Business Intelligence

G. Mudare

Market basket analysis

Market basket analysis

- Given below is a dataset consisting of 10 grocery items, with 25 market baskets.

Basket	Milk	Eggs	Bread	Beer	Water	Cola	Apples	Beans	Peas	Diapers
1				X						X
2	X		X			X	X		X	
3	X	X	X	X			X	X		X
4		X			X		X	X	X	
5	X		X	X		X	X			
6			X			X		X		
7						X	X			
8	X	X	X							X
9	X						X		X	X
10	X		X					X		
11		X		X						
12					X		X		X	
13	X		X					X		
14	X		X				X			
15				X				X		
16					X		X			X
17			X	X						
18	X	X	X	X	X	X	X	X	X	X
19	X		X			X	X	X	X	
20			X			X		X		
21		X		X						
22	X	X	X			X	X	X		X
23	X						X			
24						X	X			
25	X	X	X			X	X			

Market basket analysis

1. Find the support for :
   a. apples and milk
   b. beer and water
2. How many pairs will be generated if the minimum support is 1
3. Which item pairs are below the minimum support of 2,
4. Which item pairs meet the minimum support of 10
5. Which item pairs meet the minimum support of 5
6. If bread was in a market basket, what is the confidence that milk would also be in the basket?
7. If milk was in a market basket, what is the confidence that bread would also be in the basket?
8. If eggs were in a market basket, what is the confidence that apples would also be in the basket?

SOLUTION

1. Sort on the items with the most volume, counts for cross-sales can be generated as in Table
2. The diagonal contains the total number of market baskets containing each item.
3. Most customers who purchased apples also purchased milk, bread, and cola.
4. Of those that purchased beer, few purchased water.
5. One customer purchased everything, so there are no zeros in this matrix.[Basket 18]
6. *Support* is the number of cases for a given pair.

	Apples	Bread	Milk	Cola	Beans	Eggs	Beer	Diapers	Peas	Water
Apples	15	8	10	8	5	5	3	5	6	4
Bread	8	14	11	8	8	5	4	4	3	1
Milk	10	11	13	6	6	5	3	5	4	1
Cola	8	8	6	10	5	3	2	2	3	1
Beans	5	8	6	5	10	4	3	3	3	2
Eggs	5	5	5	3	4	8	4	4	2	2
Beer	3	4	3	2	3	4	8	3	1	1
Diapers	5	4	5	2	3	4	3	7	2	2
Peas	6	3	4	3	3	2	1	2	6	3
Water	4	1	1	1	2	2	1	2	3	4
TOTAL	15	14	13	10	10	8	8	7	6	4

SOLUTION

1. Find the support for :
   a. apples and milk = **[10]**
   b. beer and water =**[1]**
2. How many pairs will be generated if the minimum support is 1 **[all]**
3. Which item sets meet the minimum support of 10 **={Apples-Milk 10 cases; Bread-Milk 11 cases}.**
4. If bread was in a market basket, what is the confidence that milk would also be in the basket? **11/14, or 0.786.**
5. If milk was in a market basket, what is the confidence that bread would also be in the basket? 11/13, or 0.846.

Improving the Efficiency of Apriori

BELGIUM CAMPUS iTversity
It's the way we're wired

- Hash-based technique (hashing itemsets into corresponding buckets):
- Transaction reduction (reducing the number of transactions scanned in future iterations):
- Partitioning (partitioning the data to find candidate itemsets):
- Sampling (mining on a subset of the given data)
- Dynamic itemset counting (adding candidate itemsets at different points during a scan):

Hash-based technique

- A hash-based technique can be used to reduce the size of the candidate *k*-itemsets, *Ck*, for $k > 1$.
- For example, when scanning each transaction in the database to generate the frequent 1-itemsets, *L*1, from the candidate 1-itemsets in *C*1, we can generate all of the 2-itemsets for each transaction, hash (i.e., map) them into the different *buckets* of a *hash table* structure, and increase the corresponding bucket counts
- A 2-itemset whose corresponding bucket count in the hash table is below the support threshold cannot be frequent and thus should be removed from the candidate set

