Frequent Itemset Mining & Association Rules (Part 1)

Slides adapted from Stanford , UIUC data mining course, and textbook slides from Kumar etc

Outline

- **Basics**
  - Introduction & Application
  - Frequent itemsets & Association rules
- **Algorithms for finding frequent itemsets**
  - A-Priori algorithm
- **Implementation of Algorithms**
  - Finding frequent pairs
  - A-Priori algorithm
  - PCY algorithm
- **Interestingness of association rules, other types of itemsets**

Intro to Association Rule Discovery

Supermarket shelf management – Market-basket model:

- **Goal:** Identify items that are bought together by sufficiently many customers
- **Approach:** Process the sales data collected with barcode scanners to find dependencies among items
- **A classic rule:**
  - If someone buys diaper and milk, then he/she is likely to buy beer
  - Don’t be surprised if you find beer next to diapers!

The Market-Basket Model

Input:

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

- A large set of **items**
  - e.g., things sold in a supermarket
- A **large set** of **baskets** or **transactions**
- Each basket/transaction is a **small subset of items**
  - e.g., the things one customer buys on one day
- Want to discover **association rules**
  - People who bought {x,y,z} tend to buy {v,w}
    - Amazon!

Output:

Rules Discovered:

{Milk} --> {Coke}

{Diaper, Milk} --> {Beer}

Applications – (1)

- **Items** = products; **Baskets** = sets of products bought in one trip to the store
- **Real market baskets:** TBs of data about what customers buy together
  - Tells how typical customers navigate stores, lets them position tempting items
  - Suggests tie-in "tricks", e.g., run sale on diapers and raise the price of beer

Applications – (2)

- **Baskets** = sentences; **Items** = words containing in each sentences
  - Frequent set of items may represent phrase, which can be used for phase detection in natural language processing applications
- **Baskets** = patients; **Items** = genes and values
  - Frequent set of items may represent that a combination of different genes and their values, which may indicate some illness

Other applications

- It can be used for other data mining tasks, such as
  - Classification
  - Clustering
  - Outlier detection

- Extend to other type of data, e.g.,
  - From a set of graphs, mine frequent subgraphs (with applications in chemical applications and cyber security)

Frequent Itemsets/Patterns

- **Task:** Find sets of items that appear together "frequently" in baskets
- ***Support*** for itemset I: Number of transactions containing all items in I, denoted by $support(I)$
  - (can also be expressed as a fraction of the total number of transactions)
- Given a ***support threshold minsup***, then sets of items that appear in at least ***minsup*** baskets are called ***frequent itemsets (or patterns)***

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

Support of {Beer, Bread} = 2

Example: Frequent Itemsets

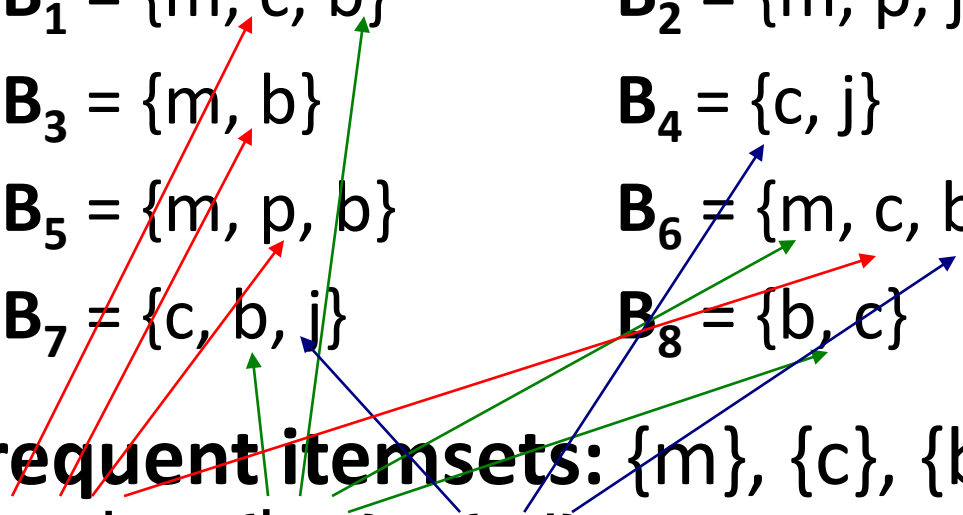
- **Items** = {milk, coke, pepsi, beer, juice}
- **Support threshold** = 3 baskets

B_1 = {m, c, b} B_2 = {m, p, j}

B_3 = {m, b} B_4 = {c, j}

B_5 = {m, p, b} B_6 = {m, c, b, j}

B_7 = {c, b, j} B_8 = {b, c}



- **Frequent itemsets:** {m}, {c}, {b}, {j}, {m,b} , {b,c} , {c,j}.

Association Rules

- **Association Rules:**
  If-then rules about the contents of transactions
- $\{i_1, i_2, \ldots, i_k\} \rightarrow j$ means: "if a transaction contains all of $i_1, \ldots, i_k$ then it is ***likely*** to contain j"
- **In practice there are too many rules**
  - Not all the generated rules are interesting
  - **want to find significant/interesting ones!**
- ***Confidence*** of this association rule is the probability of j given $\mathbf{I} = \{i_1, \ldots, i_k\}$

$$conf(I \rightarrow j) = \frac{support(I \cup j)}{support(I)}$$

Example: Confidence

$$conf(I \rightarrow j) = \frac{support(I \cup j)}{support(I)}$$

Data:

B_1 = {m, c, b} **B_2 = {m, p, j}**

B_3 = {m, b} **B_4= {c, j}**

B_5 = {m, p, b} **B_6 = {m, c, b, j}**

B_7 = {c, b, j} **B_8 = {b, c}**

- **Association rule: {m} → b**
  - **Confidence** = 4/5 = 0.8
- **Association rule: {m, b} → c**
  - **Confidence** = 2/4 = 0.5

Association Rule Mining Task

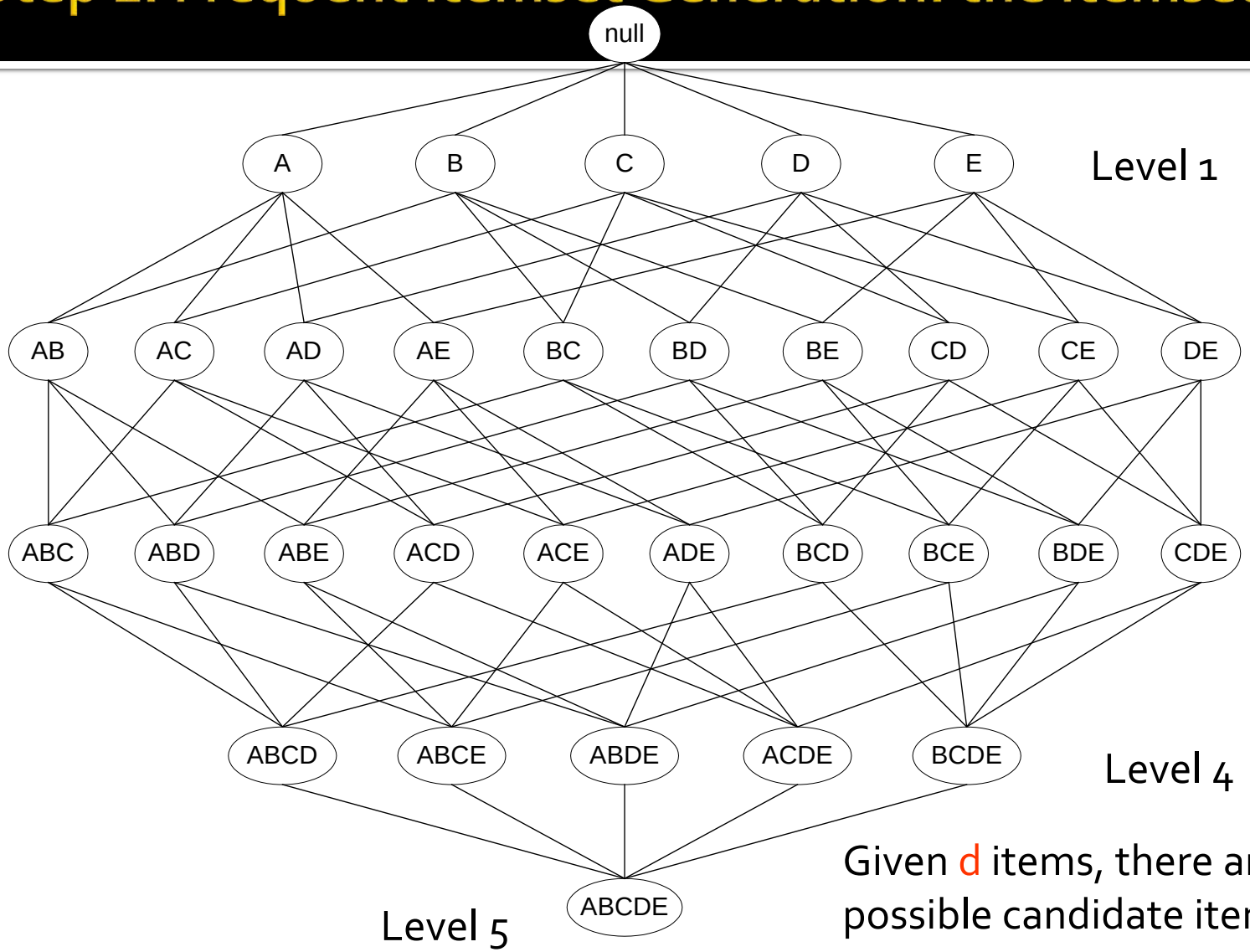
- **Problem:** Given a set of transactions T, the goal of association rule mining is to find all rules having
  - **support ≥ *minsup*** threshold
  - **confidence ≥ *minconf*** threshold
- Brute-force approach:
  - List all possible association rules
  - Compute the support and confidence for each rule
  - Prune rules that fail the *minsup* and *minconf* thresholds

  ⇒ Computationally prohibitive!

Mining Association Rules

- **Step 1: Find all frequent itemsets I**
  - Generate all itemsets whose support $\geq$ *minsup*
  - Based on number of items *l* in an itemset, we group them into *l*-itemsets
    - E,g, 2-itemsets for an itemset containing 2 items
- **Step 2: Rule generation**
  - For each frequent itemset I, find all non-empty subsets $A \subset I$ such that $A \rightarrow I - A$ satisfies the **minimum confidence requirement (*minconf*)**

Step 1: Frequent Itemset Generation: the Itemset Lattice



Level 1

Level 2

Level 3

Level 4

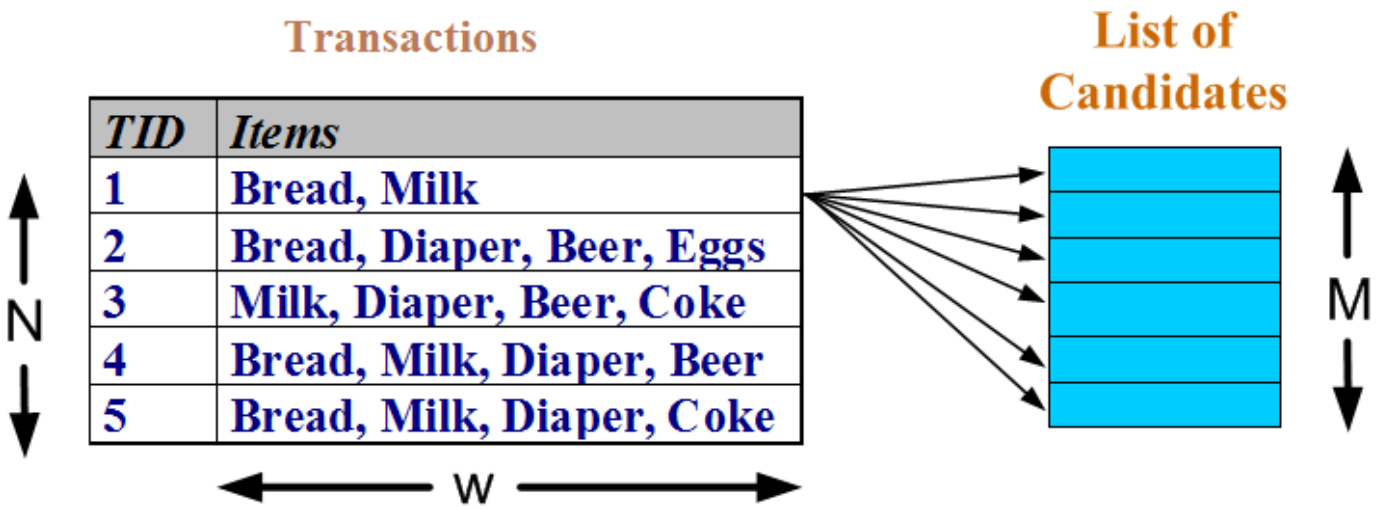
Level 5

Given d items, there are $2^d - 1$ possible candidate itemsets

Frequent itemset generation is computationally expensive!

