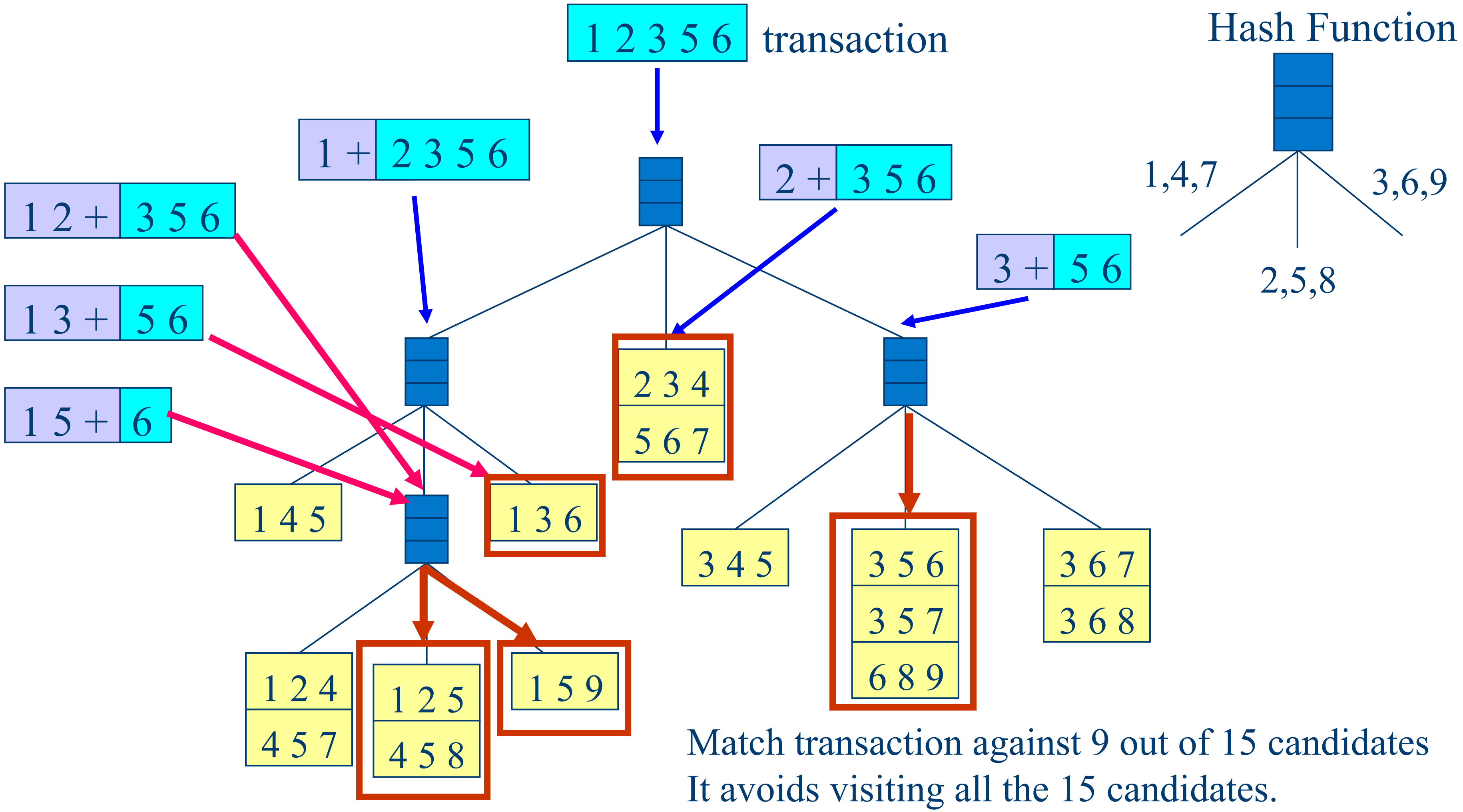
If the transaction is (1,2,3,5,6), how many of these itemsets should be checked for possible count increment?

All the 3-itemsets belonging to t must begin with items 1, 2, or 3



Subset Operation Using Hash Tree

If the transaction is (1,2,3,5,6), how many of these itemsets should be checked for possible count increment?



Match transaction against 9 out of 15 candidates
It avoids visiting all the 15 candidates.