Hash-based technique

Create hash table H_2 using hash function $h(x, y) = ((order\ of\ x) \times 10 + (order\ of\ y))\ mod\ 7$

H_2

bucket address	0	1	2	3	4	5	6
bucket count	2	2	4	2	2	4	4
bucket contents	{I1, I4} {I3, I5}	{I1, I5} {I1, I5}	{I2, I3} {I2, I3} {I2, I3} {I2, I3}	{I2, I4} {I2, I4}	{I2, I5} {I2, I5}	{I1, I2} {I1, I2} {I1, I2} {I1, I2}	{I1, I3} {I1, I3} {I1, I3} {I1, I3}

Hash table, *H*2, for candidate 2-itemsets:

This hash table was generated by scanning the transactions of thae dbTable while determining *L*1 from *C*1.

If the minimum support count is, say, 3, then the itemsets in buckets 0, 1, 3, and 4 cannot be frequent and so they should not be included in *C*2.

Transaction reduction

- Reducing the number of transactions scanned in future iterations):
- A transaction that does not contain any frequent k-itemsets cannot contain any frequent (k+1)-itemsets.
- Such a transaction can be marked or removed from further consideration because subsequent scans of the database for j-itemsets, where $j > k$, will not require it.

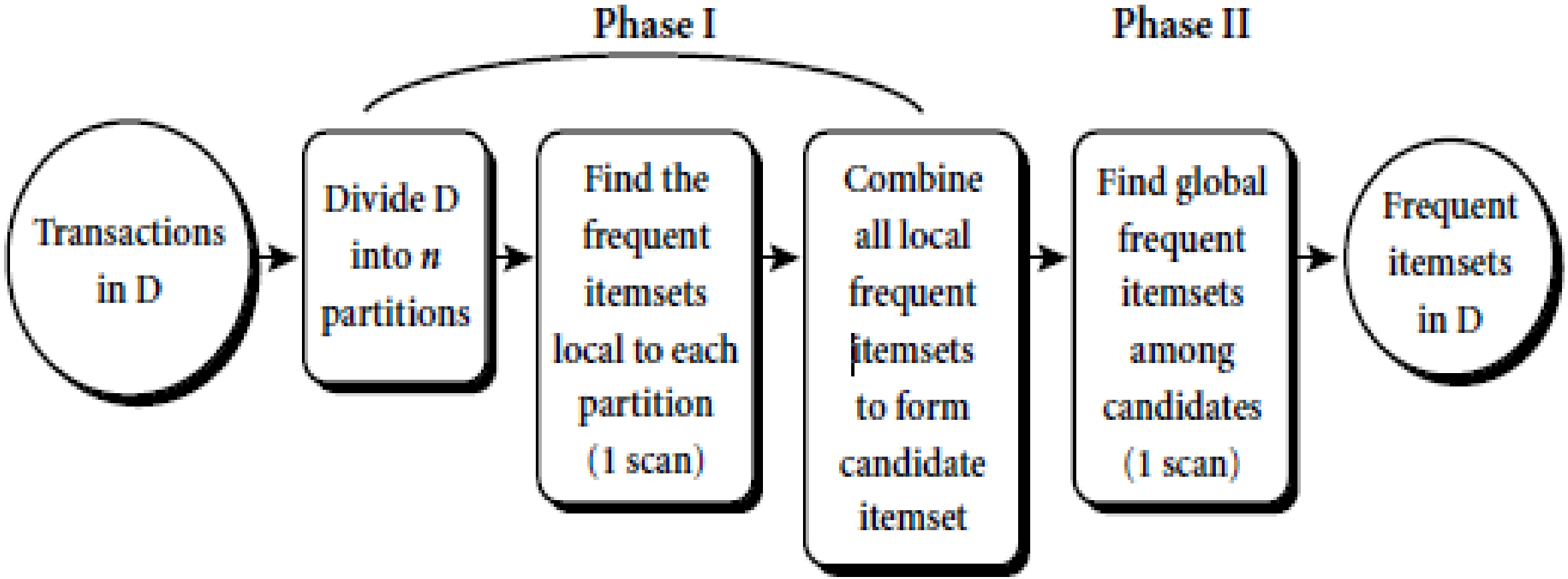
Partitioning

- Partitioning the data to find candidate itemsets):
- A partitioning technique can be used that requires just two database scans to mine the
- It consists of two phases.
- In Phase I, the algorithm subdivides the transactions of D into n nonoverlapping partitions. If the minimum support threshold for transactions in D is $min\ sup$, then the minimum support count for a partition is $min\ sup$ *the number of transactions in that partition*.
- For each partition, all frequent itemsets (local frequent itemsets) within the partition are found.
- The procedure employs a special data structure that, for each itemset, records the TIDs of the transactions containing the items in the itemset.
- A local frequent itemset may or may not be frequent with respect to the entire database, *D*. ***Any itemset that is potentially frequent with respect to D must occur as a frequent itemset in at least one of the partitions.***
- Therefore, all local frequent itemsets are candidate itemsets with respect to *D*.
- The collection of frequent itemsets from all partitions forms the global candidate itemsets with respect to D.

Partitioning

- In Phase II, a second scan of D is conducted in which the actual support of each candidate is assessed in order to determine the global frequent itemsets.
