DATA MINING
LECTURE 3

Frequent Itemsets

Association Rules

This is how it all started…

- Rakesh Agrawal, Tomasz Imielinski, Arun N. Swami: Mining Association Rules between Sets of Items in Large Databases. SIGMOD Conference 1993: 207-216
- Rakesh Agrawal, Ramakrishnan Srikant: Fast Algorithms for Mining Association Rules in Large Databases. VLDB 1994: 487-499

- These two papers are credited with the birth of Data Mining
- For a long time people were fascinated with Association Rules and Frequent Itemsets
  - Some people (in industry and academia) still are.

Market-Basket Data

- A large set of items, e.g., things sold in a supermarket.
- A large set of baskets, each of which is a small set of the items, e.g., the things one customer buys on one day.

Market-Baskets – (2)

- Really, a general many-to-many mapping (association) between two kinds of things, where the one (the baskets) is a set of the other (the items)
  - But we ask about connections among "items," not "baskets."
- The technology focuses on common events, not rare events ("long tail").

Frequent Itemsets

- Given a set of transactions, find combinations of items (itemsets) that occur frequently

Support $s(I)$: number of transactions that contain itemset I

Market-Basket transactions

Items: {Bread, Milk, Diaper, Beer, Eggs, Coke}

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**

Examples of frequent itemsets $s(I) \geq 3$

{Bread}: 4
{Milk} : 4
{Diaper} : 4
{Beer}: 3
{Diaper, Beer} : 3
{Milk, Bread} : 3

Applications – (1)

- Items = products; baskets = sets of products someone bought in one trip to the store.

- Example application: given that many people buy beer and diapers together:
  - Run a sale on diapers; raise price of beer.
- Only useful if many buy diapers & beer.

Applications – (2)

- Baskets = Web pages; items = words.

- Example application: Unusual words appearing together in a large number of documents, e.g., “Brad” and “Angelina,” may indicate an interesting relationship.

Applications – (3)

- Baskets = sentences; items = documents containing those sentences.

- Example application: Items that appear together too often could represent plagiarism.
- Notice items do not have to be “in” baskets.

Definition: Frequent Itemset

- **Itemset**
  - A collection of one or more items
    - Example: {Milk, Bread, Diaper}
  - k-itemset
    - An itemset that contains k items
- **Support (σ)**
  - **Count**: Frequency of occurrence of an itemset
  - E.g. σ({Milk, Bread,Diaper}) = 2
  - **Fraction**: Fraction of transactions that contain an itemset
  - E.g. s({Milk, Bread, Diaper}) = 40%
- **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**

$$s(I) \geq \text{minsup}$$

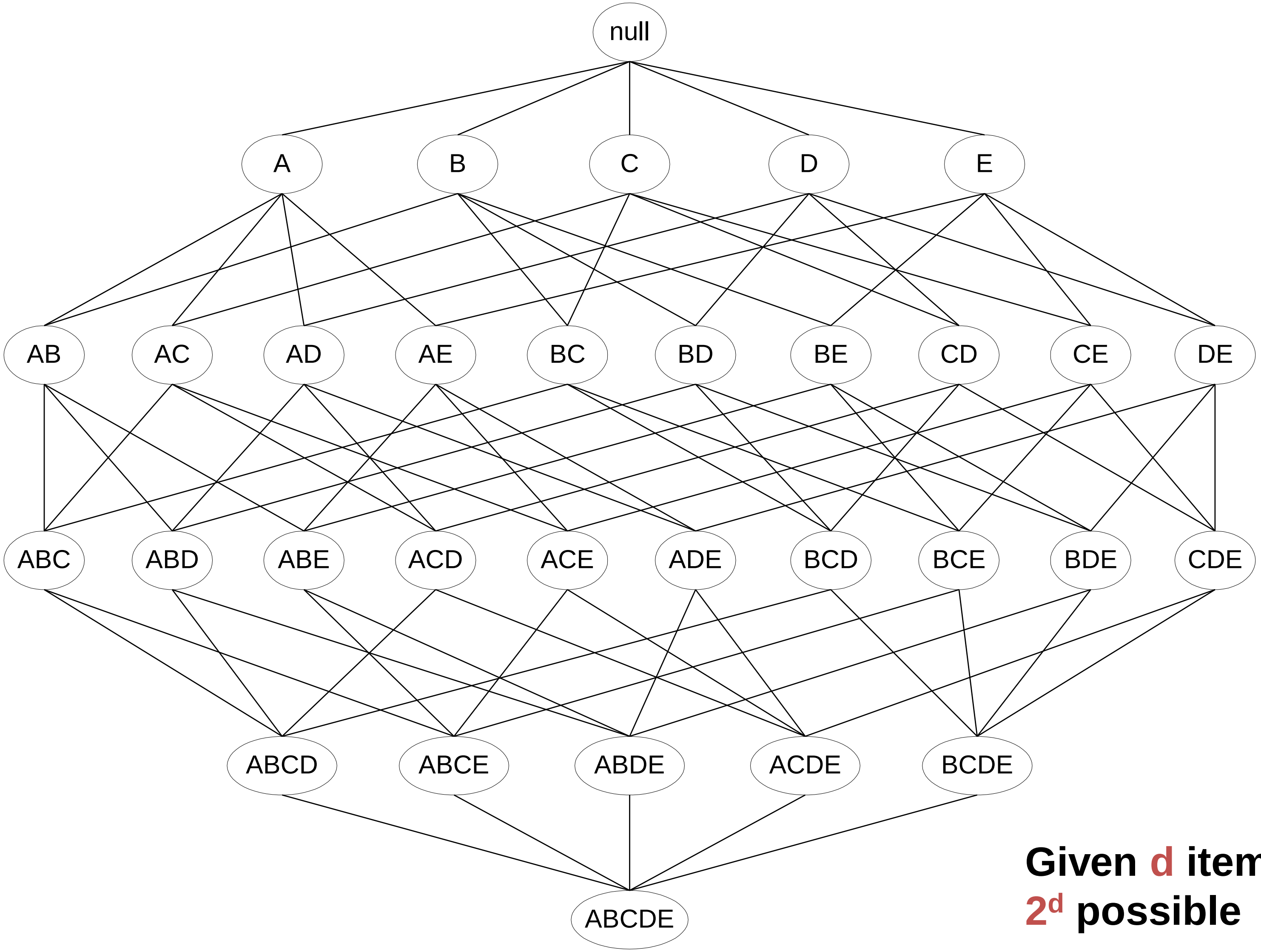
Mining Frequent Itemsets task

- Input: A set of transactions T, over a set of items I
- Output: All itemsets with items in I having
  - support ≥ *minsup* threshold

- Problem parameters:
  - N = |T|: number of transactions
  - d = |I|: number of (distinct) items
  - w: max width of a transaction
  - Number of possible itemsets? $M = 2^d$

- Scale of the problem:
  - WalMart sells 100,000 items and can store billions of baskets.
  - The Web has billions of words and many billions of pages.

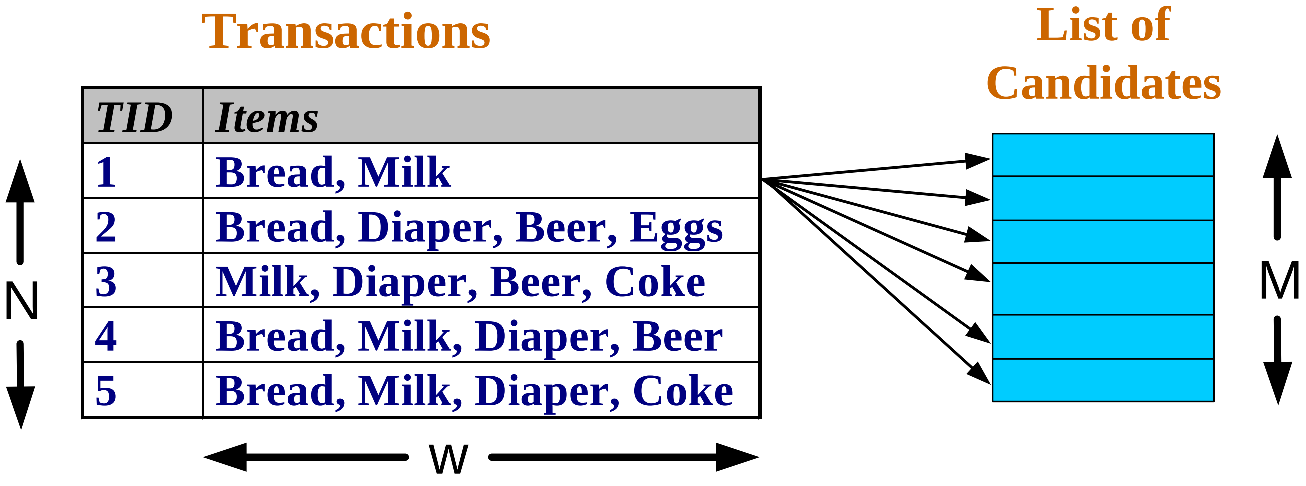
The itemset lattice



Given d items, there are 2^d possible itemsets

A Naïve Algorithm

- Brute-force approach, each itemset is a candidate :
  - Consider each itemset in the lattice, and count the support of each candidate by scanning the data
  - Time Complexity ~ O(NMw) , Space Complexity ~ O(M)
- OR
  - Scan the data, and for each transaction generate all possible itemsets. Keep a count for each itemset in the data.
  - Time Complexity ~ $O(N2^w)$, Space Complexity ~ O(M)
- Expensive since $M = 2^d$!!!



