CS 4440 A

Emerging Database Technologies

Lecture 13

02/21/24

By Hantian Zhang

If we just have a bunch of data sets in a repository, it is unlikely anyone will ever be able to find, let alone reuse, any of this data. With adequate metadata, there is some hope, but even so, challenges will remain..

[D. Agrawal, P. Bernstein, E. Bertino, S. Davidson, U. Dayal, M. Franklin, J. Gehrke, L. Haas, A. Halevy, J. Han, H. V. Jagadish, A. Labrinidis, S. Madden, Y. Papakonstantinou, J. M. Patel, R. Ramakrishnan, K. Ross, C. Shahabi, D. Suciu, S. Vaithyanathan, and J. Widom. Challenges and opportunities with Big Data. Technical report, Computing Community Consortium, http://cra.org/ccc/docs/init/bigdatawhitepaper.pdf, 2012.]

It is important to understand your data aka data mining

- Stats of the data
- Association Rule Mining
- Classification
- Regression
- Clustering
- Anomaly Detection
- etc

Two Main Categories of Algorithms

- Schema-Driven
  - Has candidate generation
  - Has pruning
  - Can quickly check if a candidate is interesting or not
  - Usually sensitive to the size of the schema

- Data-Driven
  - No candidate generation
  - Have a novel data structure to summarize the data
  - Usually sensitive to the size of the instance

Today's class

Association rules mining

- Schema-Driven: Apriori algorithm
- Data-Driven: FP-Growth algorithm

Acknowledgement: Slides adapted from Prof. Xu Chu's CS4400-X

Association Rule Mining

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

Market-Basket transactions

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

Example of Association Rules

{Diaper} $\rightarrow$ {Beer},
{Milk, Bread} $\rightarrow$ {Eggs,Coke},
{Beer, Bread} $\rightarrow$ {Milk},

Implication means co-occurrence, not causality!

Definition: Frequent Itemset

Itemset

- A collection of one or more items
  - Example: {Milk, Bread, Diaper}
- k-itemset
  - An itemset that contains k items

Support count (σ)

- Frequency of occurrence of an itemset
- E.g. σ({Milk, Bread,Diaper}) = 2

Support

- Fraction of transactions that contain an itemset
- E.g. s({Milk, Bread, Diaper}) = 2/5

Frequent Itemset

- An itemset whose support is greater than or equal to a *minsup* threshold

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

Definition: Association Rule

- Association Rule
  - An implication expression of the form X → Y, where X and Y are itemsets
  - Example:
    {Milk, Diaper} → {Beer}

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**

- Rule Evaluation Metrics
  - Support (s)
    - Fraction of transactions that contain both X and Y
  - Confidence (c)
    - Measures how often items in Y appear in transactions that contain X

Example:

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

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

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

Association Rule Mining Task

- Given a set of transactions T, the goal of association rule mining is to find all rules having
  - support ≥ *minsup* threshold
  - confidence ≥ *minconf* threshold

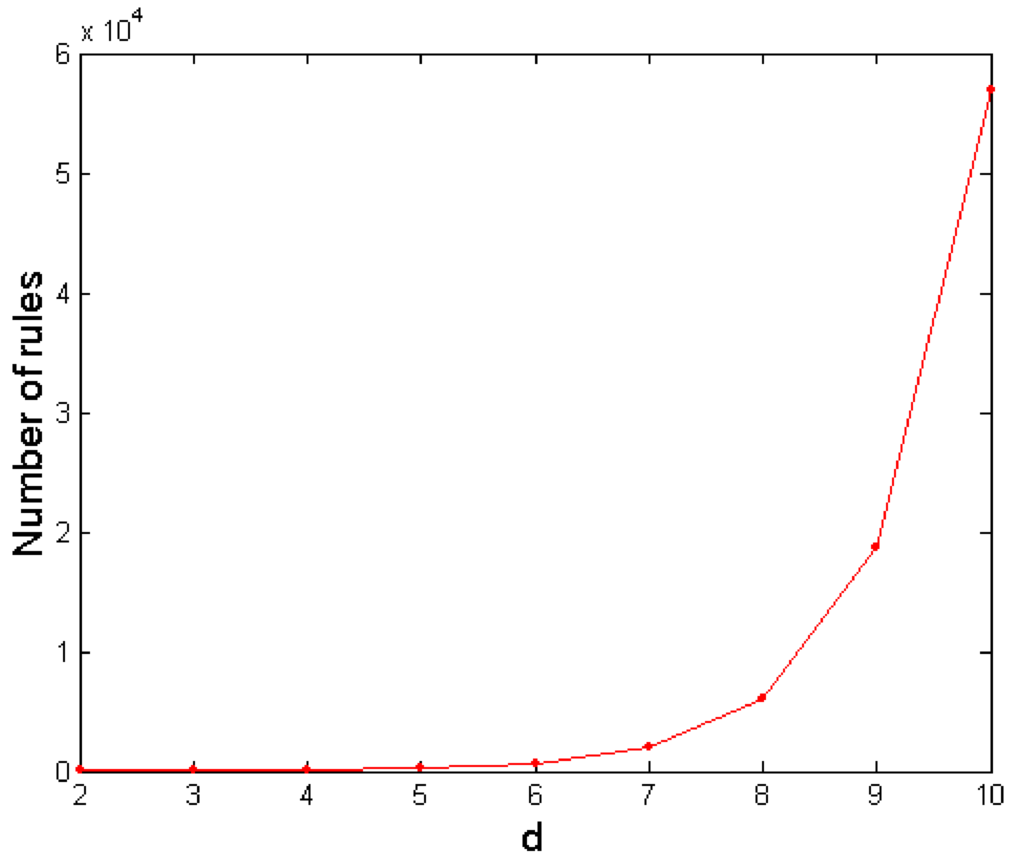
- Brute-force approach:
  - List all possible association rules
  - Compute the support and confidence for each rule
  - Prune rules that fail the *minsup* and *minconf* thresholds

  ⇒ Computationally prohibitive!

