Mining Frequent Patterns, Associations and Correlations

Cam Tu Nguyen

阮锦绣

Software Institute, Nanjing University
nguyenct@lamda.nju.edu.cn
ncamtu@gmail.com

Frequent Patterns

- **Frequent Patterns** are patterns (e.g. itemsets, subsequences, or substructures) that appear frequently in a data set.
  - Examples:
    - Milk and Bread in a transaction data
    - A subsequence, such as buying first a PC, then a digital camera, then a memory card, is a *(frequent) sequential pattern.*

- A **substructure** may be combined with itemsets or subsequences.
  - A substructure occurs frequently is called a *(frequent) structured pattern.*

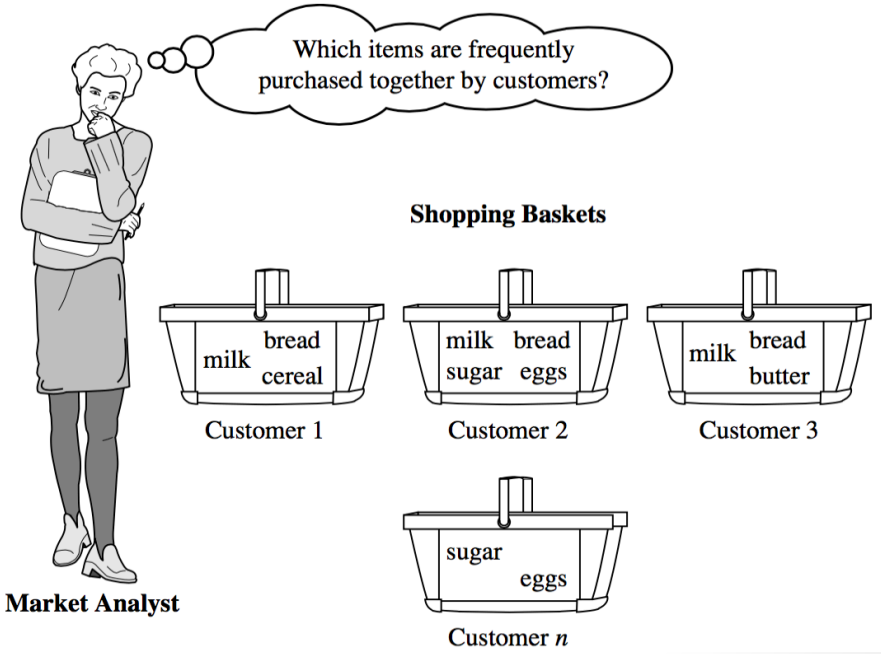
Outline

- Basic Concepts
- Frequent Itemset Mining Methods
  - Apriori Algorithm
  - Pattern Growth Approach
- Pattern Evaluation Methods

Market Basket Analysis

- Frequent itemset mining leads to the discovery of **associations** and **correlations** among items in large data sets.
  - This helps in many **business decision making** processes such as catalog design, cross-marketing, and customer shopping behavior analysis.

- Market-basket analysis analyzes customer buying habits.



Market Basket Analysis

- Each **basket** can be represented by a Boolean vector, each element represents the presence or absence of an item.

- We analyze baskets (boolean vectors) to find **buying patterns** in the form of **association rules**

$$computer \Rightarrow antivirus\_software\ [support = 2\%, confidence = 60\%].$$

- Basic Concepts
  - **Support**
  - **Confidence**
  - **Minimum Support Threshold**
  - **Mininum Confidence Threshold**

Frequent Itemsets, Closed Itemsets and Association Rules

- Let $\mathcal{I} = \{I_1, I_2, \ldots, I_m\}$ be an itemset
- D is a set of **database transactions**
  - Each transaction $T \subseteq \mathcal{I}$ is a nonempty itemset, and associated with a TID (transaction ID)

- An itemset *A* is contained in *T (i.e. T contains A)* if $A \subseteq T$.
- An association rule is of the form
  $A \Rightarrow B$, where $A \subset \mathcal{I}$, $B \subset \mathcal{I}$, $A \neq \emptyset$, $B \neq \emptyset$, and $A \cap B = \phi$.
- Support and confidence

$$support(A \Rightarrow B) = P(A \cup B)$$

$$confidence(A \Rightarrow B) = P(B|A).$$

Frequent Itemsets, Closed Itemsets and Association Rules

- Association rules that satisfy both a mininum support threshold, and a mininum confidence threshold is called **strong**.

- **Itemset** and **k-itemsets**
  - Example: the set {computer, antivirus_software} is a 2-itemset.

- **Occurrence frequency** of an itemset is the number of transactions that contain the itemset.
  - Also known as **frequency**, **support count**, or **count** of the itemset.
  - If an itemset satisfies a predefined **mininum support threshold**, then *I* is a **frequent** itemset.

Frequent Itemsets, Closed Itemsets and Association Rules

- Confidence can be calculated using support count.

$$confidence(A \Rightarrow B) = P(B|A) = \frac{support(A \cup B)}{support(A)} = \frac{support\_count(A \cup B)}{support\_count(A)}.$$

- The problem of mining association rules can be reduced to that of mining frequent itemsets.

- Association rule mining as a two-step process
  - Finding all itemsets (most computational cost coming from this step)
  - Generate association rules from the frequent itemsets.

Frequent Itemsets, Closed Itemsets and Association Rules

- Main Challenge in Mining Frequent Itemsets
  - Combinational number of itemsets.

- **Closed frequent itemset**
  - An itemset X is a **closed** itemset in a data set *D,* if there exists no *proper super-itemset* Y such that Y has the same **support count** as X in D
  - An itemset X is a **closed frequent itemset** in D if X is both closed and frequent in D.

- **Maximum frequent itemset**
  - X is a maximum frequent itemset in D if X is frequent, and there exists no super-itemset Y such that Y is frequent in D.

Frequent Itemsets, Closed Itemsets and Association Rules

- Closed and Maximum Frequent Itemsets:
  - Let *C* be the set of closed frequent itemsets for a data set *D* satisfying a minimum support threshold, *min_sup.*
  - Let *M* be the set of maximal frequent itemsets for *D* satisfying *min_sup*
  - *C contains complete information regarding its corresponding frequent itemsets while M doesn't.*
- Example:
  - D = $\{\langle a_1, a_2, \ldots, a_{100}\rangle; \langle a_1, a_2, \ldots, a_{50}\rangle\}$. *min_sup=1*
  - *C* = $\{\{a_1, a_2, \ldots, a_{100}\} : 1; \{a_1, a_2, \ldots, a_{50}\} : 2\}$
  - *M* = $\{\{a_1, a_2, \ldots, a_{100}\} : 1\}$
  - From *C,* we can derive $\{a_2, a_{45}:2\}$ and $\{a_8, a_{55}:1\}$
  - From *M,* we can assert that both itemsets ($\{a_2, a_{45}\}$, $\{a_8, a_{55}\}$) are frequent, but can't read their actual support counts.