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

List of Candidates

Computation Model

- Typically, data is kept in flat files rather than in a database system.
  - Stored on disk.
  - Stored basket-by-basket.
  - Expand baskets into pairs, triples, etc. as you read baskets.
    - Use k nested loops to generate all sets of size k.

Example file: retail

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
30 31 32
33 34 35
36 37 38 39 40 41 42 43 44 45 46
38 39 47 48
38 39 48 49 50 51 52 53 54 55 56 57 58
32 41 59 60 61 62
3 39 48
63 64 65 66 67 68
32 69
48 70 71 72
39 73 74 75 76 77 78 79
36 38 39 41 48 79 80 81
82 83 84
41 85 86 87 88
39 48 89 90 91 92 93 94 95 96 97 98 99 100 101
36 38 39 48 89
39 41 102 103 104 105 106 107 108
38 39 41 109 110
39 111 112 113 114 115 116 117 118
119 120 121 122 123 124 125 126 127 128 129 130 131 132 133
48 134 135 136
39 48 137 138 139 140 141 142 143 144 145 146 147 148 149
39 150 151 152
38 39 56 153 154 155
```

Example: items are positive integers, and each basket corresponds to a line in the file of space separated integers

Computation Model – (2)

- The true cost of mining disk-resident data is usually the number of disk I/O's.
- In practice, association-rule algorithms read the data in passes – all baskets read in turn.
- Thus, we measure the cost by the number of passes an algorithm takes.

Main-Memory Bottleneck

- For many frequent-itemset algorithms, main memory is the critical resource.
    - As we read baskets, we need to count something, e.g., occurrences of pairs.
    - The number of different things we can count is limited by main memory.
    - Swapping counts in/out is a disaster (why?).

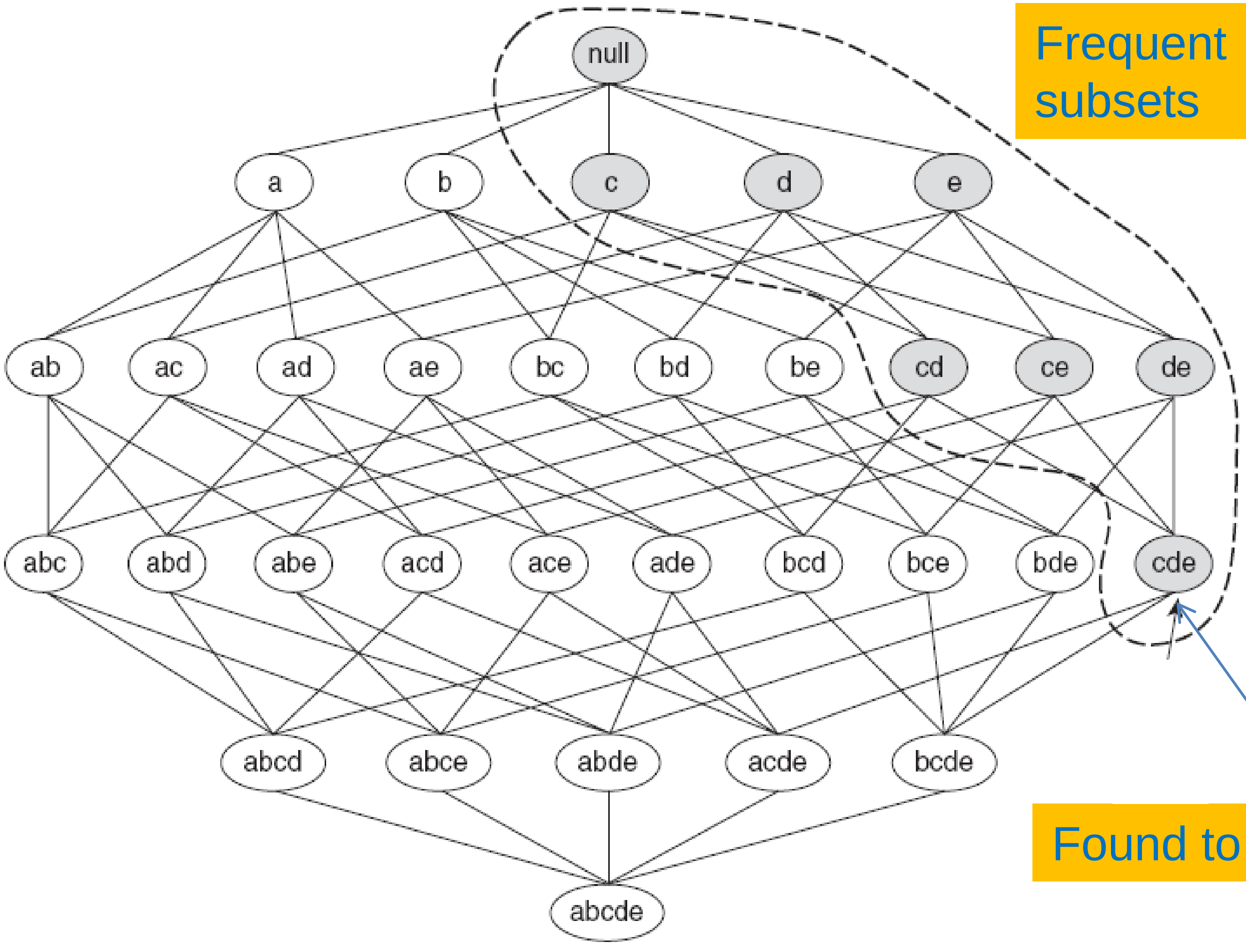
The Apriori Principle

- Apriori principle (Main observation):
  - If an itemset is frequent, then all of its subsets must also be frequent
  - If an itemset is not frequent, then all of its supersets cannot be frequent

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