Computational Complexity

Given d unique items:

- Total number of itemsets = 2^d
- Total number of possible association rules:



$$R = \sum_{k=1}^{d-1}\left[\binom{d}{k} \times \sum_{j=1}^{d-k}\binom{d-k}{j}\right] = 3^d - 2^{d+1} + 1$$

If d=6, R = 602 rules

Mining Association Rules

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

Example of Rules:

{Milk,Diaper} → {Beer} (s=0.4, c=0.67)
{Milk,Beer} → {Diaper} (s=0.4, c=1.0)
{Diaper,Beer} → {Milk} (s=0.4, c=0.67)
{Beer} → {Milk,Diaper} (s=0.4, c=0.67)
{Diaper} → {Milk,Beer} (s=0.4, c=0.5)
{Milk} → {Diaper,Beer} (s=0.4, c=0.5)

Observations:

- All the above rules are binary partitions of the same itemset:
  {Milk, Diaper, Beer}
- Rules originating from the same itemset have identical support but can have different confidence
- Thus, we may decouple the support and confidence requirements

Mining Association Rules

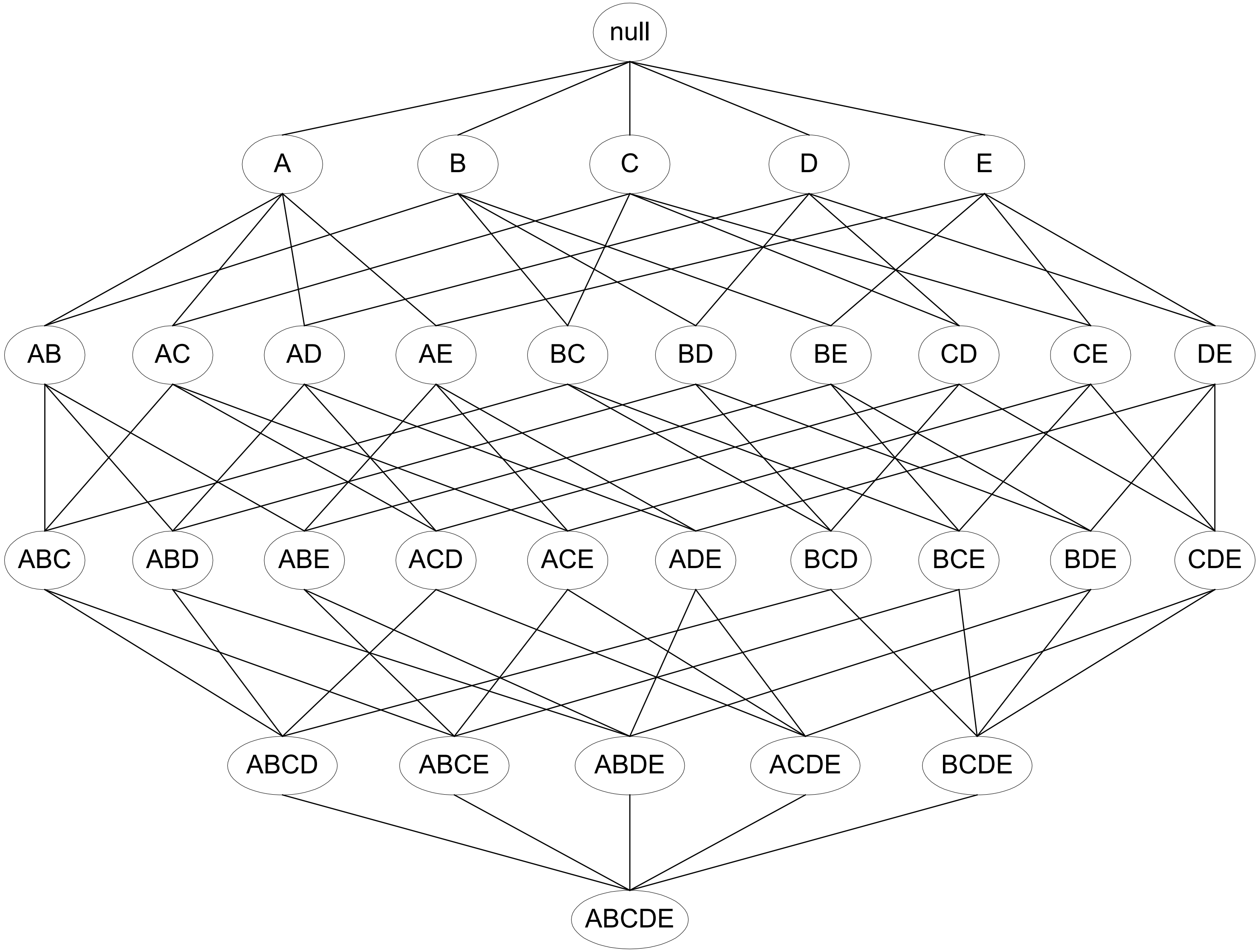
Two-step approach:

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 binary partitioning of a frequent itemset

Frequent itemset generation is still computationally expensive

Frequent Itemset Generation



Given d items, there are 2^d possible candidate itemsets

Frequent Itemset Generation

Brute-force approach:

- Each itemset in the lattice is a candidate frequent itemset
- Count the support of each candidate by scanning the database