Rule Generation

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

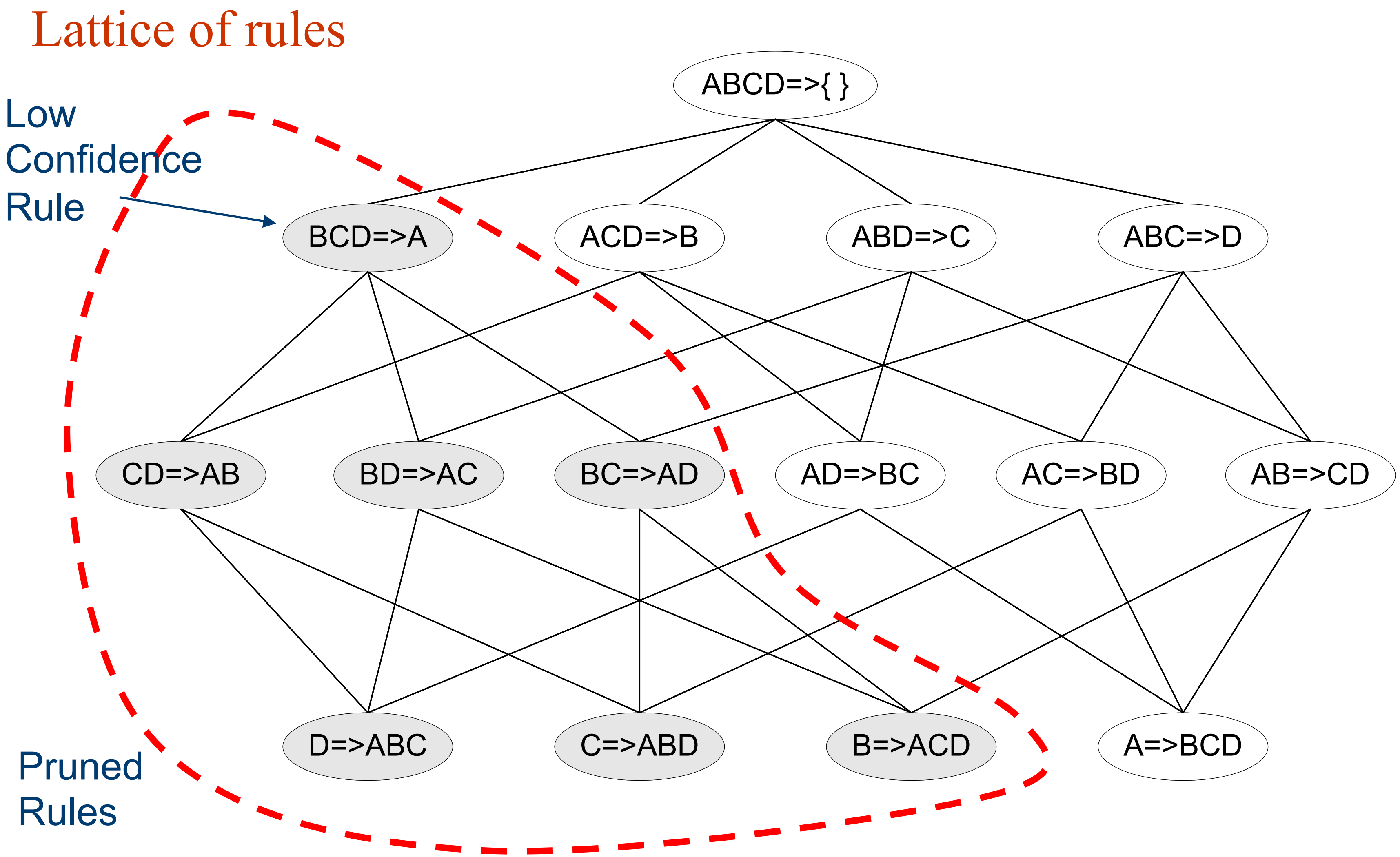
- How to efficiently generate rules from frequent itemsets?
  - In general, confidence **does** not have an anti-monotone property
    - $c(ABC \rightarrow D)$ can be larger or smaller than $c(AB \rightarrow D)$

  - 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. the number of items on the RHS of the rule

Rule Generation for Apriori Algorithm



Advanced Topics

Compact Representation of Frequent Itemsets

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

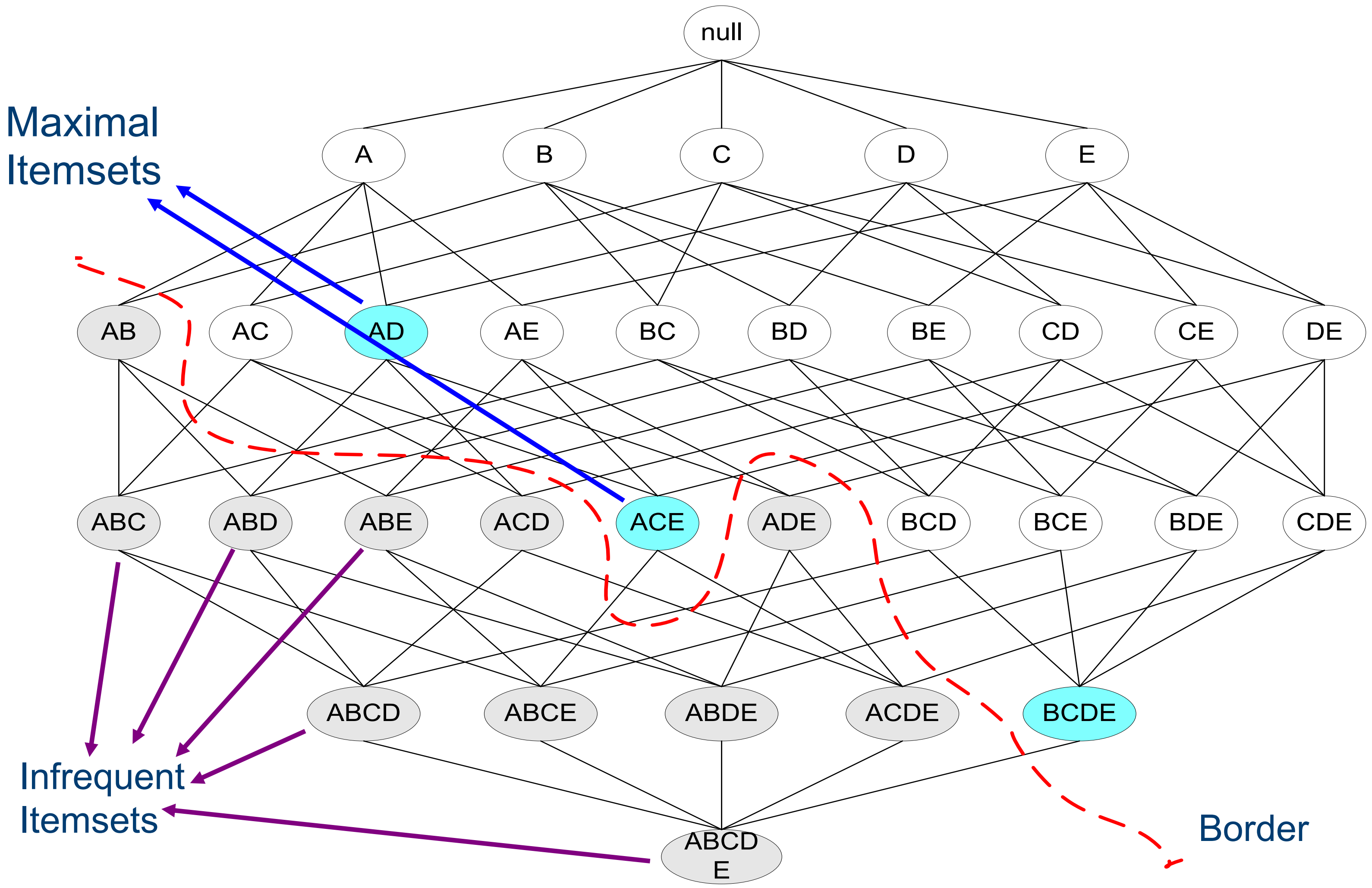
TID	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1

