Chapter 6:
Association Rules

Association rule mining

- Proposed by Agrawal et al in 1993.
- It is an important data mining model.
- Transaction data (no time-dependent)
- Assume all data are categorical. No good algorithm for numeric data.
- Initially used for Market Basket Analysis to find how items purchased by customers are related.

Bread $\rightarrow$ Milk [sup = 5%, conf = 100%]

Association rule mining

- Ανάλυση συσχετίσεων:
  - Η διαδικασία ανακάλυψης συσχετίσεων μεταξύ αντικειμένων σε μεγάλο όγκο δεδομένων, type “market basket”.
- Οι κανόνες συσχέτισης προσφέρουν μία απεικόνιση συσχετίσεων της μορφής:

  $A \rightarrow B$ (ύπαρξη συσχέτισης)
- Εφαρμογές σε bioinformatics, web mining, medical diagnosis, etc.

The model: data representation

- $I = \{i_1, i_2, \ldots, i_m\}$: a set of *items*. (alphabet)

- Transaction t :
  - t a set of items, and $t \subseteq I$.
  - Transaction data, itemset (στοιχειοσύνολο) from set I.

- Transaction set $T = \{t_1, t_2, \ldots, t_n\}$.

Transaction data: supermarket data

- Market basket transactions:

  t1: {bread, cheese, milk}

  t2: {apple, eggs, salt, yogurt}

  … …

  tn: {biscuit, eggs, milk}

- Concepts:
  - An *item*: an item/article in a basket
  - *I*: the set of all items sold in the store
  - A *transaction*: items purchased in a basket; it may have TID (transaction ID)
  - A *transactional dataset*: A set of transactions

Transaction data: a set of documents

- **A text document data set. Each document is treated as a "bag" of keywords**

doc1: Student, Teach, School

doc2: Student, School

doc3: Teach, School, City, Game

doc4: Baseball, Basketball

doc5: Basketball, Player, Spectator

doc6: Baseball, Coach, Game, Team

doc7: Basketball, Team, City, Game

The model: rules

- A transaction *t* contains *X*, a set of items (itemset) in *I*, if $X \subseteq t$.
- An association rule is an implication of the form:

  $X \rightarrow Y$, where $X, Y \subset I$, *and* $X \cap Y = \varnothing$

- An itemset is a set of items.
  - E.g., X = {milk, bread, cereal} is an itemset.
- A *k*-itemset is an itemset with *k* items.
  - E.g., {milk, bread, cereal} is a 3-itemset

Rule strength measures

- Support count of an itemset X:

$$\sigma(X) = \{ t_i \mid X \subseteq t_i , t_i \in T \}$$

- Number of transactions having itemset X

- Frequency of X.

Rule strength measures

$X \rightarrow Y$, where $X, Y \subset I$, *and* $X \cap Y = \varnothing$

- Support of a rule: The rule holds with support *sup* in *T* (the transaction data set) if sup% of transactions contain $X \cup Y$.

$$\text{support} = s(X \rightarrow Y) = \frac{\sigma(X \cup Y)}{N} = P(X \cup Y)$$

- relative frequency of **joint** appearance of union of X, Y.

Rule strength measures

- Confidence of a rule: The rule holds in *T* with confidence *conf* if *conf* % of tranactions that contain *X* also contain *Y.*

$$confidence = c(X \rightarrow Y) = \frac{\sigma(X \cup Y)}{\sigma(X)} = P(Y \mid X)$$

- **Conditional** frequency of appearance of Y given X.

Rule strength measures

- Why use support and confidence?
  - Rules with low support may occur simply by change
  - A low support rule is not interesting from a business perspective.
  - Confidence measures the reliability of the inference made by a rule.
    - The higher the confidence, the more likely it is for Y to be present in transactions containing X.

Goal and key features

- **Assume** transaction set $T = \{t_1, t_2, \ldots, t_n\}$
- **Goal:** Find all rules that **satisfy** user-specified
  - *minimum support* (**minsup**) and
  - *minimum confidence* (**minconf**).
- **Key Features**
  - Completeness: find all rules.
  - No target item(s) on the right-hand-side

An example (I)

t1:	Beef, Chicken, Milk
t2:	Beef, Cheese
t3:	Cheese, Boots
t4:	Beef, Chicken, Cheese
t5:	Beef, Chicken, Clothes, Cheese, Milk
t6:	Chicken, Clothes, Milk
t7:	Chicken, Milk, Clothes

- Transaction data
- Assume:

  minsup = 30%

  minconf = 80%
- An example frequent *itemset*:

{Chicken, Clothes, Milk} [sup = 3/7]

- Association rules from the itemset:

Clothes → Milk, Chicken [sup = 3/7, conf = 3/3]

…

Clothes, Chicken → Milk, [sup = 3/7, conf = 3/3]

An example (II)

- Itemset $I = \{i_1, i_2, \ldots, i_n\}$
- Find all rules $X \Rightarrow Y$ such that:
  - $min\_support = 50\%$
  - $min\_conf = 50\%$

TID	Items
T1	A, B, C
T2	A, C
T3	A, D
T4	B, E, F

$\boldsymbol{A \Rightarrow D}$ $(25\%, 33.3\%)$ ✘

$\boldsymbol{A \Rightarrow B}$ $(25\%, 33.3\%)$ ✘

$\boldsymbol{A \Rightarrow C}$ $(50\%, 66.7\%)$ ✔

$\boldsymbol{C \Rightarrow A}$ $(50\%, 100\%)$ ✔

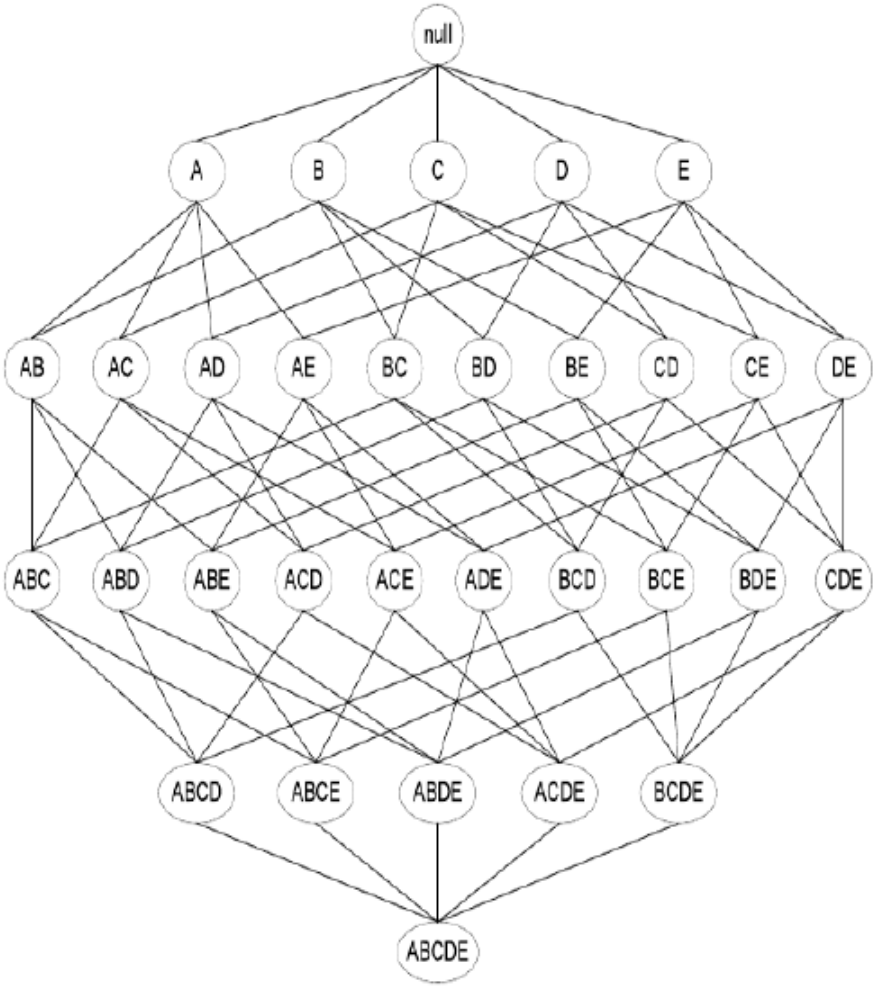
Mining Association Rules

- Two-step approach:
  1. Frequent Itemset Generation
     - Generate all itemsets whose *support* $\geq$ *minsup*
     - Large itemsets
  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 computationally **expensive**