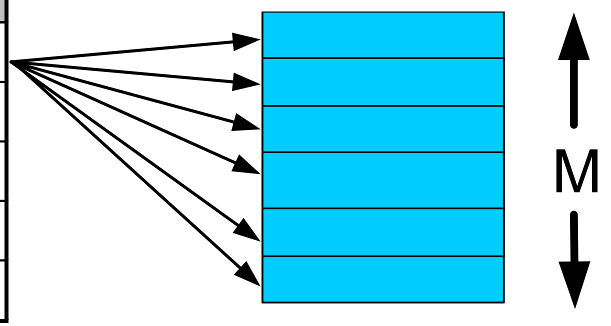
Transactions

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

N

w

List of Candidates



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

Frequent Itemset Generation Strategies

Reduce the number of candidates (M)

- Complete search: $M=2^d$
- Use pruning techniques to reduce M

Reduce the number of comparisons (NM)

- Use efficient data structures to store the candidates or transactions
- No need to match every candidate against every transaction

Reducing Number of Candidates

Apriori principle:

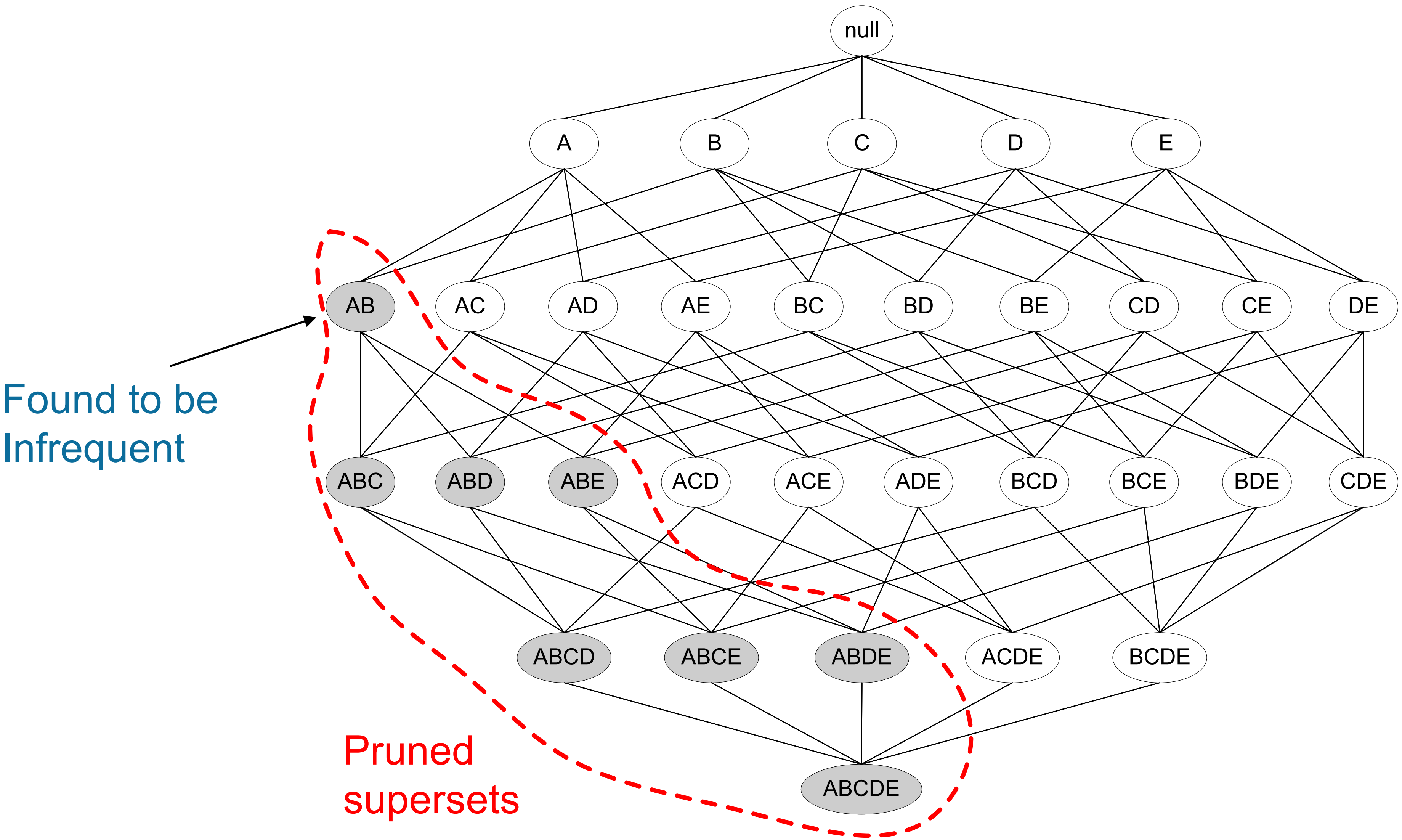
- If an itemset is frequent, then all of its subsets must also be frequent

Apriori principle holds due to the following property of the support measure:

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

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

Illustrating Apriori Principle



Illustrating Apriori Principle

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 (1-itemsets)

Item	Count
Bread	**4**
Coke	**2**
Milk	**4**
Beer	**3**
Diaper	**4**
Eggs	**1**

Minimum Support = 3

If every subset is considered,

$^6C_1 + {}^6C_2 + {}^6C_3$

$6 + 15 + 20 = 41$

With support-based pruning,

$6 + 6 + 4 = 16$

Illustrating Apriori Principle

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 (1-itemsets)