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

Illustration of the Apriori principle

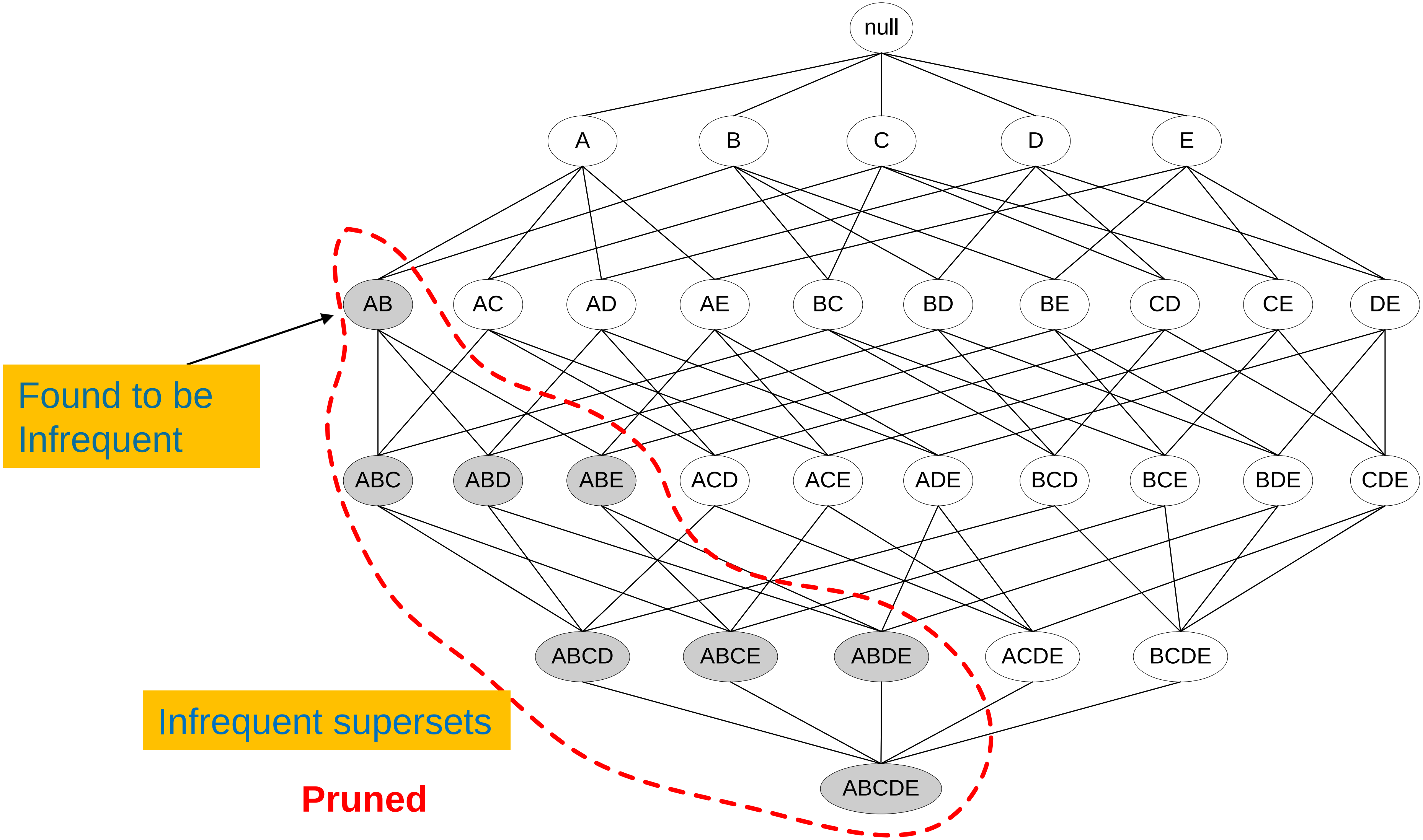


Frequent subsets

Found to be frequent

Figure 6.3. An illustration of the *Apriori* principle. If $\{c, d, e\}$ is frequent, then all subsets of this itemset are frequent.

Illustration of the Apriori principle



Found to be Infrequent

Infrequent supersets

Pruned

The Apriori algorithm

Level-wise approach

C_k = candidate itemsets of size k
L_k = frequent itemsets of size k

1. $k = 1$, C_1 = all items
2. While C_k not empty

Frequent itemset generation

3. Scan the database to find which itemsets in C_k are frequent and put them into L_k

Candidate generation

4. Use L_k to generate a collection of candidate itemsets C_{k+1} of size $k+1$
5. $k = k+1$

R. Agrawal, R. Srikant: "Fast Algorithms for Mining Association Rules",
Proc. of the 20th Int'l Conference on Very Large Databases, 1994.

Illustration of the 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

minsup = 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}	**2**

Only this triplet has all subsets to be frequent
But it is below the minsup threshold

If every subset is considered,

$$\binom{6}{1} + \binom{6}{2} + \binom{6}{3} = 6 + 15 + 20 = 41$$

With support-based pruning,

$$\binom{6}{1} + \binom{4}{2} + 1 \ = 6 + 6 + 1 = 13$$

Candidate Generation

- Basic principle (Apriori):
  - An itemset of size k+1 is candidate to be frequent only if all of its subsets of size k are known to be frequent
- Main idea:
  - Construct a candidate of size k+1 by combining frequent itemsets of size k
    - If k = 1, take the all pairs of frequent items
    - If k > 1, join pairs of itemsets that differ by just one item
    - For each generated candidate itemset ensure that all subsets of size k are frequent.

Generate Candidates C_{k+1}

- Assumption: The items in an itemset are ordered