Step 1: Mining all frequent itemsets

- For a set with k items
  - 2^k–1 possible itemsets
- We can use a lattice structure consisting of all possible itemsets.
- Each of these is called *candidate* frequent itemset.



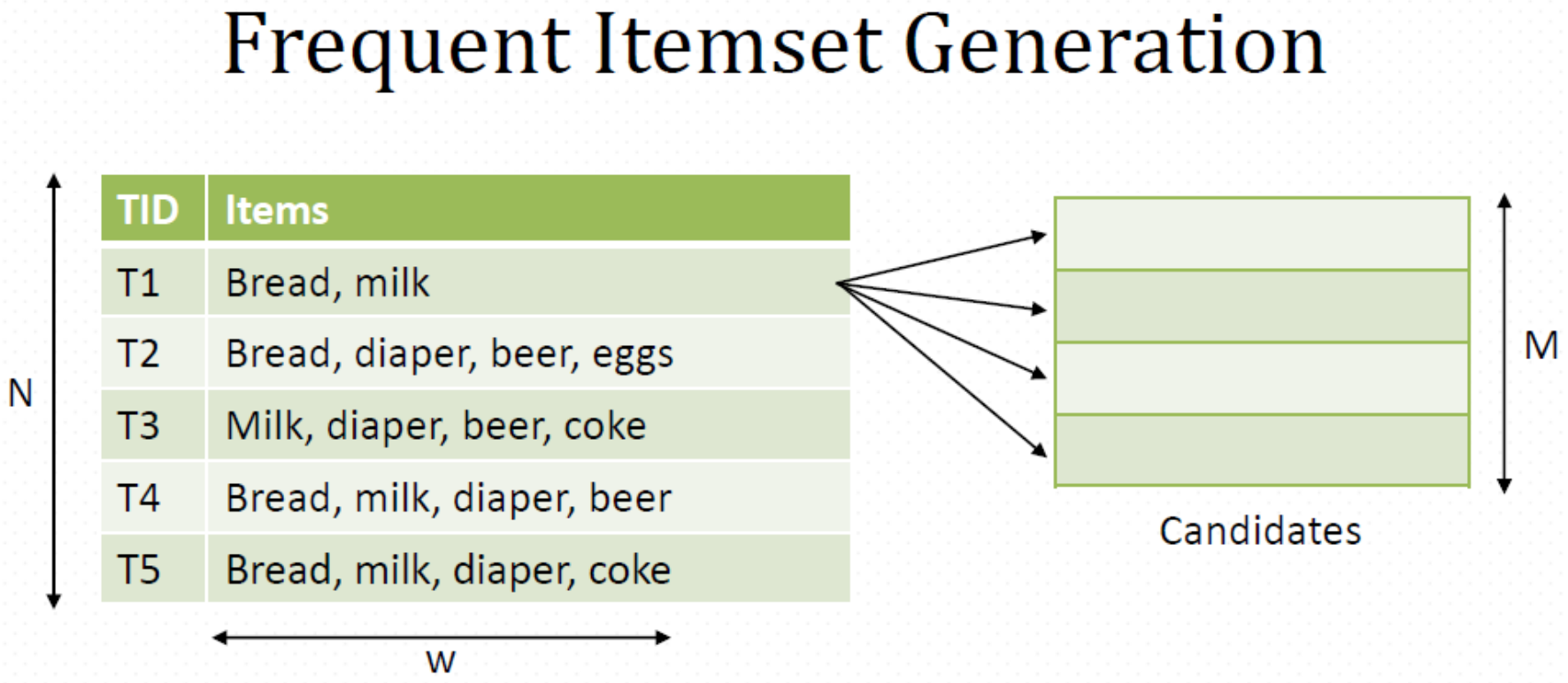
Itemset lattice

Step 1: Mining all frequent itemsets

Frequent item generation

- The brute-force approach
  - Compute the support and confidence for every possible k-itemset
- Prohibitively expensive

Step 1: Mining all frequent itemsets



- Compare each candidate against every transaction
- Complexity: $O(NMw)$

Step 1: Mining all frequent itemsets

- Several ways to reduce the computational complexity:
  - Reduce the number of candidate itemsets by pruning the itemset lattice (**Apriori Algorithm**)
  - Reduce the number of comparisons by using advanced data structures to store the candidate itemsets or to compress the dataset (**FP Growth**)

A-priori algorithm

Key idea:

- any subset of a frequent itemset is also frequent itemset
- Any subset of a non-frequent itemset is also non-frequent
- Pruning procedure that reduce searching

