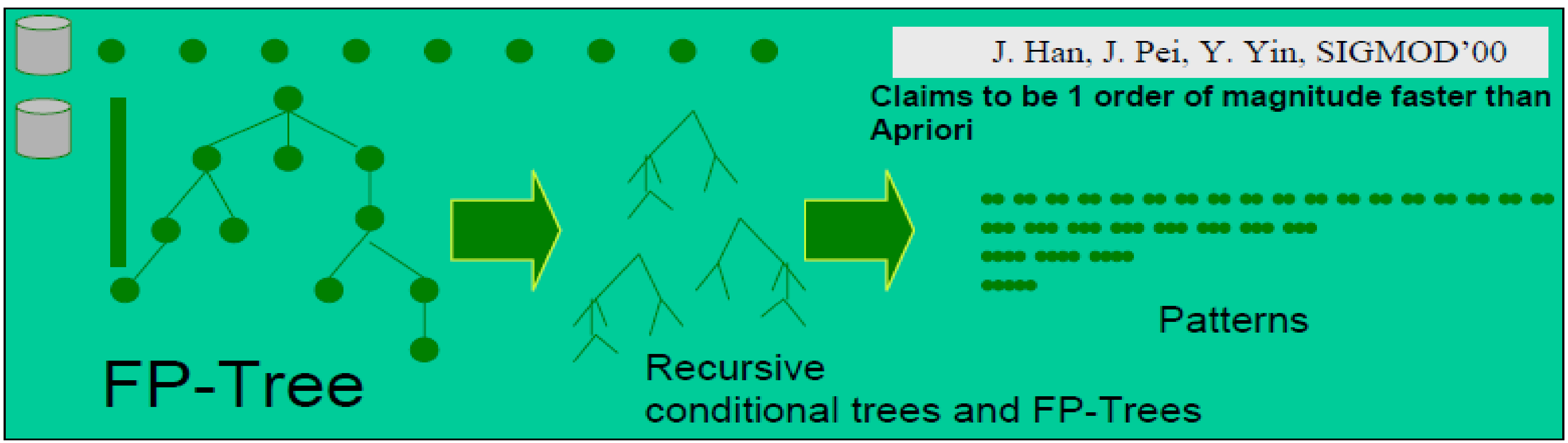
Frequent Pattern Growth (FP)

- First algorithm that allows frequent pattern mining without generating candidate sets



Construct FP-tree from a Transaction Database

TID	Items bought	(ordered) frequent items
100	{f, a, c, d, g, i, m, p}	{f, c, a, m, p}
200	{a, b, c, f, l, m, o}	{f, c, a, b, m}
300	{b, f, h, j, o, w}	{f, b}
400	{b, c, k, s, p}	{c, b, p}
500	{a, f, c, e, l, p, m, n}	{f, c, a, m, p}

min_support = 3

Frequent Items
f-c-a-b-m-p

1. Scan DB once, find frequent 1-itemset
2. Sort frequent items in frequency descending order
3. Scan DB again, construct FP-tree

Header Table

Item	*frequency*
f	4
c	4
a	3
b	3
m	3
p	3

Construct FP-tree from a Transaction Database

TID	Items bought	(ordered) frequent items
100	{f, a, c, d, g, i, m, p}	{f, c, a, m, p}
200	{a, b, c, f, l, m, o}	{f, c, a, b, m}
300	{b, f, h, j, o, w}	{f, b}
400	{b, c, k, s, p}	{c, b, p}
500	{a, f, c, e, l, p, m, n}	{f, c, a, m, p}

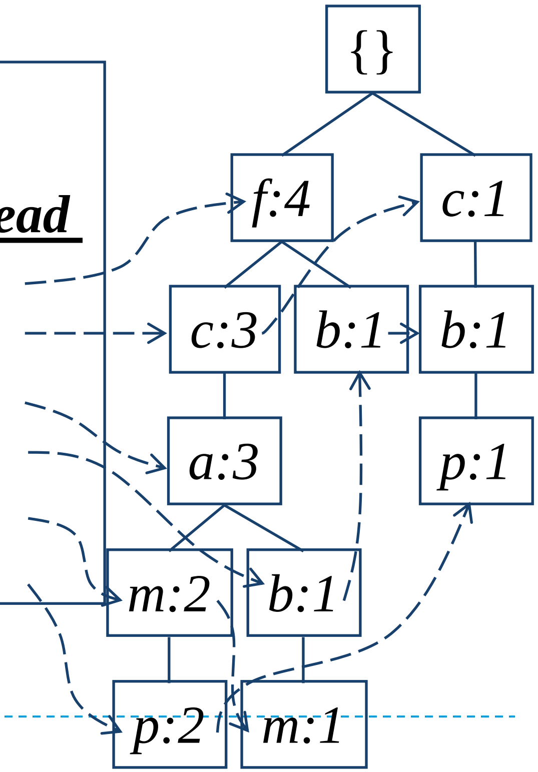
min_support = 3

Frequent Items
f-c-a-b-m-p

1. Scan DB once, find frequent 1-itemset
2. Sort frequent items in frequency descending order
3. Scan DB again, construct FP-tree