Item	Count
Bread	**4**
Coke	**2**
Milk	**4**
Beer	**3**
Diaper	**4**
Eggs	**1**

Minimum Support = 3

If every subset is considered,

$^6C_1 + {}^6C_2 + {}^6C_3$

$6 + 15 + 20 = 41$

With support-based pruning,

$6 + 6 + 4 = 16$

Illustrating Apriori Principle

Item	Count
Bread	4
Coke	2
Milk	4
Beer	3
Diaper	4
Eggs	1

Items (1-itemsets)

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

Pairs (2-itemsets)

(No need to generate candidates involving Coke or Eggs)

Minimum Support = 3

If every subset is considered,

$^6C_1 + {}^6C_2 + {}^6C_3$

$6 + 15 + 20 = 41$

With support-based pruning,

$6 + 6 + 4 = 16$

Illustrating Apriori Principle

Item	Count
Bread	**4**
Coke	**2**
Milk	**4**
Beer	**3**
Diaper	**4**
Eggs	**1**

Items (1-itemsets)

Itemset	Count
{Bread,Milk}	**3**
{Beer, Bread}	**2**
{Bread,Diaper}	**3**
{Beer,Milk}	**2**
{Diaper,Milk}	**3**
{Beer,Diaper}	**3**

Pairs (2-itemsets)

(No need to generate candidates involving Coke or Eggs)

Minimum Support = 3

If every subset is considered,

$$^6C_1 + {}^6C_2 + {}^6C_3$$

$$6 + 15 + 20 = 41$$

With support-based pruning,

$$6 + 6 + 4 = 16$$

Illustrating Apriori Principle

Item	Count
Bread	**4**
Coke	**2**
Milk	**4**
Beer	**3**
Diaper	**4**
Eggs	**1**

Items (1-itemsets)

Itemset	Count
{Bread,Milk}	**3**
{Beer, Bread}	**2**
{Bread,Diaper}	**3**
{Beer,Milk}	**2**
{Diaper,Milk}	**3**
{Beer,Diaper}	**3**

Pairs (2-itemsets)

(No need to generate candidates involving Coke or Eggs)

Triplets (3-itemsets)

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

Minimum Support = 3

If every subset is considered,

$^6C_1 + {}^6C_2 + {}^6C_3$

$6 + 15 + 20 = 41$

With support-based pruning,

$6 + 6 + 4 = 16$

Illustrating Apriori Principle

Item	Count
Bread	**4**
Coke	**2**
Milk	**4**
Beer	**3**
Diaper	**4**
Eggs	**1**

Items (1-itemsets)

Itemset	Count
{Bread,Milk}	**3**
{Beer, Bread}	**2**
{Bread,Diaper}	**3**
{Beer,Milk}	**2**
{Diaper,Milk}	**3**
{Beer,Diaper}	**3**

Pairs (2-itemsets)

(No need to generate candidates involving Coke or Eggs)

Minimum Support = 3

If every subset is considered,

$^6C_1 + {}^6C_2 + {}^6C_3$

$6 + 15 + 20 = 41$

With support-based pruning,

$6 + 6 + 4 = 16$

Triplets (3-itemsets)

Itemset	Count
{ Beer, Diaper, Milk}	**2**
{ Beer,Bread, Diaper}	**2**
{Bread, Diaper, Milk}	**2**
{Beer, Bread, Milk}	**1**

Apriori Algorithm

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

Algorithm

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

Candidate Generation: F_{k-1} x F_{k-1} Method

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

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

  - Do not merge(<u>A</u>BD,<u>A</u>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

- L_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: L_4 = {ABCD}

Illustrating Apriori Principle

Item	Count
Bread	**4**
Coke	**2**
Milk	**4**
Beer	**3**
Diaper	**4**
Eggs	**1**

Items (1-itemsets)

Itemset	Count
{Bread,Milk}	**3**
{Beer, Bread}	**2**
{Bread,Diaper}	**3**
{Beer,Milk}	**2**
{Diaper,Milk}	**3**
{Beer,Diaper}	**3**

Pairs (2-itemsets)

(No need to generate candidates involving Coke or Eggs)

Minimum Support = 3

Triplets (3-itemsets)

Itemset	Count
{Bread, Diaper, Milk}	**2**

If every subset is considered,

$^6C_1 + {}^6C_2 + {}^6C_3$

$6 + 15 + 20 = 41$

With support-based pruning,

$6 + 4 + 1 = 11$

Use of $F_{k-1}xF_{k-1}$ method for candidate generation results in only one 3-itemset. This is eliminated after the support counting step.

Support Counting of Candidate Itemsets

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

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

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

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

Rule Generation

- Given a frequent itemset L, find all non-empty subsets $f \subset L$ such that $f \rightarrow L - f$ satisfies the minimum confidence requirement
  - If {A,B,C,D} is a frequent itemset, candidate rules:

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

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

Rule Generation

- In general, confidence does not have an anti-monotone property

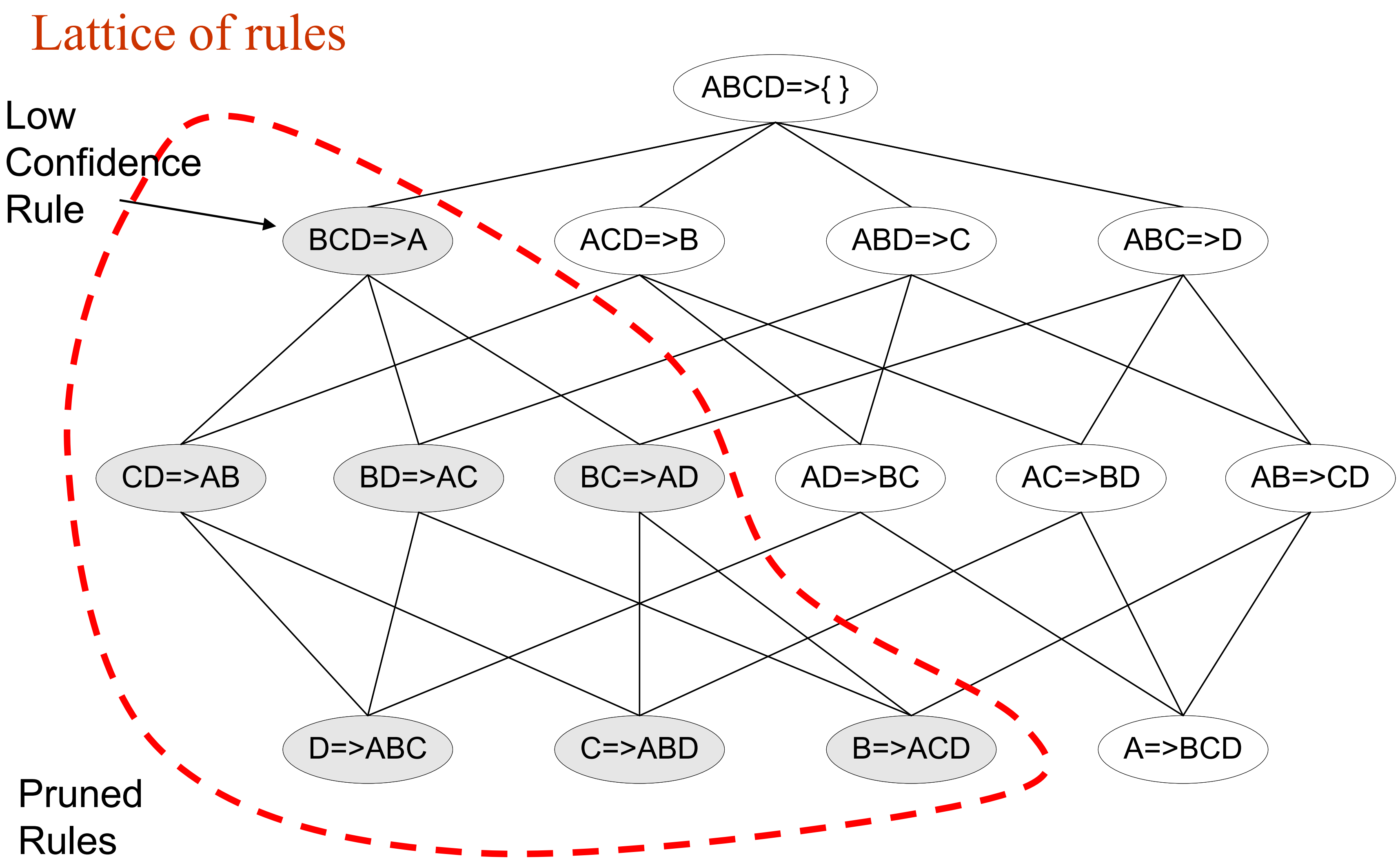
  $c(ABC \rightarrow D)$ can be larger or smaller than $c(AB \rightarrow D)$

- But confidence of rules generated from the same itemset has an anti-monotone property
  - E.g., 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

Rule Generation for Apriori Algorithm

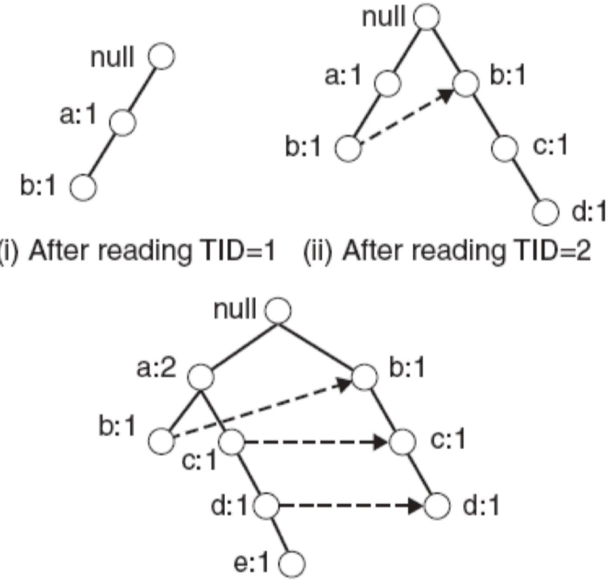


The Data Driven Approach: The FP-TREE and THE FP-Growth Algorithm

Core Data Structure: FP-Tree

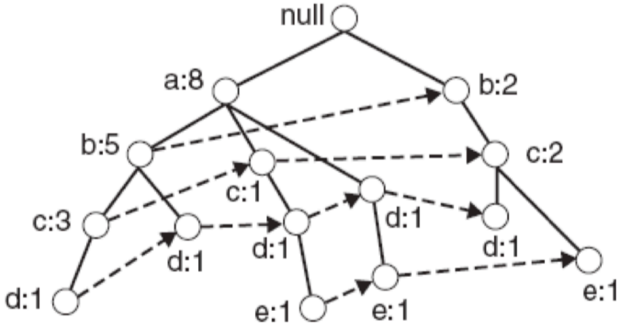
Transaction Data Set

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

- 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 linked 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.

Step 1: FP-Tree Construction (Example)

FP-Tree is constructed using 2 passes over the data-set:

- **Pass 1**:
  - Scan data and find support for each item.
  - Discard infrequent items.
  - Sort frequent items in 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.