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 support ≥ *minsup*



  - Match each transaction against every candidate
  - Complexity ~ O(NMw), w is the maximum transaction width

    => Expensive since $M = 2^d - 1$!!!

Apriori Principle: 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:

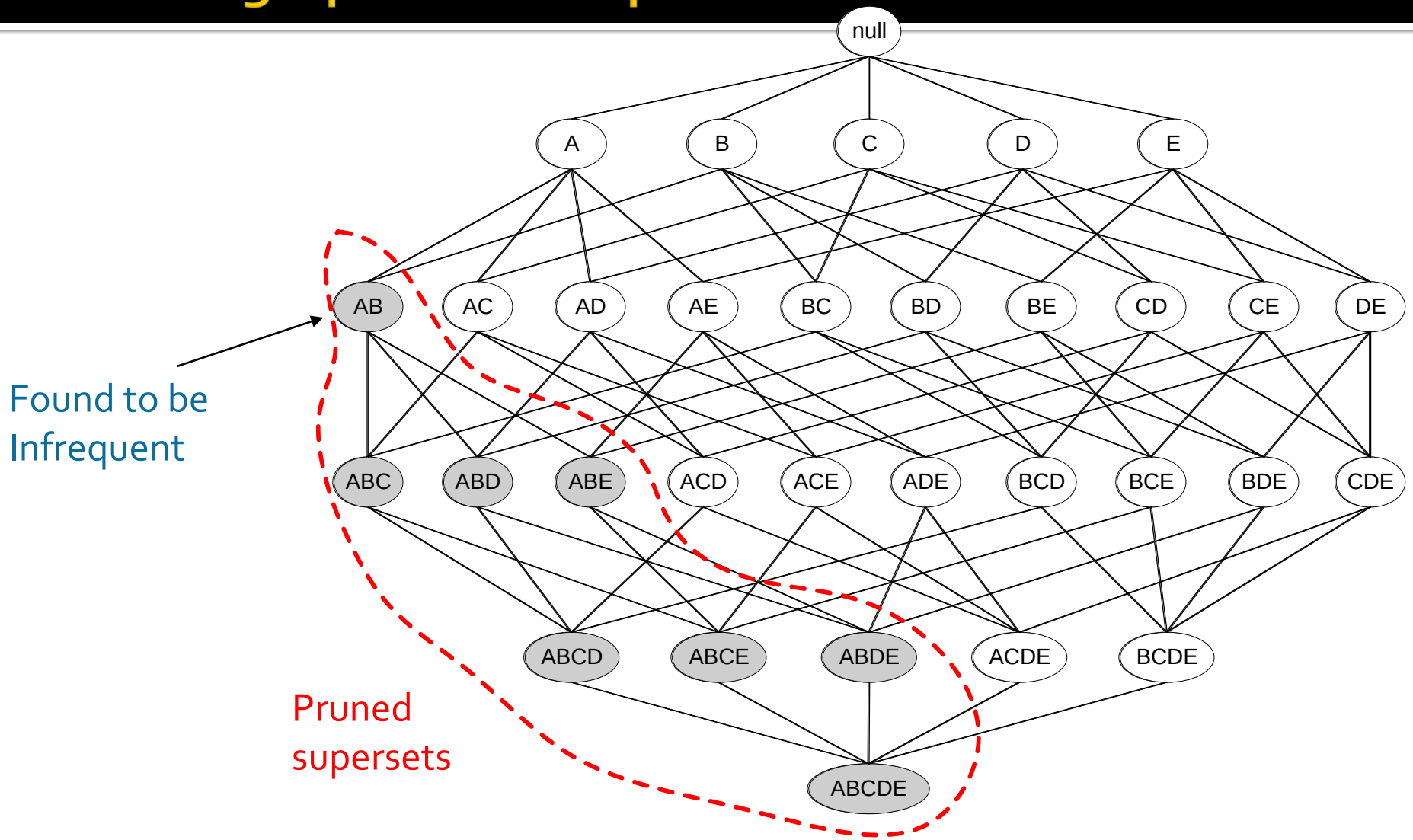
$$\forall X, Y: (X \subseteq Y) \Rightarrow s(X) \geq s(Y)$$

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

Anti-Monotone Property

- Any subset of a frequent itemset must be also frequent — an anti-monotone property
  - Any transaction containing {beer, diaper, milk} also contains {beer, diaper}
  - {beer, diaper, milk} is frequent → {beer, diaper} must also be frequent
- In other words, any superset of an infrequent itemset must also be infrequent
  - No superset of any infrequent itemset should be generated or tested
  - Many item combinations can be pruned!

Illustrating Apriori Principle



Apriori Algorithm

- F_k: frequent k-itemsets (containing k items)
- C_k: candidate k-itemsets

- Algorithm
  - Let k=1
  - Generate F_1 = {frequent 1-itemsets}
  - Repeat until F_k is empty
    - **Candidate Generation**: Generate C_{k+1} from F_k
    - **Candidate Pruning**: Prune candidate itemsets in C_{k+1} containing subsets of length k that are infrequent
    - **Support Counting**: Count the support of each candidate in C_{k+1} by scanning the DB
    - **Candidate Elimination**: Eliminate candidates in C_{k+1} that are infrequent, leaving only those that are frequent => F_{k+1}

Candidate Generation: Brute-force method

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

Items

Item
Beer
Bread
Cola
Diapers
Eggs
Milk

Candidate Generation

Itemset
{Beer, Bread, Cola}
{Beer, Bread, Diapers}
{Beer, Bread, Eggs}
{Beer, Bread, Milk}
{Beer, Cola, Diapers}
{Beer, Cola, Eggs}
{Beer, Cola, Milk}
{Beer, Diapers, Eggs}
{Beer, Diapers, Milk}
{Beer, Eggs, Milk}
{Bread, Cola, Diapers}
{Bread, Cola, Eggs}
{Bread, Cola, Milk}
{Bread, Diapers, Eggs}
{Bread, Diapers, Milk}
{Bread, Eggs, Milk}
{Cola, Diapers, Eggs}
{Cola, Diapers, Milk}
{Cola, Eggs, Milk}
{Diapers, Eggs, Milk}

Candidate Pruning

Itemset
{Bread, Diapers, Milk}

Frequent 2-itemset

Itemset
{Beer, Diapers}
{Bread, Diapers}
{Bread, Milk}
{Diapers, Milk}

Figure 5.6. A brute-force method for generating candidate 3-itemsets.

Candidate Generation: $F_{k-1} \times F_{k-1}$ Method

- To generate C_{k+1} from F_k: Merge two frequent (k)-itemsets if their first (k-1) items are identical

- F_3 = {ABC,ABD,ABE,ACD,BCD,BDE,CDE}
  - Merge(**AB**C, **AB**D) = **AB**CD
  - Merge(**AB**C, **AB**E) = **AB**CE
  - Merge(**AB**D, **AB**E) = **AB**DE

  - Do not merge(**A**BD,**A**CD) because they share only prefix of length 1 instead of length 2

Candidate Pruning

- Let F_3 = {ABC,ABD,ABE,ACD,BCD,BDE,CDE} be the set of frequent 3-itemsets
- C_4 = {ABCD,ABCE,ABDE} is the set of candidate 4-itemsets generated (from previous slide)
- Candidate pruning
  - Prune ABCE because ACE and BCE are infrequent
  - Prune ABDE because ADE is infrequent
- After candidate pruning: C_4 = {ABCD}

Candidate Generation: $F_{k-1} \times F_{k-1}$ Method

Another example:

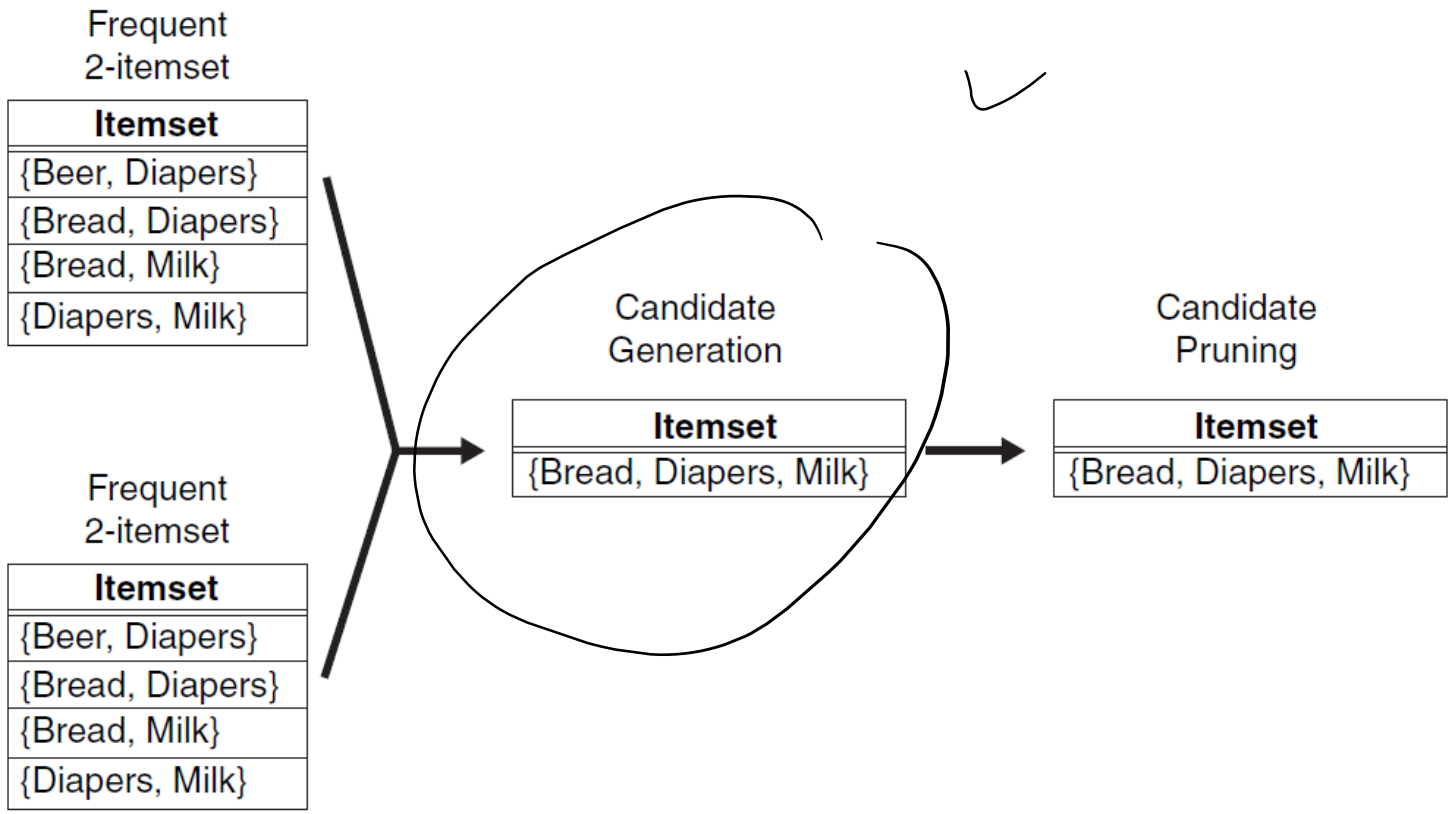
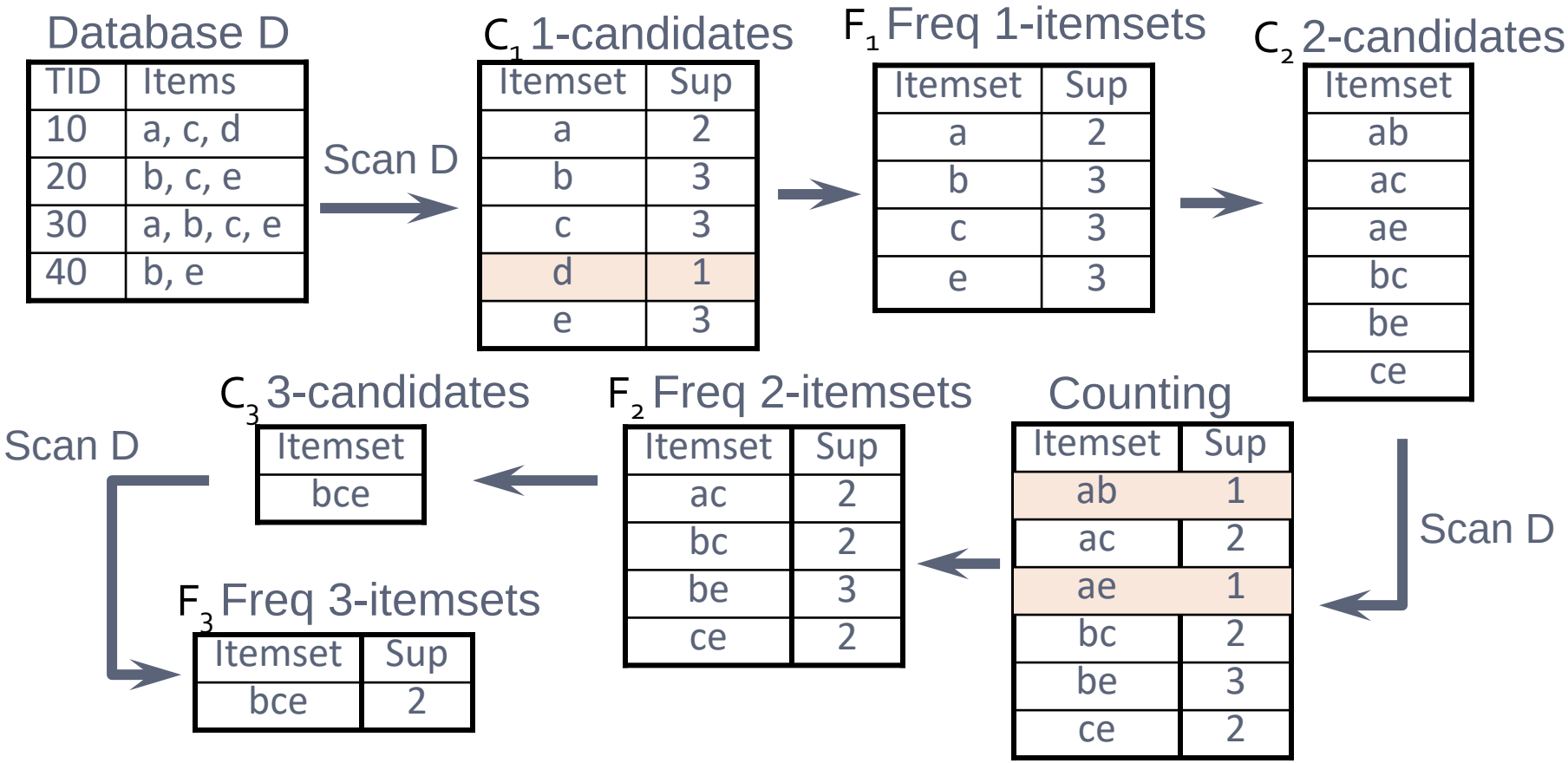