Header Table

Item	*frequency*	*head*
f	*4*	
c	*4*	
a	*3*	
b	*3*	
m	*3*	
p	*3*	

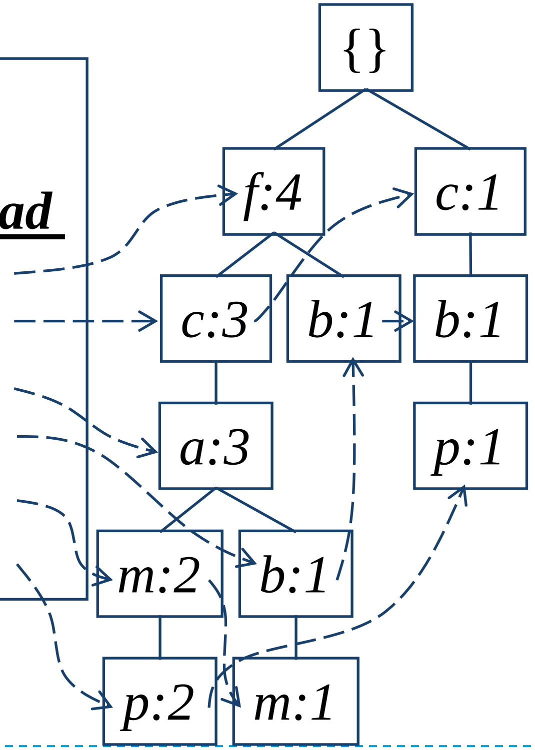


Step 1: Construct Conditional Pattern Base

- Starting at the frequent item header table in the FP-tree
- Traverse the FP-tree by following the link of each frequent item
- Accumulate all of *transformed prefix paths* of that item to form *a* conditional pattern base

Header Table

Item	*frequency*	*head*
f	*4*	
c	*4*	
a	*3*	
b	*3*	
m	*3*	
p	*3*	



***Conditional* pattern bases**

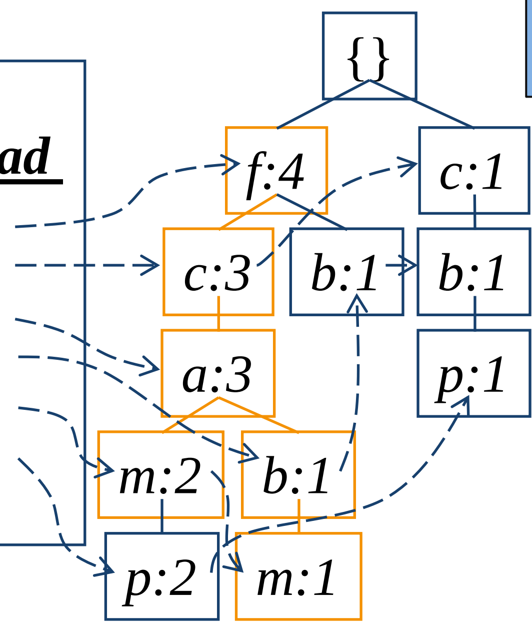
item	***Cond. pattern base***
c	***f:3***
a	***fc:3***
b	***fca:1, f:1, c:1***
m	***fca:2, fcab:1***
p	***fcam:2, cb:1***

Step 2: Construct Conditional FP-tree

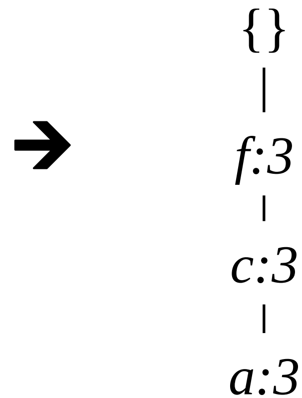
- For each pattern-base
  - Accumulate the count for each item in the base
  - Construct the conditional FP-tree for the frequent items of the pattern base

Header Table

Item	***frequency***	***head***
f	*4*	
c	*4*	
a	*3*	
b	*3*	
m	*3*	
p	*3*	



m-conditional **pattern base:**
fca:2, fcab:1



m-conditional **FP-tree**

Conditional Pattern Bases and Conditional FP-Tree

Item	Conditional pattern base	Conditional FP-tree
p	{(fcam:2), (cb:1)}	{(c:3)}|p
m	{(fca:2), (fcab:1)}	{(f:3, c:3, a:3)}|m
b	{(fca:1), (f:1), (c:1)}	Empty
a	{(fc:3)}	{(f:3, c:3)}|a
c	{(f:3)}	{(f:3)}|c
f	Empty	Empty

Single FP-tree Path Generation

- For single path the frequent patterns can be generated by enumeration of all the combinations of the sub-paths