Reducing Number of Candidates: Apriori

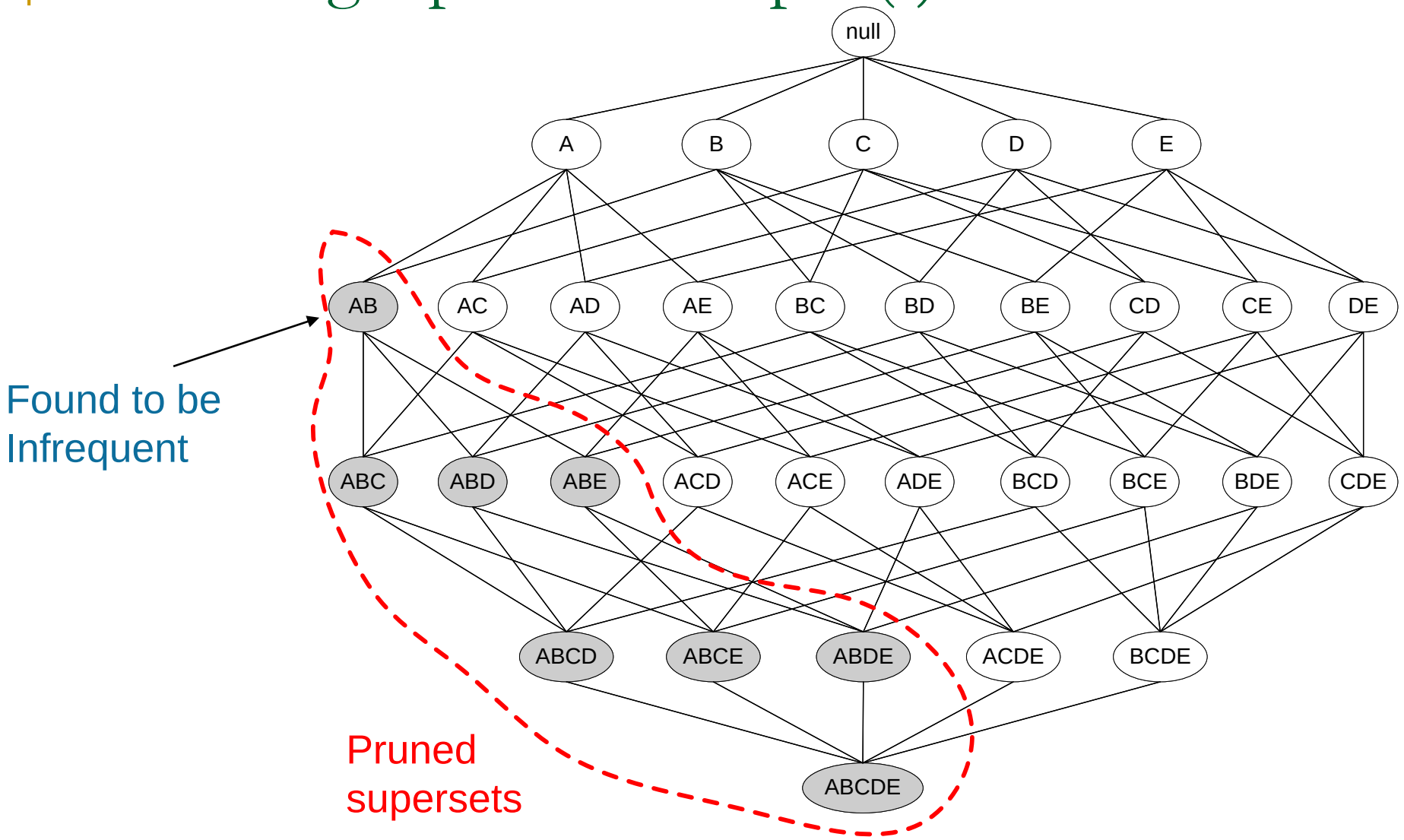
- Apriori principle:
  - If an itemset is frequent, then all of its subsets must also be frequent

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

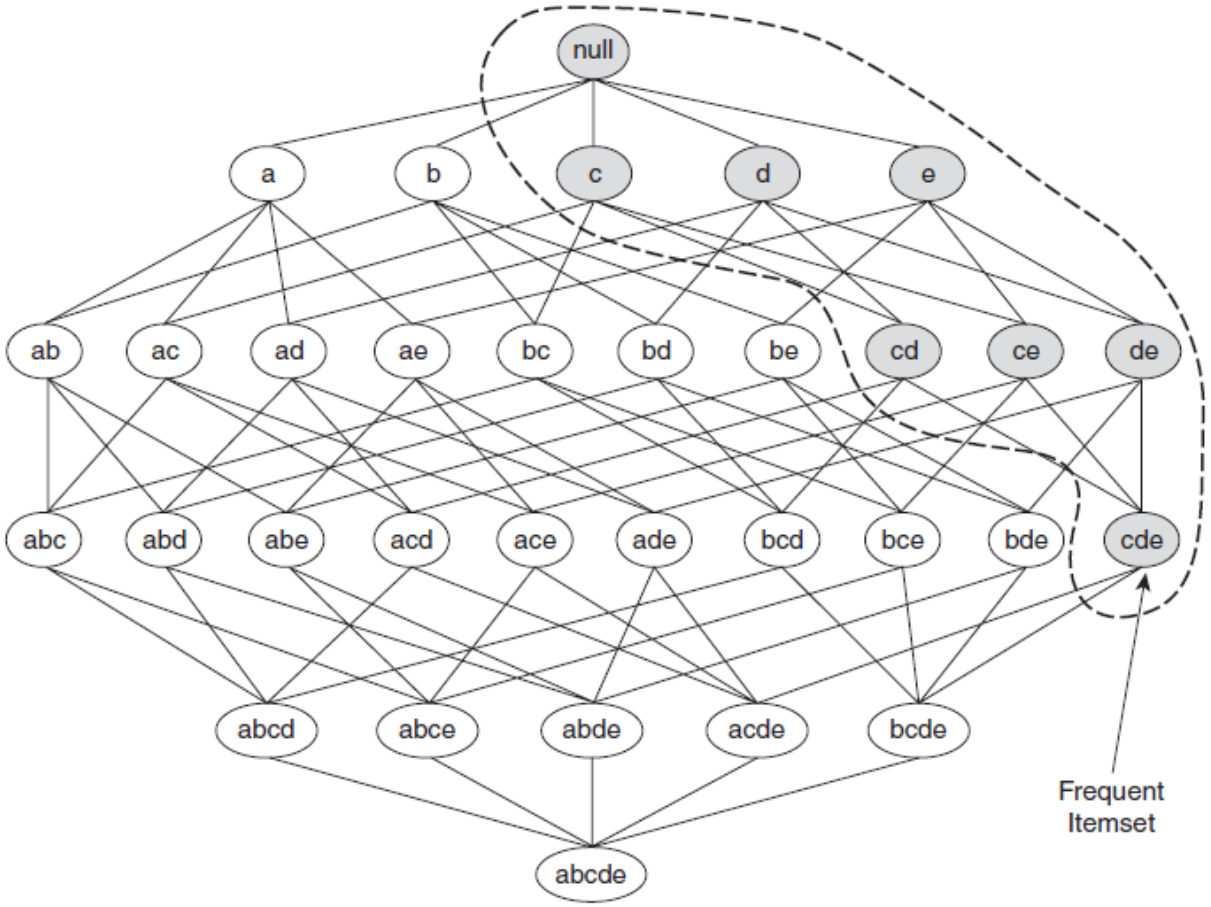
$$\forall X, Y : (X \subseteq Y) \Rightarrow s(X) \geq s(Y)$$

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

Illustrating Apriori Principle (I)



Illustrating Apriori Principle (II)



The A-priori Algorithm

- Find frequent itemsets of size k=1 (1-itemset) F_1 with support >= nminsup.
- repeat
  - k=k+1
  - Create candidate itemsets of size k C_k given F_{k-1}
  - Discover those itemsets that are actually frequent F_k , $F_k \subseteq C_k$ satisfying the support threshold (need to scan the database once) .
- until $F_k = \varnothing$
- *Frequent itemsets* $F = \cup_k F_k$

The A-priori Algorithm

- Find frequent itemsets of size k=1 (1-itemset) F_1 with support >= nminsup.
- repeat
  - k=k+1
  - Create candidate itemsets of size k C_k given F_{k-1}
  - Discover those itemsets that are actually frequent F_k , $F_k \subseteq C_k$ satisfying the support threshold (need to scan the database once) .
- until $F_k = \varnothing$
- *Frequent itemsets* $F = \cup_k F_k$

Details: the algorithm

Algorithm Apriori(*T*)

$C_1 \leftarrow$ init-pass(T);
$F_1 \leftarrow \{f \mid f \in C_1, \sigma(f) \geq N*minsup\}$; // n: no. of transactions in T
for ($k = 2$; $F_{k-1} \neq \emptyset$; k++) **do**
  $C_k \leftarrow$ candidate-gen(F_{k-1});
  for each transaction $t \in T$ **do**
    for each candidate $c \in C_k$ **do**
      if c is contained in t **then**
        $\sigma(c)=\sigma(c)+1$
    end
  end
  $F_k \leftarrow \{c \in C_k \mid \sigma(c) \geq N*minsup\}$
end
return $F \leftarrow \bigcup_k F_k$;