Outline

- Basic Concepts
- Frequent Itemset Mining Methods
  - Apriori Algorithm
  - Pattern Growth Approach
- Pattern Evaluation Methods

The Apriori Algorithm

- The Apriori algorithm was proposed by Agrawal and Srikant in 1994.

- Main ideas:
  - Find the frequent itemsets with *min_sup*
    - Apriori Property: A subset of a frequent itemset must also be a frequent itemset
      - i.e., if {AB} is a frequent itemset, both {A} and {B} should be a frequent itemset.
    - Iteratively find frequent itemsets with cardinality from 1 to k (k-itemset)

  - Use the frequent itemsets to generate association rules.

The Apriori Algorithm

- C_k: Candidate itemset of size k
- L_k: frequent itemset of size k (frequent k-itemset)

- Join step: C_k is generated by joining J_{k-1} with itself

- Prune step: Any (k-1)-itemset that is not frequent cannot be a subset of a frequent k-itemset.

The Apriori Algorithm

Input:

- D, a database of transactions;
- *min_sup*, the minimum support count threshold.

Output: L, frequent itemsets in D.

Method:

(1) L_1 = find_frequent_1-itemsets(D);
(2) **for** ($k = 2; L_{k-1} \neq \phi; k$++) {
(3) C_k = apriori_gen(L_{k-1});
(4) **for each** transaction $t \in D$ { // scan D for counts
(5) C_t = subset(C_k, t); // get the subsets of t that are candidates
(6) **for each** candidate $c \in C_t$
(7) c.count++;
(8) }
(9) $L_k = \{c \in C_k | c.count \geq min\_sup\}$
(10) }
(11) **return** $L = \cup_k L_k$;

The Apriori Algorithm

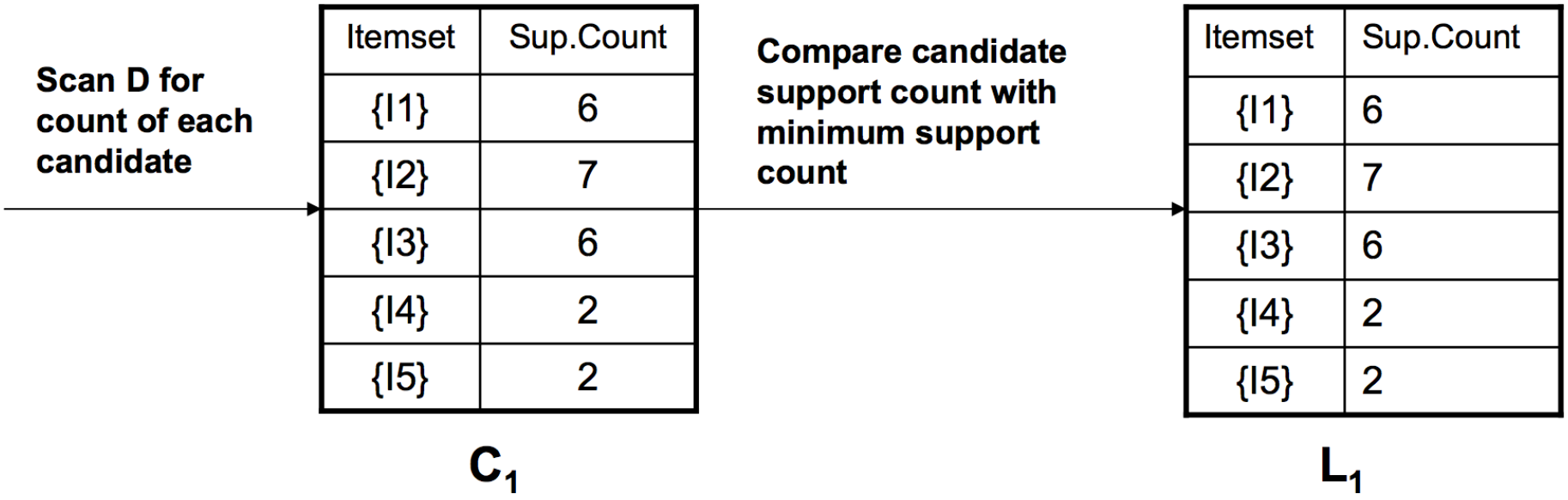
```
procedure apriori_gen(L_{k-1}:frequent (k-1)-itemsets)
(1)     for each itemset l_1 ∈ L_{k-1}
(2)         for each itemset l_2 ∈ L_{k-1}
(3)             if (l_1[1] = l_2[1]) ∧ (l_1[2] = l_2[2])
                    ∧... ∧ (l_1[k-2] = l_2[k-2]) ∧ (l_1[k-1] < l_2[k-1]) then {
(4)                 c = l_1 ⋈ l_2; // join step: generate candidates
(5)                 if has_infrequent_subset(c, L_{k-1}) then
(6)                     delete c; // prune step: remove unfruitful candidate
(7)                 else add c to C_k;
(8)             }
(9)     return C_k;

procedure has_infrequent_subset(c: candidate k-itemset;
            L_{k-1}: frequent (k-1)-itemsets); // use prior knowledge
(1)     for each (k-1)-subset s of c
(2)         if s ∉ L_{k-1} then
(3)             return TRUE;
(4)     return FALSE;
```

The Apriori Algorithm: Example

- Consider a database, D, consisting of 9 transactions
- Suppose min. support count required is 2 (i.e., min_sup=2/9= 22%).
- Let **minimum confidence required is 70%.**
- We have to first find out the frequent itemset using Apriori algorithm.
- Then, Association rules will be generated using min. support & min. confidence.