{}
|
f:3
|
c:3
|
a:3

m-conditional FP-tree

→

All frequent patterns concerning *m*

m,

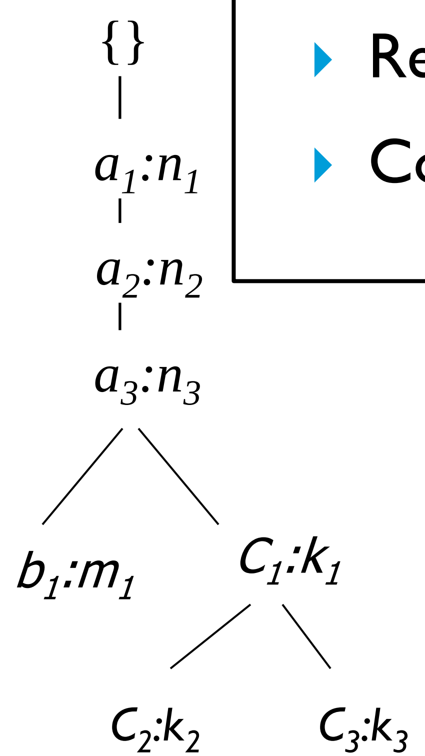
fm, *cm*, *am*,

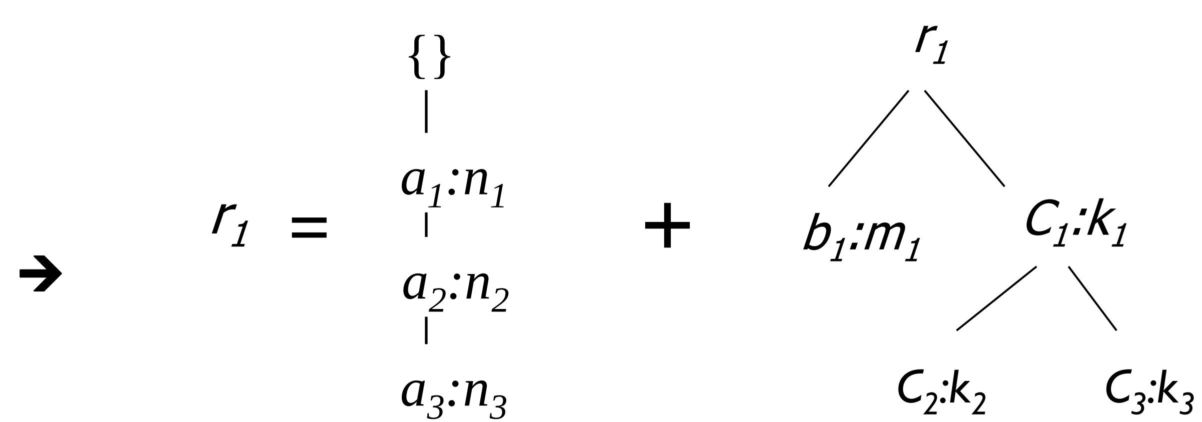
fcm, *fam*, *cam*,

fcam

A Special Case: Single Prefix Path in FP-tree

- Suppose a (conditional) FP-tree T has a shared single prefix-path P
- Mining can be decomposed into two parts
  - Reduction of the single prefix path into one node
  - Concatenation of the mining results of the two parts