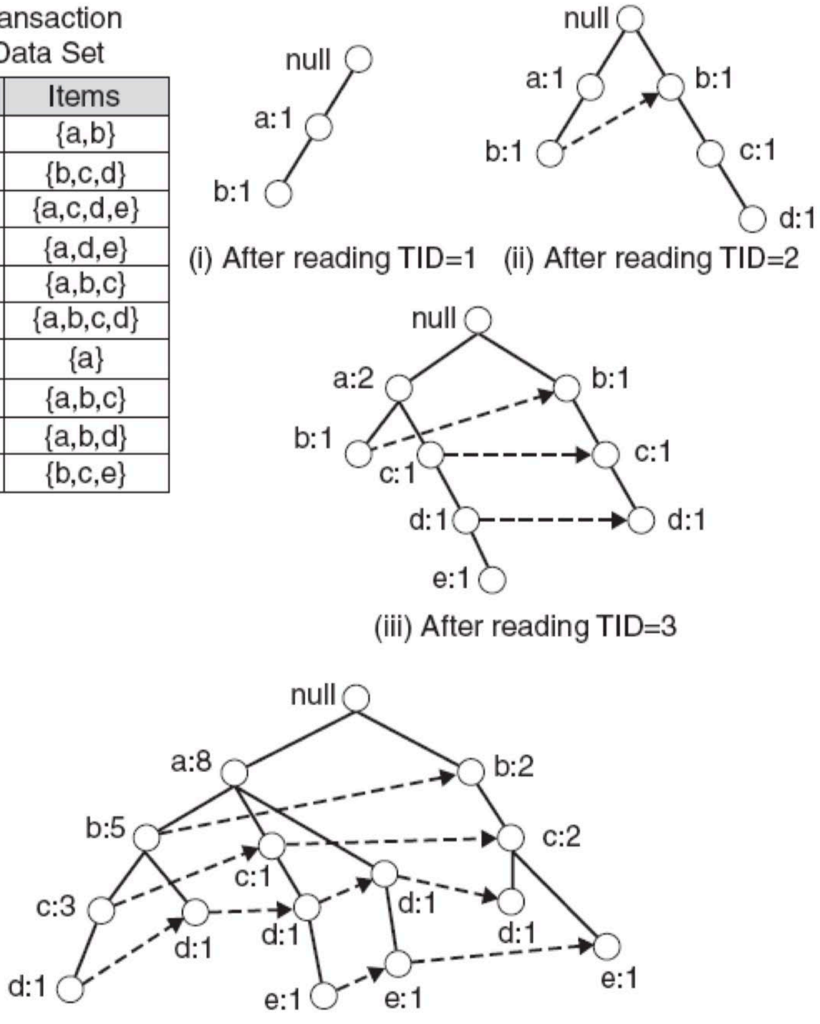
Step 1: FP-Tree Construction (Example)

- **Pass 2**: construct the FP-Tree (see diagram on next slide)
  - Read transaction 1: $\{a, b\}$
    - Create 2 nodes a and b and the path $null \rightarrow a \rightarrow b$. 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 $null \rightarrow b \rightarrow c \rightarrow d$. 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

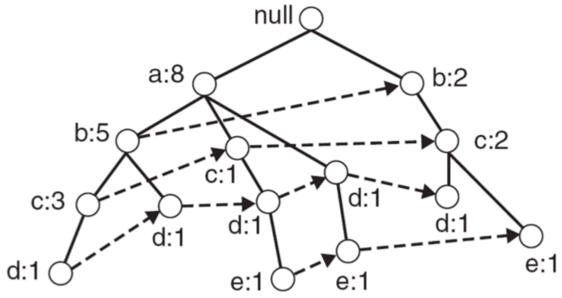
(iv) After reading TID=10

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 requirements 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 does not always lead to the smallest tree (it's a heuristic).