Figure 5.8. Generating and pruning candidate k-itemsets by merging pairs of frequent $(k-1)$-itemsets.

Example: Apriori-based Mining

Minsup=2



Step2: Rule generation

- **Step 1:** Find all frequent itemsets I
  - Generate all itemsets whose support $\geq$ *minsup*
- **Step 2:** **Rule generation**
  - Given a frequent itemset I, find all non-empty subsets $A \subset I$ such that $A \rightarrow I - A$ satisfies the **minimum confidence requirement *minconf***
    - 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

    - An example to compute the rule confidence
      - confidence(***A,B→C,D***) = support(**A,B,C,D**) / support(**A,B**)
  - Do the above for every frequent itemset, and **output all the rules above the confidence threshold**

Example

B_1 = {m, c, b}
B_2 = {m, p, j}
B_3 = {m, c, b, n}
B_4= {c, j}
B_5 = {m, p, b}
B_6 = {m, c, b, j}
B_7 = {c, b, j}
B_8 = {b, c}

Support threshold
minsup = 3,
confidence
minconf = 0.75

1) Frequent itemsets:

- {b,m}: $s = 4$ {b,c}:4 {c,m}:3 {c,j}:3 {m,c,b}:3

2) Generate rules:

{b,m}: b→m: c=4/6; m→b: c=4/5

{b,c}: b→c: c=5/6; c→b: c=5/6

…

{m,c,b}: b,c →m: c=3/5; b,m→c: c=3/4; m,c→b:c=3/3

b →c,m: c=3/6; c →b,m: c=3/6; m →b,c : c=3/5

Rule Generation

- If | $\boldsymbol{I}$ | = k, then there are $2^k - 2$ candidate association rules (ignoring $\boldsymbol{I} \rightarrow \varnothing$ and $\varnothing \rightarrow \boldsymbol{I}$)
  - For example $\boldsymbol{I}$ *is* {A,B,C,D}, 14 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		

Another example of Rule Generation

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

Example of Rules:

{Milk,Diaper} → {Beer} (s=2, c=2/3)
{Milk,Beer} → {Diaper} (s=2, c=2/2)
{Diaper,Beer} → {Milk} (s=2, c=2/3)
{Beer} → {Milk,Diaper} (s=2, c=2/3)
{Diaper} → {Milk,Beer} (s=2, c=2/4)
{Milk} → {Diaper,Beer} (s=2, c=2/4)

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**

Prunning in Rule Generation

- Confidence does not have an anti-monotone property

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

- How to prune?
  - Confidence of rules generated from the same itemset has an anti-monotone property
  - E.g., Suppose {A,B,C,D} is a frequent 4-itemset:

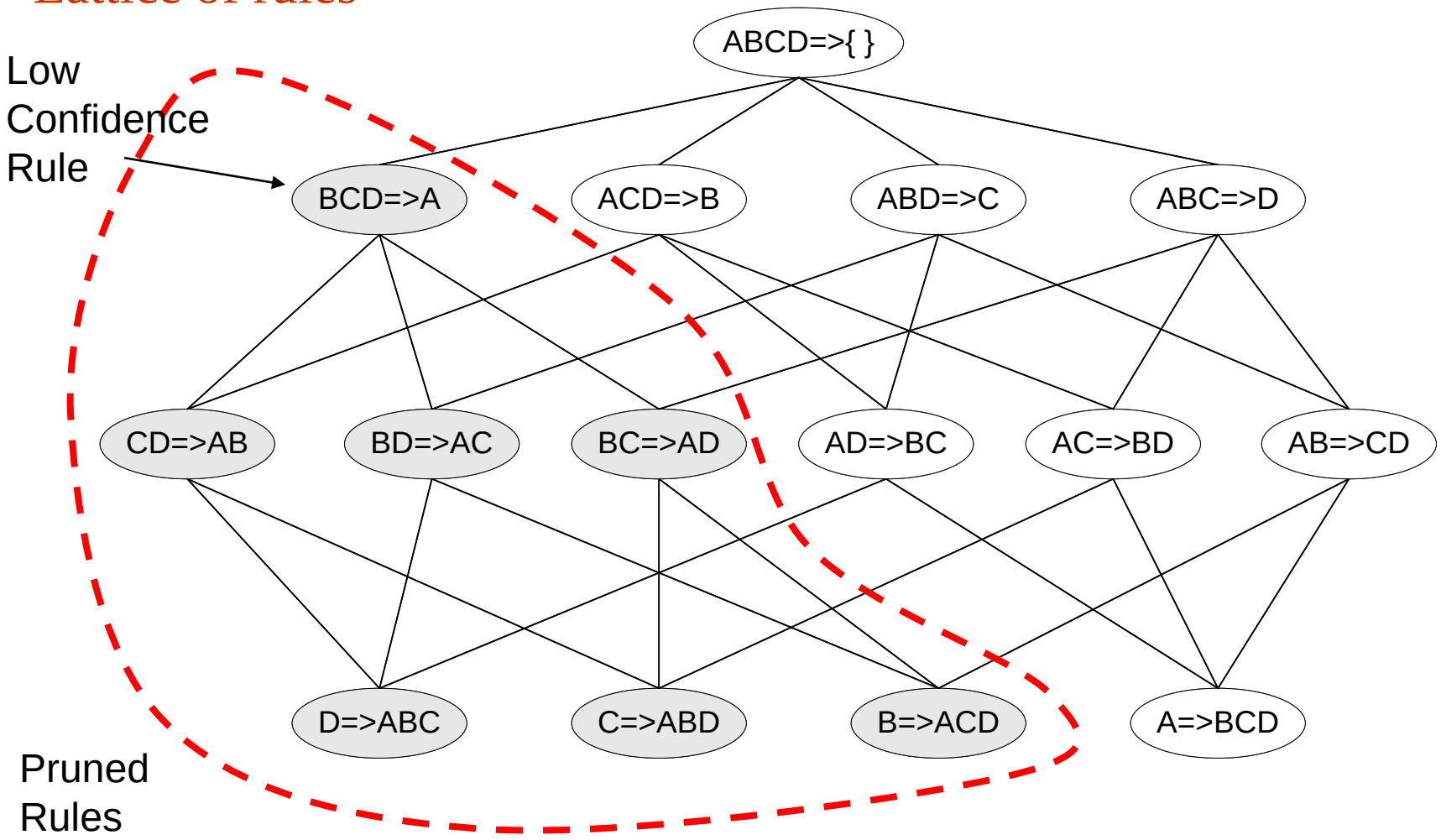
$$c(ABC \rightarrow D) \geq c(AB \rightarrow CD) \geq c(A \rightarrow BCD)$$

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

  - **Observation:** If **A,B,C$\rightarrow$D** is below confidence, so is **A,B$\rightarrow$C,D**
  - Can generate "bigger" rules from smaller ones (RHS)!

Rule Generation for Apriori Algorithm

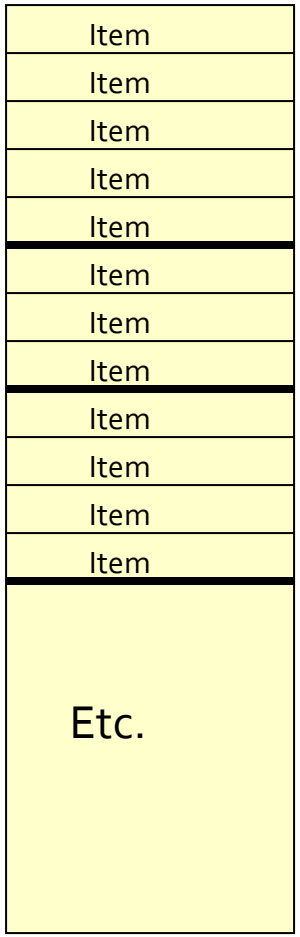
Lattice of rules



Implementation of algorithms for finding Frequent Itemsets on BIG data

Itemsets: Computation Model

- **Back to finding frequent itemsets**
- Typically, **data** is kept in flat files rather than in a database system:
  - Stored on disk
  - Stored basket-by-basket
  - Baskets are **small** but we have many baskets and many items



Items are positive integers, and boundaries between baskets are –1.

Computation Model

- The dominating cost of mining disk-resident data is usually the **number of disk I/Os**
- In practice, association-rule algorithms read the data in ***passes*** – all baskets read in turn
- We measure the cost by the **number of passes** an algorithm makes over the data

Main-Memory Bottleneck

- **Itemsets:** To find frequent itemsets, we have to count them. To count them, we have to generate them, and usually store them in **memory.**

- 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 of items
  - The number of different things we can count is limited by main memory
  - Swapping counts in/out is a disaster (why?)