  - E.g., if integers ordered in increasing order, if strings ordered in lexicographic order
    - The order ensures that if item y > x appears before x, then x is not in the itemset
- The items in L_k are also listed in an order

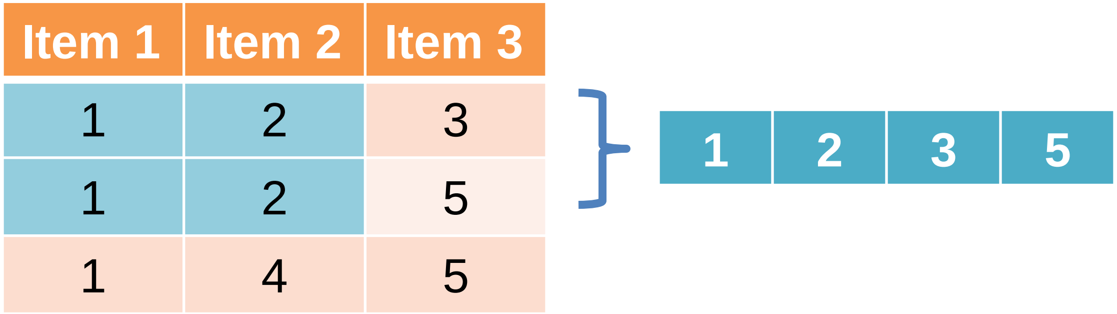
Create a candidate itemset of size k+1, by joining two itemsets of size k, that share the first k-1 items

Item 1	Item 2	Item 3
1	2	3
1	2	5
1	4	5

Generate Candidates C_{k+1}

- Assumption: The items in an itemset are ordered
  - E.g., if integers ordered in increasing order, if strings ordered in lexicographic order
    - The order ensures that if item y > x appears before x, then x is not in the itemset
- The items in L_k are also listed in an order

Create a candidate itemset of size k+1, by joining two itemsets of size k, that share the first k-1 items



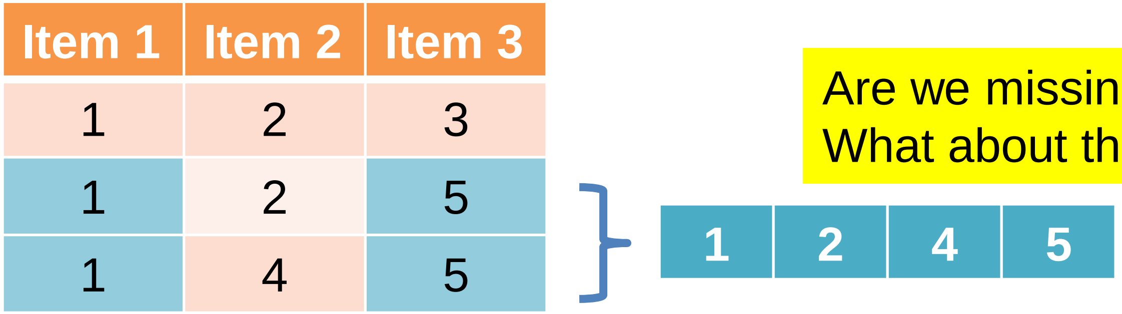
Item 1	Item 2	Item 3
1	2	3
1	2	5
1	4	5

1	2	3	5

Generate Candidates C_{k+1}

- Assumption: The items in an itemset are ordered
  - E.g., if integers ordered in increasing order, if strings ordered in lexicographic order
    - The order ensures that if item y > x appears before x, then x is not in the itemset
- The items in L_k are also listed in an order

Create a candidate itemset of size k+1, by joining two itemsets of size k, that share the first k-1 items



Item 1	**Item 2**	**Item 3**
1	2	3
1	2	5
1	4	5

1	2	4	5

Are we missing something?
What about this candidate?

Generating Candidates C_{k+1} in SQL

- ***self-join*** L_k

  insert into C_{k+1}

  select $p.item_1, p.item_2, \ldots, p.item_k, q.item_k$

  from $L_k\ p, L_k\ q$

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

Example I

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

item1	item2	item3
a	b	c
a	b	d
a	c	d
a	c	e
b	c	d

item1	item2	item3
a	b	c
a	b	d
a	c	d
a	c	e
b	c	d

$p.item_1 = q.item_1, p.item_2 = q.item_2, p.item_3 < q.item_3$

Example I

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

item1	item2	item3
a	b	c
a	b	d
a	c	d
a	c	e