Example – Finding frequent itemsets

Dataset T
minsup=0.5

TID	Items
T100	1, 3, 4
T200	2, 3, 5
T300	1, 2, 3, 5
T400	2, 5

itemset:count

1. scan T ➔ C_1: {1}:2, {2}:3, {3}:3, {4}:1, {5}:3

   ➔ F_1: {1}:2, {2}:3, {3}:3, {5}:3

   ➔ C_2: {1,2}, {1,3}, {1,5}, {2,3}, {2,5}, {3,5}

2. scan T ➔ C_2: {1,2}:1, {1,3}:2, {1,5}:1, {2,3}:2, {2,5}:3, {3,5}:2

   ➔ F_2: **{1,3}**:2, **{2,3}**:2, **{2,5}**:3, **{3,5}**:2

   ➔ C_3: {2, 3,5}

3. scan T ➔ C_3: **{2, 3, 5}**:2 ➔ F_3: **{2, 3, 5}**

Candidate generation methods

- Brute-force: examine all possible itemsets candidates and then delete useless.
  - *weakness*: too many candidates

- F_{k-1} x F_1 : extend frequent k-1 itemsets found from previous step with the frequent 1-itemsets.
  - *weakness*: it is possible to create copies of candidate itemsets, ex. {a,b} $\cup$ {c} , {b,c} $\cup$ {a}

A possible solution: ordering of items

- The items are sorted in lexicographic order (which is a total order).
  - {*w*[1],*w*[2],…,*w*[*k*]} : a *k*-itemset *w* consisting of k items where *w*[1]<*w*[2]<…<*w*[*k*] according to the total lexicographic order.
- Then, k-1 itemsets can be extended only to items which are lexicographically higher than its own, e.g. {a,b}∪{c} is allowed but not {a,c}∪{b}
  - weakness: again, numerous of possible useless candidates itemsets, e.g. merge {a,b} {c} is useless since itemset {a,c} is not frequent.

A-priori candidate generation

- A-priori uses $F_{k-1} \times F_{k-1}$ method:
- Examines all possible pairs of F_{k-1} itemsets and keeps only those having *k-2* common items.
  - *A, B* are merged *iff* $a_i = b_i$ *i=1, …, k-2 and* $a_{k-2} \neq b_{k-2}$.
- Example (assume lexicographic order):

  F_2 = {a,c}, {b,c}, {b,d}, {c,d}

  then candidate

  C_3 = {b,c,d} merging {b,c} with {b,d}

A-priori candidate generation

- Two steps:
  - *join* step: Generate all possible candidate itemsets C_k of length k
  - *prune* step: Remove those candidates in C_k that cannot be frequent.

Candidate-gen function

Function candidate-gen(F_{k-1})

$C_k \leftarrow \emptyset$;

forall $f_1, f_2 \in F_{k-1}$

with $f_1 = \{i_1, \ldots, i_{k-2}, i_{k-1}\}$

and $f_2 = \{i_1, \ldots, i_{k-2}, i'_{k-1}\}$

and $i_{k-1} < i'_{k-1}$ **do**

$c \leftarrow \{i_1, \ldots, i_{k-1}, i'_{k-1}\}$; // join f_1 and f_2

$C_k \leftarrow C_k \cup \{c\}$;

for each (k-1)-subset s of c **do**

if ($s \notin F_{k-1}$) **then**

delete c from C_k; // prune

end

end

return C_k;

An example

- F_3 = {{1, 2, 3}, {1, 2, 4}, {1, 3, 4}, {1, 3, 5}, {2, 3, 4}}

- After join
  - C_4 = {{1, 2, 3, 4}, {1, 3, 4, 5}}
- After pruning:
  - C_4 = {{1, 2, 3, 4}}
    because {1, 4, 5} is not in F_3

    (and so {1, 3, 4, 5} is removed)

Step 2: Generating rules from frequent itemsets

- Frequent itemsets ≠ association rules
- Every *k*-itemset *Y* generates 2^k-2 possible rules.
- Every association rule divides the *k*-itemset *Y* into two non-empty subsets, *X* and *Y-X*

$$X \rightarrow Y\text{-}X$$

satisying that $c(X \rightarrow Y\text{-}X) \geq minconf$

- Important: every possible rule satisfies the *minsup* threshold.

Step 2: Generating rules from frequent itemsets (cont.)

An example

3-itemset $Y=\{a,b,c\}$

6 possible rules:

$\{a, b\} \rightarrow \{c\}$

$\{a, c\} \rightarrow \{b\}$

$\{b, c\} \rightarrow \{a\}$

$\{a\} \rightarrow \{b, c\}$

$\{b\} \rightarrow \{a, c\}$

$\{c\} \rightarrow \{a, b\}$

Generating rules: an example

- Suppose {2,3,4} is frequent, with sup=50%
  - Proper nonempty subsets: {2,3}, {2,4}, {3,4}, {2}, {3}, {4}, with sup=50%, 50%, 75%, 75%, 75%, 75% respectively
  - These generate these association rules:
    - 2,3 $\rightarrow$ 4, confidence=100%
    - 2,4 $\rightarrow$ 3, confidence=100%
    - 3,4 $\rightarrow$ 2, confidence=67%
    - 2 $\rightarrow$ 3,4, confidence=67%
    - 3 $\rightarrow$ 2,4, confidence=67%
    - 4 $\rightarrow$ 2,3, confidence=67%
    - All rules have support = 50%

Generating rules: complexity

- In order to obtain $A \rightarrow B$, we need to have support($A \cup B$) and support(A).
- But, all the required information for confidence computation has already been recorded in itemset generation. No need to see the data *T* any more.
- This step is not as time-consuming as frequent itemsets generation.

Generating rules: complexity

- Important: Confidence does not have monotonic property
  - Rule A $\rightarrow$ B can have less, equal or greater than confidence of any rule A' $\rightarrow$ B' , where A'$\subseteq$ A and B'$\subseteq$ B.
- However, if rule *X* $\rightarrow$ *Y-X* does not cover the confidence threshold, then any rule *X'* $\rightarrow$ *Y-X'* , X'$\subseteq$X does not cover confidence threshold, too

$$(s(X') \geq s(X))$$

A-priori Rule generation

- A-priori creates rules incrementally with respect to ”then” part of rule.
- Considering k-itemsets f_k
- Initially, we examine all rules with *m=1* items in “then part” of rule:

for each frequent k-itemset f_k *(k≥2)* do

H_1 = {i | i ∈ f_k } 1-item then part

ap_genrules(f_k, H_1)

end

e.g.

{a,c.d}→{b}
{a,b,d}→{c}

Recursive procedure ap_genrules(f_k, H_m)

k=|f_k| (number of frequent k-itemsets)

m=|H_m| (number of then part of rule)

if *k>m+1*

H_{m+1} = apriori-gen(H_m) */ create candidates by merging */

for $h_{m+1} \in H_{m+1}$

conf($f_k - h_{m+1} \rightarrow h_{m+1}$) = sup(f_k) / sup($f_k - h_{m+1}$)

if conf ≥ minconf

accept rule $f_k - h_{m+1} \rightarrow h_{m+1}$

else

delete h_{m+1}

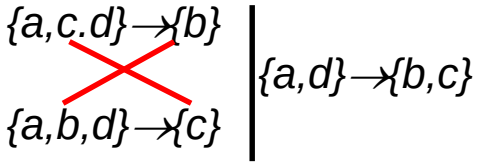
endif

endfor

ap_genrules(f_k, H_{m+1})

endif

e.g.