- Number of frequent itemsets $= 3 \times \sum_{k=1}^{10} \binom{10}{k}$

- Need a compact representation

Maximal Frequent Itemset

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



Closed Itemset

- An itemset is **closed** if none of its immediate supersets has the same support as the itemset
- Example:

TID	Items
1	{A,B}
2	{B,C,D}
3	{A,B,C,D}
4	{A,B,D}
5	{A,B,C,D}

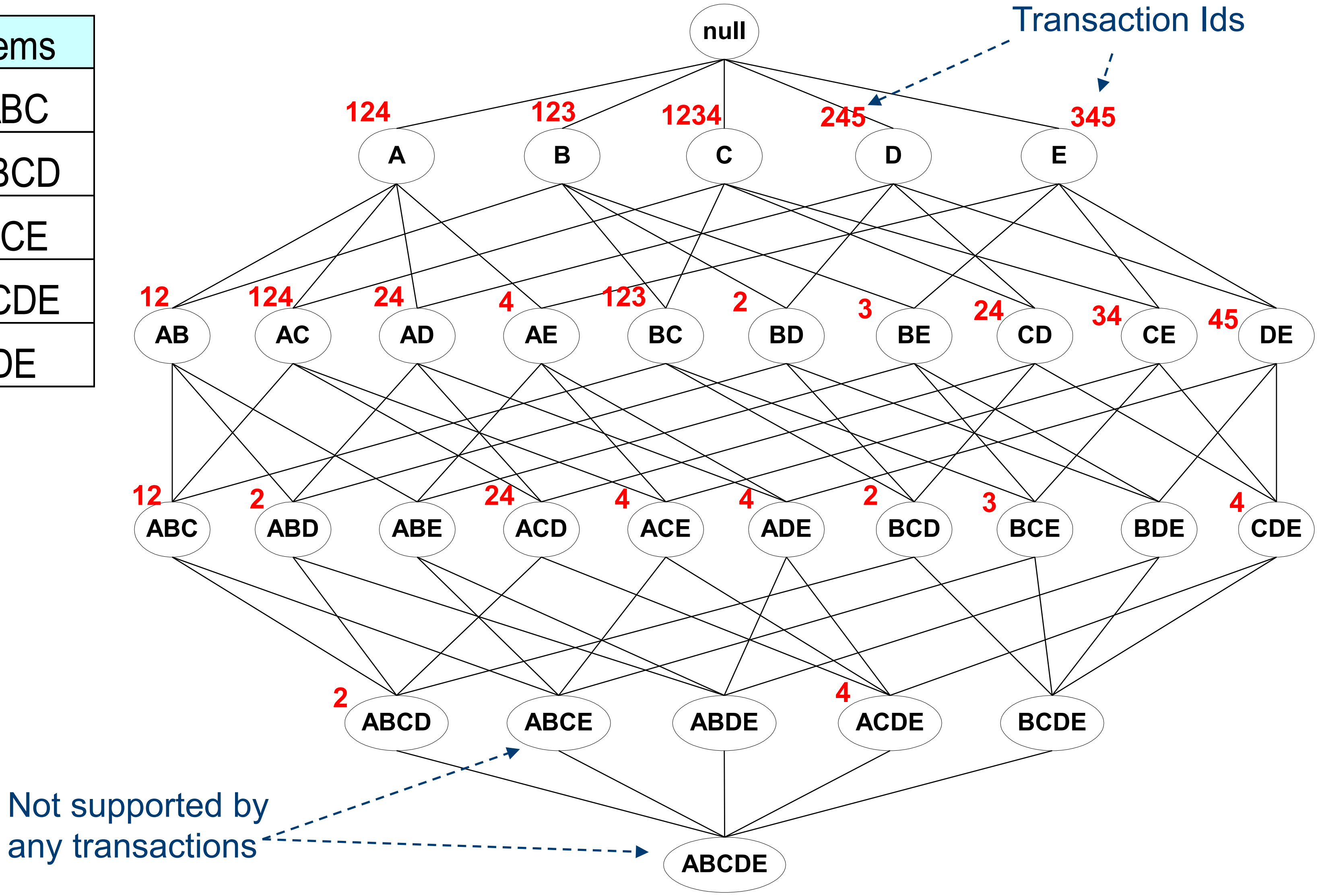
closed

Itemset	Support
{A}	4
{B}	5
{C}	3
{D}	4
{A,B}	4
{A,C}	2
{A,D}	3
{B,C}	3
{B,D}	4
{C,D}	3