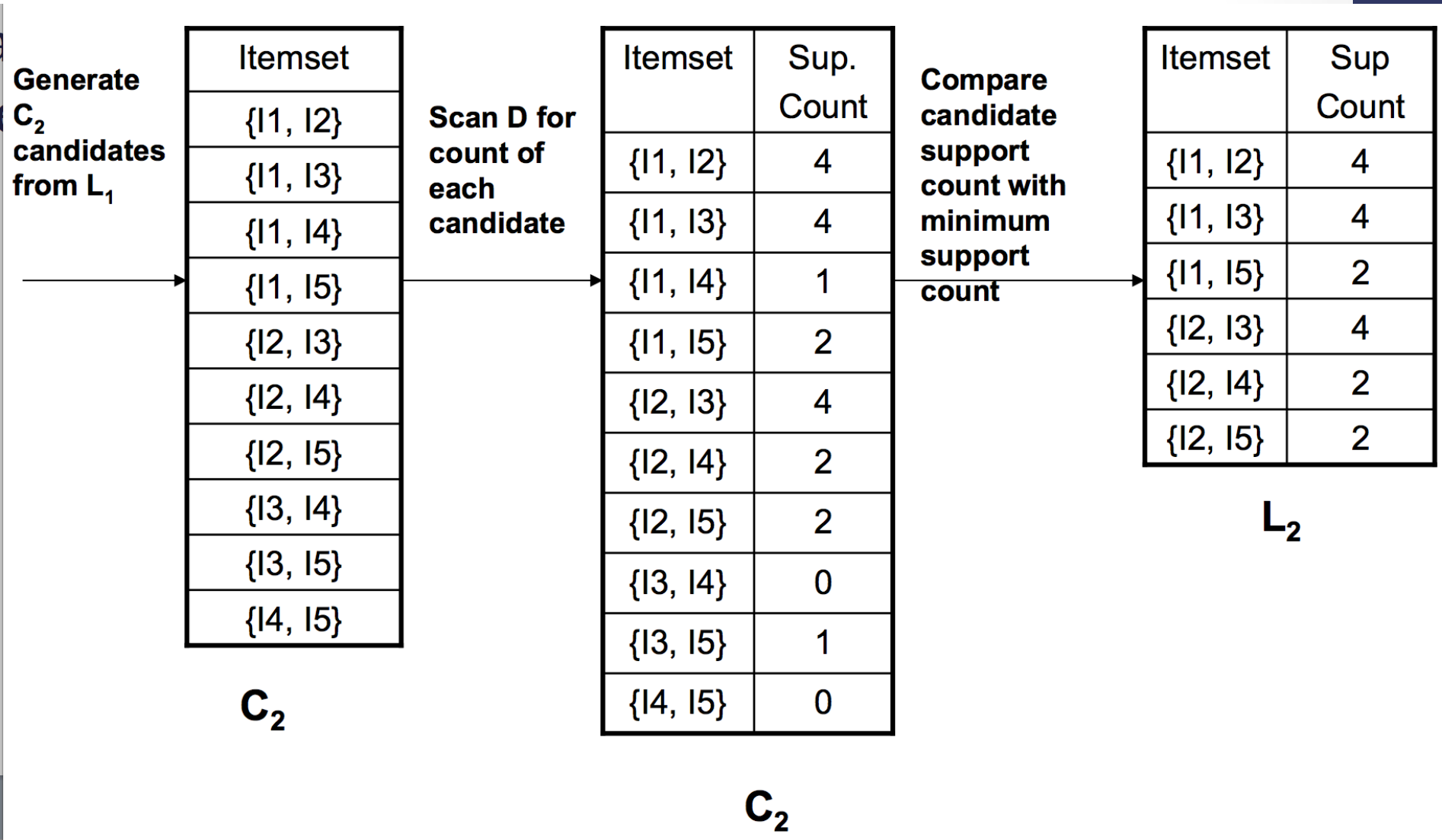
TID	List of Items
T100	I1, I2, I5
T100	I2, I4
T100	I2, I3
T100	I1, I2, I4
T100	I1, I3
T100	I2, I3
T100	I1, I3
T100	I1, I2 ,I3, I5
T100	I1, I2, I3

Step 1: Generating 1-itemset frequent pattern



- The set of frequent 1-itemsets, L1, consists of the candidate 1-itemsets satisfying minimum support.
- In the first iteration, each item is a member of the set of candidate.

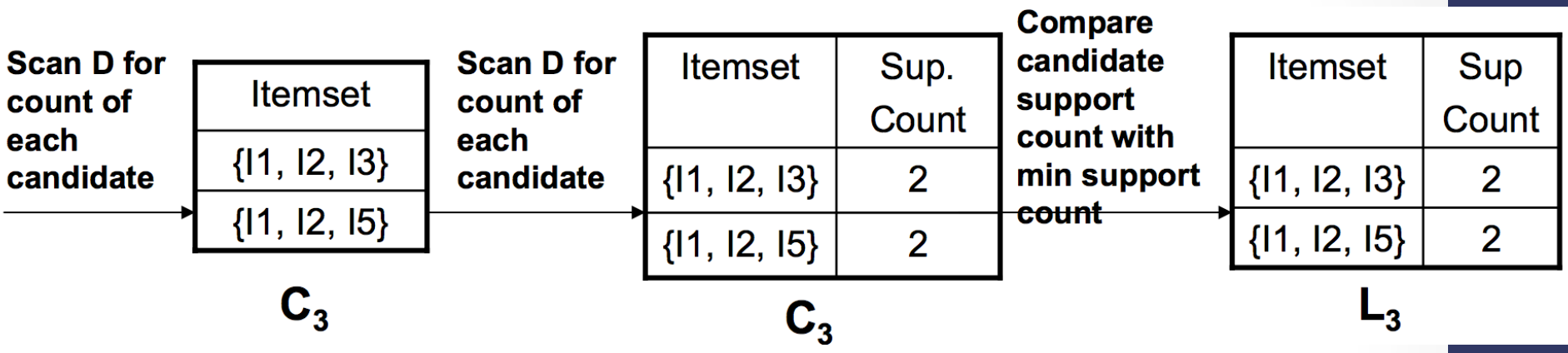
Step 2: Generating 2-itemset Frequent Pattern



Step 2: Generating 2-itemset Frequent Pattern

- To discover the set of frequent 2-itemsets, L2, the algorithm uses L1 join L1 to generate a candidate set of 2-itemsets, C2.
- Next, the transactions in D are scanned and the support count for each candidate itemset in C2 is accumulated (as shown in the middle table).
- The set of frequent 2-itemsets, L2, is then determined consisting of those candidate 2-itemsets in C2 having minimum support.
  - Note: we haven't used Apriori Property yet.

Step 3: Generating 3-itemset Frequent Pattern



- The generation of the set of candidate 3-itemsets, C3, involves use of the Apriori Property.
- In order to find C3, we compute L2 join L2.
- C3= L2 join L2 = {{I1, I2, I3}, {I1, I2, I5}, {I1, I3, I5}, {I2,I3, I4}, {I2, I3, I5}, {I2, I4, I5}}
- Now, Join step is complete, and the Prune step will be used to reduce the size of C3. Prune step helps avoid heavy computation due to large Ck.