Naïve Algorithm for finding frequent pairs

- **Let us consider finding frequent pairs**
- Read file once, counting in main memory the occurrences of each pair:
  - From each basket of ***n*** items, generate its ***n(n-1)/2*** pairs by two nested loops
- **Fails if (#items)^2 exceeds main memory**
  - **Remember:** #items can be 100K (Wal-Mart) or 10B (Web pages)
    - Suppose 10^5 items, counts are 4-byte integers
    - Number of pairs of items: $10^5(10^5-1)/2 = 5*10^9$
    - Therefore, $2*10^{10}$ (20 gigabytes) of memory needed

Counting Pairs in Memory

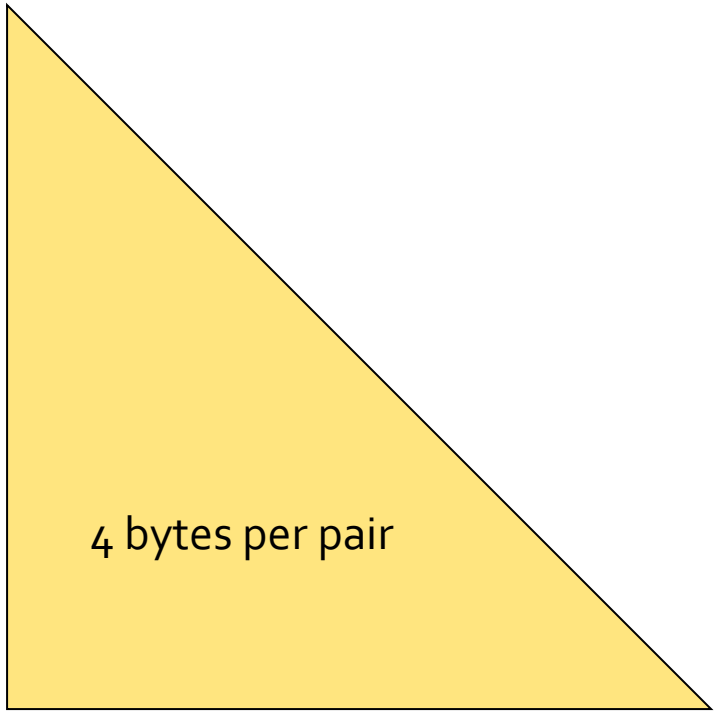
Two approaches:

- **Approach 1:** Count all pairs using a matrix
- **Approach 2:** Keep a table of triples [*i*, *j*, *c*] = "the count of the pair of items {*i*, *j*} is *c*."
  - If integers and item ids are 4 bytes, we need approximately 12 bytes for pairs with count > 0
  - Plus some additional overhead for the hashtable

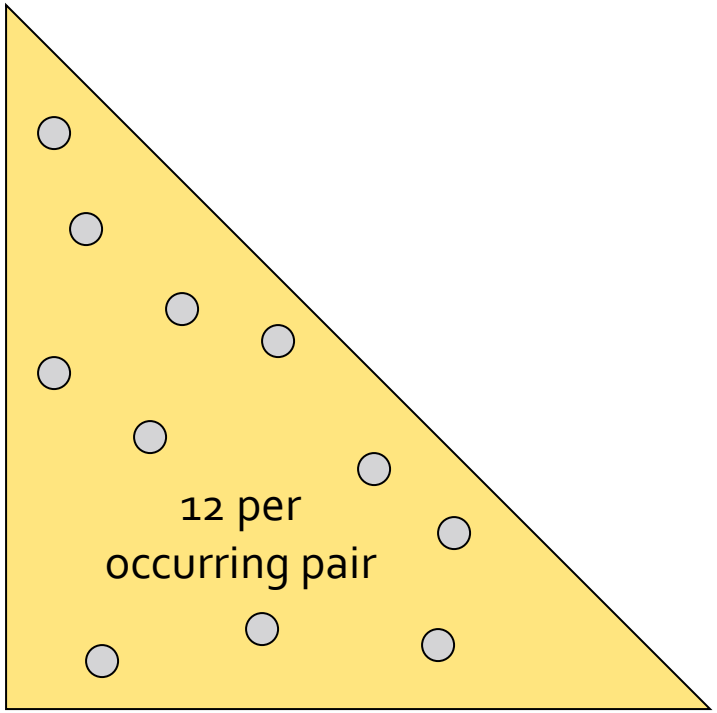
Note:

- **Approach 1** only requires 4 bytes per pair
- **Approach 2** uses 12 bytes per pair (but only for pairs with count > 0)

Comparing the 2 Approaches



Triangular Matrix



Triples

Comparing the two approaches

- **Approach 1: Triangular Matrix**
  - **n** = total number items
  - Count pair of items {*i*, *j*} only if *i*<*j*
  - Keep pair counts in lexicographic order:
    - {1,2}, {1,3},…, {1,*n*}, {2,3}, {2,4},…,{2,*n*}, {3,4},…
  - Pair {*i*, *j*} is at position $(i-1)(n-i/2)+j-i$
  - Total number of pairs $\boldsymbol{n(n-1)/2}$; total bytes= $\boldsymbol{2n^2}$
  - **Triangular Matrix** requires 4 bytes per pair
- **Approach 2** uses **12 *bytes*** per occurring pair *(but only for pairs with count > 0)*
  - Beats Approach 1 if less than **1/3** of possible pairs actually occur

Comparing the two approaches

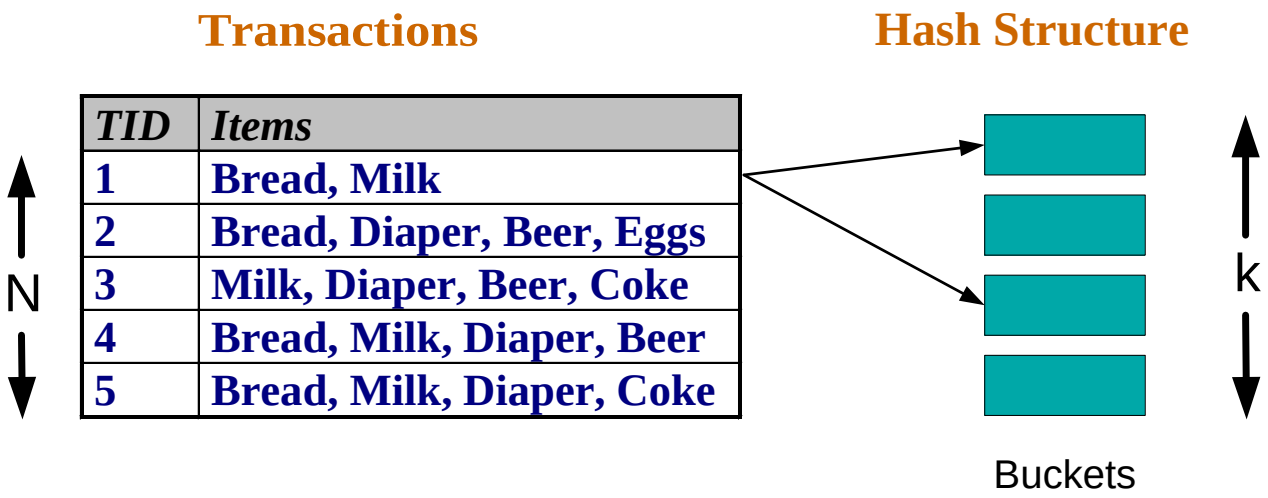
- **Approach 1: Triangular Matrix**
  - **n** = total number items
  - Count pair of items {*i*, *j*} only if *i*<*j*
  - Keep pair counts in lexicographic order:
    - {1,2}, {1,3},…, {1,*n*}, {2,3}, {2,4},…,{2,*n*}, {3,4},…
  - Pair {*i*, *j*} is at position $(i-1)(n-i/2)+j-i$
  - Total number of pairs $\boldsymbol{n(n-1)/2}$; total bytes= $\boldsymbol{2n^2}$
  - **Triangular Matrix** requires 4 bytes per pair
- **Approach 2** uses 12 *bytes* per pair *(but only for pairs with count > 0)*
  - Beats Approach 1 if less than **1/3** of possible pairs actually occur

Problem is if we have too many items so the pairs do not fit into memory.

Can we do better?

Support Counting of Candidate Itemsets

- When itemsets are longer, matching every candidate itemset against every transaction is an expensive operation
- 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



Support Counting Using a Hash Tree **(the next 5 slides for your information only, not examinable)**

Suppose you have 15 candidate itemsets of length 3:

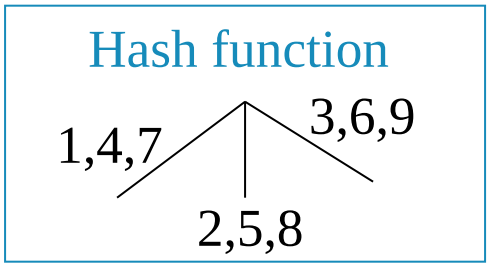
{1 4 5}, {1 2 4}, {4 5 7}, {1 2 5}, {4 5 8}, {1 5 9}, {1 3 6}, {2 3 4}, {5 6 7}, {3 4 5}, {3 5 6}, {3 5 7}, {6 8 9}, {3 6 7}, {3 6 8}

Use a data structure to organize these itemsets to reduce computation:

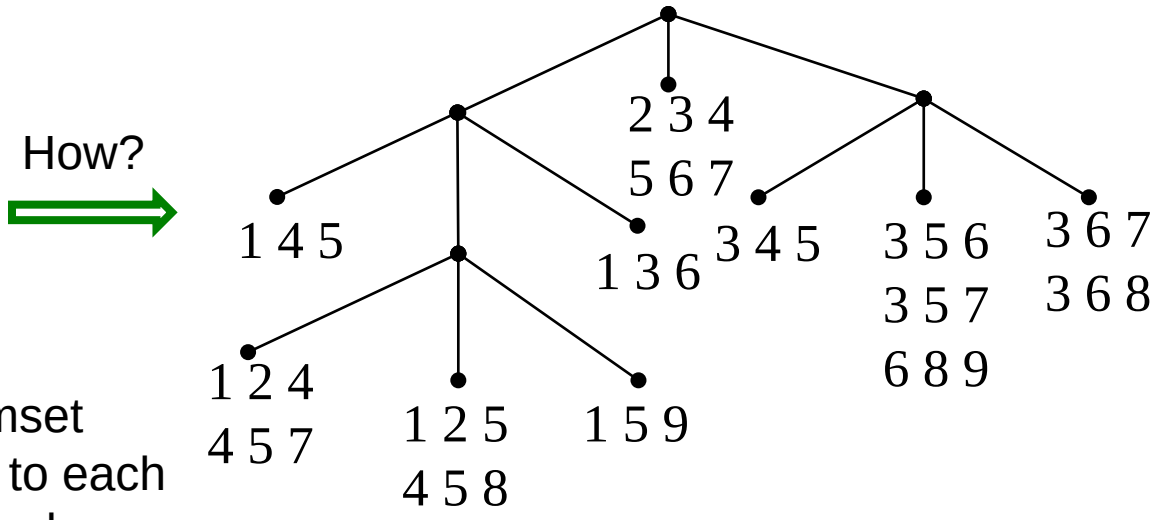
• Hash function

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

$h(p) = p \ mod \ 3$

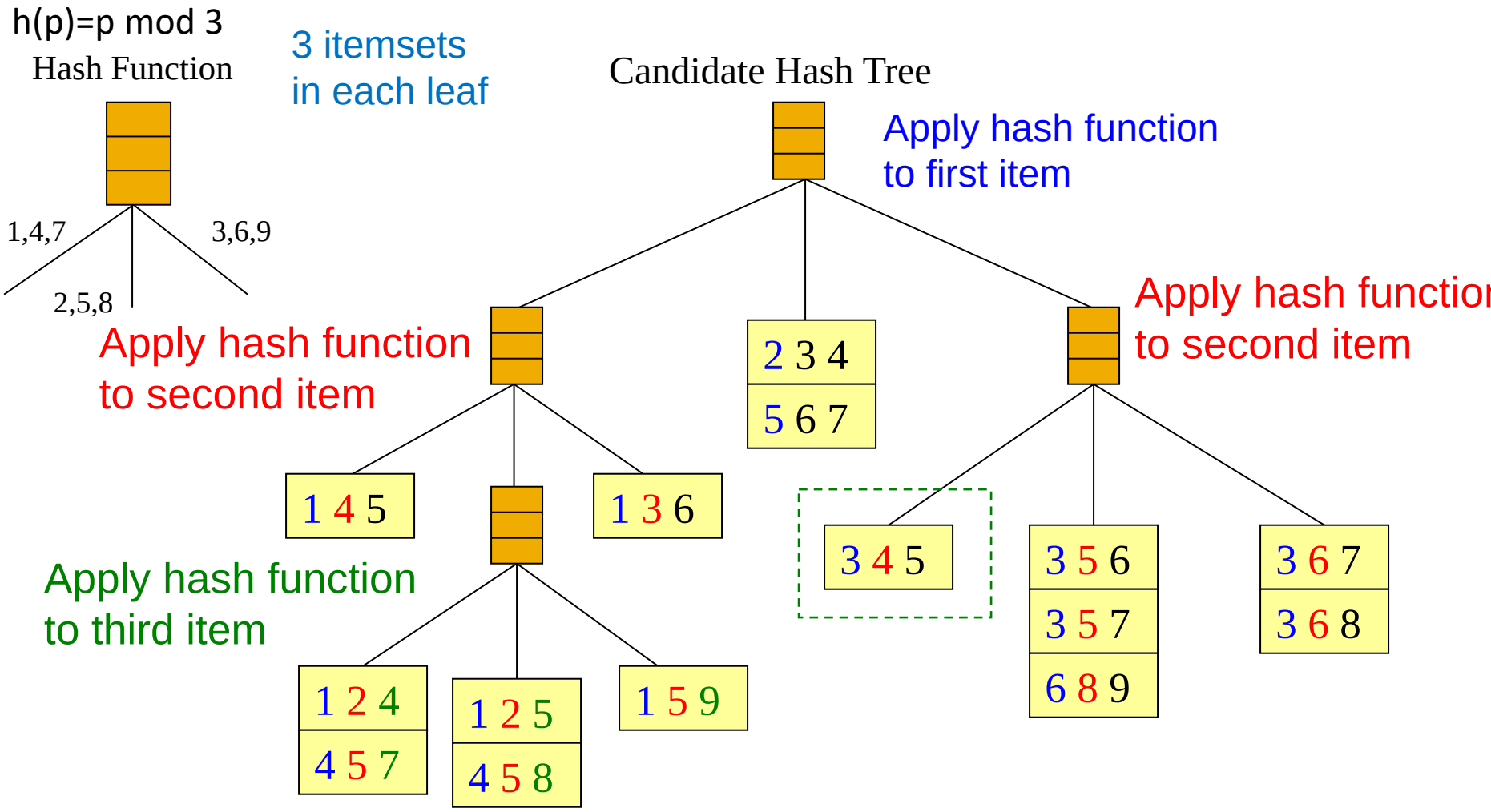


How?