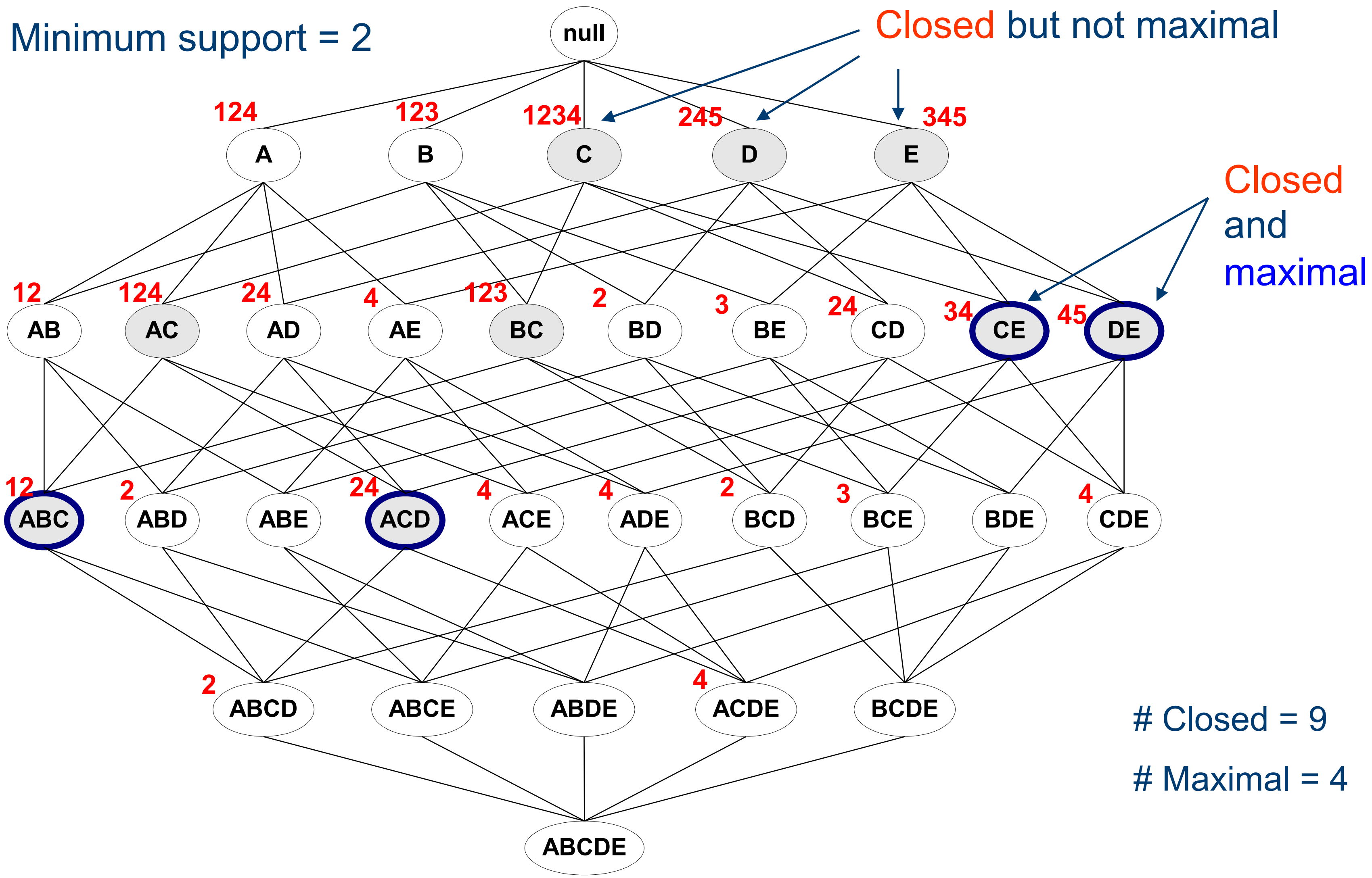
Itemset	Support
{A,B,C}	2
{A,B,D}	3
{A,C,D}	2
{B,C,D}	3
{A,B,C,D}	2