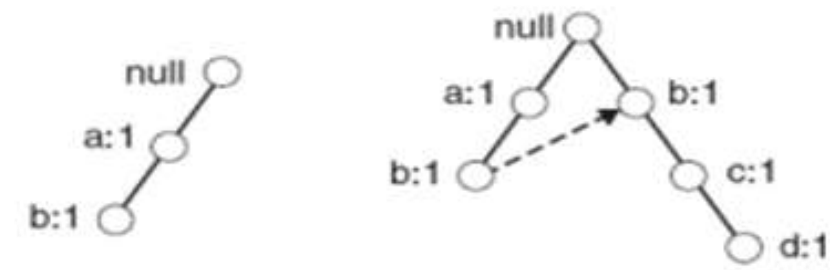




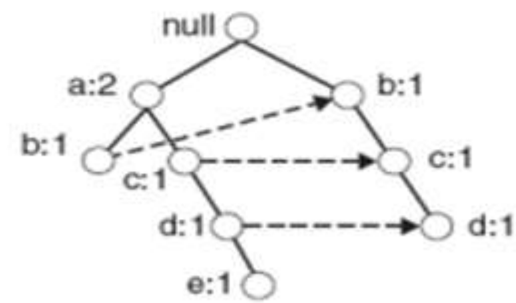
Example 2: FP-Tree Construction

Transaction
Data Set

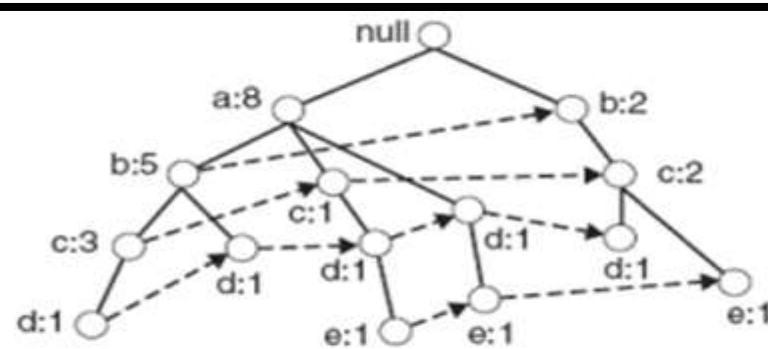
TID	Items
1	{a,b}
2	{b,c,d}
3	{a,c,d,e}
4	{a,d,e}
5	{a,b,c}
6	{a,b,c,d}
7	{a}
8	{a,b,c}
9	{a,b,d}
10	{b,c,e}