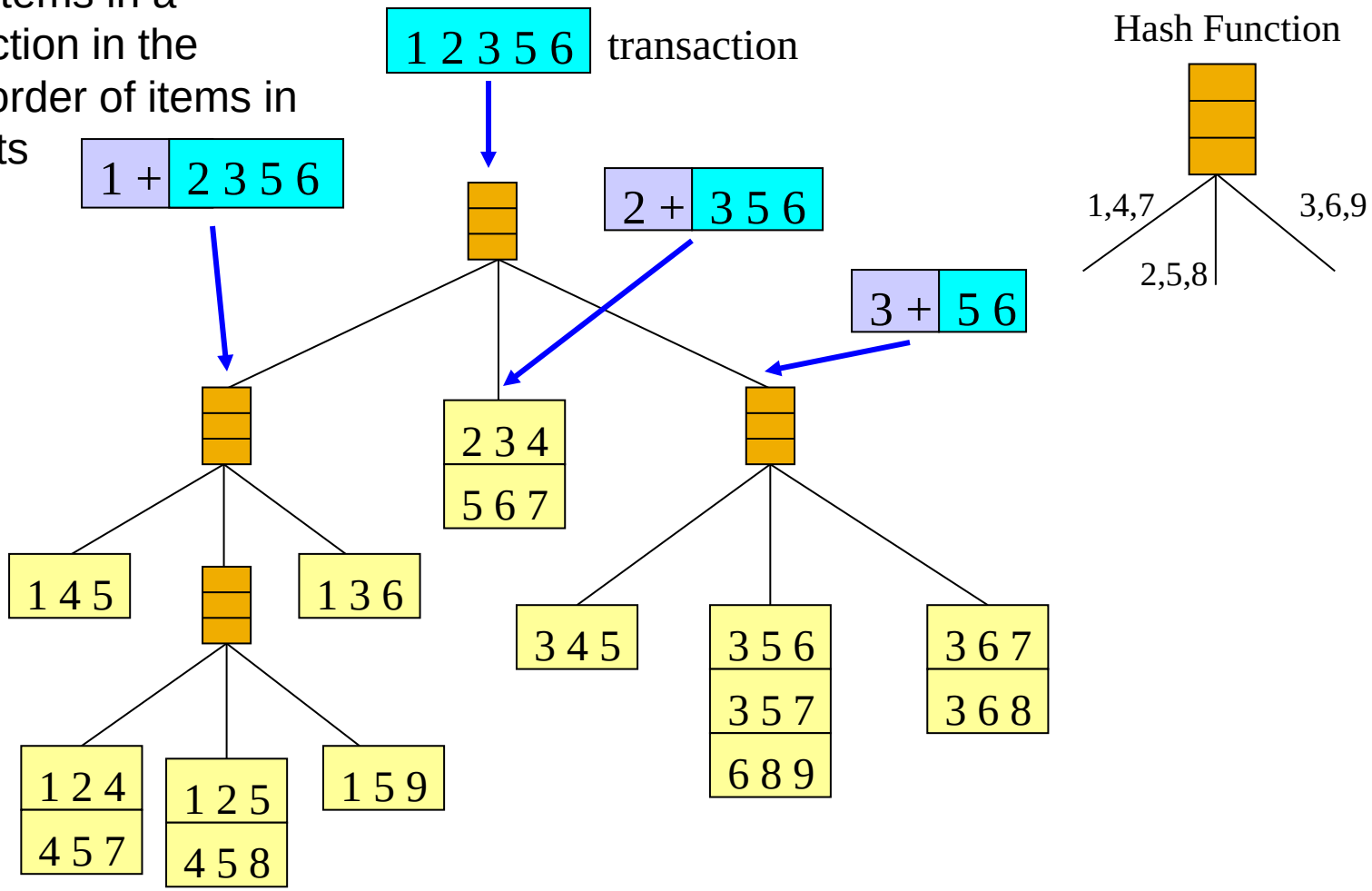
- Order items in each itemset
- Apply the hash function to each item in each itemset in order

Building a Hash Tree of Itemsets

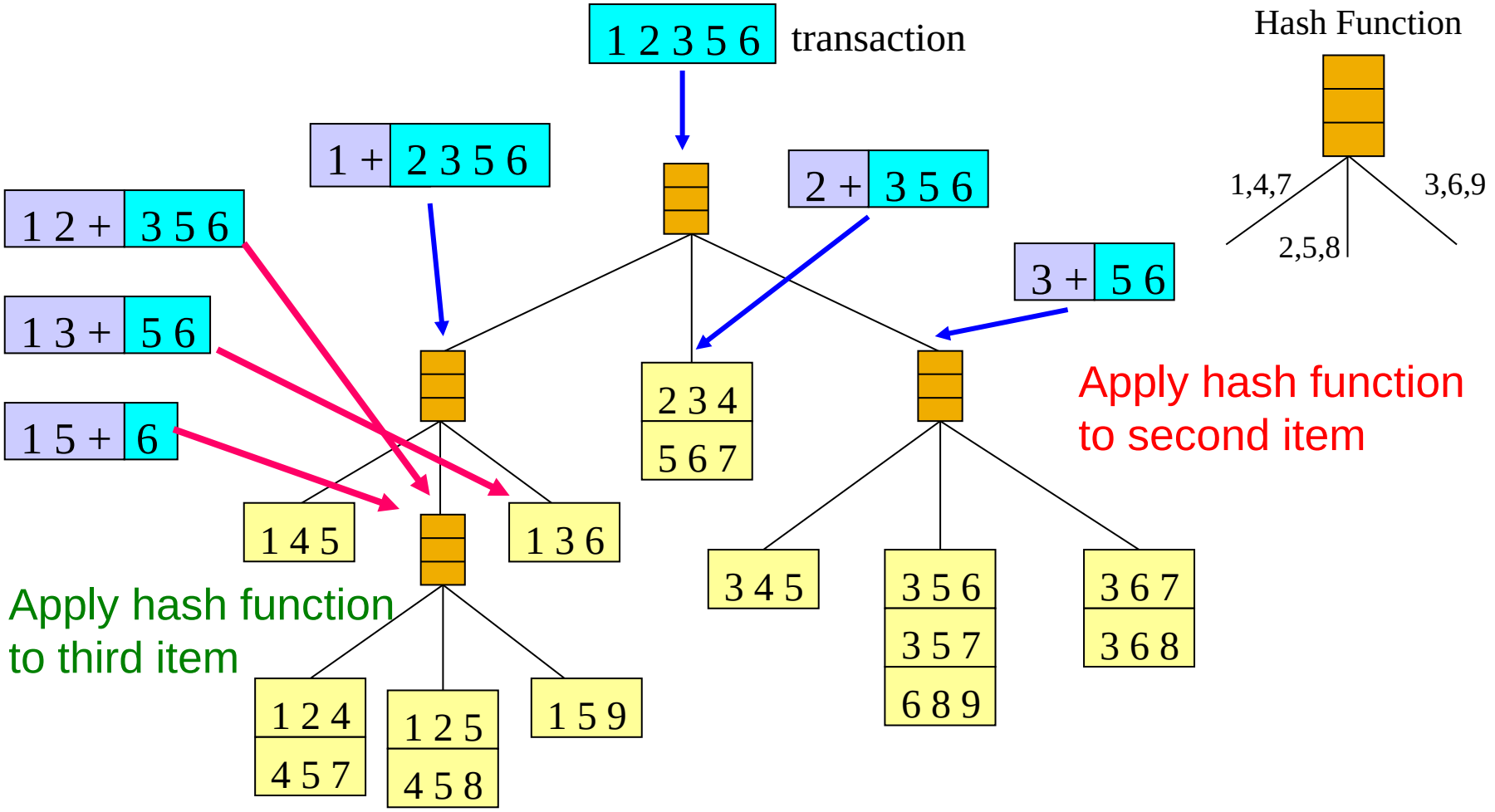


Support Counting Using a Hash Tree

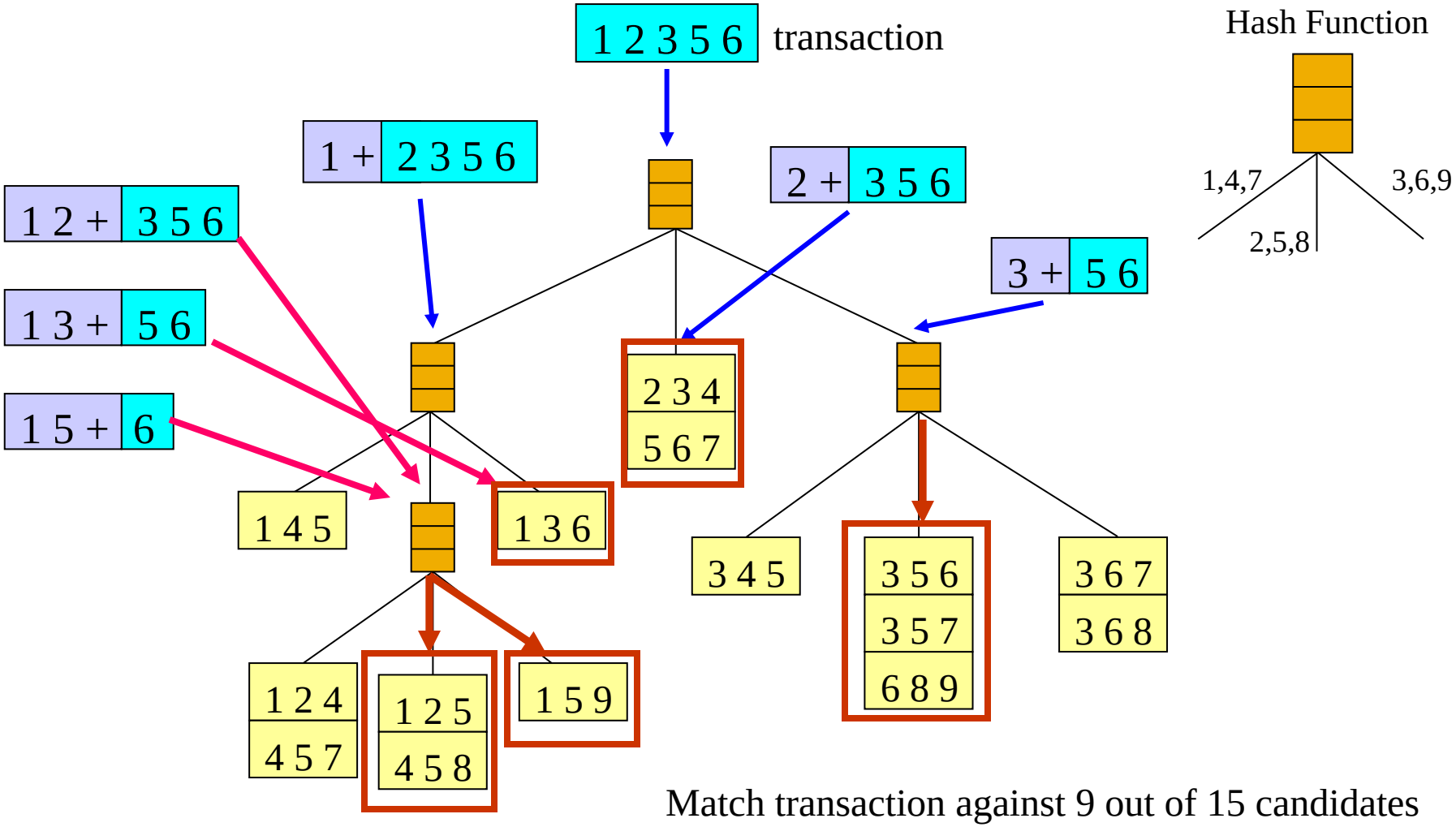
Order items in a transaction in the same order of items in itemsets