Maximal vs Closed Itemsets

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



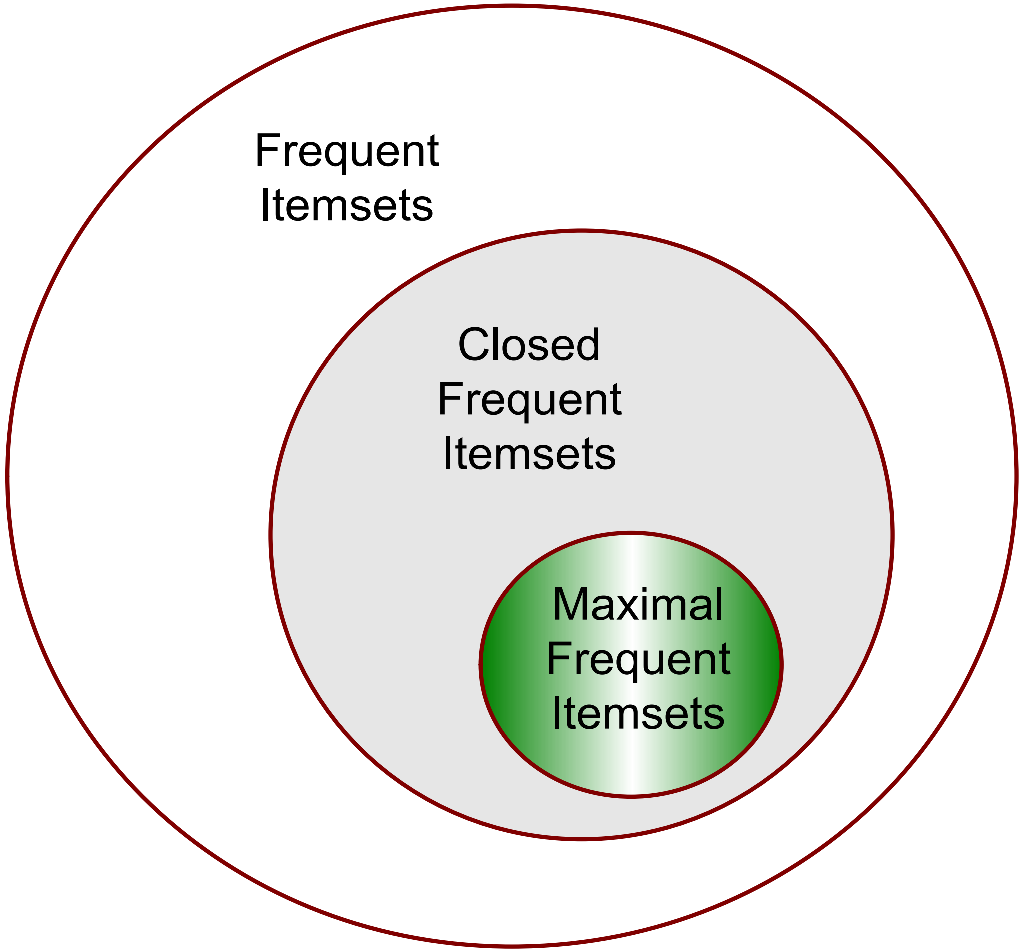
Maximal vs Closed Frequent Itemsets



Closed = 9

Maximal = 4

Maximal vs Closed Itemsets



FP-growth Algorithm

- To discover frequent itemsets by a compressed representation of the database using an FP-tree (Frequent Pattern Tree)

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

- It does not employ the generate-and-test paradigm of the Apriori algorithm.

FP-growth Ideas

- Grow long patterns from short ones using local frequent items
  - "abc" is a frequent pattern
  - Get all transactions having "abc":

    DB|abc (projected database on abc)

  - "d" is a local frequent item in DB|abc → abcd is a frequent pattern
  - Get all transactions having "abcd" (projected database on "abcd") and find longer itemsets

FP-growth Ideas

- 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: examine sub-database (conditional pattern base) 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_sup= 50%

Freq=> f: 4,c:4,a:3,b:3,m:3, p:3

Steps:

1. Scan DB once, find frequent 1-itemset (single item pattern)
2. Order frequent items in frequency descending order: f, c, a, b, m, p (L-order)
3. Process DB based on L-order

a	3	~~i~~	~~1~~
b	3	~~j~~	~~1~~
c	4	~~k~~	~~1~~
~~d~~	~~1~~	~~l~~	~~2~~
~~e~~	~~1~~	m	3
f	4	~~n~~	~~1~~
~~g~~	~~1~~	~~o~~	~~2~~
~~h~~	~~1~~	p	3

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}

{}

Header Table

Item	*frequency*	*head*
f	*0*	*nil*
c	*0*	*nil*
a	*0*	*nil*
b	*0*	*nil*
m	*0*	*nil*
p	*0*	*nil*