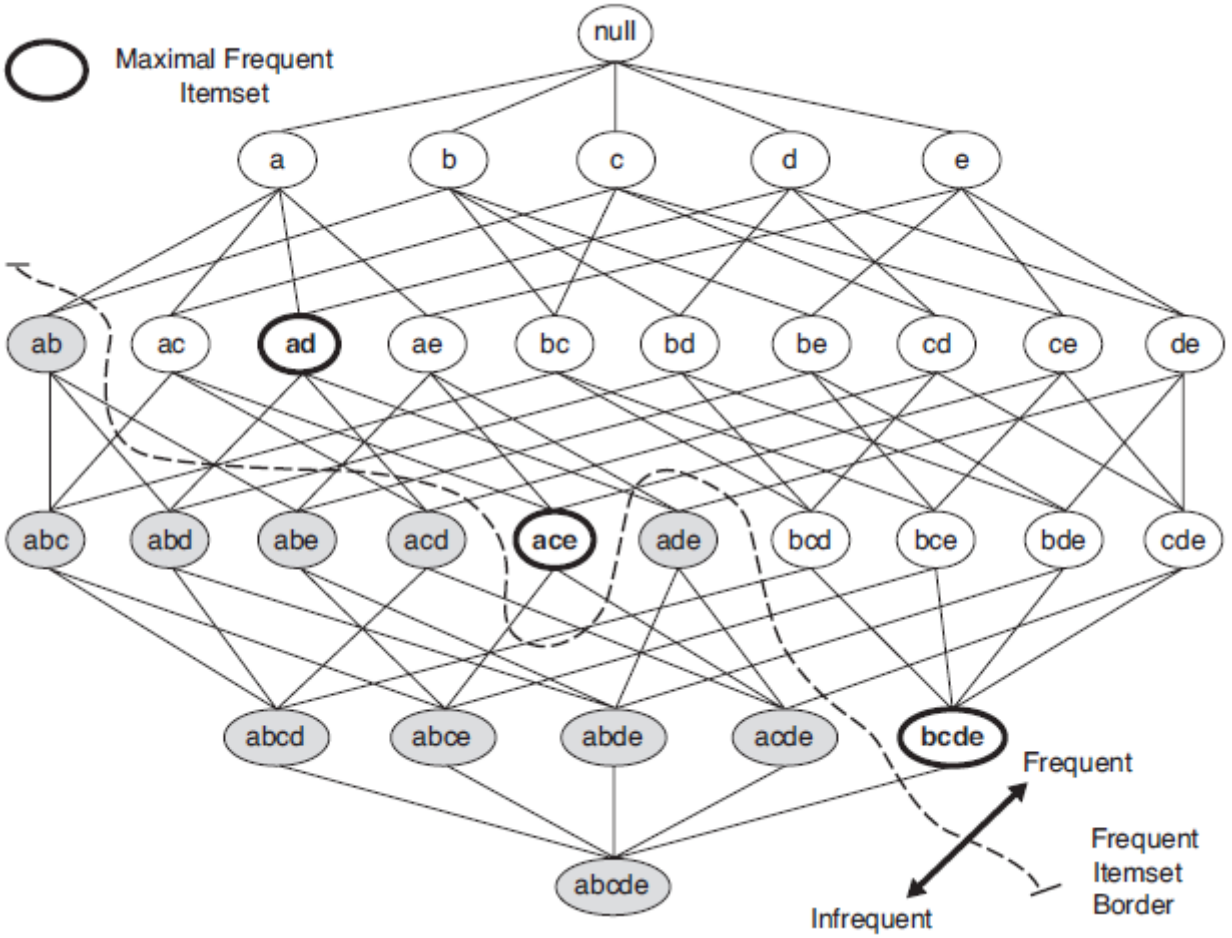
More: Compact representation of frequent itemsets

- Number of frequent itemsets produced can be very large.
- Need for a small representative set of itemsets from which all other itemsets can be derived.
- Two such representations
  - Maximal frequent itemsets
  - Closed frequent itemsets

Maximal frequent itemsets

- A Maximal frequent itemset is defined as a frequent itemset for which **none of its immediate** supersets are frequent.
- It stands in the border between frequent and non-frequent supersets.
- Smallest set of itemsets from which all frequent itemsets can be derived.
- However, they do not contain the support information of their subsets.

Maximal frequent itemsets



Closed frequent itemsets

- An itemset is closed if none of its immediate supersets has exactly the same support count.

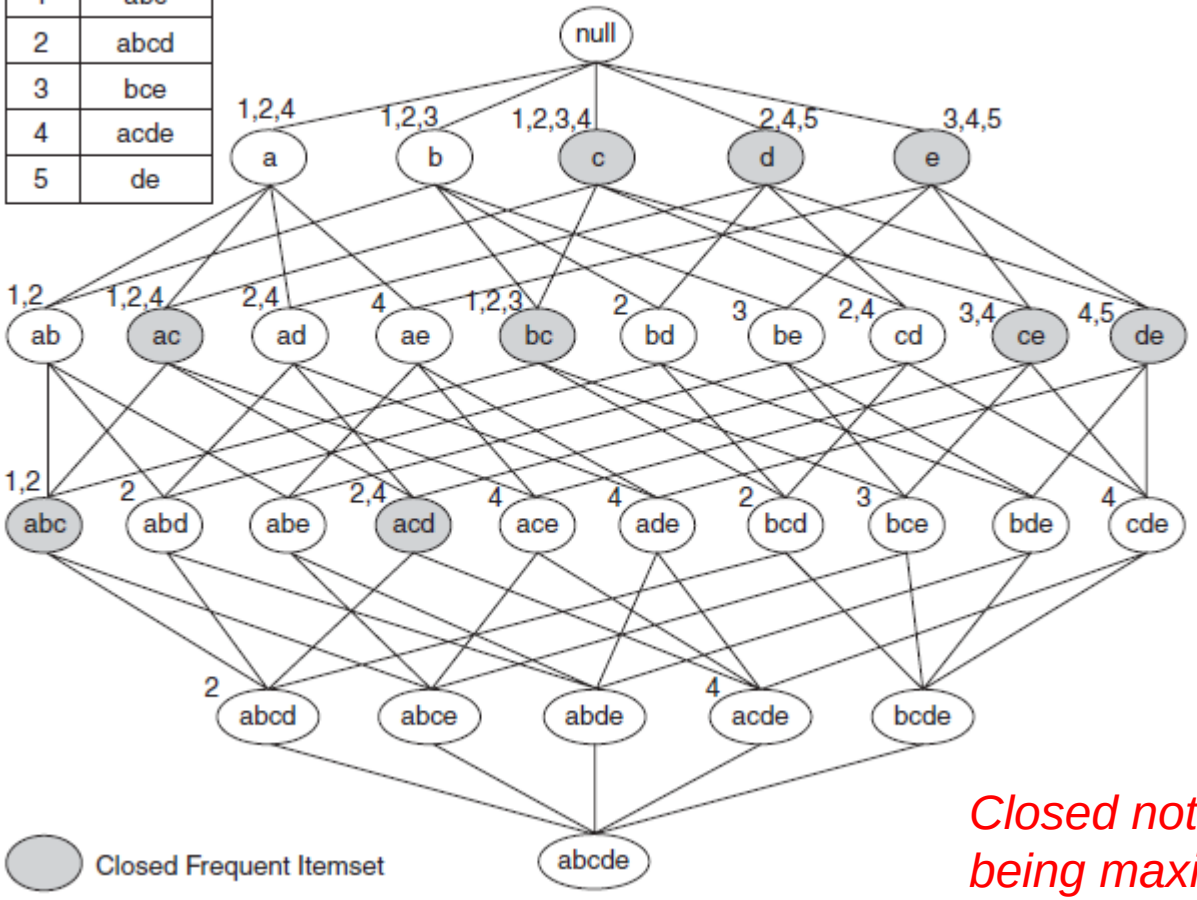
- Furthermore, a closed frequent itemset is a closed itemset where its support is ≥ minsup.