b	c	d

item1	item2	item3
a	b	c
a	b	d
a	c	d
a	c	e
b	c	d

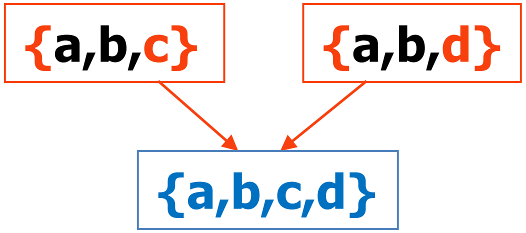
$p.item_1=q.item_1, p.item_2=q.item_2, p.item_3< q.item_3$

Example I

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

item1	item2	item3
a	b	c
a	b	d
a	c	d
a	c	e
b	c	d

item1	item2	item3
a	b	c
a	b	d
a	c	d
a	c	e
b	c	d



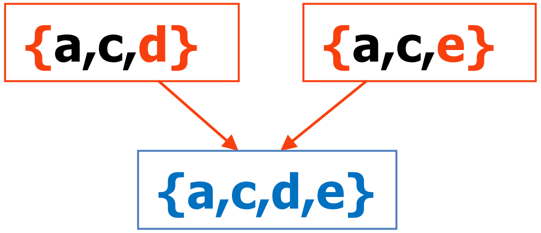
$$p.item_1=q.item_1, p.item_2=q.item_2, p.item_3< q.item_3$$

Example I

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

item1	item2	item3
a	b	c
a	b	d
a	c	d
a	c	e
b	c	d

item1	item2	item3
a	b	c
a	b	d
a	c	d
a	c	e
b	c	d



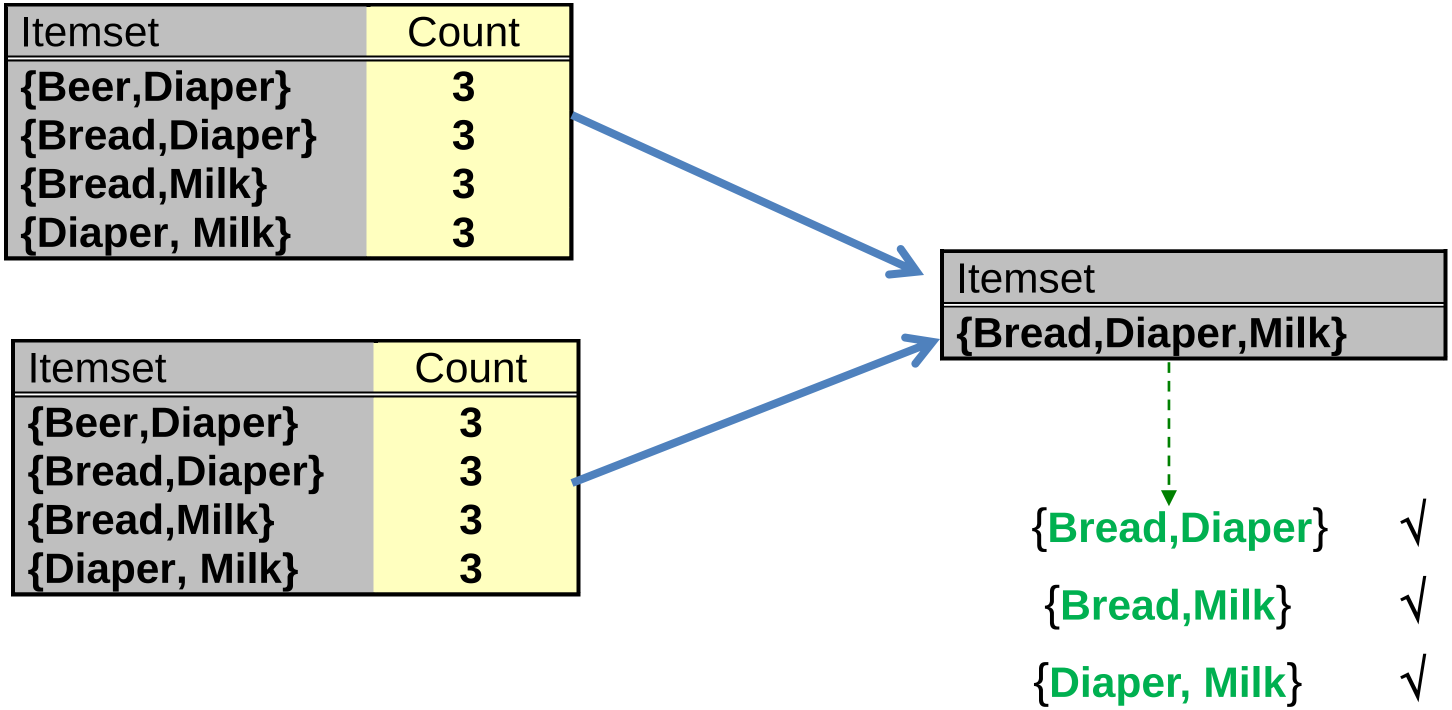
{a,c,d}

{a,c,e}

{a,c,d,e}

$p.item_1=q.item_1, p.item_2=q.item_2, p.item_3< q.item_3$

Example II



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

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

Itemset
{Bread,Diaper,Milk}

{**Bread,Diaper**}	√
{**Bread,Milk**}	√
{**Diaper, Milk**}	√

Generate Candidates C_{k+1}

- Are we done? Are all the candidates valid?

Item 1	Item 2	Item 3
1	2	3
1	2	5
1	4	5

1	2	3	5

Is this a valid candidate?

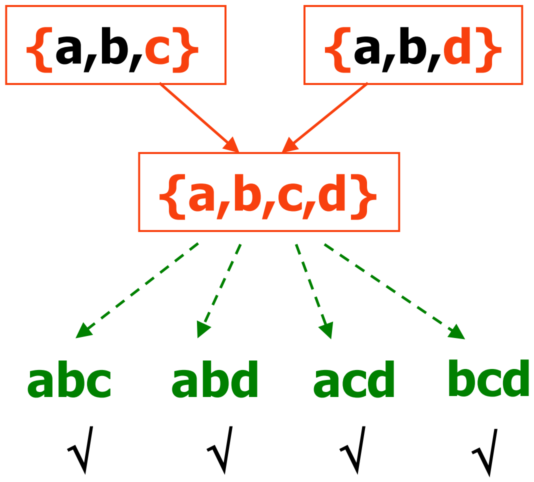
No. Subsets (1,3,5) and (2,3,5) should also be frequent

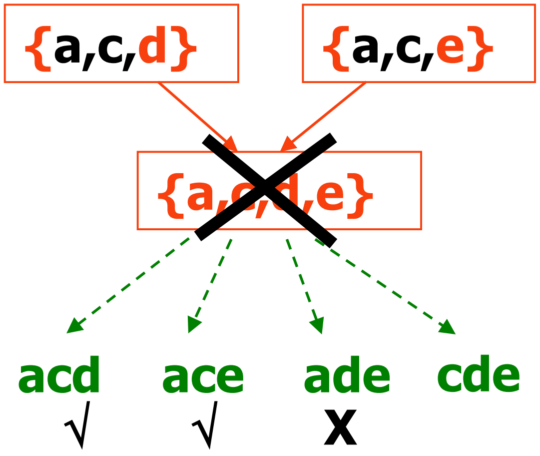
Apriori principle

- Pruning step:
  - For each candidate (k+1)-itemset create all subset k-itemsets
  - Remove a candidate if it contains a subset k-itemset that is not frequent

Example I

- L_3={*abc, abd, acd, ace, bcd*}
- ***Self-joining***: L_3*L_3
  - ***abcd*** from ***abc*** and ***abd***
  - ***acde*** from ***acd*** and ***ace***
- ***Pruning:***
  - ***abcd*** is kept since all subset itemsets are in L_3
  - ***acde*** is removed because ***ade*** is not in L_3
- C_4={*abcd*}



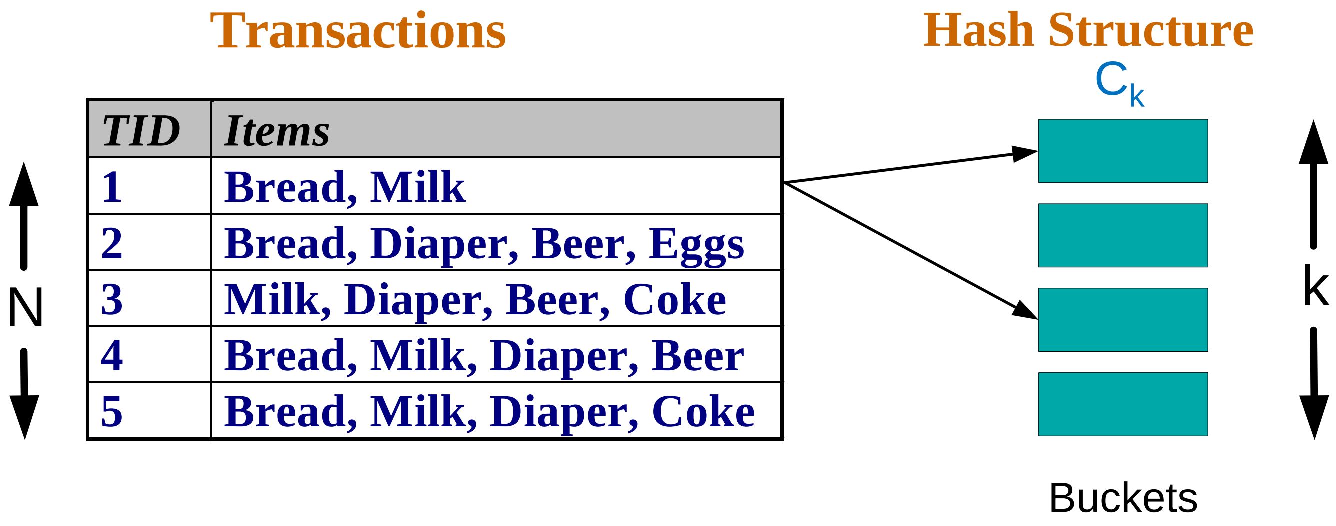


Generate Candidates C_{k+1}

- We have all frequent k-itemsets L_k
- **Step 1: self-join L_k**
  - Create set C_{k+1} by joining frequent k-itemsets that share the first k-1 items
- **Step 2: prune**