Support Counting Using a Hash Tree



Support Counting Using a Hash Tree



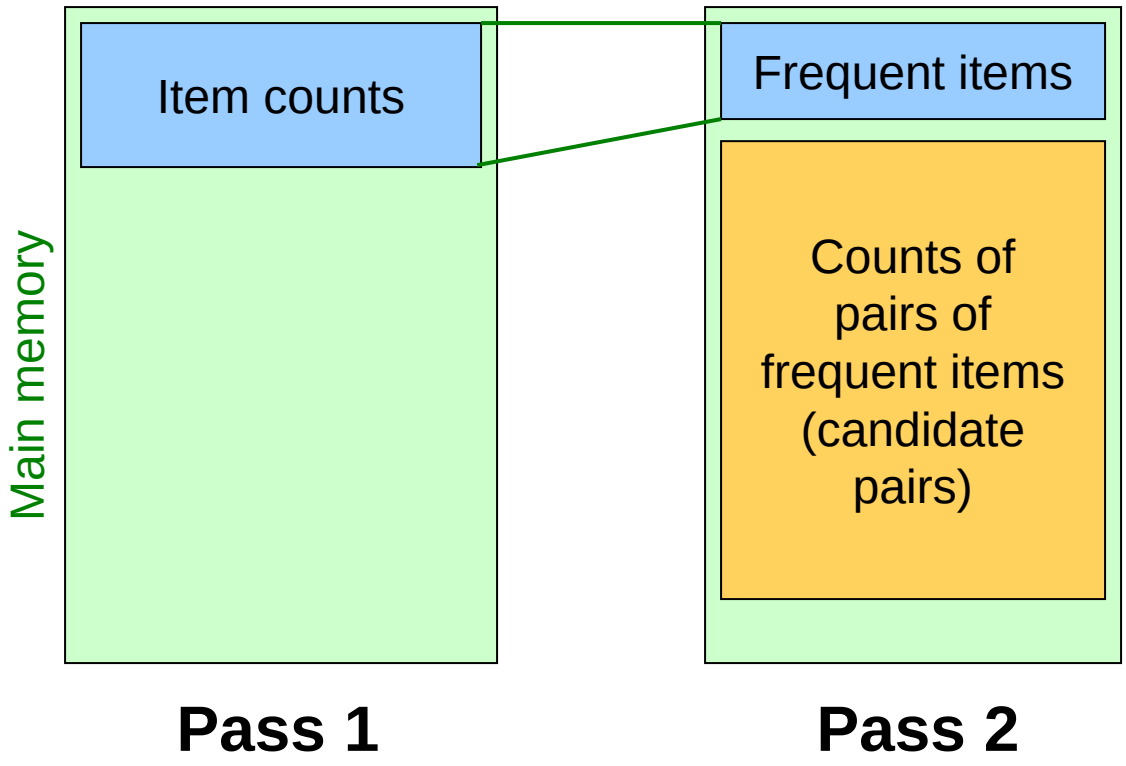
Match transaction against 9 out of 15 candidates

Implementation issues of A-Priori Algorithm

Implementation of A-Priori Algorithm

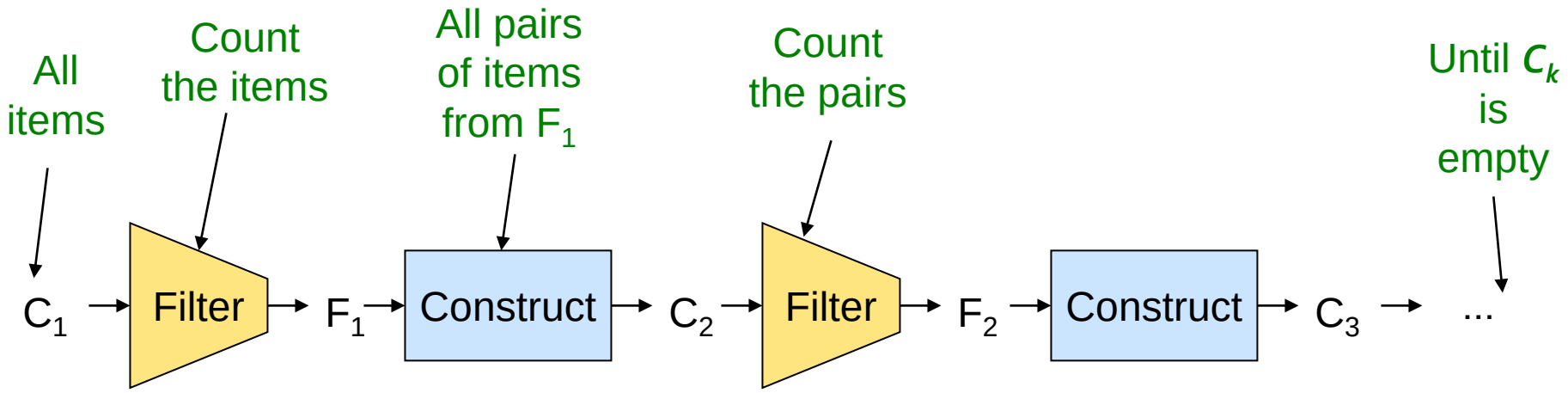
- **Pass 1:** Read baskets and count in main memory the occurrences of each **individual item**
  - Requires only memory proportional to #items
- **Items that appear $\geq s$ times are the frequent items**
- **Pass 2:** Read baskets again and count in main memory only those pairs where both elements are frequent (from Pass 1)
  - Requires memory proportional to square of **frequent** items only (for counts)
  - Plus a list of the frequent items (so you know what must be counted)

Main-Memory: Picture of A-Priori



Frequent Triples, Etc.

- **Other passes: For each *k*, we construct two sets of *k-itemsets*** (sets of size *k*):
  - C_k = ***candidate k-itemsets***
  - F_k = the set of truly frequent ***k***-itemsets
  - For each k, need room in main memory to count each candidate, and need to make a pass of the data to count the occurrences for itemsets in C_k



PCY (Park-Chen-Yu) Algorithm

PCY (Park-Chen-Yu) Algorithm

- **Observation:**
  In pass 1 of A-Priori, most memory is idle
  - We store only individual item counts
  - **Can we use the idle memory to reduce memory required in pass 2?**
- **Pass 1 of PCY:** In addition to item counts, maintain a hash table with as many buckets as fit in memory
  - Keep a **count** for each bucket into which **pairs** of items are hashed
    - **For each bucket just keep the count, not the actual pairs that hash to the bucket!**

PCY Algorithm – First Pass

```
FOR (each basket) :
    FOR (each item in the basket) :
        add 1 to item's count;
    FOR (each pair of items) :
        hash the pair to a bucket;
        add 1 to the count for that bucket;
```

New in PCY

- **Few things to note:**
  - Pairs of items need to be generated from the input file; they are not present in the file
  - We are not just interested in the presence of a pair, but we need to see whether it is present at least ***s*** (support) times

Observations about Buckets

- **Observation: If a bucket contains a frequent pair, then the bucket is surely frequent**
- However, even without any frequent pair, a bucket can still be frequent ☹
  - So, we cannot use the hash to eliminate any member (pair) of a "frequent" bucket
- **But, for a bucket with total count less than s, none of its pairs can be frequent ☺**
  - Pairs that hash to this bucket can be eliminated as candidates (even if the pair consists of 2 frequent items)
- **Pass 2:**
  Only count pairs that hash to frequent buckets

Example: PCY Algorithm First Pass

- **Items** = {milk, coke, beer, pepsi, juice}
- **Baskets**:

$$\mathbf{B_1} = \{\mathbf{m}, \mathbf{c}, \mathbf{b}\} \qquad \mathbf{B_2} = \{\mathbf{m}, \mathbf{p}, \mathbf{j}\}$$
$$\mathbf{B_3} = \{\mathbf{m}, \mathbf{b}\} \qquad \mathbf{B_4} = \{\mathbf{c}, \mathbf{j}\}$$
$$\mathbf{B_5} = \{\mathbf{m}, \mathbf{p}, \mathbf{b}\} \qquad \mathbf{B_6} = \{\mathbf{m}, \mathbf{c}, \mathbf{b}, \mathbf{j}\}$$
$$\mathbf{B_7} = \{\mathbf{c}, \mathbf{b}, \mathbf{j}\} \qquad \mathbf{B_8} = \{\mathbf{b}, \mathbf{c}\}$$

- By assigning $milk = 1, coke = 2,\ beer = 3,$ $pepsi = 4, juice = 5$ then **Baskets**:

$$\mathbf{B_1} = \{\mathbf{1}, \mathbf{2}, \mathbf{3}\} \qquad \mathbf{B_2} = \{\mathbf{1}, \mathbf{4}, \mathbf{5}\}$$
$$\mathbf{B_3} = \{\mathbf{1}, \mathbf{3}\} \qquad \mathbf{B_4} = \{\mathbf{2}, \mathbf{5}\}$$
$$\mathbf{B_5} = \{\mathbf{1}, \mathbf{3}, \mathbf{4}\} \qquad \mathbf{B_6} = \{\mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{5}\}$$
$$\mathbf{B_7} = \{\mathbf{2}, \mathbf{3}, \mathbf{5}\} \qquad \mathbf{B_8} = \{\mathbf{2}, \mathbf{3}\}$$

Example: PCY Algorithm First Pass

- Define a hash function (Hashing pair (i, j) to bucket K):

$$h(i, j) = (i + j)\%5 = K$$

Example:

$$h(1,3) = (1 + 3)\%5 = 4 \rightarrow Bucket\ 4$$

Example: PCY Algorithm First Pass

- First Pass:

```
FOR (each basket) :
     FOR (each item in the basket) :
          add 1 to item's count;
     FOR (each pair of items) :
          hash the pair to a bucket;
          add 1 to the count for that bucket;
```

Baskets:

$B_1 = \{1, 2, 3\}$ $B_2 = \{1, 4, 5\}$

$B_3 = \{1, 3\}$ $B_4 = \{2, 5\}$

$B_5 = \{1, 3, 4\}$ $B_6 = \{1, 2, 3, 5\}$

$B_7 = \{2, 3, 5\}$ $B_8 = \{2, 3\}$

Item #	Count
1	5 +1
2	5
3	5
4	2
5	4

Example: PCY Algorithm First Pass

- First Pass:

```
FOR (each basket) :
    FOR (each item in the basket) :
        add 1 to item's count;
FOR (each pair of items) :
        hash the pair to a bucket;
        add 1 to the count for that bucket;
```

$(1,4),(2,3) \rightarrow h(i,j) = 0$
$(1,5),(2,4) \rightarrow h(i,j) = 1$
$(2,5),(3,4) \rightarrow h(i,j) = 2$
$(1,2),(3,5) \rightarrow h(i,j) = 3$
$(1,3),(4,5) \rightarrow h(i,j) = 4$

Baskets:

$B_1 = \{1,2,3\}$
$B_2 = \{1,4,5\}$
$B_3 = \{1,3\}$
$B_4 = \{2,5\}$
$B_5 = \{1,3,4\}$
$B_6 = \{1,2,3,5\}$
$B_7 = \{2,3,5\}$
$B_8 = \{2,3\}$

$h(2,5) = 2 \rightarrow Bucket\ 2$

$h(1,2) = 3 \rightarrow Bucket\ 3$

Hash Table:

Bucket #	Count
0	6
1	2
2	4 +1
3	4 +1
4	5

Example: PCY Algorithm First Pass

- For support threshold $s = 3$

Item Count

Item #	Count
1	5
2	5
3	5
4	2
5	4

Itemsets in each bucket

$(\mathbf{1},\mathbf{4}), (\mathbf{2},\mathbf{3}) \rightarrow \mathbf{0}$

$(\mathbf{1},\mathbf{5}), (\mathbf{2},\mathbf{4}) \rightarrow \mathbf{1}$

$(\mathbf{2},\mathbf{5}), (\mathbf{3},\mathbf{4}) \rightarrow \mathbf{2}$

$(\mathbf{1},\mathbf{2}), (\mathbf{3},\mathbf{5}) \rightarrow \mathbf{3}$

$(\mathbf{1},\mathbf{3}), (\mathbf{4},\mathbf{5}) \rightarrow \mathbf{4}$

Hash Table

Bucket #	Count
0	6
1	2
2	4
3	4
4	5

- For $s = 3$, $L_1 = \{1,2,3,5\}$, and Bitmap $\{1,0,1,1,1\}$

PCY Algorithm – Between Passes

- **Optimization: Replace the buckets by a bit-vector:**
  - **1** means the bucket count exceeded the support s (call it a **frequent bucket**); **0** means it did not
  - **4-byte integer counts are replaced by bits, so the bit-vector requires 1/32 of memory**
- Also, decide which items are frequent and list them for the second pass

