A pattern growth Approach for Mining Frequent Itemsets

Pattern Growth Approach

- Suffer two nontrivial costs:
  - Generation of huge number of candidate sets.
  - Need to repeatedly scan the database and check large set of candidates by pattern matching
- Needs a method that mines the complete set of frequent itemsets without costly candidate generation process
- Frequent pattern growth adopts divide-and-conquer strategy

Frequent Pattern -growth

- Encompasses the database representing frequent itemsets into FP-tree
- FP-tree retains the item association information.
- Divides the compressed db into set of conditional databases
  - Each consists of one frequent item or pattern fragment
  - For each pattern fragment only its associated data sets need to be examined
  - Approach reduce the size of data set to be searched along with growth of patterns being examined.

FP-Growth without candidate generation

- Scan and derive the set of frequent itemsets(1-frequent) and their support counts.
- The frequent itemsets are sorted in descending order of support count
- Resulting set is denoted by L
- L= {{I2:7},{I1:6},{I3:6},{I4:2},
- {I5:2}}

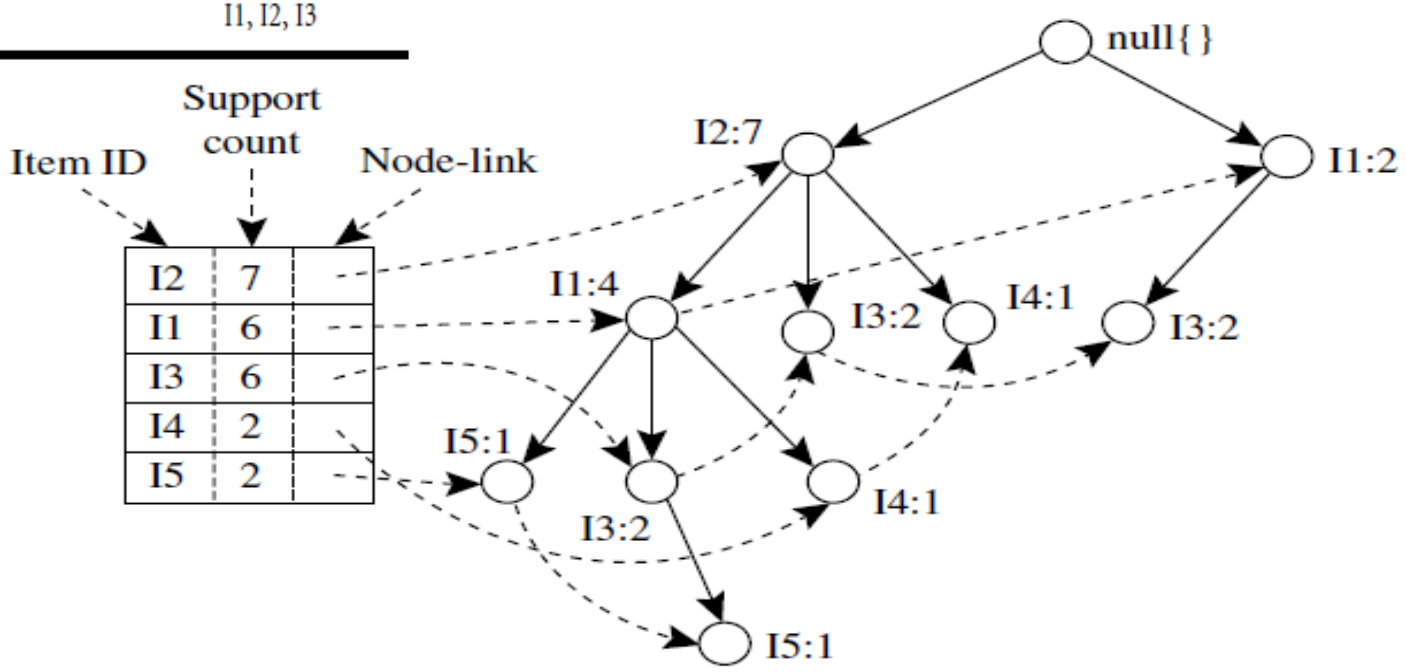
TID	*List of item_IDs*
T100	I1, I2, I5
T200	I2, I4
T300	I2, I3
T400	I1, I2, I4
T500	I1, I3
T600	I2, I3
T700	I1, I3
T800	I1, I2, I3, I5
T900	I1, I2, I3

FP-Growth without candidate generation

- FP tree is constructed as follows:
  - Create the root of the tree, labeled with NULL
  - Process the transactions in L order
  - Create a branch for each transactions
  - The items in the transactions acts a node in the branches
- Eg: T100:I1,I2,I5 & I2,I1,I5 in L order
  - Construct the first branch with three nodes <I2:1>, <I1:1>, <I5:1>,
  - Connect I2 to the root and I1 to I2 and I5 to I1
- The next transactions T200 contains L{I2,I4}, connect I2 to the root and attach l4 to I2
- The transaction T200 shares a common prefix I2 with T100 so increment the count of the node I2 by 1.

FP-Growth without candidate generation

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



FP-Growth without candidate generation

- When branch to be added for a transaction
  - The node of the common prefix is incremented by 1
  - Nodes of the prefix are created and linked accordingly
- To facilitate tree traversal, an item header table is built so that each item points to its occurrences in the tree via a chain of **node-links**

FP-Tree mining

- **Algorithm**
  - Start from each frequent length-1 pattern
  - Construct conditional pattern base (sub database consists set of prefix paths in the FP tree co-occurring with the suffix pattern)
  - Construct conditional FP-tree using minimum support
  - Generate frequent patterns
  - Pattern growth achieved by concatenation of the suffix pattern with frequent patterns generated from conditional FP-tree

Table 6.2 Mining the FP-Tree by Creating Conditional (Sub-)Pattern Bases

Item	*Conditional Pattern Base*	*Conditional FP-tree*	*Frequent Patterns 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: 1}, {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}

FP-Tree mining

- The FP-growth method transforms the problem of finding long frequent patterns into searching for shorter ones in much smaller conditional databases recursively.
- Method reduces the search costs.
- Efficient and scalable for mining both long and short frequent patterns.
- Faster than Apriori algorithm.