  - Remove from C_{k+1} the itemsets that contain a subset k-itemset that is not frequent

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



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**

A simple hash structure

- 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

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},

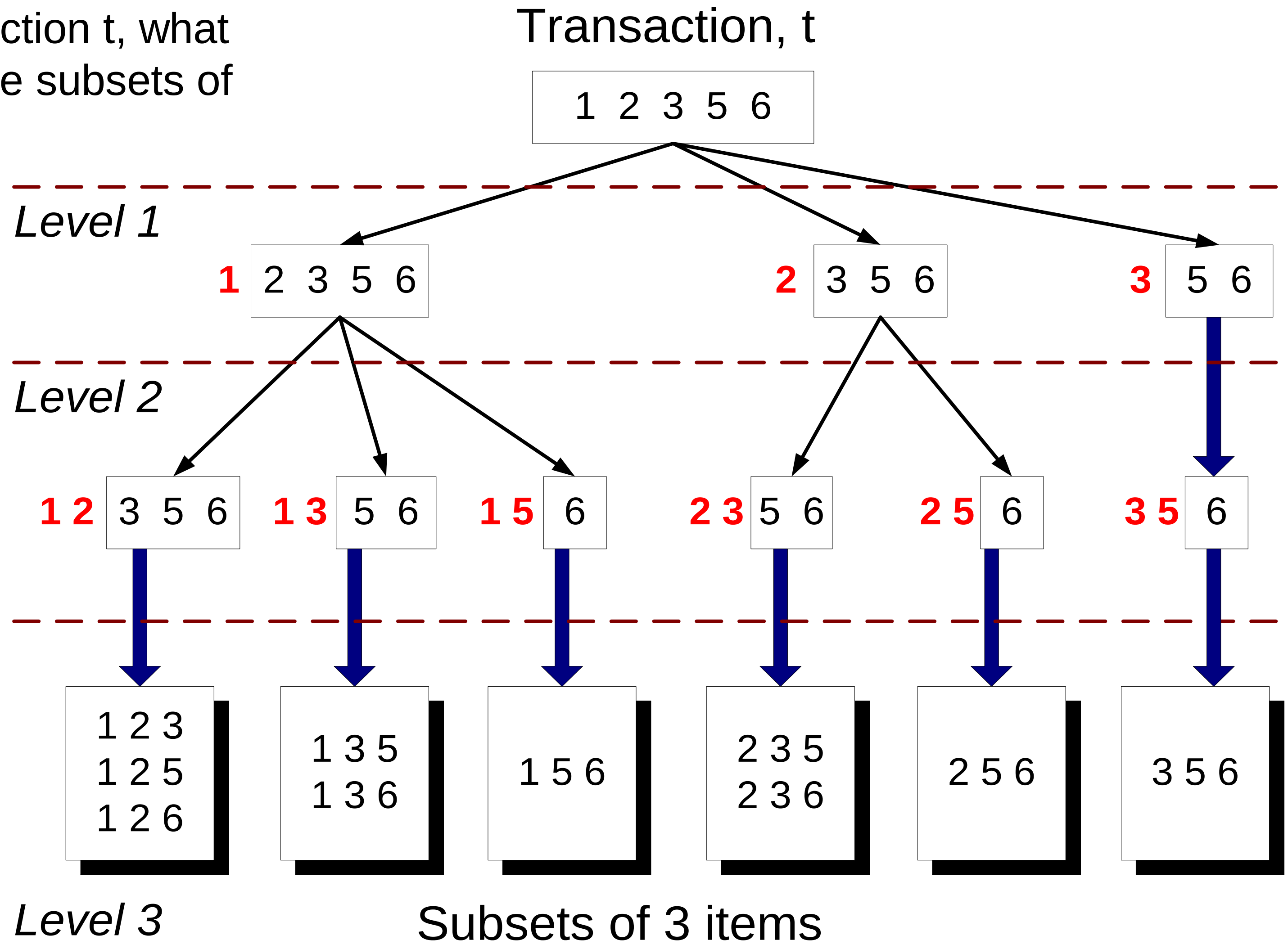
{3 5 6}, {3 5 7}, {6 8 9}, {3 6 7}, {3 6 8}

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

The Hash Tree Structure

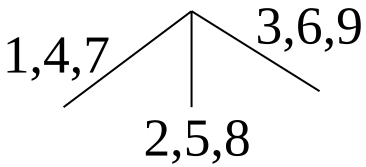
Suppose you have the same 15 candidate itemsets of length 3:

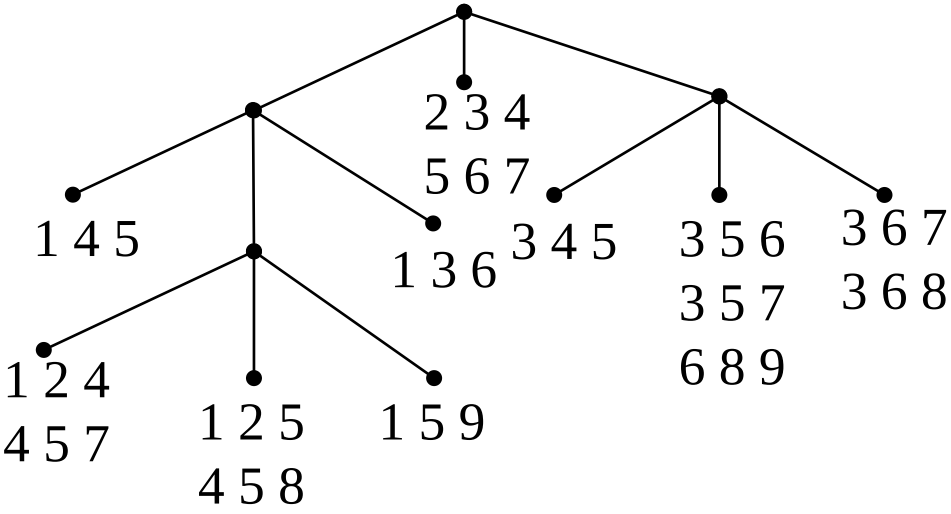
{1 4 5}, {1 2 4}, {4 5 7}, {1 2 5}, {4 5 8}, {1 5 9}, {1 3 6}, {2 3 4}, {5 6 7}, {3 4 5}, {3 5 6}, {3 5 7}, {6 8 9}, {3 6 7}, {3 6 8}

You need:

- Hash function
- Leafs: Store the itemsets

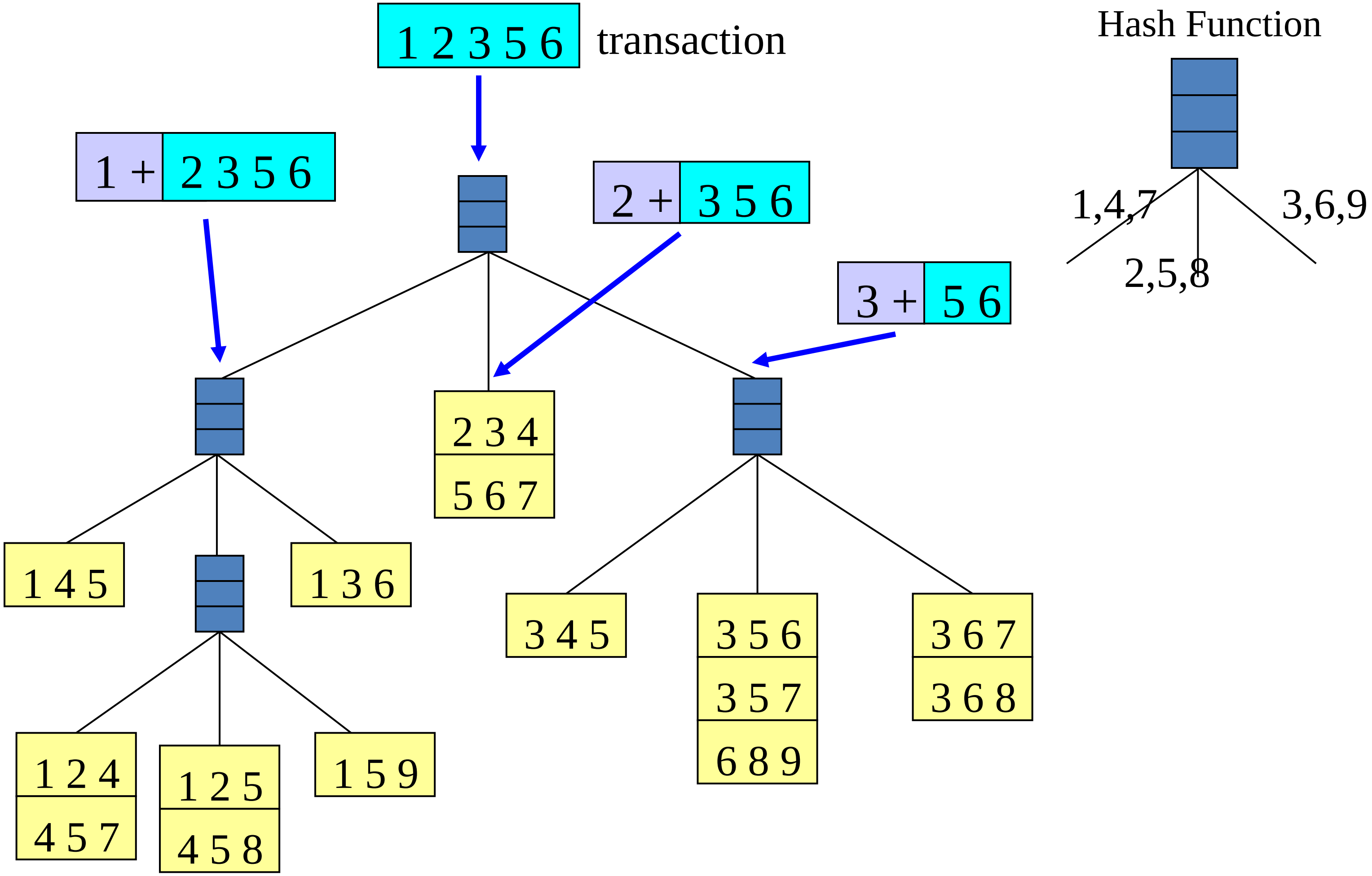
Hash function = x mod 3



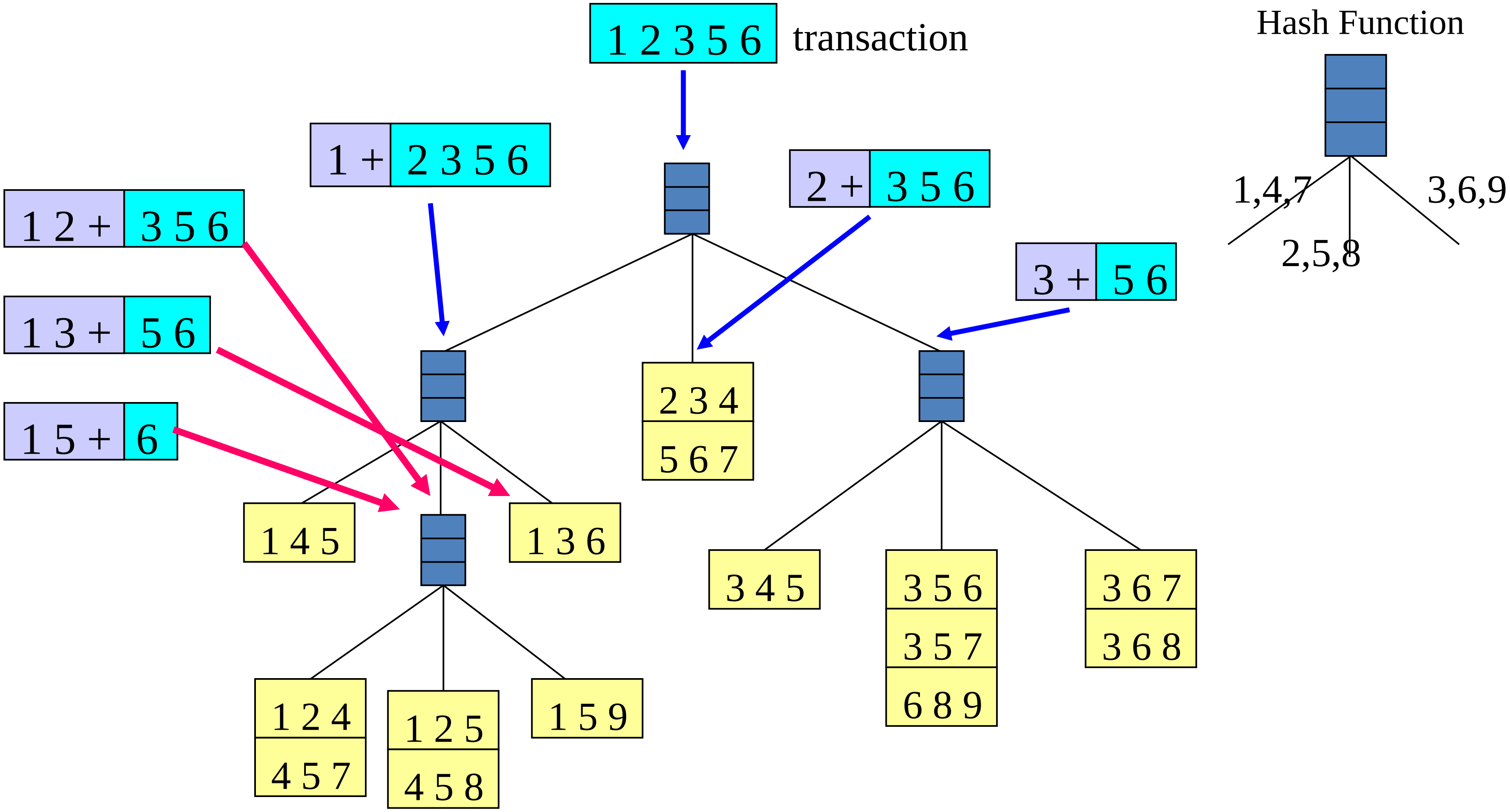


At the i-th level we hash on the i-th item

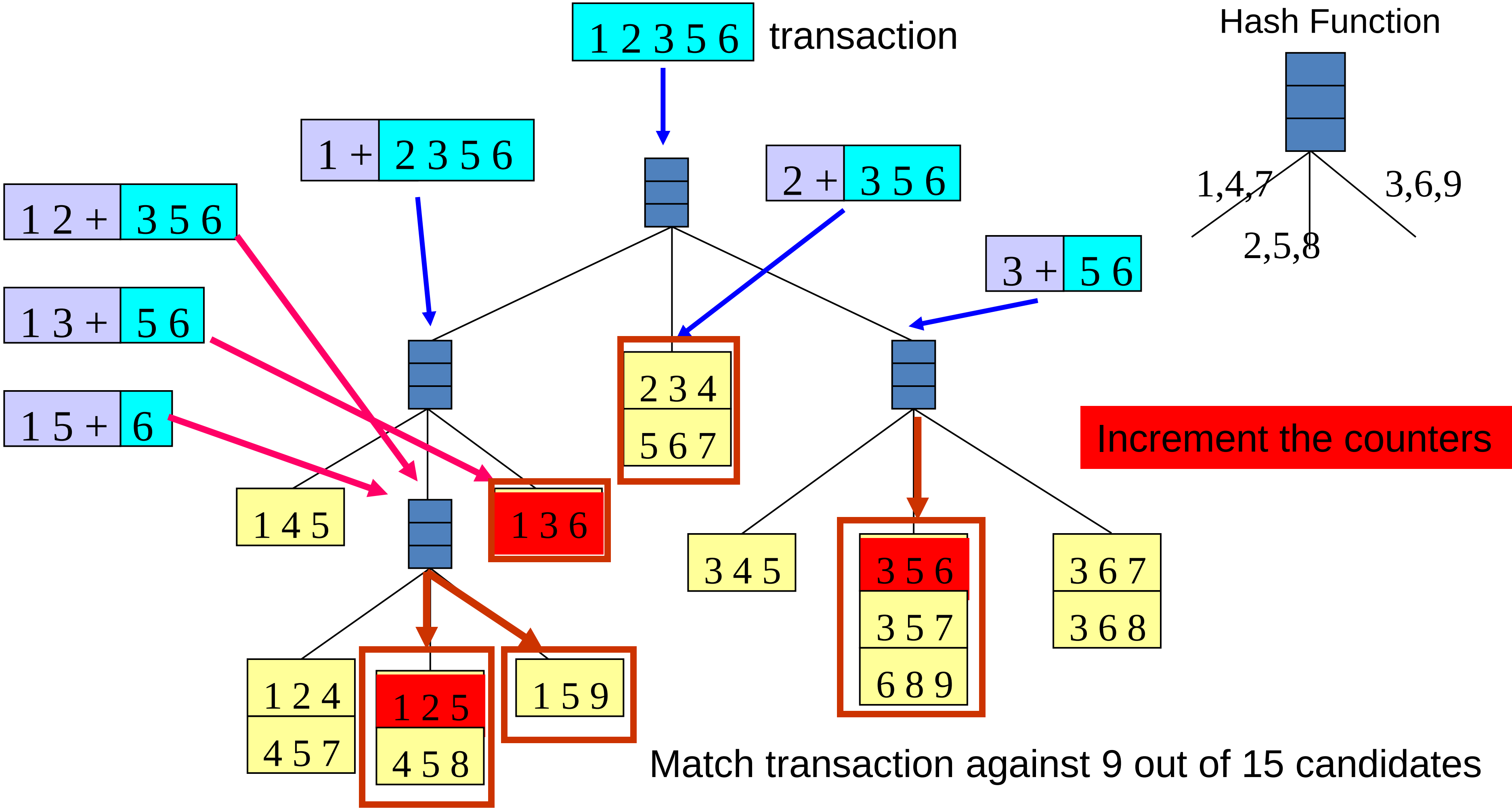
Subset Operation Using Hash Tree



Subset Operation Using Hash Tree

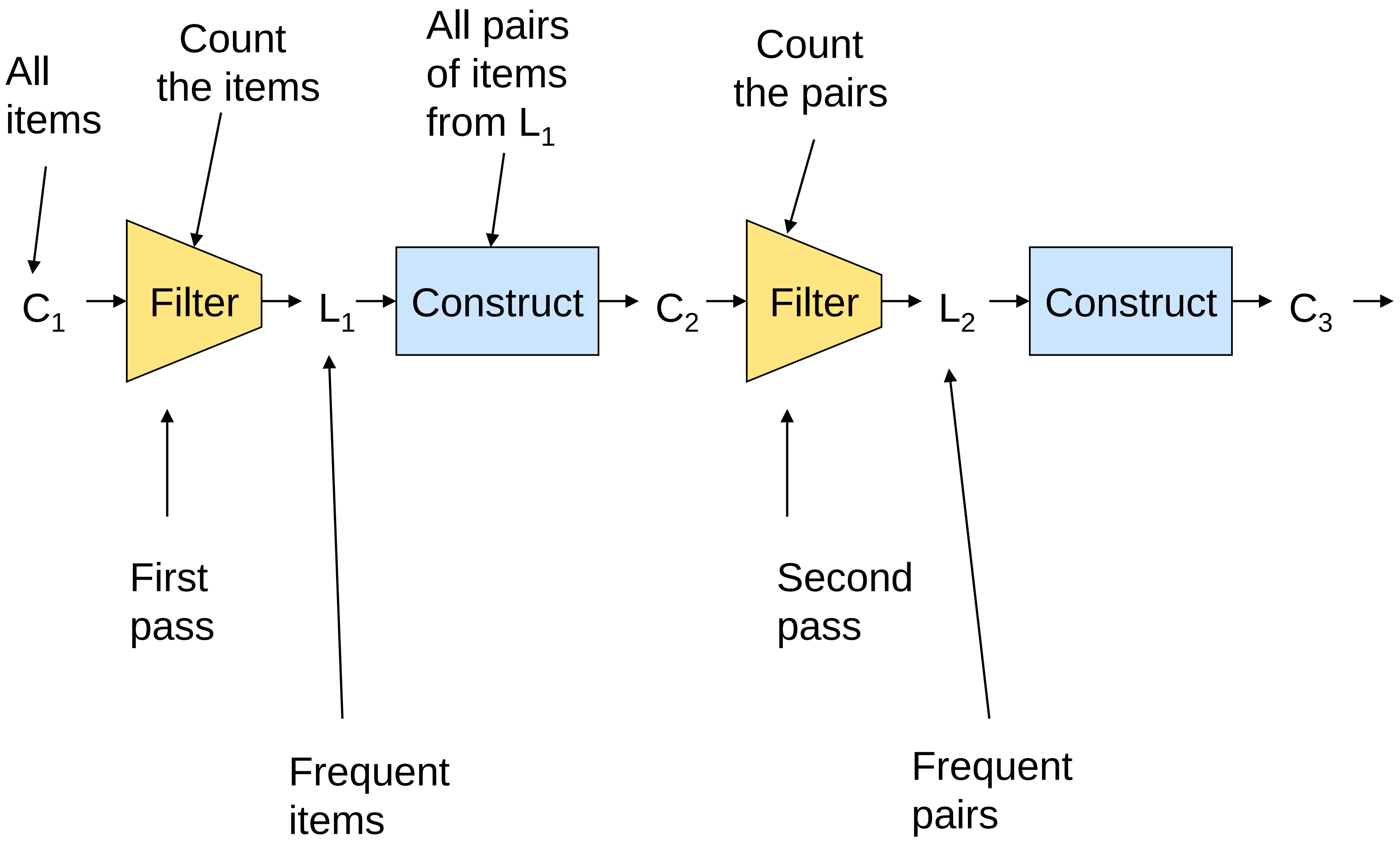


Subset Operation Using Hash Tree



Match transaction against 9 out of 15 candidates

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



All items

Count the items

All pairs of items from L_1

Count the pairs

C_1 → Filter → L_1 → Construct → C_2 → Filter → L_2 → Construct → C_3 →

First pass

Second pass

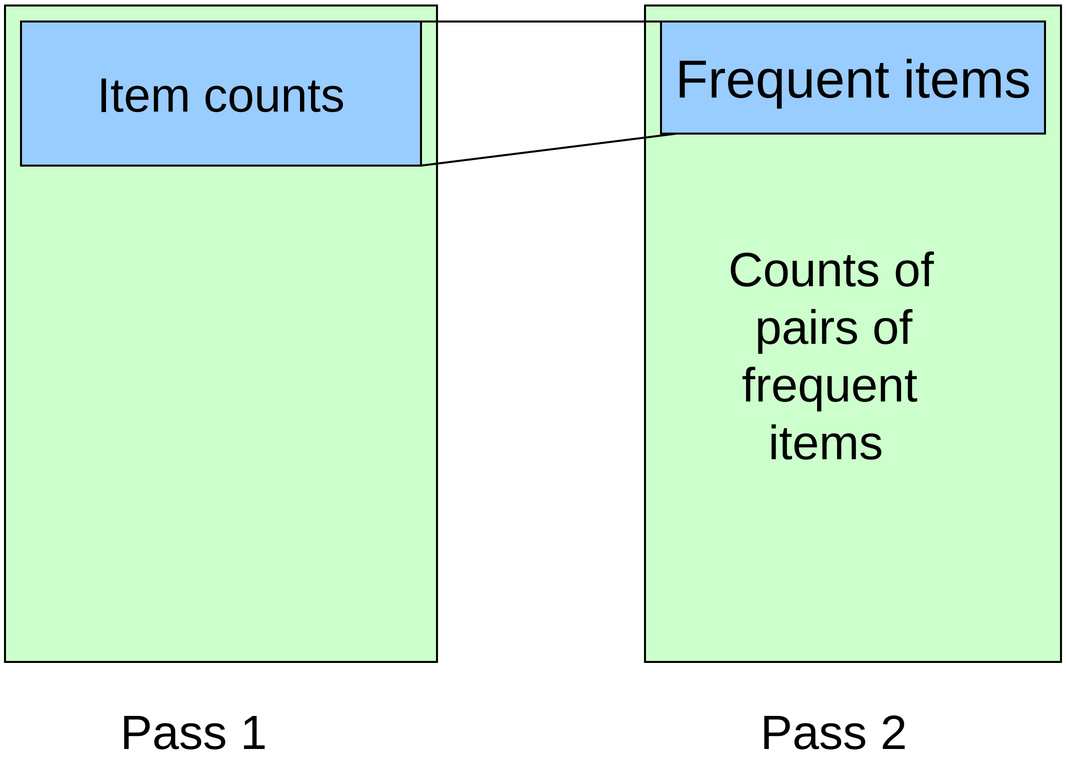
Frequent items

Frequent pairs

A-Priori for All Frequent Itemsets

- One pass for each *k*.
- Needs room in main memory to count each candidate *k* -set.
- For typical market-basket data and reasonable support (e.g., 1%), *k* = 2 requires the most memory.

Picture of A-Priori



Item counts

Frequent items

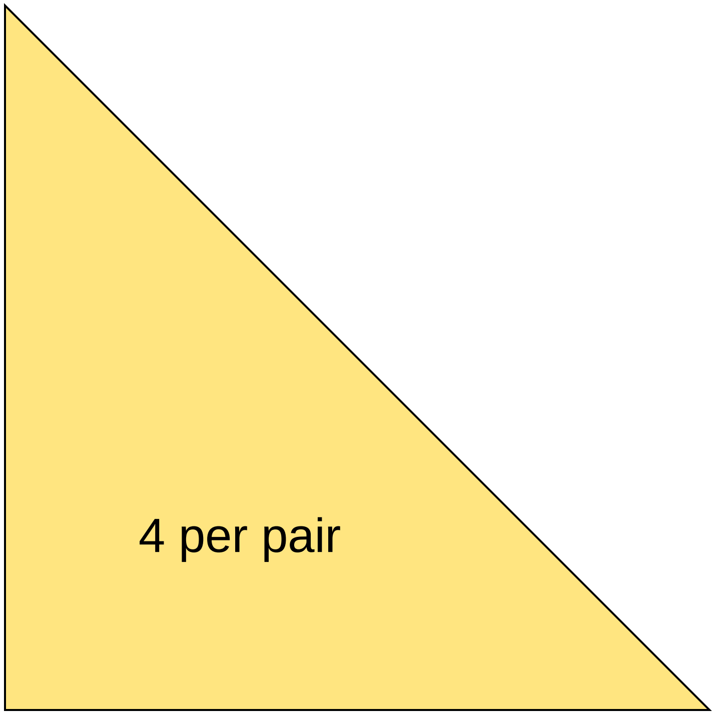
Counts of pairs of frequent items

Pass 1

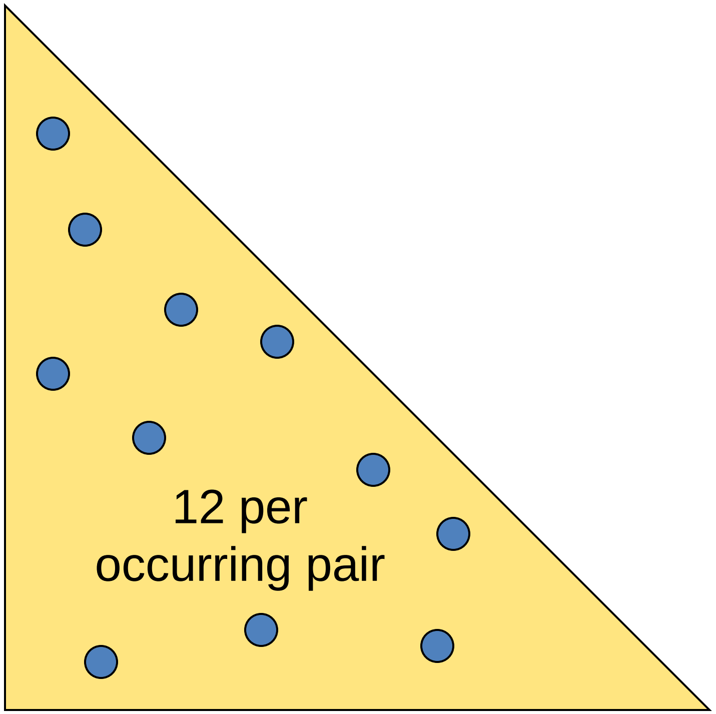
Pass 2

Details of Main-Memory Counting