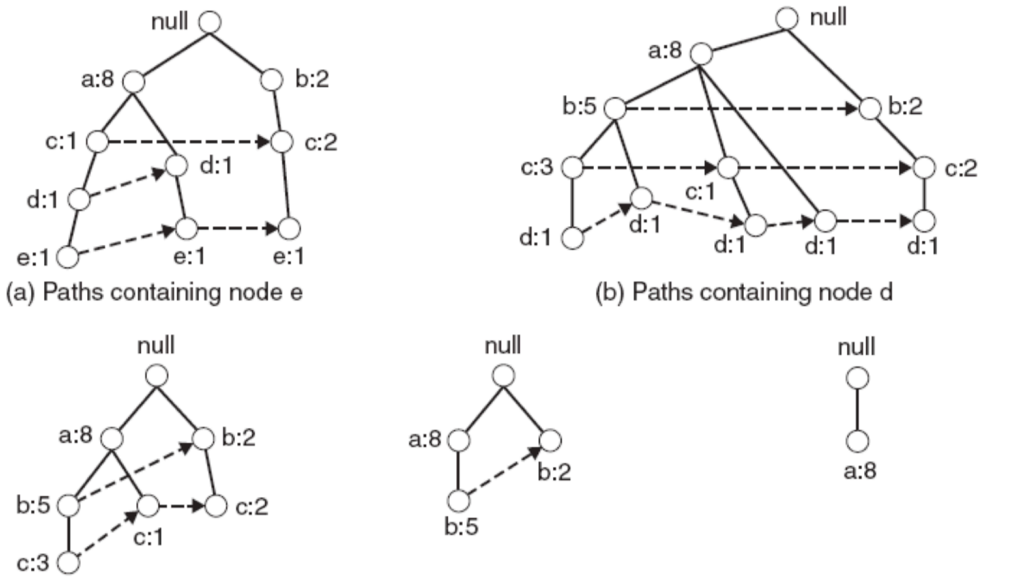
Step 2: Frequent Itemset Generation

- 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

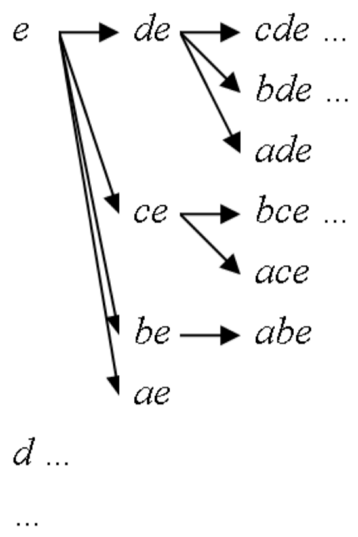
(c) Paths containing node c

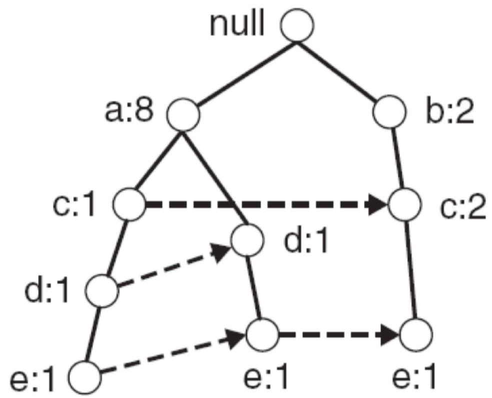
(d) Paths containing node b

(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.
  - **E.g.** 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



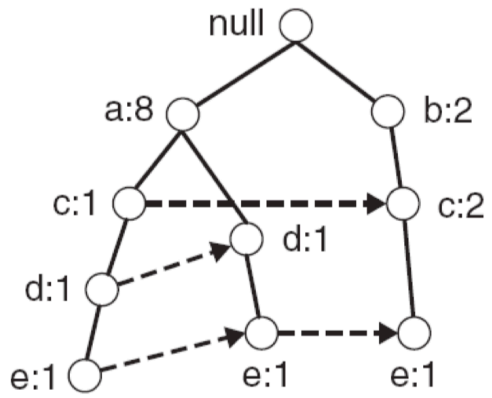


Prefix path sub-tree ending in *e*.

Example

Let $minSup = 2$ and extract all frequent itemsets containing e.

- 1. Obtain the prefix path sub-tree for e:

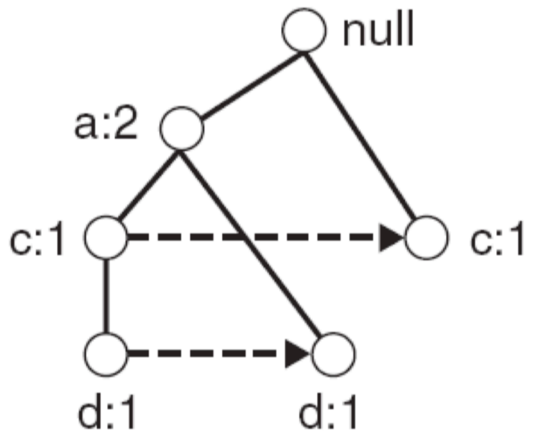


- 2. Check if e is a frequent item by adding the counts along the linked list (dotted line). If so, extract it.
  - Yes, count $=3$ so $\{e\}$ is extracted as a frequent itemset.
- 3. As e is frequent, find frequent itemsets ending in e. i.e. de, ce, be and ae.
  - i.e. decompose the problem recursively.
  - 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**: FP-Tree conditional on *e*.

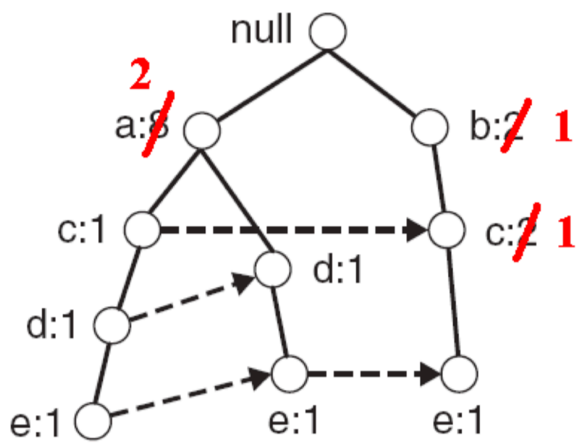
TID	Items
~~1~~	~~{a,b}~~
~~2~~	~~{b,c,d}~~
3	{a,c,d,~~e~~}
4	{a,d,~~e~~}
~~5~~	~~{a,b,c}~~
~~6~~	~~{a,b,c,d}~~
~~7~~	~~{a}~~
~~8~~	~~{a,b,c}~~
~~9~~	~~{a,b,d}~~
10	{b,c,~~e~~}



Conditional FP-Tree

To obtain the *conditional FP-tree* for *e* from the *prefix sub-tree* ending in *e*:

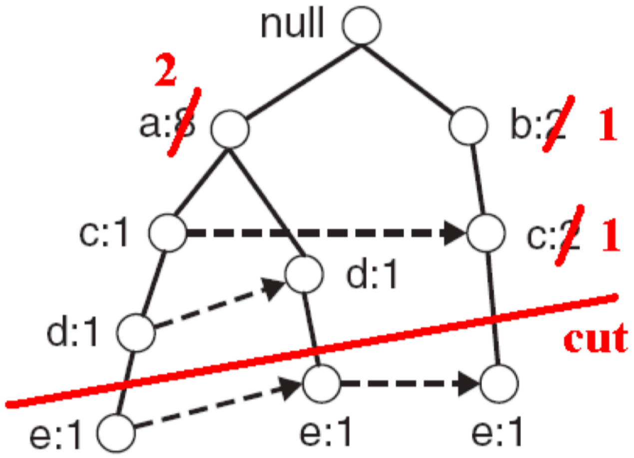
- Update the support counts along the prefix paths (from *e*) to reflect the number of transactions containing *e*.
  - *b* and *c* should be set to 1 and *a* to 2.