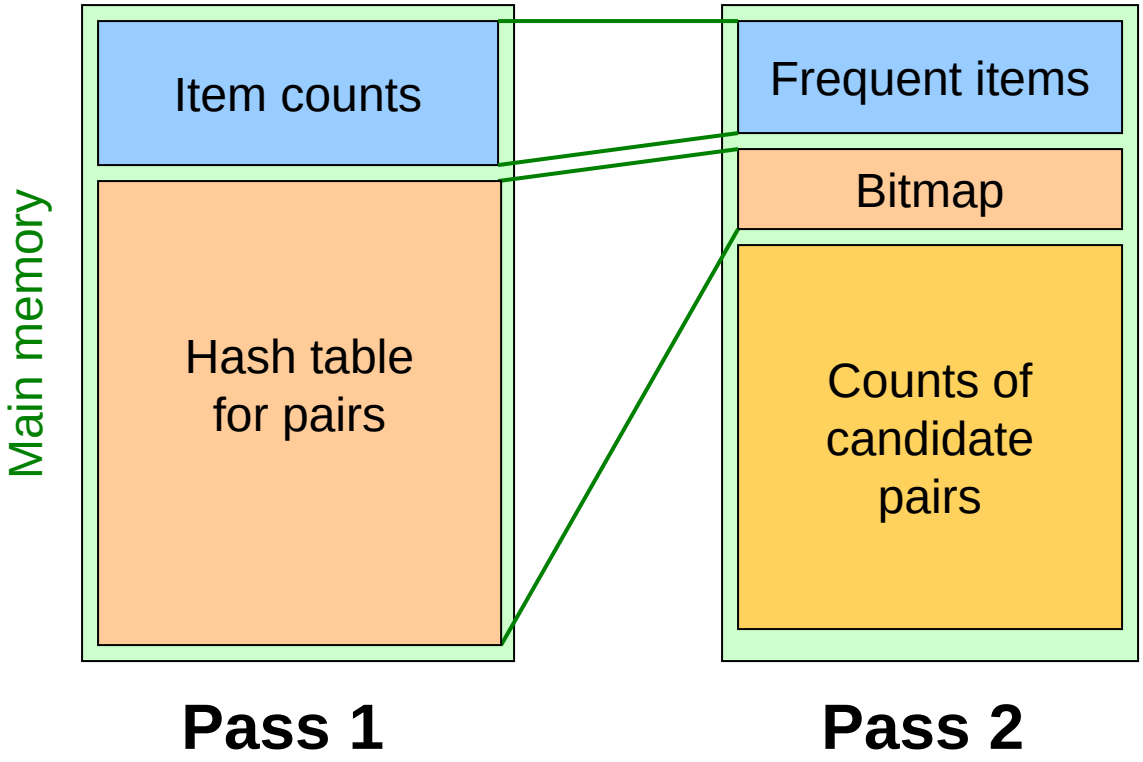
PCY Algorithm – Pass 2

- Count all pairs ***{i, j}*** that meet the conditions for being a **candidate pair**:
  1. Both ***i*** and ***j*** are frequent items
  2. The pair ***{i, j}*** hashes to a bucket whose bit in the bit vector is **1** (i.e., a **frequent bucket**)

- **Both conditions are necessary for the pair to have a chance of being frequent**

- Implementation details: On second pass, a table of (item, item, count) triples is essential (we cannot use triangular matrix approach, why?)
  - Thus, hash table must eliminate approx. 2/3 of the candidate pairs for PCY to beat A-Priori

Main-Memory: Picture of PCY



Example: PCY Algorithm 2nd Pass

For $\{i, j\}$ to be a **candidate pair**:

1. Both i and j are frequent items
2. The pair $\{i, j\}$ hashes to a bucket whose bit in the bit vector is **1** (i.e., a **frequent bucket**)

Item Count

Item #	Count
1	5
2	5
3	5
4	2
5	4

Hash Table

Bucket #	Count
0	6
1	2
2	4
3	4
4	5

$\sout{(1,4)}, (2,3) \rightarrow h(i,j) = 0$

$(1,5), \sout{(2,4)} \rightarrow h(i,j) = 1$

$(2,5), \sout{(3,4)} \rightarrow h(i,j) = 2$

$(1,2), (3,5) \rightarrow h(i,j) = 3$

$(1,3), \sout{(4,5)} \rightarrow h(i,j) = 4$

Frequent Items: $\{1, 2, 3, 5\}$, frequent buckets: 0,2,3,4

Example: PCY Algorithm 2nd Pass

For $\{i, j\}$ to be a **candidate pair**:

1. Both i and j are frequent items
2. The pair $\{i, j\}$ hashes to a bucket whose bit in the bit vector is **1** (i.e., a **frequent bucket**)

Item Count

Item #	Count
1	5
2	5
3	5
4	2
5	4

Hash Table

Bucket #	Count
0	6
1	2
2	4
3	4
4	5

$\cancel{(1,4)}, (2,3) \rightarrow h(i,j) = 0$

$\cancel{(1,5), (2,4) \rightarrow h(i,j) = 1}$

$(2,5), \cancel{(3,4)} \rightarrow h(i,j) = 2$

$(1,2), (3,5) \rightarrow h(i,j) = 3$

$(1,3), \cancel{(4,5)} \rightarrow h(i,j) = 4$

Frequent Items: $\{1, 2, 3, 5\}$, frequent buckets: 0,2,3,4

Example: PCY Algorithm 2nd Pass

For $\{i, j\}$ to be a **candidate pair**:

1. Both i and j are frequent items
2. The pair $\{i, j\}$ hashes to a bucket whose bit in the bit vector is **1** (i.e., a **frequent bucket**)

$\sout{(1,4)}, (2,3) \rightarrow h(i,j) = 0$
$\sout{(1,5), (2,4) \rightarrow h(i,j) = 1}$
$(2,5), \sout{(3,4)} \rightarrow h(i,j) = 2$
$(1,2), (3,5) \rightarrow h(i,j) = 3$
$(1,3), \sout{(4,5)} \rightarrow h(i,j) = 4$

Candidate Pairs & Counts

Pair	Count
(2,3)	4
(2,5)	3
(1,2)	2
(3,5)	2
(1,3)	4

Frequent Itemsets are: $\{1\}, \{2\}, \{3\}, \{5\}, \{1,3\}, \{2,3\}, \{2,5\}$

Interestingness of association rules, other types of patterns

Interesting Association Rules

- **Not all high-confidence rules are interesting**
  - The rule $X \rightarrow \boldsymbol{milk}$ $\qquad \text{conf}(X \rightarrow milk) = \dfrac{\text{support}(X \cup milk)}{\text{support}(X)}$
  - The rule $X \rightarrow \boldsymbol{milk}$ may have high confidence for many itemsets $\boldsymbol{X}$, because milk is just purchased very often (independent of $\boldsymbol{X}$) and the confidence will be high
- **Interest** of an association rule $\boldsymbol{I \rightarrow j}$: difference between its confidence and the fraction of baskets that contain $\boldsymbol{j}$

  $$Interest(I \rightarrow j) = conf(I \rightarrow j) - \Pr[j]$$

  - Interesting rules are those with high positive or negative interest values (ABS usually above 0.5)

Example: Interest

B_1 = {m, c, b} B_2 = {m, p, j}

B_3 = {m, c, b} B_4 = {c, j}

B_5 = {m, p, c, b} B_6 = {m, c, b, j}

B_7 = {c, b, j} B_8 = {b, c}

$$Interest(I \rightarrow j) = conf(I \rightarrow j) - \Pr[j]$$

- **Association rule: {m, b} →c**
  - **Confidence** = 4/4 = 1
  - **Interest** = |4/4 – 7/8| = 1/8
    - Item ***c*** appears in 7/8 of the baskets
    - Rule is not very interesting!
- Another rule: **{j} →c**
  - **Interest** = |3/4 – 7/8| = 1/8

More about Judging if a Rule/Pattern Is Interesting?

- Interestingness measures: Objective vs. subjective
  - Objective interestingness measures
    - Support, confidence, interest, correlation(lift), …
  - Subjective interestingness measures:
    - Let a user specify
      - Query-based: Relevant to a user's particular request
    - Judge against one's knowledge-base
      - unexpected, freshness, timeliness

Limitation of the Support-Confidence

- Another Example: Suppose one school may have the following statistics on # of students who may play basketball and/or eat cereal:

	play-basketball	not play-basketball	sum (row)
eat-cereal	400	350	750
not eat-cereal	200	50	250
sum(col.)	600	400	1000

2-way contingency table

- Association rule mining may generate the following:
  - *play-basketball* $\Rightarrow$ *eat-cereal* [support: 40%, conf: 66.7%]
- But the overall % of students eating cereal is 75% > 66.7% (eat-cereal appears in most of data). a more telling rule:
  - ¬ *play-basketball* $\Rightarrow$ *eat-cereal* [support: 35%, conf: 87.5%]

Interestingness Measure: Lift

- Measure of dependent/correlated events: **lift**

$$lift(B,C) = \frac{conf(B \rightarrow C)}{support(C)} = \frac{support(B \cup C)}{support(B) \times support(C)}$$

Note that here support(.) is defined as relative support

	B	¬B	Σ_{row}
C	400	350	750
¬C	200	50	250
$\Sigma_{col.}$	600	400	1000

- Lift(B, C) may tell how B and C are correlated
  - Lift(B, C) = 1: B and C are independent
  - > 1: positively correlated
  - < 1: negatively correlated
- For our example,

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

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

- B and C are negatively correlated since lift(B, C) < 1;
- B and ¬C are positively correlated since lift(B, ¬C) > 1

Compact Representation of Frequent Itemsets

- Some frequent itemsets are redundant because their supersets are also frequent

Consider the following data set. Assume support threshold =5

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

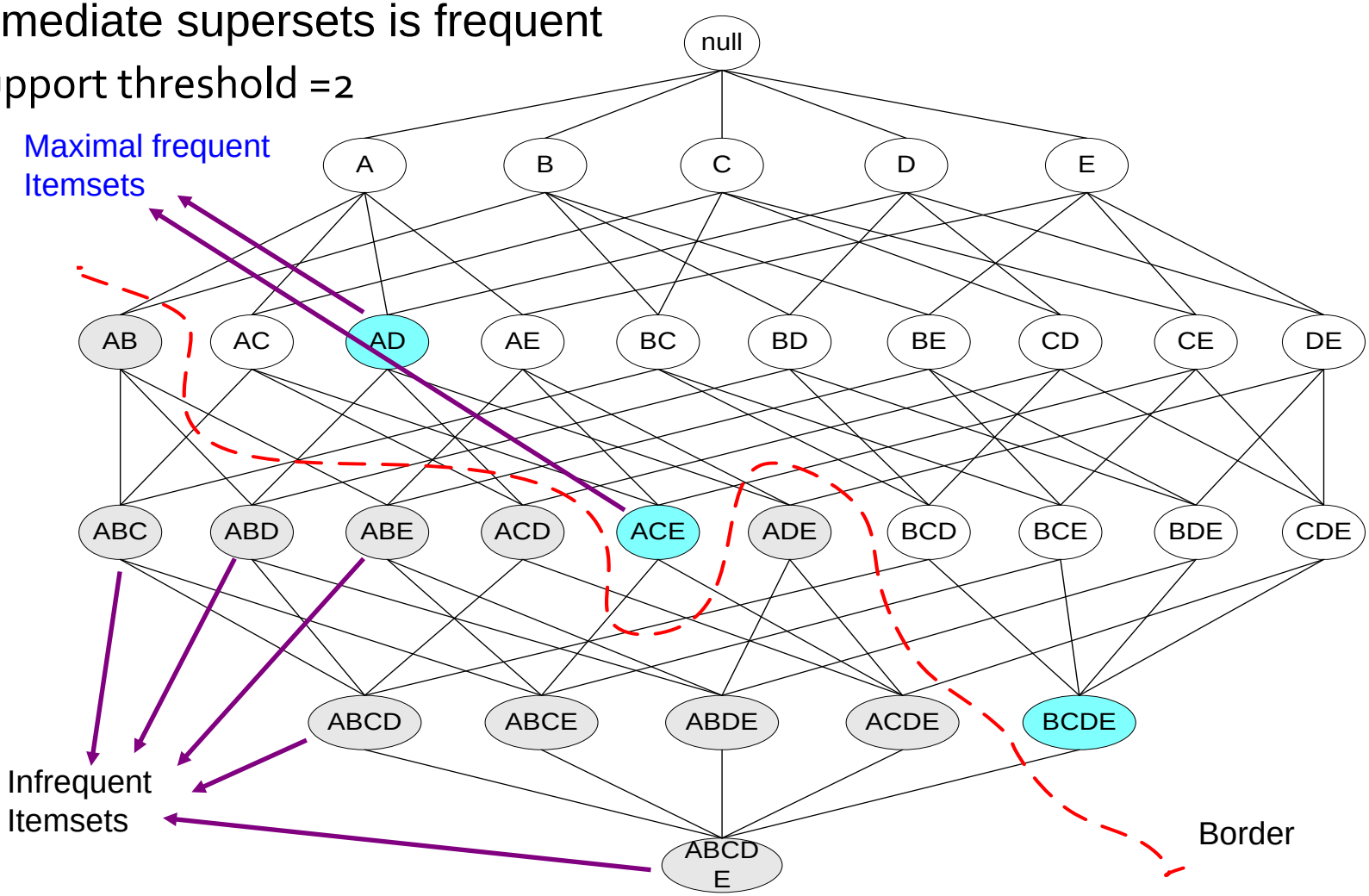
- Number of frequent itemsets $= 3 \times \sum_{k=1}^{10} \binom{10}{k}$

- Need a compact representation

Maximal Frequent Itemset

An itemset is maximal frequent if it is frequent and none of its immediate supersets is frequent

support threshold =2



What are the Maximal Frequent Itemsets in this Data?

