Bottleneck of Frequent-pattern Mining

- Multiple database scans are costly
- Mining long patterns needs many passes of scanning and generates lots of candidates
  - To find frequent itemset $i_1 i_2 \ldots i_{100}$
    - # of scans: 100
    - # of Candidates: $\binom{100}{1} + \binom{100}{2} + \ldots + \binom{100}{100} = 2^{100} - 1 = 1.27 \times 10^{30}$!
- Bottleneck: candidate-generation-and-test
- Can we avoid candidate generation?

FP- Tree

- Definition: A frequent pattern (FP) tree is a tree structure consisting of an item-prefix-tree and a frequent item header table
- Item-prefix-tree
  - Root node (label-null)
  - 3 fields for each non root node
    - Item name
    - Support count
    - Node link (with link to node with same item name)

- Frequent item header table
  - Item name
  - Support count
  - Head of node link which points to the first node in the FP tree carrying the item name
- FP-tree features
  - Dependent on support threshold σ. (i.e., for different values of σ, the trees are different)
  - Depends on ordering of items (decreasing order of support count)

FP-growth

- Adopts divide and conquer strategy
- First, compresses the database representing frequent items into a frequent pattern tree (FP-tree) which retains itemset association information
- Divides compressed database into a set of conditional databases each associated with one frequent item or pattern segment

Benefits

- Completeness
  - Preserve complete information for frequent pattern mining
- Compactness
  - Reduce irrelevant info—infrequent items are gone
  - Items in frequency descending order: the more frequently occurring, the more likely to be shared
  - Never be larger than the original database

1. Scan DB once, find frequent 1-itemset (single item pattern)
2. Sort frequent items in frequency descending order, L-list
3. Scan DB again, construct FP-tree based on L-list
4. Mine frequent item sets using conditional pattern base, derived from FP-tree

NOTE:

The transformed prefix path of a node ‘a’ from a truncated database of patterns which co-occur with a is called a conditional pattern base

Find Patterns Having P From P-conditional Database

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

From Conditional Pattern-bases to Conditional FP-trees

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

Recursion: Mine Each Conditional FP-tree

Example: FP-growth

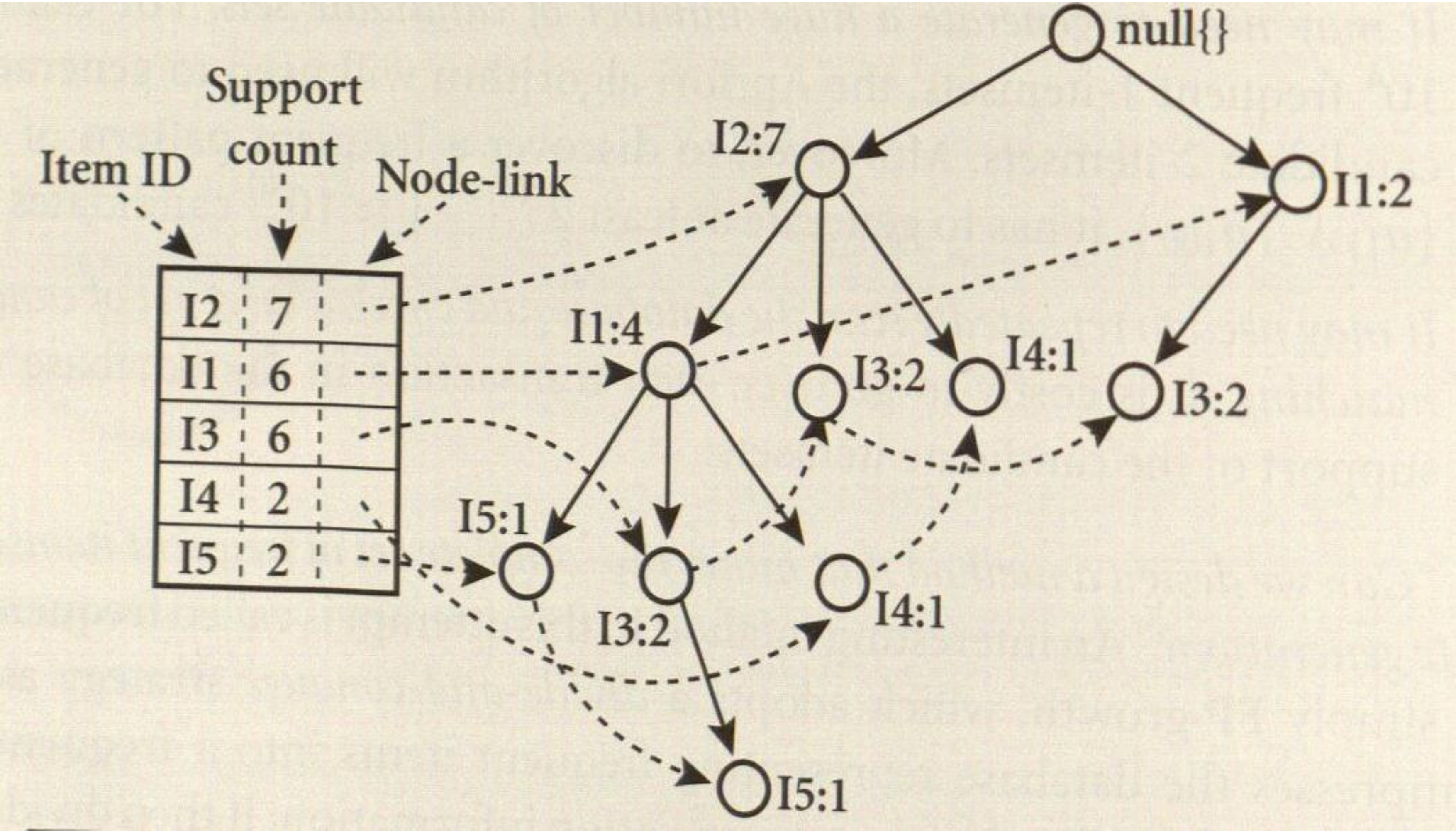
- The first scan of data is the same as Apriori
- Derive the set of frequent 1-itemsets
- Let min-sup=2
- Generate a set of ordered items