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

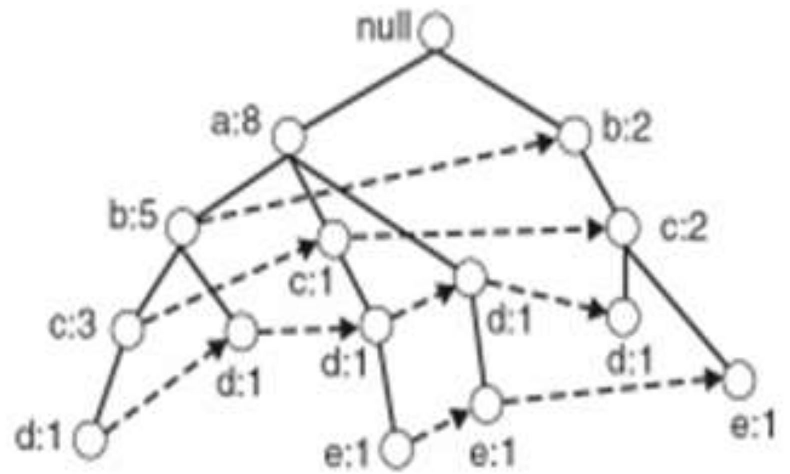


(iii) After reading TID=3

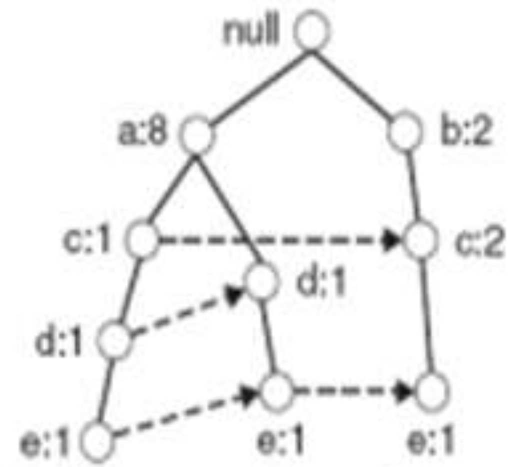


(iv) After reading TID=10

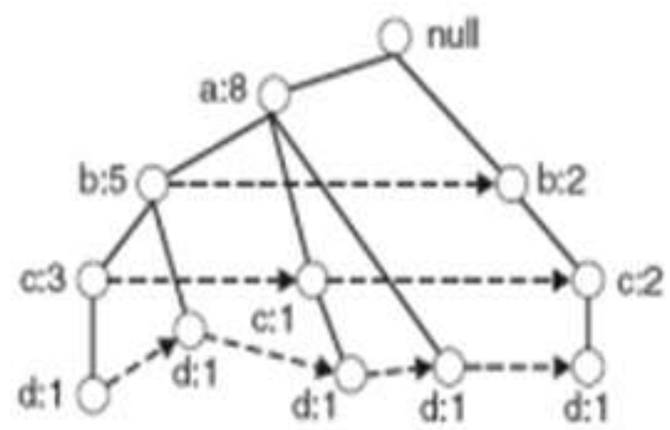
Example 2: Conditional Pattern Base



↑ Complete FP-tree

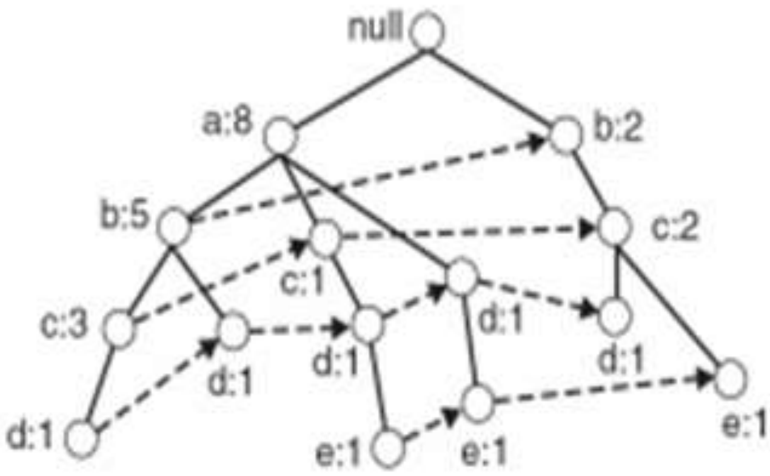


(a) Paths containing node e

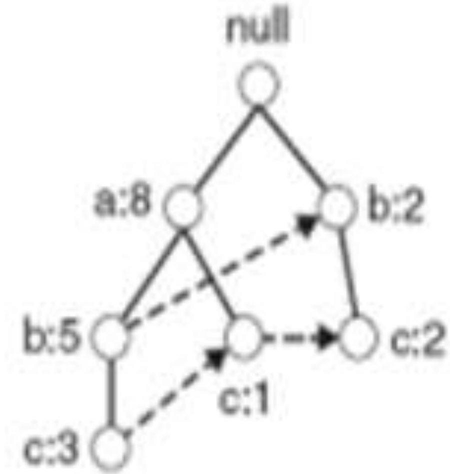


(b) Paths containing node d

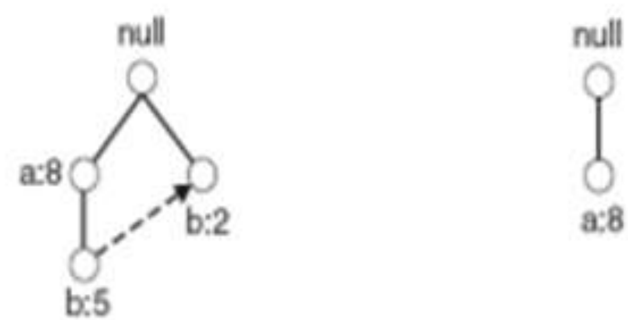
Example 2: Conditional Pattern Base



↑ Complete FP-tree

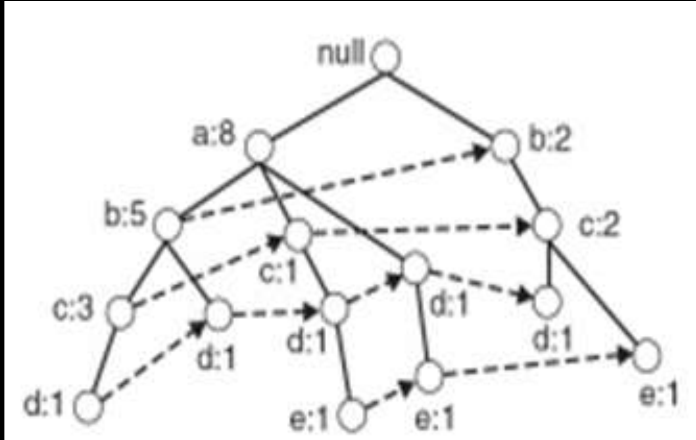


(c) Paths containing node c

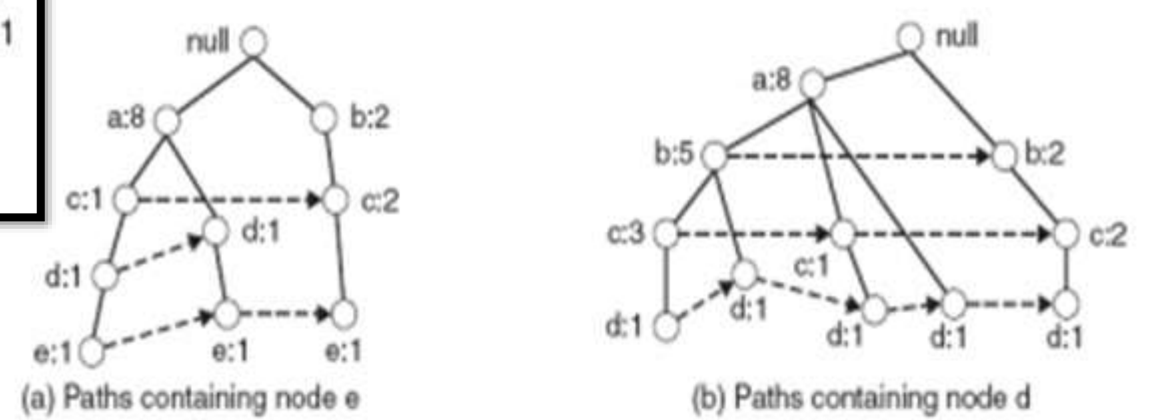