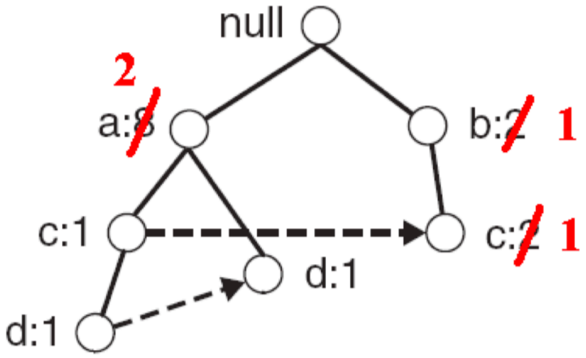


Conditional FP-Tree

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

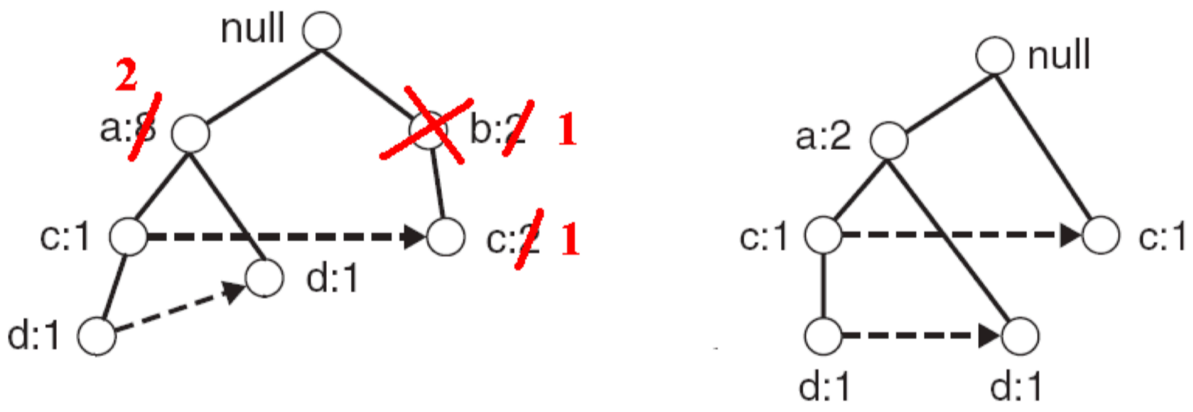




Conditional FP-Tree

To obtain the *conditional FP-tree* for *e* from the *prefix sub-tree* ending in *e*:

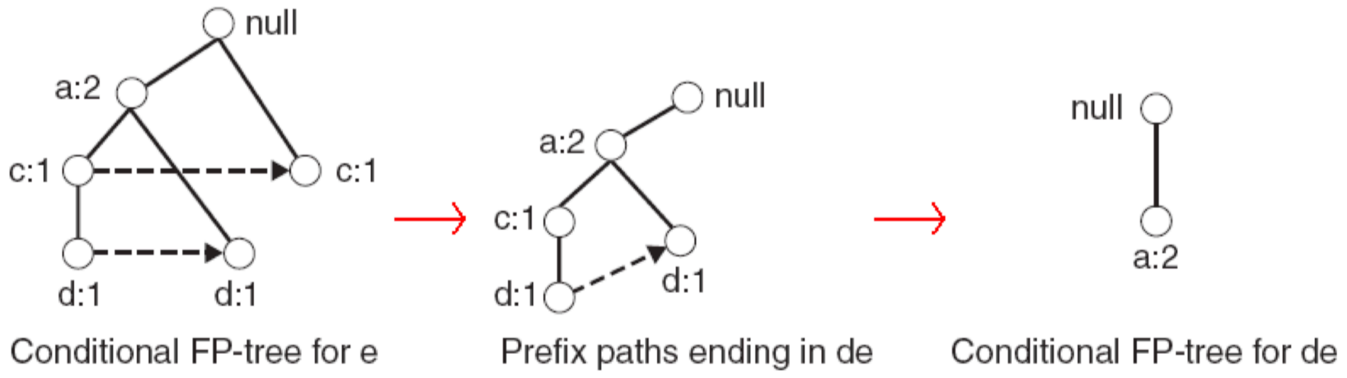
- Remove infrequent items (nodes) from the prefix paths
- **E.g.** *b* has a support of 1 (note this really means *be* has a support of 1). i.e. there is only 1 transaction containing *b and e* so *be* is infrequent – can remove b.



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

Example (continued)

- 4. Use the 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 conditional FP-tree, etc... (recursive)
  - **Example:** $e \rightarrow de \rightarrow ade$ ($\{d, e\}$,$\{a, d, e\}$ are found to be frequent)



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 data-set
  - "compresses" data-set
  - 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 support threshold is high), as the only pruning that can be done is on *single items*.
    - support can only be calculated once the entire data-set is added to the FP-Tree.

- **Apriori:** uses a generate-and-test approach – generates candidate itemsets and tests if they are frequent
  - Generation of candidate itemsets is expensive (in both space and time)
  - Support counting is expensive
    - Subset checking (computationally expensive)
    - Multiple Database scans (I/O)
- **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
    - Traversal through FP-Tree