- Two approaches:
  1. Count all pairs, using a “triangular matrix” = one dimensional array that stores the lower diagonal.
  2. Keep a table of triples [*i*, *j*, *c*] = “the count of the pair of items {*i*, *j* } is *c*.”
- (1) requires only 4 bytes/pair.
  - Note: always assume integers are 4 bytes.
- (2) requires 12 bytes, but only for those pairs with count > 0.



Method (1)



Method (2)

Triangular-Matrix Approach – (1)

- Number items 1, 2,…
  - Requires table of size O(n) to convert item names to consecutive integers.
- Count $\{i, j\}$ only if $i < j$.
- Keep pairs in the order {1,2}, {1,3},…, {1,n }, {2,3}, {2,4},…,{2,n }, {3,4},…, {3,n },…{n -1,n }.

Triangular-Matrix Approach – (2)

- Find pair $\{i, j\}$ at the position

  $(i - 1)(n - i/2) + j - i$.

- Total number of pairs $n(n - 1)/2$; total bytes about $2n^2$.

Details of Approach #2

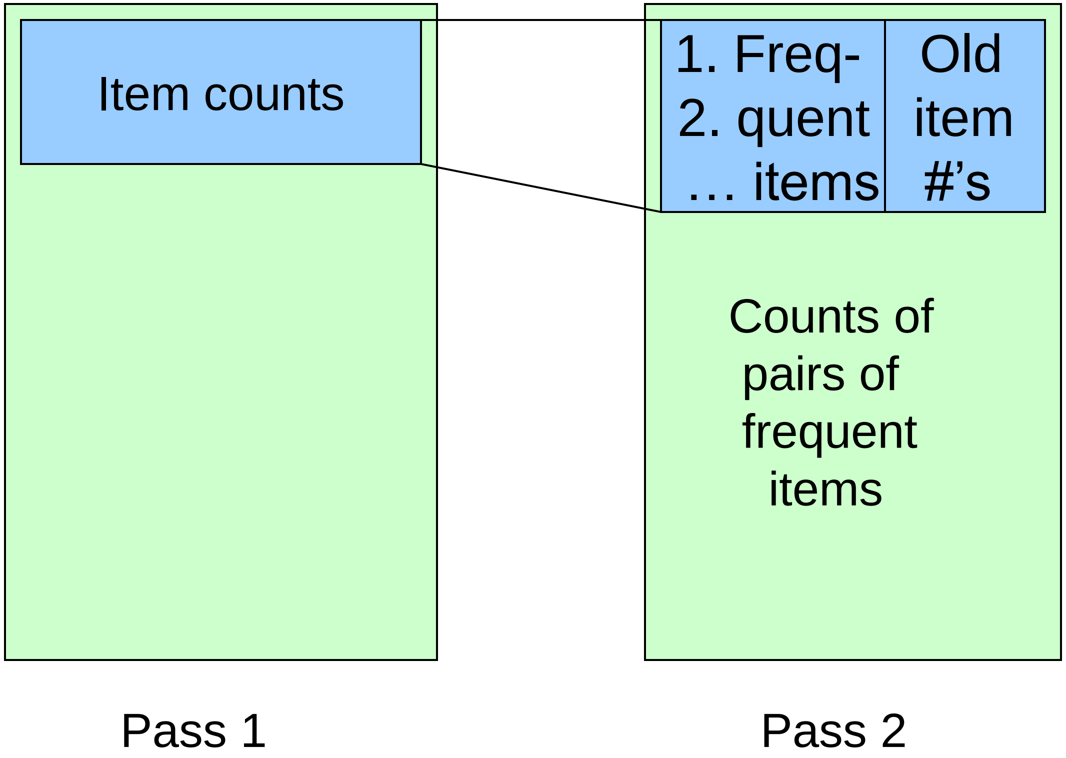
- Total bytes used is about $12p$, where p is the number of pairs that actually occur.
  - Beats triangular matrix if at most 1/3 of possible pairs actually occur.

- May require extra space for retrieval structure, e.g., a hash table.

Detail for A-Priori

- You can use the triangular matrix method with n = number of frequent items.
  - May save space compared with storing triples.
- Trick: number frequent items 1,2,… and keep a table relating new numbers to original item numbers.

A-Priori Using Triangular Matrix for Counts



Item counts

1. Freq-
2. quent
… items

Old
item
#'s

Counts of
pairs of
frequent
items

Pass 1

Pass 2

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)

ASSOCIATION RULES

Association Rule Mining

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

Market-Basket transactions

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

Example of Association Rules

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

Implication means co-occurrence, not causality!

Definition: Association Rule

- **Association Rule**