Step 3: Generating 3-itemset Frequent Pattern

- Based on the Apriori property that all subsets of a frequent itemset must also be frequent, we can determine that four letter candidates cannot possibly be frequent. How?

- For example, lets take {I1, I2, I3}. The 2-item subsets of it are {I1, I2}, {I1, I3}, & {I2, I3}. Since all 2-item subsets of {I1, I2, I3}are members of L2. We will keep {I1, I2, I3} in C3.

- Lets take another example {I2, I3, I5} which shows how the pruning is performed. The 2-itemsets are {I2, I3}, {I2, I5}, & {I3, I5}.
  - But {I3, I5} is not a member of L2, and hence it is not frequent violating Apriori property. Thus, we will remove {I2, I3, I5} from C3.
  - There for, C3={{I1, I2, I3}, {I1, I2, I5}} after checking for all members of result of Join operation for Prune.

- Now, the transactions in D are scanned in order to determine L3, consisting of those candidates 3-itemset C3 having mininum support.

Step 4: Generating 4-itemset Frequent Pattern

- The algorithm uses L3 join L3 to generate a candidate of 4-itemsets, C4. Although the join results in {{I1, I2, I3, I5}}, this itemset is pruned since its subset {{I2, I3, I5}} is not frequent.
- Thus, C4 is empty, and algorithm terminates, having found all of the frequent items. This completes our Apriori algorithm.

- What's Next?
  - These frequent itemsets will be used to generate strong association rules (where strong association rules satisfy both minimum support and minimum confidence).

Step 5: Generating Association rules from Frequent Itemsets.

- Procedure:
  - For each frequent itemset, *l,* generate all nonempty subsets of *l.*
  - For every nonempty subset *s of l,* output the rule "s → (*l-s)"* if support_count(l)/support_count(s) >= min_conf.

- Back to Example:
  - We have L={{I1}, {I2}, {I3}, {I4}, {I5}, {I1, I2}, {I1, I3}, {I1, I5}, {I2, I3}, {I2, I4}, {I2, I5}, {I1, I2, I3}, {I1, I2, I5}}
    - Lets take I={I1, I2, I5}
    - Its all nonempty subsets are {I1, I2}, {I1, I5}, {I2, I5}, {I1}, {I2}, {I5}.

Step 5: Generating Association rules from Frequent Itemsets.

- Let mininum confidence thereshold is 70%.
- The resulting association rules are shown below, each list with its confidence.
  - R1: I1 ^ I2 → I5
    - Confidence support_count{I1, I2, I5}/support_count(I1, I2} = 2/5=50%
    - R1 is rejected
  - R2: I1 ^ I5 → I2
    - Confidence = 100%
    - R2 is selected.
  - R3: I2 ^ I5 → I1
    - Confidence = 100%
    - S3 is selected.

Step 5: Generating Association rules from Frequent Itemsets.

- R4: I1 → I2 ^ I5
  - Confidence: 33%
  - R4 is rejected
- R5: I2 → I1 ^ I5
  - Confidence: 29%
  - R3 is rejected.
- R6: I5 → I1 ^ I2
  - Confidence: 100%
  - R6 is selected
  - In this way, we have found three strong association rules.

Methods to improve Apriori's Efficiency.

- Hash-based itemset counting: A k-itemset whose corresponding hashing bucket count is below the threshold cannot be frequent.
- Transaction reduction: A transaction that doesn't contain any frequent k-itemset is useless in subsequent scans.
- Partitioning: Any itemset that is potentially frequent in DB must be frequent in at least one of the partitions of DB.
- Sampling: mining on a subset of given data, lower support threshold + a method to determine the completeness.
- Dynamic itemset counting: add new candidate itemsets only when all of their subsets are eliminated to be frequent.

FP-Growth Algorithm

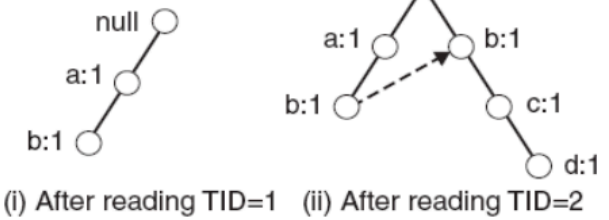
- FP-Growth: allows frequent itemset discovery without candidate itemset generation. Two-step approach:
  - Step 1: Build a compact data structure called the FP-tree
    - Built using 2-passes over the data set.
  - Step 2: Extracts frequent itemsets directly from the FP-tree

Core Data Structure: FP-tree

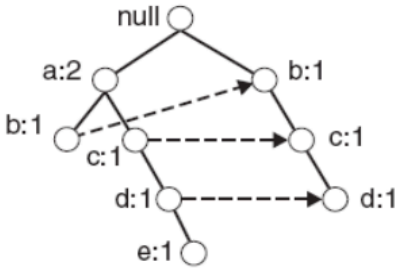
- Nodes correspond to items and have a counter
- FP-Growth reads 1 transaction at a time and maps it to a path
- Fixed order is used, so paths can overlap when transactions share items (when they have the same prefix).
- In this case, counters are incremented.
- Pointers are maintained between nodes containing the same item, creating singly lists (dotted lines).
- The more paths that overlap, the higher the compression. FP-tree may fit in memory.
- Frequent itemsets extracted from the FP-tree

Transaction Data Set

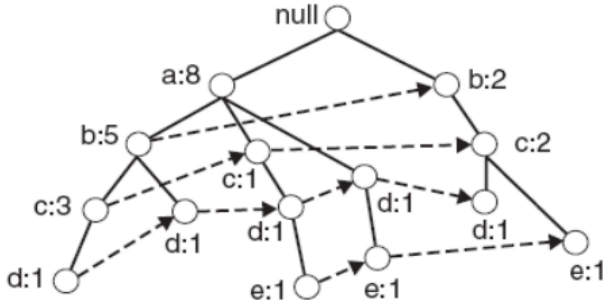
TID	Items
1	{a,b}
2	{b,c,d}
3	{a,c,d,e}
4	{a,d,e}
5	{a,b,c}
6	{a,b,c,d}
7	{a}
8	{a,b,c}
9	{a,b,d}
10	{b,c,e}



(i) After reading TID=1 (ii) After reading TID=2



(iii) After reading TID=3



(iv) After reading TID=10

Step 1: FP-tree Construction (Example)

- FP-tree is constructed using 2-passes over the dataset.
  - Pass 1:
    - Scan data and find support for each item.
    - Discard infrequent items
    - Sort frequent items decreasing order based on their support.
      - For our example: a, b, c, d, e
      - Use this order when building the FP-tree, so common prefixes can be shared.