(d) Paths containing node b (e) Paths containing node a

Example 2: Conditional Pattern Base



↑ Complete FP-tree

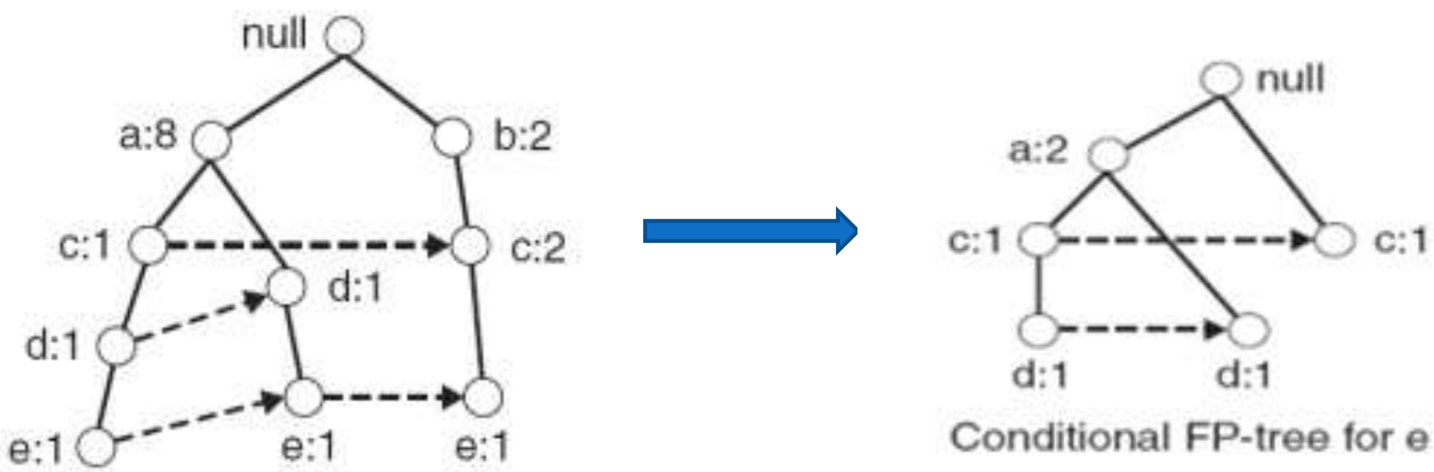


Example 2: Extract all frequent itemsets containing e

minSup = 2

Extract all frequent itemsets containing e

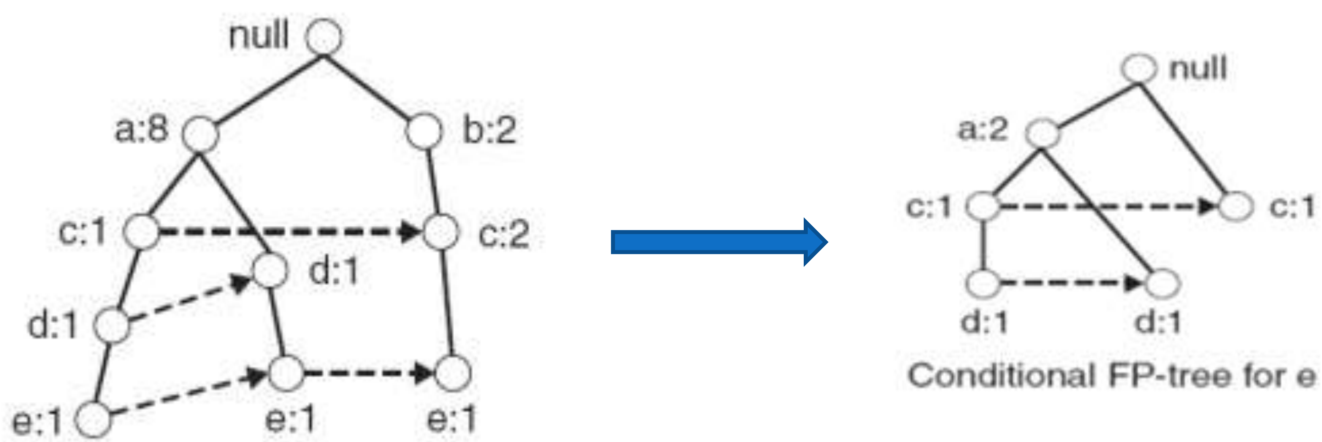
Conditional Pattern base and FP tree for e



Example

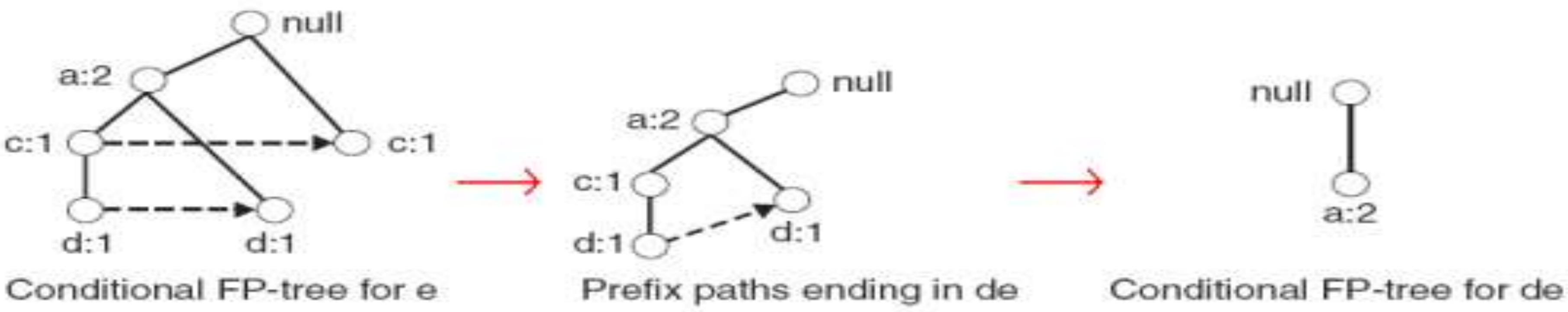
- **Check if e is a frequent item**
  - Add the counts along the linked list (dotted line).
  - If, count = min support, then {e} is extracted as a frequent itemset.
- As e is frequent, find frequent itemsets ending in e. i.e. de, ce and ae.