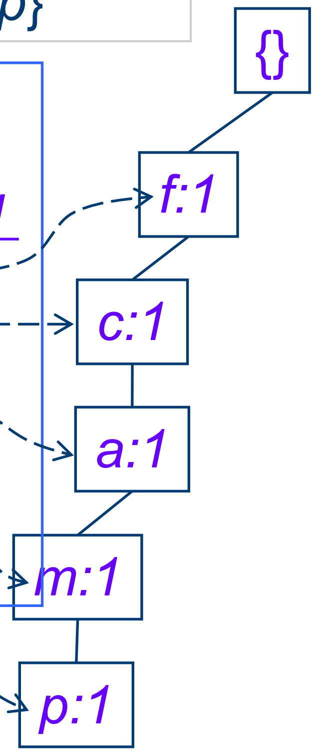
Initial FP-tree

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*}

Header Table

Item	*frequency*	*head*
f	*1*	
c	*1*	
a	*1*	
b	*0*	*nil*
m	*1*	
p	*1*	



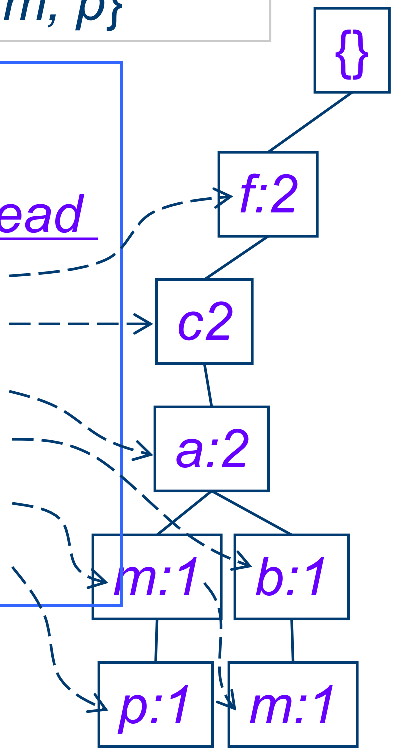
Insert {*f, c, a, m, p*}

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}*

Header Table

Item	*frequency*	*head*
f	*2*	
c	*2*	
a	*2*	
b	*1*	
m	*2*	
p	*1*	



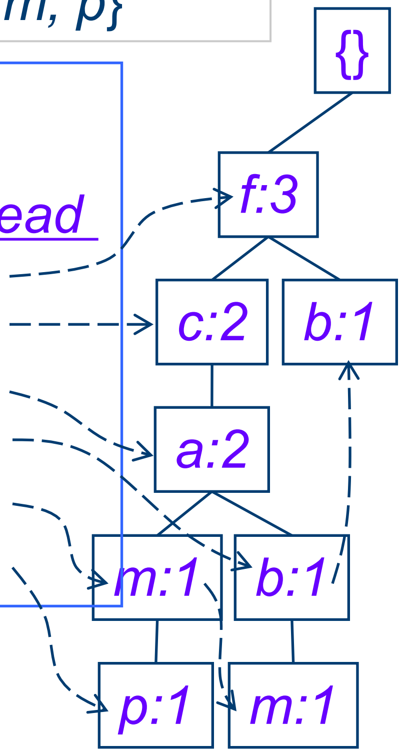
Insert *{f, c, a, b, m}*

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*}

Header Table

Item	*frequency*	*head*
f	*3*	
c	*2*	
a	*2*	
b	*2*	
m	*2*	
p	*1*	



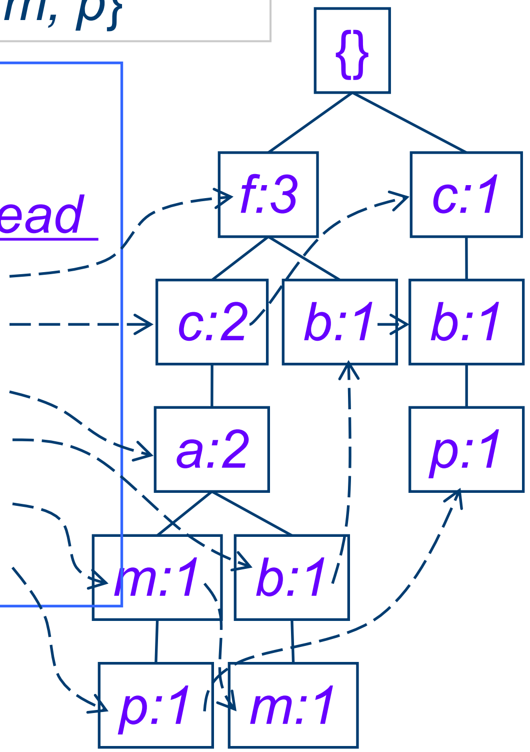
Insert *{f, b}*

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*}

Header Table

Item	*frequency*	*head*
f	*3*	
c	*3*	
a	*2*	
b	*3*	
m	*2*	
p	*2*	



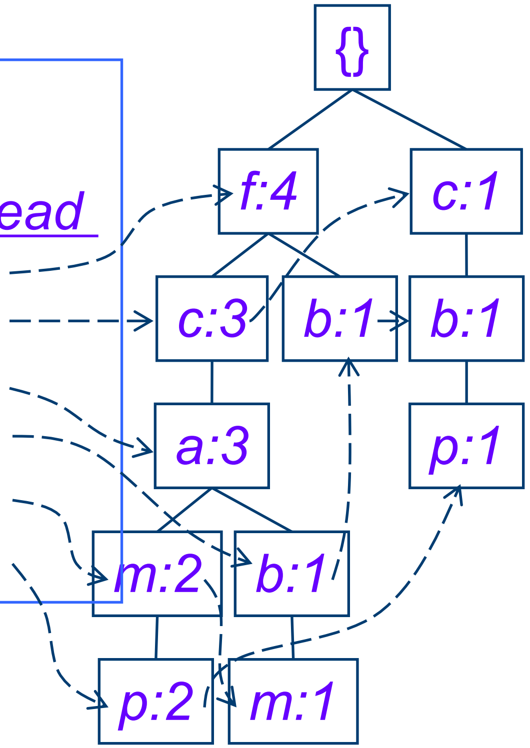
Insert {*c, b, p*}

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}*

Header Table

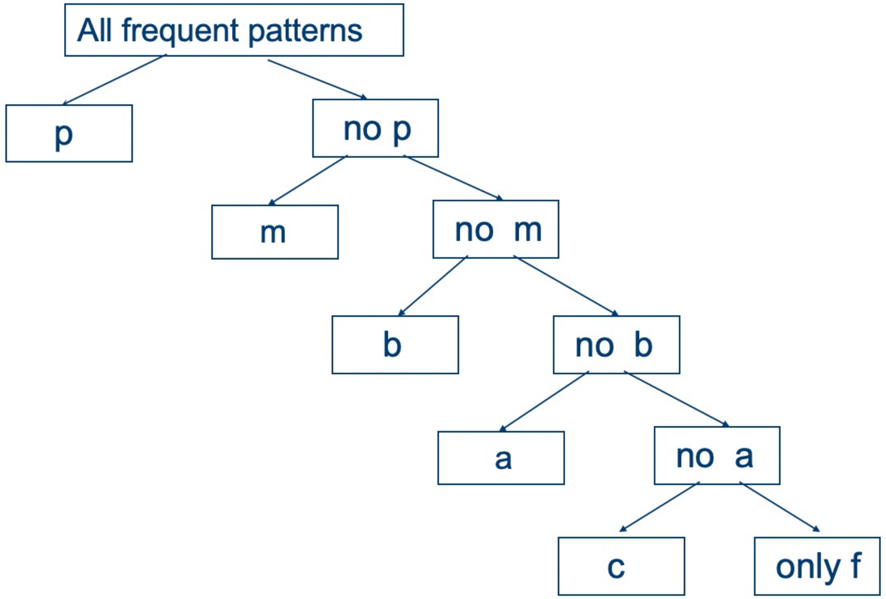
Item	*frequency*	*head*
f	*4*	
c	*4*	
a	*3*	
b	*3*	
m	*3*	
p	*3*	