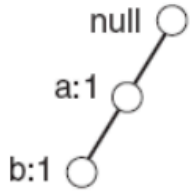
Step 1: FP-tree Construction (Example)

- Pass 2: construct the FP-tree (see diagram on next slide)
  - Read transaction 1: {a,b}
    - Create 2 nodes a, b and the path null → a → b. Set counts of a and b to 1.
  - Read transaction 2: {b, c, d}
    - Create 3 nodes for b, c and d and the path null→ b→ c→ d. Set counts to 1.
    - Note that although transaction 1 and 2 share b, the paths are disjoint as they don't share a common prefix. Add the link between the b's.
  - Read transaction 3: {a, c, d, e}
    - It shares common prefix item **a** with transaction 1 so the path for transaction 1 and 3 will overlap and the frequency count for node a will be incremented by 1. Add links between the c's and d's.
  - Continue until all transactions are mapped to a path in the FP-tree.

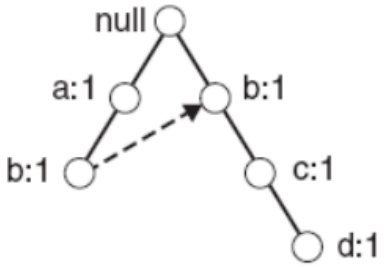
Step 1: FP-tree Construction (Example)

Transaction Data Set

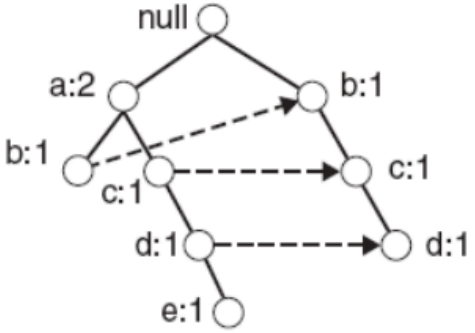
TID	Items
1	{a,b}
2	{b,c,d}
3	{a,c,d,e}
4	{a,d,e}
5	{a,b,c}
6	{a,b,c,d}
7	{a}
8	{a,b,c}
9	{a,b,d}
10	{b,c,e}



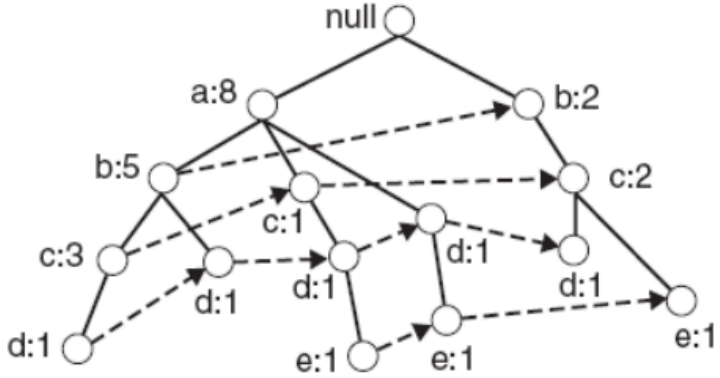
(i) After reading TID=1



(ii) After reading TID=2



(iii) After reading TID=3



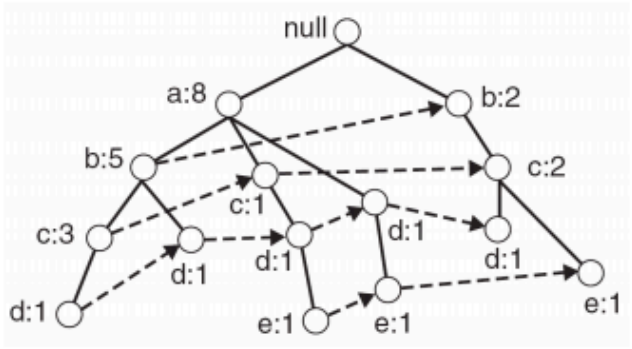
(iv) After reading TID=10

FP-Tree size

- The FP-Tree usually has a smaller size than the uncompressed data – typically many transactions share items (and hence prefixes).
  - Best case scenario: all transactions contain the same set of items
    - 1 path in the FP-tree
  - Worst case scenario: every transaction has a unique set of items (no items in common)
    - Size of the FP-tree is at least as large as the original data.
    - Storage requirement for the FP-tree are higher –need to store the pointers between the nodes and the counters.
  - The size of the FP-tree depends on how the items are ordered
    - Ordering by decreasing support is typically used but it doesn't always lead to the smallest tree (it's a heuristic).

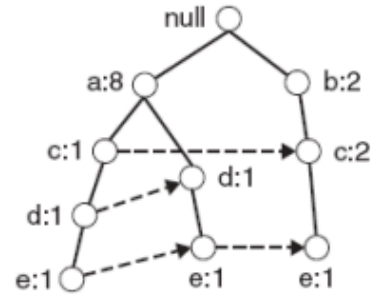
Step 2: Frequent Itemset Generation

- FP-Growth extracts frequent itemsets from the FP-tree
- Bottom up algorithm – from the leaves towards the root.
  - Divide and conquer: first look for frequent itemsets ending in **e,** then **de,** etc. then **d,** then **cd,** etc.
  - First, extract prefix path sub-trees ending in an item (set)
    - Hint: use the linked lists.