  - An implication expression of the form X → Y, where X and Y are itemsets
  - Example:
    {Milk, Diaper} → {Beer}
- Rule Evaluation Metrics
  - Support (s)
    - Fraction of transactions that contain both X and Y
    - the probability P(X,Y) that X and Y occur together
  - Confidence (c)
    - Measures how often items in Y appear in transactions that contain X
    - the conditional probability P(X|Y) that X occurs given that Y has occurred.

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:

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

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

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

Association Rule Mining Task

- Input: A set of transactions T, over a set of items I
- Output: All rules with items in I having
  - support ≥ *minsup* threshold
  - confidence ≥ *minconf* threshold

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 partitioning of a frequent itemset into Left-Hand-Side (LHS) and Right-Hand-Side (RHS)

Frequent itemset: {A,B,C,D}
Rule: AB→CD

Rule Generation

- We have all frequent itemsets, how do we get the rules?
  - For every frequent itemset S, we find rules of the form $L \rightarrow S - L$, where $L \subset S$, that satisfy the minimum confidence requirement
  - Example: L = {A,B,C,D}
  - Candidate rules:

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

- 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

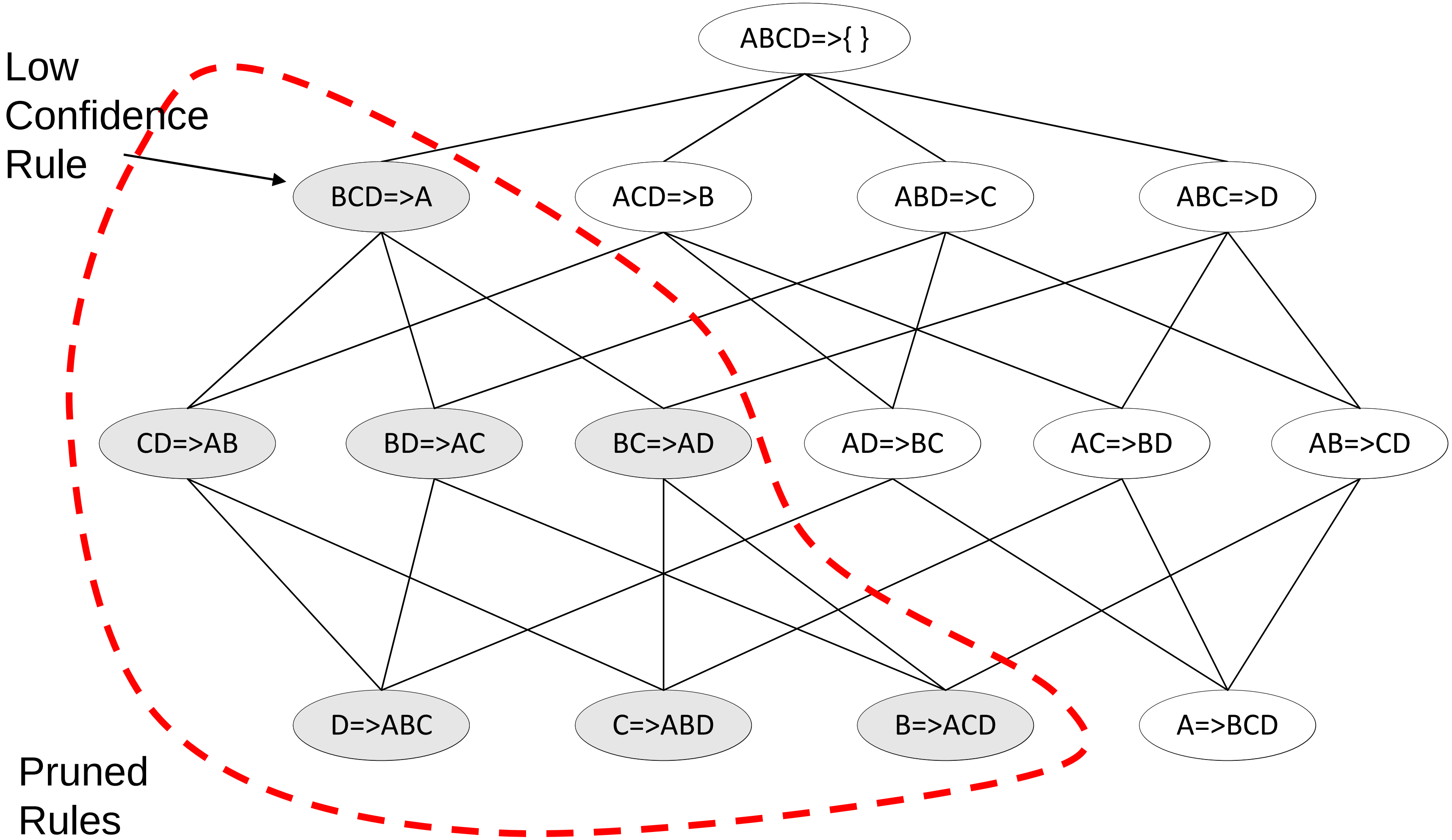
    c(ABC →D) can be larger or smaller than c(AB →D)

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

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

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

Rule Generation for Apriori Algorithm



Lattice of rules created by the RHS

Rule Generation for APriori Algorithm

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

- 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

- Essentially we are doing APriori on the RHS