Item ID	Support count
I2	7
I1	6
I3	6
I4	2
I5	2

Transactional Database

TID	List of item IDS
T100	I1,I2,I5
T200	I2,I4
T300	I2,I3
T400	I1,I2,I4
T500	I1,I3
T600	I2,I3
T700	I1,I3
T800	I1,I2,I3,I5
T900	I1,I2,I3

Example:



null{}
Support
count
Item ID
Node-link
I2:7
I1:2
I1:4
I3:2
I4:1
I3:2
I2
7
I1
6
I3
6
I4
2
I5
2
I5:1
I3:2
I4:1
I5:1

Item	Conditional Pattern Base	Conditional FP-tree	Frequent Pattern Generated
I5	{{I2, I1:1}, {I2, I1, I3:1}}	(I2:2, I1:2)	{I2, I5:2}, {I1, I5:2}, {I2, I1, I5:2}
I4	{{I2, I1:2}, {I2:1}}	(I2:2)	{I2, I4:2}
I3	{{I2, I1:2}, {I2:2}, {I1:2}}	(I2:4, I1:2), (I1:2)	{I2, I3:4}, {I1, I3:4}, {I2, I1, I3:2}
I1	{{I2:4}}	(I2:4)	{I2, I1:4}

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

min_support = 3

Pattern Count Tree (PC-Tree)

- PC-Tree is a data structure used to store all the patterns occurring in the tuples of a transaction database
- Node structure

item_name	count	child_pointer(c)	sibling_pointer(s)

- Construction of the PC-tree
  - Scan DB only once
  - Preprocess each transaction to put items in lexicographical order
  - For each transaction
    - If subpattern/prefix does not exist
      - Create a new branch
    - Else
      - Increment count

- As the data base is scanned for constructing pc tree, maintain a table with items & count initialised to zero. As transactions are read increment the count of respective items in the transaction
- Use this table to create a list L1 based on decreasing order of frequency.
- Read the pc tree from the root, and insert the transaction in fp-tree; decrement the count in pc tree as the transaction is inserted in fp tree
- First traverse child links, then sibling links
- At the end count of all items become zero in pctree

PC Tree - properties

- PC-tree is a complete representation of the database
- PC-tree is a compact representation of the database
- PC-tree corresponding to a given database is unique