- Useful for removing redundant association rules:
  - Rule X' $\rightarrow$ Y' is redundant if rule X $\rightarrow$ Y has the same support and confidence, where X' $\subseteq$ X and Y' $\subseteq$ Y.

Closed frequent itemsets

TID	Items
1	abc
2	abcd
3	bce
4	acde
5	de

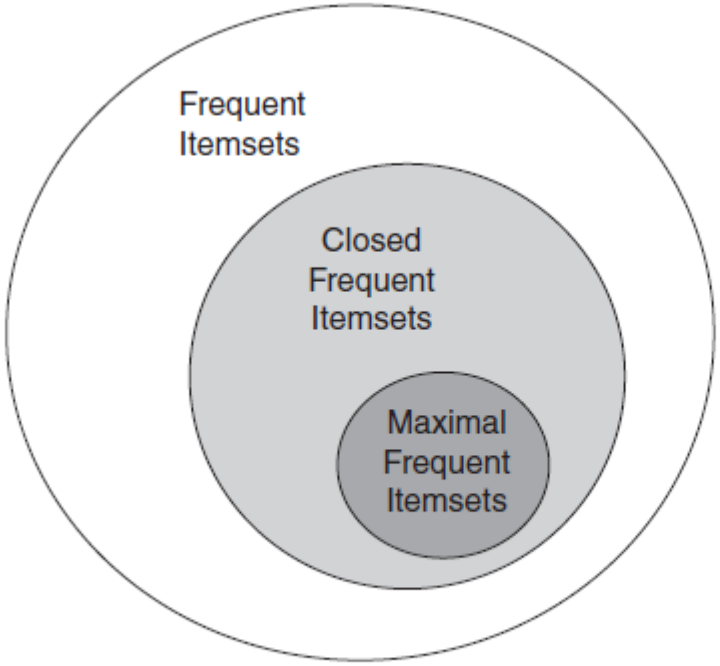
minsup = 40%



Closed not necessary being maximal

Relationships

All maximal frequent itemsets are closed since none of the maximal frequent itemsets can have the same support count as their Immediate supersets.



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

ECLAT: Another Method for Frequent Itemset Generation

- ECLAT: for each item, store a list of transaction ids (tids); vertical data layout

Horizontal Data Layout

TID	Items
1	A,B,E
2	B,C,D
3	C,E
4	A,C,D
5	A,B,C,D
6	A,E
7	A,B
8	A,B,C
9	A,C,D
10	B

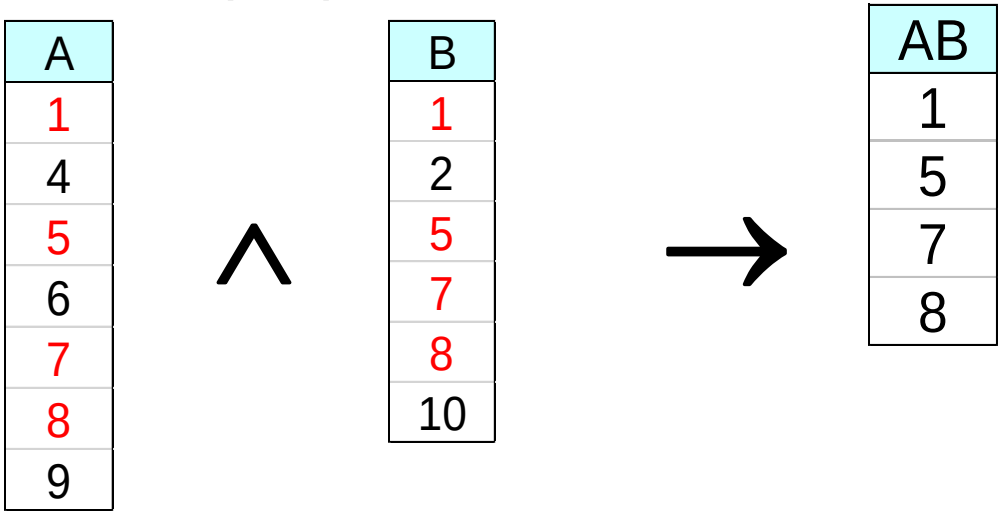
Vertical Data Layout

A	B	C	D	E
1	1	2	2	1
4	2	3	4	3
5	5	4	5	6
6	7	8	9	
7	8	9		
8	10			
9				

TID-list

ECLAT: Another Method for Frequent Itemset Generation

- Determine support of any k-itemset by intersecting tid-lists of two of its (k-1) subsets.



- 3 traversal approaches:
  - top-down, bottom-up and hybrid
- Advantage: very fast support counting
- Disadvantage: intermediate tid-lists may become too large for memory

FP–Growth Algorithm

- Encodes the data using a compact data structure called as FP-tree.

- Extracts frequent itemsets directly from this structure
  - uses a recursive divide-and-conquer approach to mine the frequent itemsets.

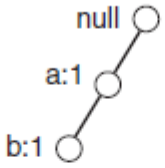
FP–Tree representation

- Sequentially mapping each transaction onto a path in the FP-tree (the more the paths overlap the more compression we achieve).
- Each node in the tree contains the label of an item along with a counter that shows the number of transactions mapped onto the given path (frequency).
- Tree constructions depends on order

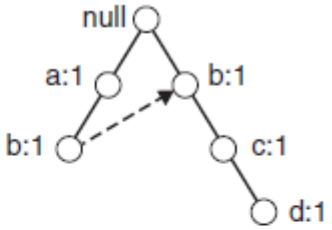
FP–Tree representation

Transaction Data Set

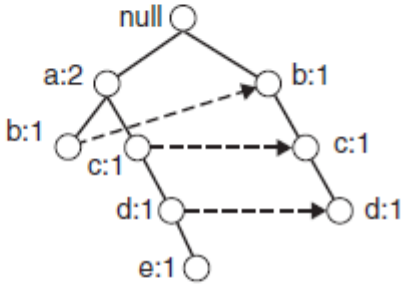
TID	Items
1	{a,b}
2	{b,c,d}
3	{a,c,d,e}
4	{a,d,e}
5	{a,b,c}
6	{a,b,c,d}
7	{a}
8	{a,b,c}
9	{a,b,d}
10	{b,c,e}