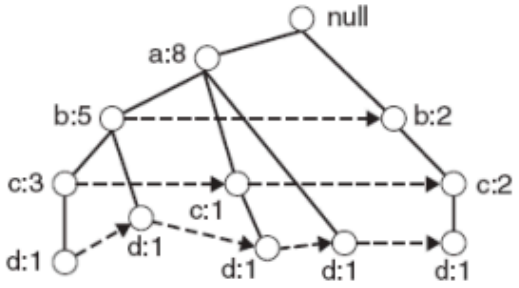


↑ Complete FP-tree

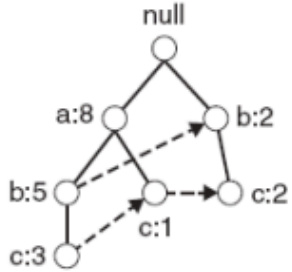
→ **Example:** prefix path sub-trees



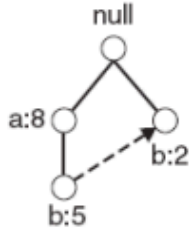
(a) Paths containing node e



(b) Paths containing node d



(c) Paths containing node c

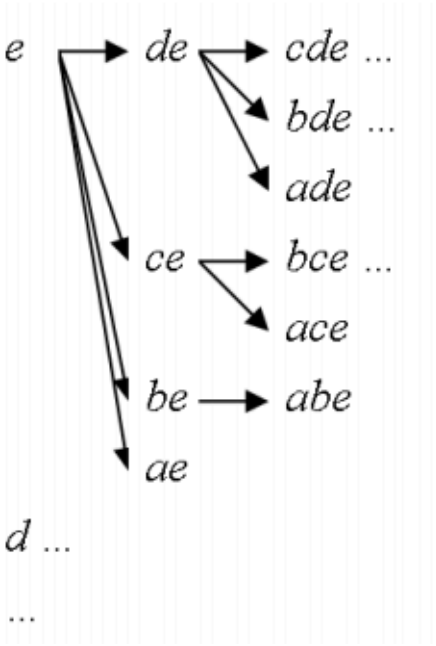


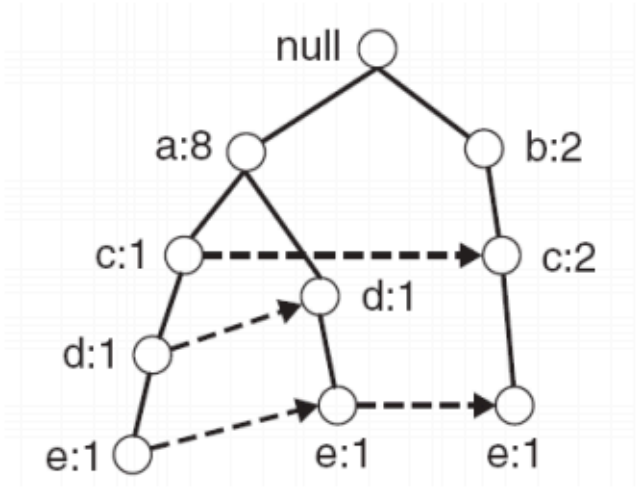
(d) Paths containing node b

(e) Paths containing node a

Step 2: Frequent Itemset Generation

- Each prefix path sub-tree is processed recursively to extract the frequent itemsets. Solutions are then merged.
  - Example: the *prefix path sub-tree* for ***e*** will be used to extract frequent itemsets ending in ***e,*** then in **de**, **ce**, **be**, and **ae**, then in **cde**, **bde**, **cde**, etc.
  - Divide and conquer approach

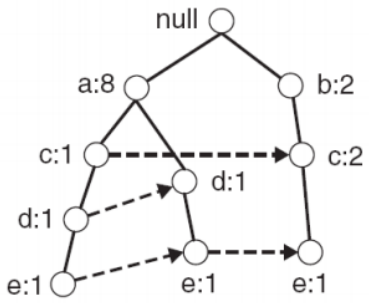




Prefix path sub-tree ending in *e*.

Example

- Let min_supp=2 and extract all frequent itemsets containing e.
- 1. Obtain the prefix path sub-tree for **e**

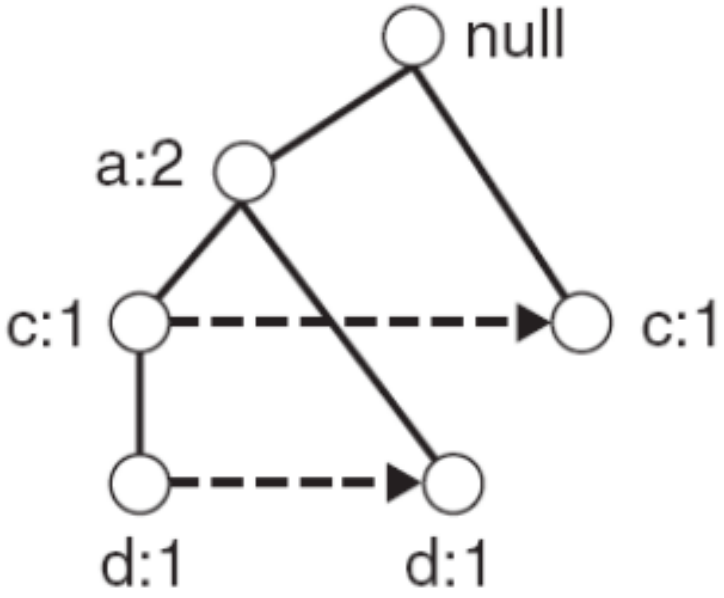


- 2. Check if e is a frequent item by adding the counts along the linked list (dotted line). If so, extract it.
  - {e} is frequent since count=3
- 3. As **e** is frequent, find frequent itemsets ending in **e**, i.e. **de, ce, be,** and **ae**
  - i.e. decompose the problem recursively
  - To do this, we must first to obtain the conditional FP-tree for e.

Conditional FP-tree

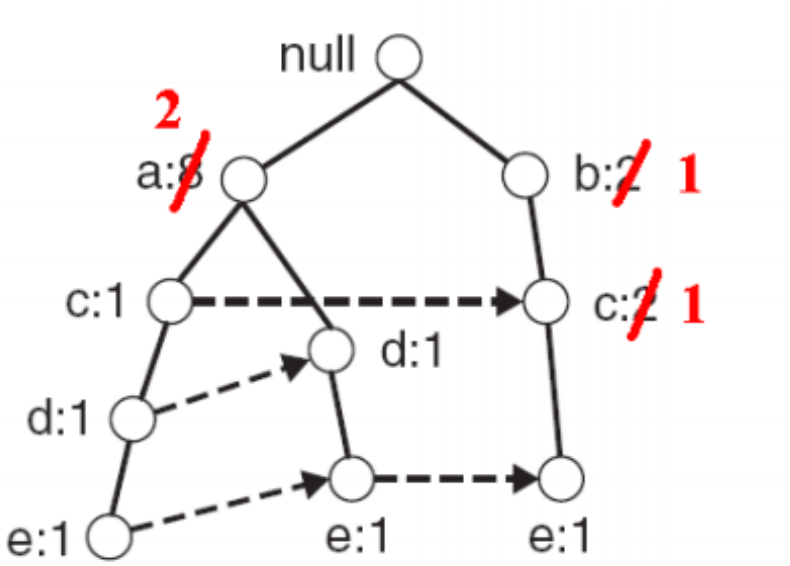
- The FP-tree that would be built if we only consider transactions containing a particular itemset (and then removing that itemset from all transactions).
- **Example:**