Insert *{f, c, a, m, p}*

Mining Frequent Patterns Using FP-tree

- General idea (divide-and-conquer)
  - partition the set of frequent patterns
  - build *conditional pattern base* and *conditional FP-tree* for each partition
- Frequent patterns can be partitioned into subsets according to L-order
  - L-order=f-c-a-b-m-p
  - Patterns containing p
  - Patterns having m but no p
  - Patterns having b but no m or p
  - …
  - Patterns having c but no a nor b, m, p
  - Pattern f



Mining Frequent Patterns Using FP-tree

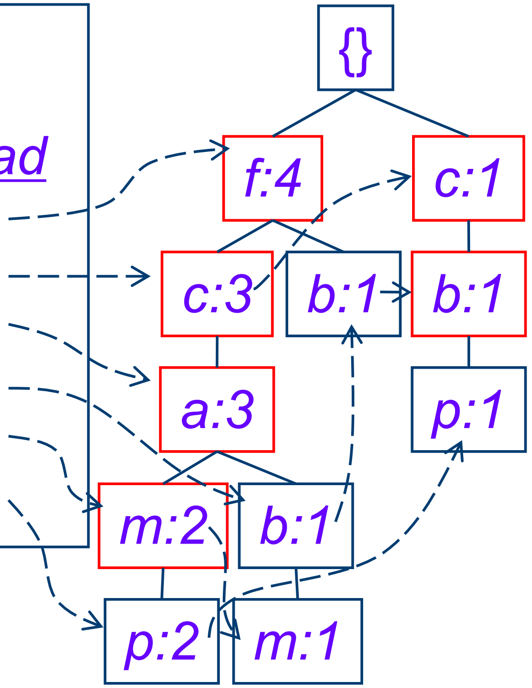
- Step 1 : Construct conditional pattern base for each item in header table
- Step 2: Construct conditional FP-tree from each conditional pattern-base
- Step 3: Recursively mine conditional FP-trees and grow frequent patterns obtained so far
  - If conditional FP-tree contains **a single path**, simply **enumerate all patterns**

Step 1: Construct Conditional Pattern Base

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

Header Table

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



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*

Step 2: Construct Conditional FP-tree

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

min_sup= 50%

transaction =5

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*

TDB|$_p$ →

fcam
fcam
cb

~~f~~	~~2~~
c	3
~~a~~	~~2~~
~~m~~	~~2~~
~~b~~	~~1~~

p-conditional FP-tree

Item	*frequency*	*head*
c	*3*	

{}

c:3

Local frequent item: c:3

Frequent patterns containing p

Freq=> cp: 3

Step 2: Construct Conditional FP-tree

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

min_sup= 50%

transaction =5

Conditional pattern bases

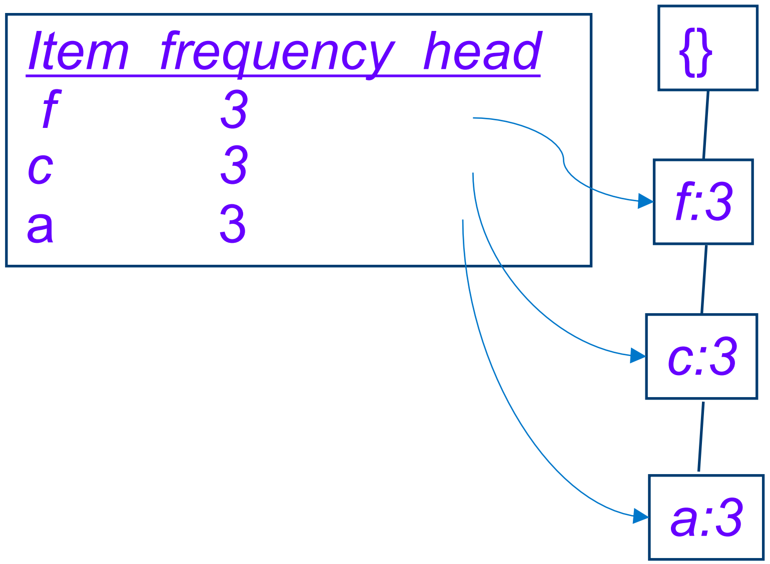
item	cond. pattern base
c	*f:3*
a	*fc:3*
b	*fca:1, f:1, c:1*
m	*fca:2, fcab:1*

TDB|m → *m*

fca
fca
fcab

f	3
c	3
a	3
~~b~~	~~1~~

p-conditional FP-tree



Frequent patterns containing m

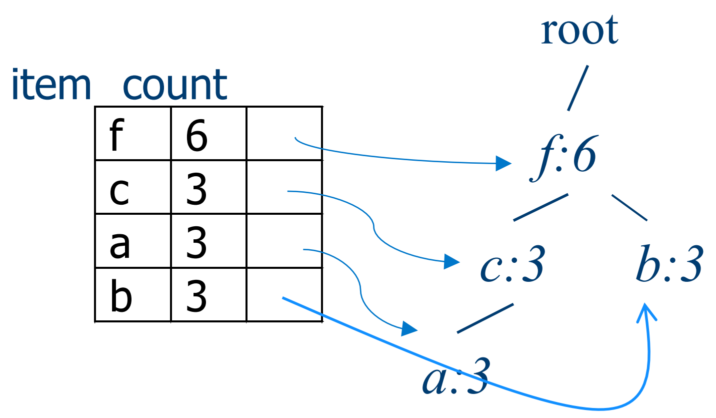
Freq=> fm, cm, am, fcm, fam, cam, fcam

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

Find Frequent Patterns Having Item m But No p (*more complex situation*)