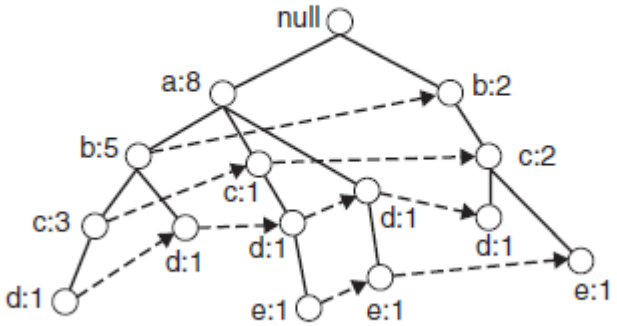
(i) After reading TID=1



(ii) After reading TID=2



(iii) After reading TID=3

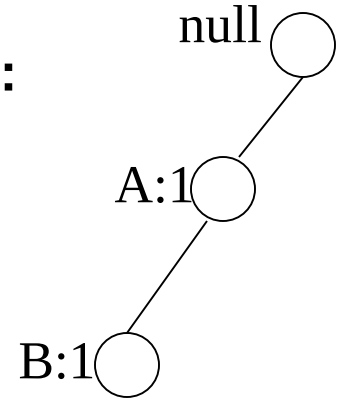


(iv) After reading TID=10

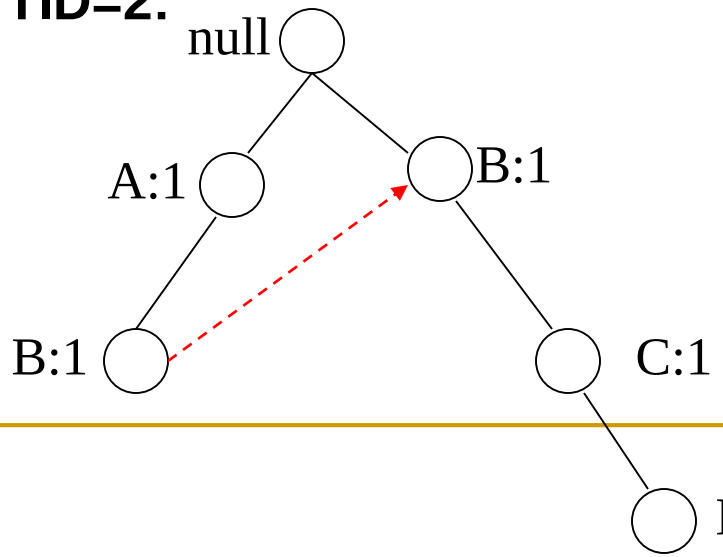
FP-Tree Construction

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	{B,C}
8	{A,B,C}
9	{A,B,D}
10	{B,C,E}

After reading TID=1:



After reading TID=2:



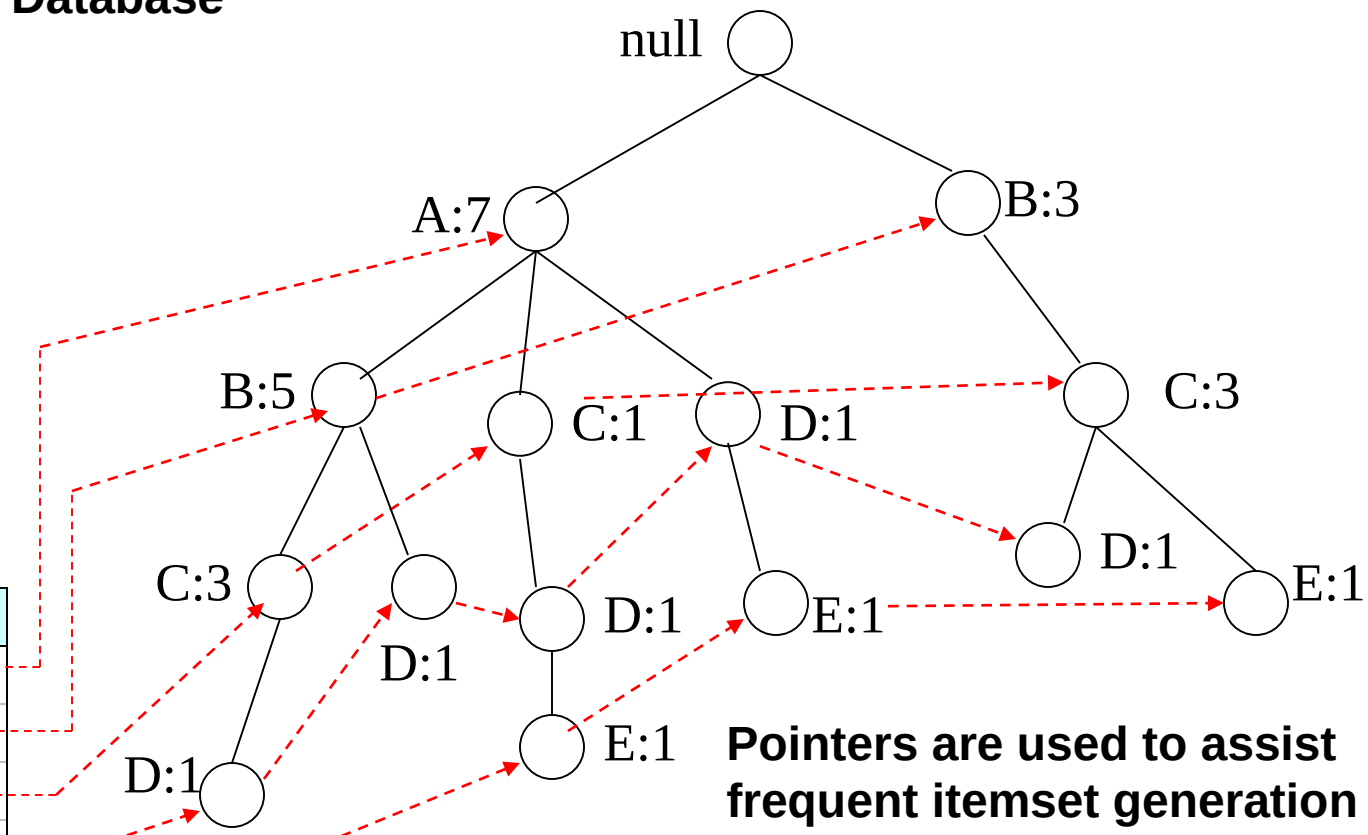
FP-Tree Construction

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	{B,C}
8	{A,B,C}
9	{A,B,D}
10	{B,C,E}

Transaction Database

Header table

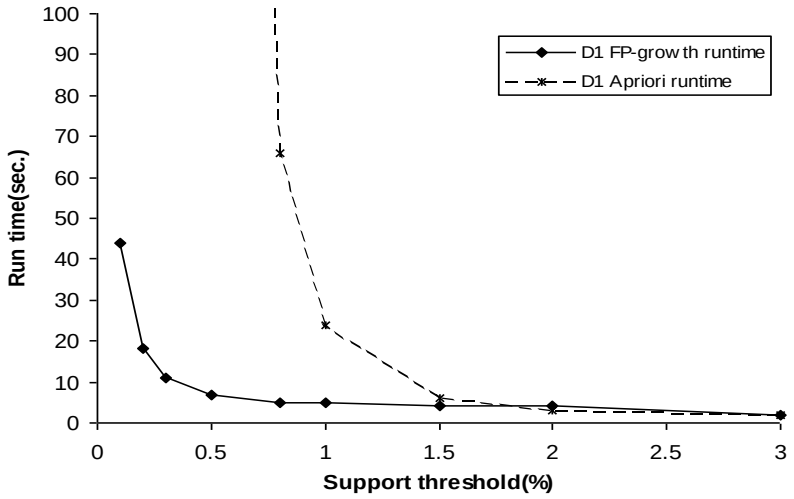
Item	Pointer
A	
B	
C	
D	
E	



Pointers are used to assist frequent itemset generation

Benefits of the FP-tree Structure

- Performance study shows
  - FP-growth is an order of magnitude faster than Apriori, and is also faster than tree-projection
- Reasoning
  - No candidate generation, no candidate test
  - Use compact data structure
  - Eliminate repeated database scan
  - Basic operation is counting and FP-tree building