TID	Items
~~1~~	~~{a,b}~~
~~2~~	~~{b,c,d}~~
3	{a,c,d,~~e~~}
4	{a,d,~~e~~}
~~5~~	~~{a,b,c}~~
~~6~~	~~{a,b,c,d}~~
~~7~~	~~{a}~~
~~8~~	~~{a,b,c}~~
~~9~~	~~{a,b,d}~~
10	{b,c,~~e~~}



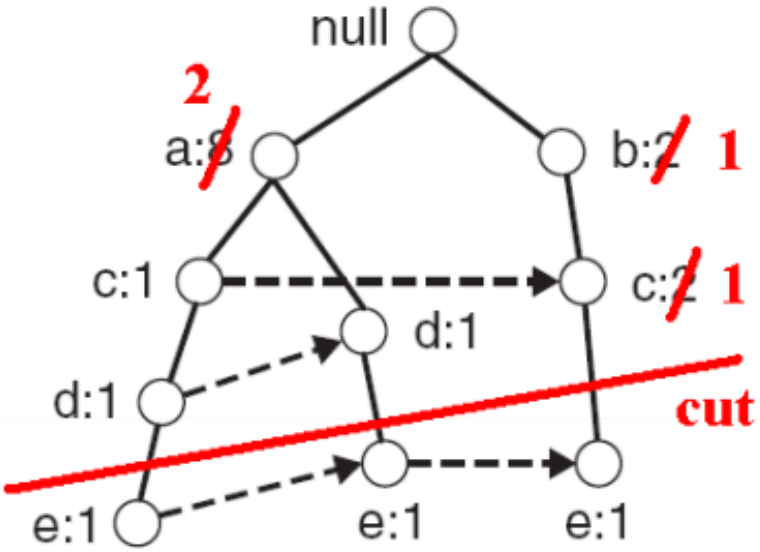
Conditional FP-tree

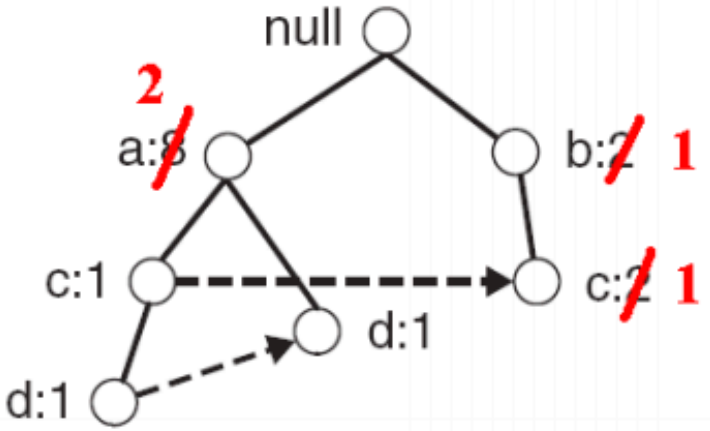
- To obtain the conditional FP-tree for **e** from the prefix sub-tree ending in **e:**
  - Update the support counts along the prefix paths (from **e**) to reflect the number of transactions containing e.
    - b and c should be set to 1 and a to 2



Conditional FP-tree

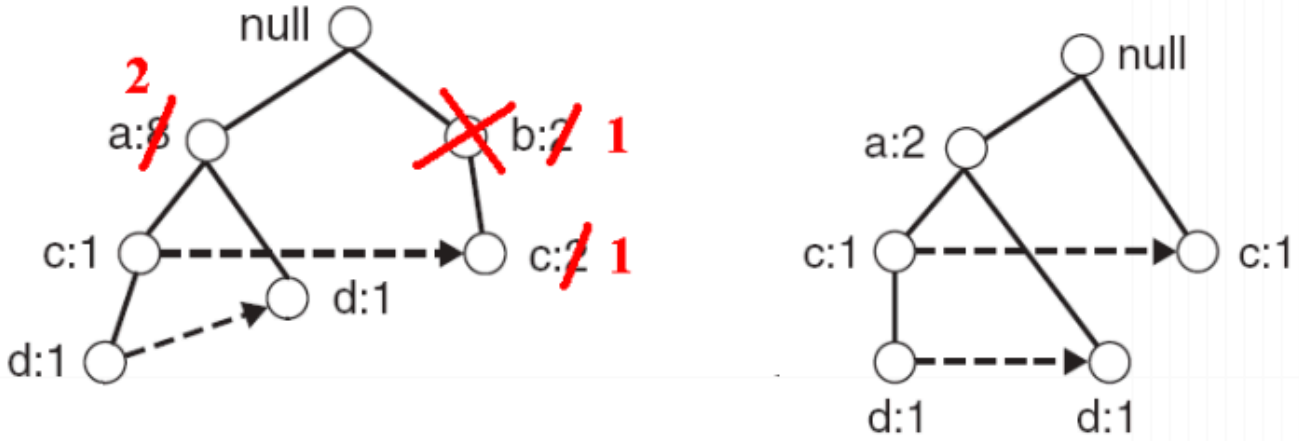
- To obtain the conditional FP-tree for e from the prefix sub-tree ending in e.
  - Remove the nodes containing **e** – information about node e is no longer needed because of the previous step





Conditional FP-tree

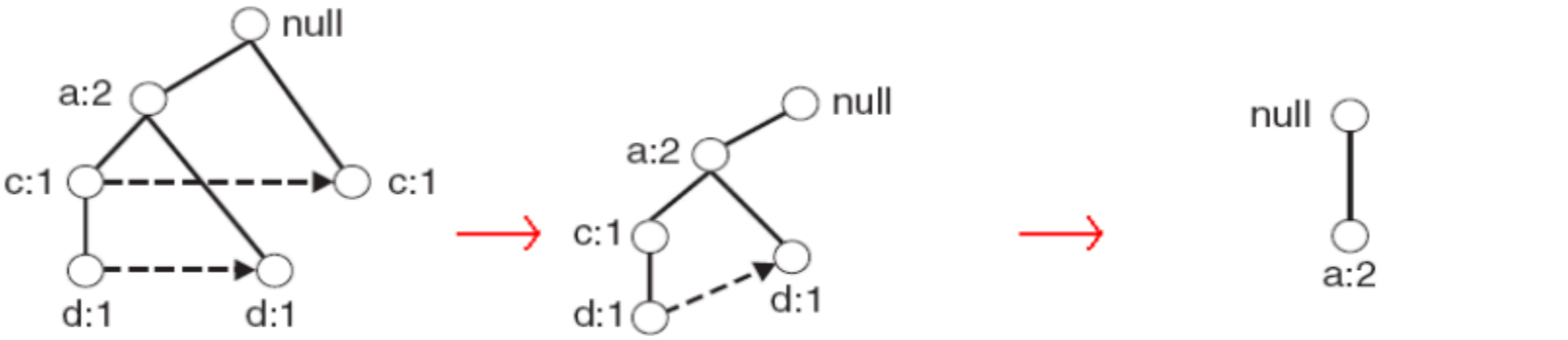
- To obtain the conditional FP-tree for e from the prefix sub-tree ending in e.
  - Remove infrequent items (nodes) from the prefix paths
  - E.g. **b** has a support of 1 (note this really mean **be** has a support of 1). i.e. there is only 1 transaction containing b and e so **be** is infrequent – thus, we can remove b.