- Suppose m-conditional pattern base is: *fca:3, fb:3*
- Local frequent items: *f:6, c:3, a:3, b: 3*
- Build m-conditional FP-tree
- First generate:
  - *fm: 6, cm: 3, am:3, bm:3*



Compute *ym*-conditional pattern bases:

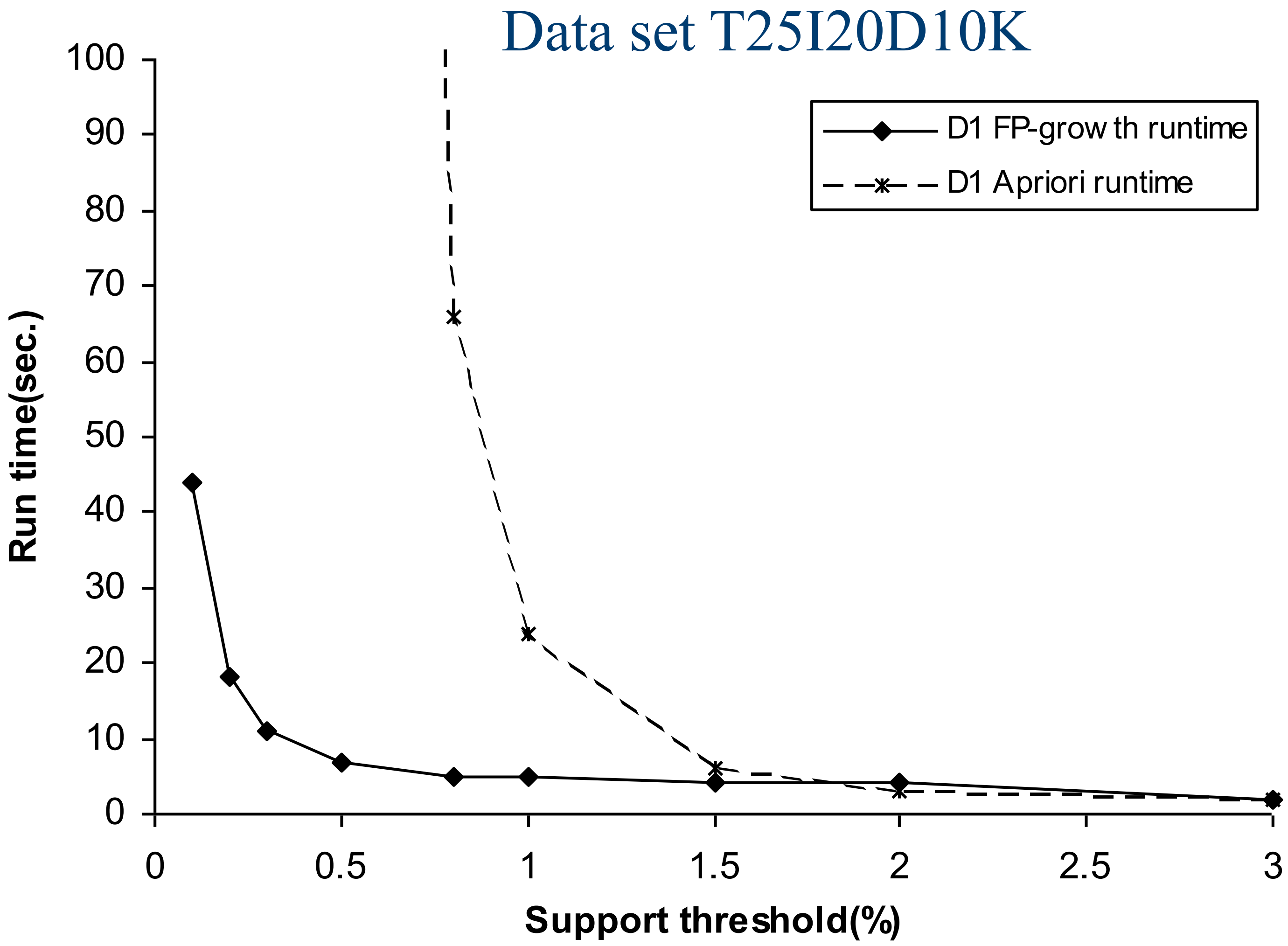
ym	conditional pattern base
bm	*f:3*
am	*fc:3*
cm	*f:3*

TDB|bm → root | *f:3* — **Freq: fbm**

TDB|am → root | *f:3* | *c:3* — **Freq: fam, cam, fcam**

TDB|cm → root | *f:3* — **Freq: fcm**

FP-growth vs. Apriori: Scalability With the Support Threshold



Why Is Frequent Pattern Growth Fast?

- 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 operations are counting and FP-tree building

Weaknesses of FP-growth

- Support dependent; cannot accommodate dynamic support threshold
- Cannot accommodate incremental DB update
- Mining requires recursive operations

Participant Leaders

Points	Participant	Points	Participant
3	Al-Azzawe, Mustafa		
3	Alibhai, Aleem		
3	Baek, Yong Joon		
3	Henderson, Connor		
3	Nagy, Peter		
3	Ramanathan, Sinthujan		
3	Tran, Luke		
2	Abdul, Moiz		
2	Alvarez-Grekoff, Mathieu Alexander		
2	Armstrong, Ben		