minSup = 2



Example

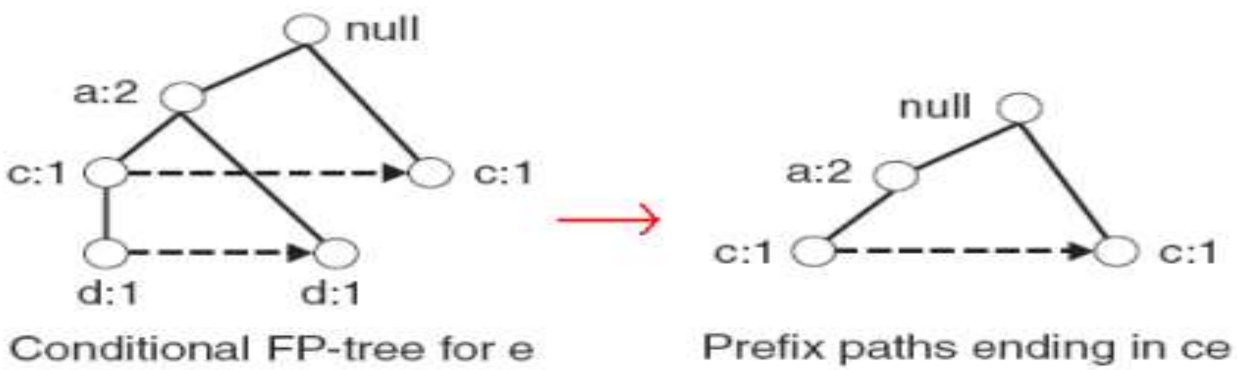
- Example: e -> de -> ade
- {d,e}, {a,d,e} are found to be frequent



minSup = 2

Example

- Example: e -> ce
- {c,e} is found to be frequent)



Result

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

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}

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}

Frequent Pattern Growth Mining Method

- **Idea: Frequent pattern growth**
  - Recursively grow frequent patterns by pattern and database partition
- **Method**
  - For each frequent item, construct its conditional pattern-base, and then its conditional FP-tree
  - Repeat the process on each newly created conditional FP-tree
  - Until the resulting FP-tree is empty, or it contains only one path—single path will generate all the combinations of its sub-paths, each of which is a frequent pattern

FP-Tree size

- 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.

Discusion

- 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

The Partitioning Algorithm

- Divide database into *n partitions*.
  - so that each portion can fit into memory.

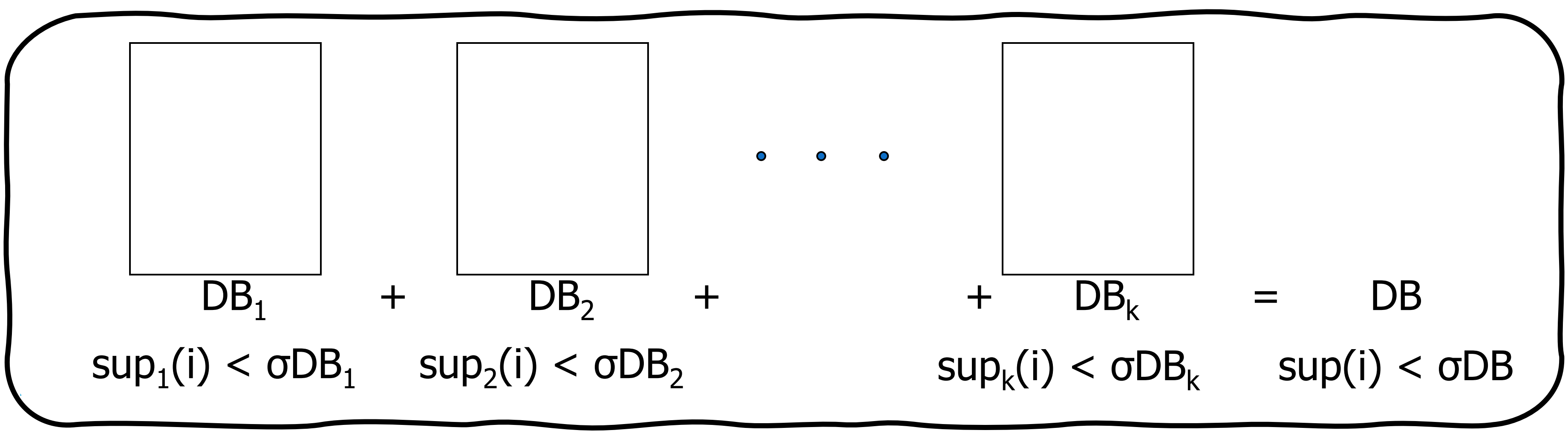
- Any itemset that is frequent in DB must be frequent in at least one of the n partitions (pigeon hole principle)