Evaluation of Association Patterns

	B	$\overline{B}$	
A	f_{11}	f_{10}	f_{1+}
$\overline{A}$	f_{01}	f_{00}	f_{0+}
	f_{+1}	f_{+0}	N

- Data-driven for quality evaluation of association patterns
  - Contingency Table (based on frequency counts)
  - Calculate easily support and confidence measures

Evaluation of Association Patterns

- Evaluation metrics (degree of independence)
  - **Lift**: $\frac{c(A \to B)}{s(B)}$
  - **Interest**: $I(A,B) = \frac{s(A,B)}{s(A) \times s(B)} = \frac{Nf_{11}}{f_{1+} f_{+1}}$
  - **IS Measure**: $IS(A,B) = \sqrt{I(A,B) \times \sigma(A,B)} = \frac{s(A,B)}{\sqrt{s(A) \times s(B)}} = \cos(A,B)$

Sequential pattern mining

- Association rule mining does not consider the order of transactions.
- In many applications such orderings are significant. E.g.,
  - in market basket analysis, it is interesting to know whether people buy some items in sequence,
    - e.g., buying bed first and then bed sheets some time later.
  - In Web usage mining, it is useful to find navigational patterns of users in a Web site from sequences of page visits of users

Basic concepts

- Let $I = \{i_1, i_2, \ldots, i_m\}$ be a set of items.
- **Sequence:** An ordered list of itemsets.
- **Itemset/element**: A non-empty set of items $X \subseteq I$. We denote a sequence s by $\langle a_1 a_2 \ldots a_r \rangle$, where a_i is an itemset, which is also called an **element** of s.
- An element (or an itemset) of a sequence is denoted by $\{x_1, x_2, \ldots, x_k\}$, where $x_j \in I$ is an item.
- We assume without loss of generality that items in an element of a sequence are in **lexicographic order**.

Basic concepts (contd)

- **Size**: The **size** of a sequence is the number of elements (or itemsets) in the sequence.
- **Length**: The **length** of a sequence is the number of items in the sequence.
  - A sequence of length k is called ***k*-sequence**.
- A sequence $s_1 = \langle a_1 a_2 \ldots a_r \rangle$ is a **subsequence** of another sequence $s_2 = \langle b_1 b_2 \ldots b_v \rangle$, or s_2 is a **supersequence** of s_1, if there exist integers $1 \leq j_1 < j_2 < \ldots < j_{r-1} < j_r \leq v$ such that $a_1 \subseteq b_{j1}$, $a_2 \subseteq b_{j2}$, …, $a_r \subseteq b_{jr}$. We also say that s_2 **contains** s_1.

An example

- Let *I* = {1, 2, 3, 4, 5, 6, 7, 8, 9}.
- Sequence ⟨{3}{4, 5}{8}⟩ is **contained** in (or is a **subsequence** of) ⟨{6} {3, 7}{9}{4, 5, 8}{3, 8}⟩
  - because {3} ⊆ {3, 7}, {4, 5} ⊆ {4, 5, 8}, and {8} ⊆ {3, 8}.
  - However, ⟨{3}{8}⟩ is not contained in ⟨{3, 8}⟩ or vice versa.
  - The size of the sequence ⟨{3}{4, 5}{8}⟩ is 3, and the length of the sequence is 4.

Objective

- Given a set *S* of input data sequences (or sequence database), the problem of mining sequential patterns is to find all the sequences that have a user-specified minimum support.
- Each such sequence is called a **frequent sequence**, or a **sequential pattern**.
- The **support** for a sequence is the fraction of total data sequences in *S* that contains this sequence.

Example

Table 1. A set of transactions sorted by customer ID and transaction time

Customer ID	Transaction Time	Transaction (items bought)
1	July 20, 2005	30
1	July 25, 2005	90
2	July 9, 2005	10, 20
2	July 14, 2005	30
2	July 20, 2005	40, 60, 70
3	July 25, 2005	30, 50, 70
4	July 25, 2005	30
4	July 29, 2005	40, 70
4	August 2, 2005	90
5	July 12, 2005	90

Example (cond)

Table 2. Data sequences produced from the transaction database in Table 1.

Customer ID	Data Sequence
1	⟨{30} {90}⟩
2	⟨{10, 20} {30} {40, 60, 70}⟩
3	⟨{30, 50, 70}⟩
4	⟨{30} {40, 70} {90}⟩
5	⟨{90}⟩

Table 3. The final output sequential patterns

	Sequential Patterns with Support ≥ 25%
1-sequences	⟨{30}⟩, ⟨{40}⟩, ⟨{70}⟩, ⟨{90}⟩
2-sequences	⟨{30} {40}⟩, ⟨{30} {70}⟩, ⟨{30} {90}⟩, ⟨{40, 70}⟩
3-sequences	⟨{30} {40, 70}⟩

GSP mining algorithm

- Very similar to the Apriori algorithm

```
Algorithm GSP(S)
1    C1 ← init-pass(S);                                  // the first pass over S
2    F1 ← {⟨{f}⟩ | f ∈ C1, f.count/n ≥ minsup};          // n is the number of sequences in S
3    for (k = 2; Fk−1 ≠ ∅; k++) do                       // subsequent passes over S
4       Ck ← candidate-gen-SPM(Fk−1);
5       for each data sequence s ∈ S do                  // scan the data once
6          for each candidate c ∈ Ck do
7             if c is contained in s then
8                c.count++;                             // increment the support count
9          end
10      end
11      Fk ← {c ∈ Ck | c.count/n ≥ minsup}
12   end
13   return ⋃k Fk;
```

Fig. 12. The GSP Algorithm for generating sequential patterns

Candidate generation

Function candidate-gen-SPM(F_{k-1})

1. **Join step.** Candidate sequences are generated by joining F_{k-1} with F_{k-1}. A sequence s_1 joins with s_2 if the subsequence obtained by dropping the first item of s_1 is the same as the subsequence obtained by dropping the last item of s_2. The candidate sequence generated by joining s_1 with s_2 is the sequence s_1 extended with the last item in s_2. There are two cases:
   - the added item forms a separate element if it was a separate element in s_2, and is appended at the end of s_1 in the merged sequence, and
   - the added item is part of the last element of s_1 in the merged sequence otherwise.

   When joining F_1 with F_1, we need to add the item in s_2 both as part of an itemset and as a separate element. That is, joining $\langle\{x\}\rangle$ with $\langle\{y\}\rangle$ gives us both $\langle\{x, y\}\rangle$ and $\langle\{x\}\{y\}\rangle$. Note that x and y in $\{x, y\}$ are ordered.
2. **Prune step**. A candidate sequence is pruned if any one of its $(k-1)$-subsequence is infrequent (without minimum support).

Fig. 13. The candidate-gen-SPM() function

An example

Table 4. Candidate generation: an example

Frequent 3-sequences	Candidate 4-sequences	
	after joining	after pruning
⟨{1, 2} {4}⟩	⟨{1, 2} {4, 5}⟩	⟨{1, 2} {4, 5}⟩
⟨{1, 2} {5}⟩	⟨{1, 2} {4} {6}⟩	
⟨{1} {4, 5}⟩		
⟨{1, 4} {6}⟩		
⟨{2} {4, 5}⟩		
⟨{2} {4} {6}⟩		