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

Minimum support threshold = 5

(A1-A10)

(B1-B10)

(C1-C10)

Closed Itemset

- An itemset X is closed if none of its immediate supersets has the same support as the itemset X.
- X is NOT closed if at least one of its immediate supersets has support count as X, i.e., contained by the same set of baskets as X .

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

- {C}:4 is closed: none of its immediate supersets (AC:3, BC:3, DC:2; EC:2)has the same support.
- {A}:3 is NOT closed: its immediate superset AC has the same support.
- {AB}:2 is NOT closed
- {ABC}:2 is closed

- Closed Frequent itemset: Closed & frequent

What are the Closed Itemsets in this Data?

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

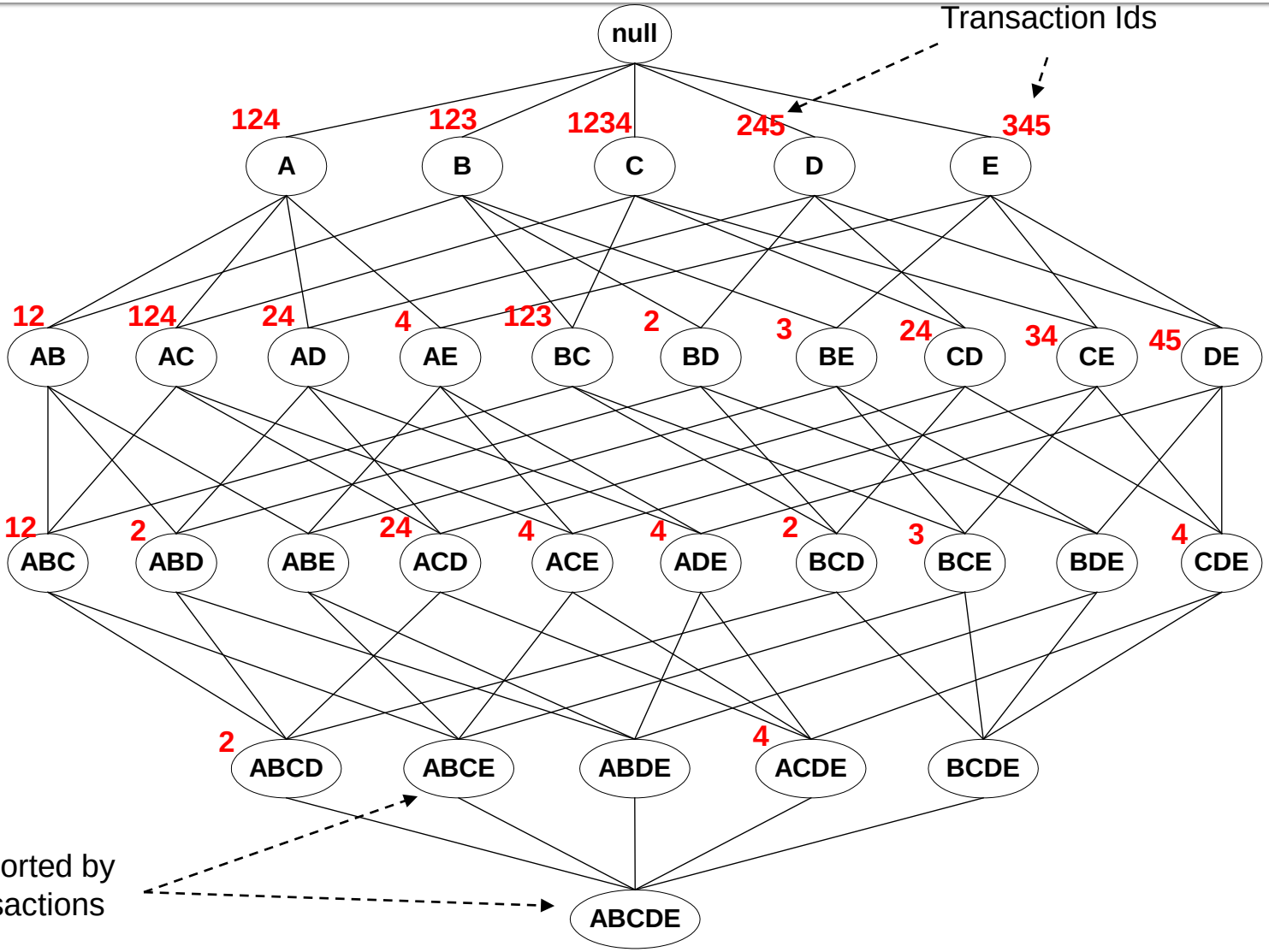
(A1-A10)

(B1-B10)

(C1-C10)

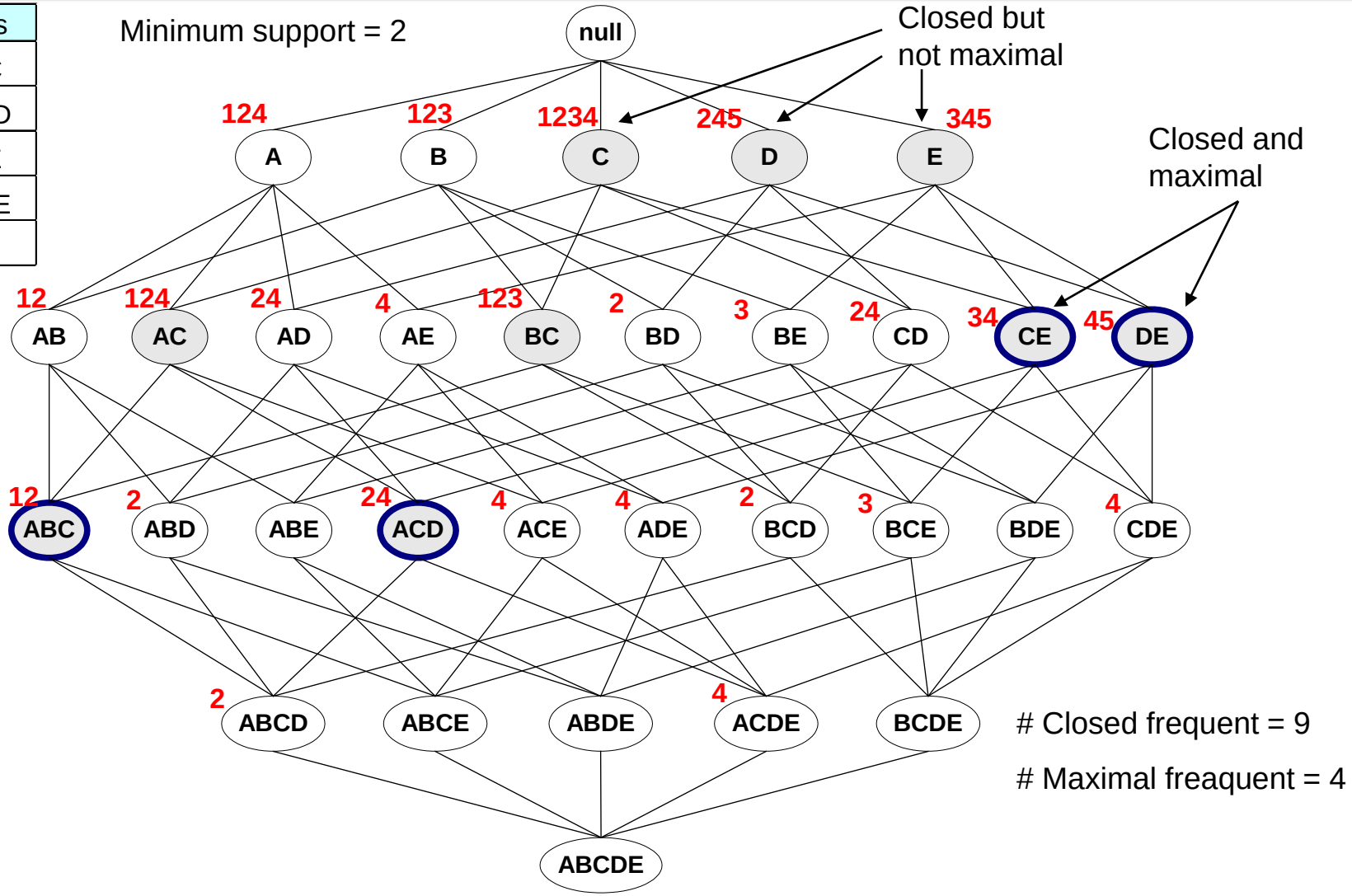
Lattice representation of itemsets

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



Maximal Frequent vs Closed Frequent Itemsets

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



Closed frequent = 9

Maximal freaquent = 4

Maximal vs Closed Itemsets

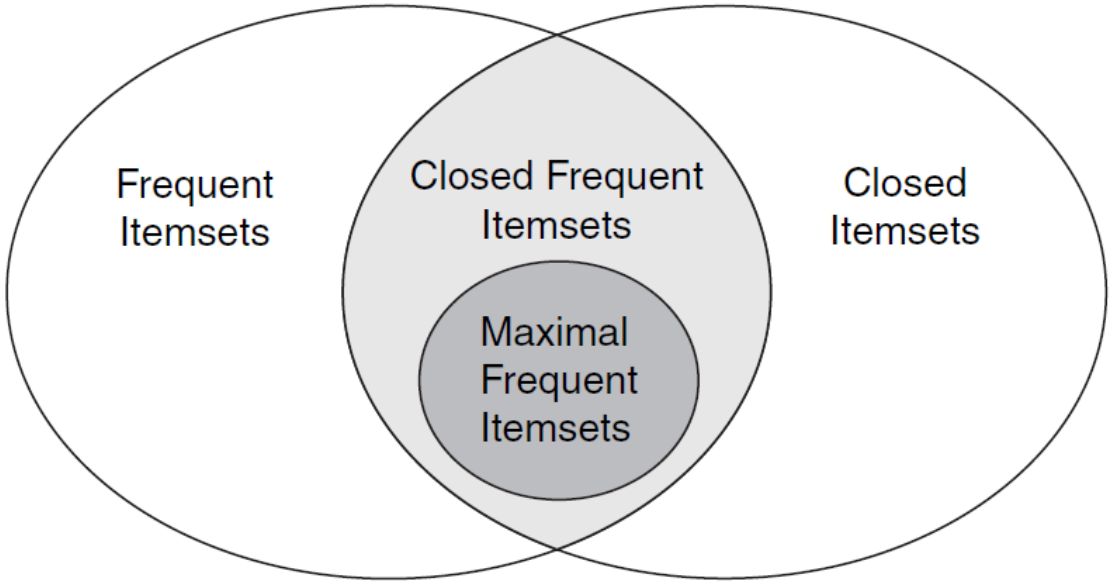


Figure 5.18. Relationships among frequent, closed, closed frequent, and maximal frequent itemsets.

Summary

- **Frequent itemsets and Association rules**
  - Understand the concepts
- **Algorithms for finding frequent itemsets**
  - Understand the A-Priori algorithm
- **Efficient Implementation of Algorithms**
  - Understand how to implement Apriori and PCY algorithms
- **Interestingness of association rules**
  - Understand the basic concept
- **Closed itemset and maximal itemset.**
  - Understand the concepts and their differences