$$DB_1 + DB_2 + \cdots + DB_k = DB$$

$$sup_1(i) < \sigma DB_1 \quad sup_2(i) < \sigma DB_2 \quad sup_k(i) < \sigma DB_k \quad sup(i) < \sigma DB$$

The Partitioning Algorithm

- Require 2 DB Scans
  - Scan 1: partition database and find local frequent patterns
  - Scan 2: consolidate global frequent patterns



The Partitioning Algorithm

- Scan 1: partition database and find local frequent patterns
- Process one partition in main memory at a time:
  - For each partition, generate frequent itemsets using the Apriori algorithm
    - min_support for a partition = min_support of DB x no of transactions in that partition (here min_support is in %)
  - Form *Tidlist* for all item sets to facilitate counting in the merge phase

$DB_1 + DB_2 + \cdots + DB_k = DB$

$sup_1(i) < \sigma DB_1$ $\quad sup_2(i) < \sigma DB_2$ $\quad sup_k(i) < \sigma DB_k$ $\quad sup(i) < \sigma DB$

The Partitioning Algorithm

- After all partitions are processed, the local frequent itemsets are merged into global frequent sets
  - support can be computed from the *tidlists*.

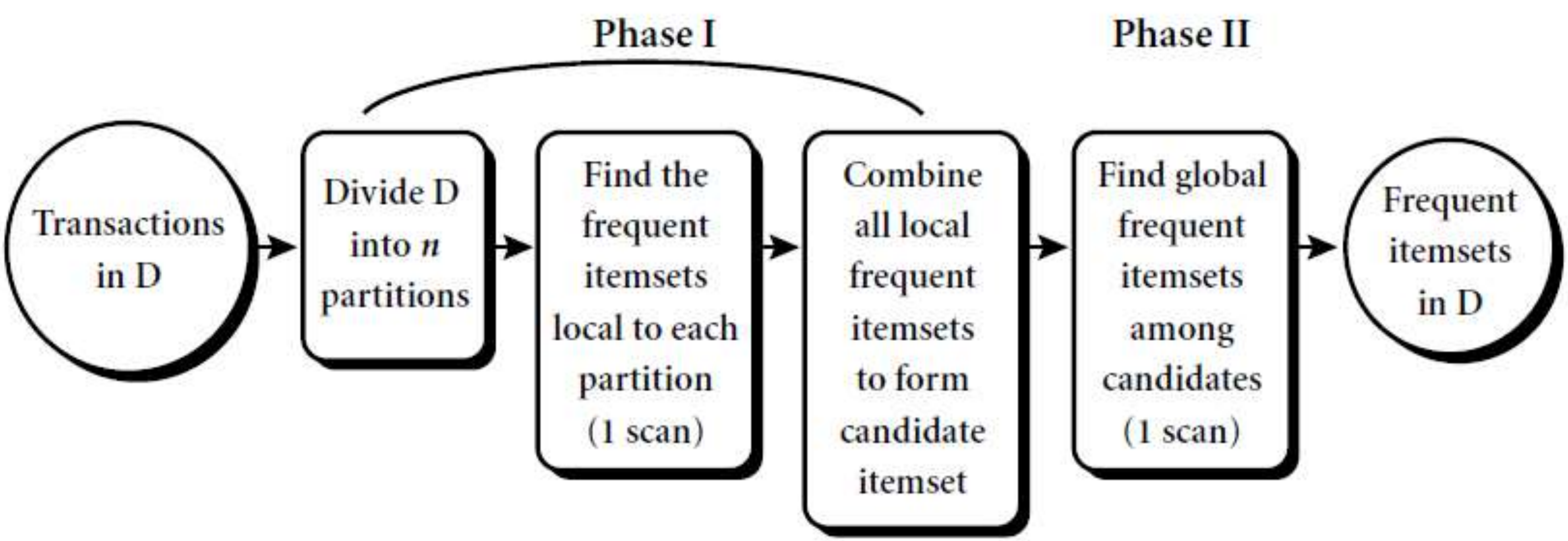


Figure 6.6: Mining by partitioning the data.

- Partition (Savasere, Omiecinski, & Navathe, VLDB'95).