Question: why were *c* and *d* not removed?

Example (continued)

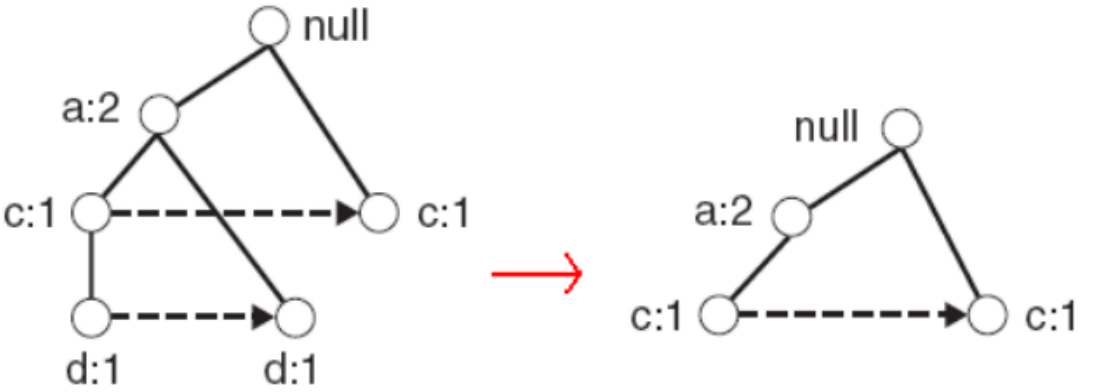
- 4. Use the conditional FP-tree for e to find frequent itemsets ending in **de, ce,** and **ae.**
  - Note that **be** is not considered as b is not in the conditional FP-tree for e.
  - For each of them (e.g. **de**), find the prefix paths from the conditional tree for **e**, extract frequent itemsets, generate FP-tree, etc. … (recursive)
  - **Example:** e → de → ade ({d,e}, {a, d, e} are found to be frequent)



Conditional FP-tree for e Prefix paths ending in de Conditional FP-tree for de

Example (cont.)

- 4. Use the conditional FP-tree for e to find frequent itemsets ending in **de, ce,** and **ae.**
  - **Example**: e → ce ({c,e} is found to be frequent)



Conditional FP-tree for e

Prefix paths ending in ce

  - **Etc.** (**ae**, then do the whole thing for **b,** … etc)

Result

- Frequent itemsets found (ordered by suffix and order in which they are found):

Suffix	Frequent Itemsets
e	{e}, {d,e}, {a,d,e}, {c,e},{a,e}
d	{d}, {c,d}, {b,c,d}, {a,c,d}, {b,d}, {a,b,d}, {a,d}
c	{c}, {b,c}, {a,b,c}, {a,c}
b	{b}, {a,b}
a	{a}

Discussion

- Advantages of FP-Growth
  - Only 2-passes over dataset
  - “compresses” dataset
  - No candidate generation
  - Much faster than Apriori

- Disadvantages of FP-growth
  - FP-tree may not fit in memory
  - FP-tree is expensive to build
    - Trade-off: takes time to build, but once it is built, frequent itemsets are read off easily
    - Time is wasted (especially if min_supp is high), as the only pruning that can be done is on single items.
    - Support can only be calculated once the entire dataset is added to the FP-tree.

Outline

- Basic Concepts
- Frequent Itemset Mining Methods
  - Apriori Algorithm
  - Pattern Growth Approach
- Pattern Evaluation Methods

Drawback of Support-Confidence Evaluation Framework

- Strong rules are not necessarily interesting.
- Example:
  - Analyzing transactions from AllElectronics, we find of **10,000** transactions, **6000** included computer games, **7500** included videos, and **4000** included both.

$$buys(X, \text{“}computer\ games\text{”}) \Rightarrow buys(X, \text{“}videos\text{”})$$

$$[support = 40\%, confidence = 66\%].$$

  - The above association rule is strong, however, the probability of purchasing videos is 75%, which is even larger than 60%.
  - In fact, computer games and videos are negatively associated because the purchase of one of these items decreases the likelihood of purchasing the other.

Correlation measure

- Correlation rules:

$$A \Rightarrow B \, [support, confidence, correlation].$$

- Correlation measures:
  - Lift (=1 if A and B are independent, > 1 if positively correlated and < 1 if negatively correlated)

$$lift(A, B) = \frac{P(A \cup B)}{P(A)P(B)}.$$

  - Chi-square measure

$$\chi^2 = \Sigma \frac{(observed - expected)^2}{expected}$$

Other measurements

- All confidence

$$all\_conf(A,B) = \frac{sup(A \cup B)}{max\{sup(A), sup(B)\}} = min\{P(A|B), P(B|A)\},$$

- Max confidence

$$max\_conf(A, B) = max\{P(A \mid B), P(B \mid A)\}.$$

- Kulczynski measure

$$Kulc(A, B) = \frac{1}{2}(P(A|B) + P(B|A)).$$

- Cosine measure

$$cosine(A, B) = \frac{P(A \cup B)}{\sqrt{P(A) \times P(B)}} = \frac{sup(A \cup B)}{\sqrt{sup(A) \times sup(B)}}$$
$$= \sqrt{P(A|B) \times P(B|A)}.$$

Comparison of measurements

- Support & confidence measures to mine association rules may generate a large number of rules
- We can augment the support-confidence framework with a pattern interestingness measure.
- Null-invariance property
  - Null transaction is a transaction that does not contain any of the itemsets being examined.
  - A measure is null-invariant if its value is free from the influence of null-transactions.
  - Lift and Chi-square are not null-invariant
- Among all_confidence, max_confidence, Kulczynski and cosine measures, Kulc is recommended in conjunction with imbalance ratio.

Summary

- Basic Concepts
- Frequent Itemset Mining Methods
  - Apriori Algorithm
  - Pattern Growth Approach
- Pattern Evaluation Methods