- Partition size and the number of partitions are set so that each partition can fit into main memory and therefore be read only once in each phase.

Partitioning



Sampling

- Mining on a subset of the given data: The basic idea of the sampling approach is to pick a random sample S of the given data D, and then search for frequent itemsets in S instead of D.
- In this way, we trade off some degree of accuracy against efficiency.
- The sample size of *S* is such that the search for frequent itemsets in *S* can be done in main memory, and so only one scan of the transactions in *S* is required overall.

Counting

- Adding candidate itemsets at different points during a scan the database is partitioned into blocks marked by start points.
- In this variation, new candidate itemsets can be added at any start point, unlike in Apriori, which determines new candidate itemsets only immediately before each complete database scan.
- The technique is dynamic in that it estimates the support of all of the itemsets that have been counted so far, adding new candidate itemsets if all of their subsets are estimated to be frequent.
- The resulting algorithm requires fewer database scans than Apriori.

Business Intelligence

G. Mudare

Mining Frequent Itemsets without Candidate Generation

Mining Frequent Itemsets without Candidate Generation

- The Apriori candidate generate-and-test method significantly reduces the size of candidate sets, leading to good performance gain
- Apriori suffer from two costs:
  1. *It may need to generate a huge number of candidate sets.*
  2. *It may need to repeatedly scan the database and check a large set of candidates by pattern matching.*
- An interesting method called frequent-pattern growth, adopts a *divide-and-conquer* strategy

Frequent-pattern growth (FP-Growth

1. **FP-Growth** Compresses the database representing frequent items into a frequent-pattern tree, or FP-tree,which retains the itemset association information.
2. It then divides the compressed database into a set of *conditional databases* (a special kind of projected database), each associated with one frequent item or **"pattern fragment**,"
3. It then mines each sub database separately.

Frequent-pattern growth (FP-Growth)

Example

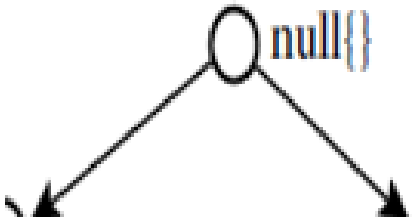
TID	List of item_ID
T100	I1, I2, I5
T200	I2, I4
T300	I2, I3
T400	I1, I2, I4
T500	I1, I3
T600	I2, I3
T700	I1, I3
T800	I1, I2, I3, I5
T900	I1, I2, I3

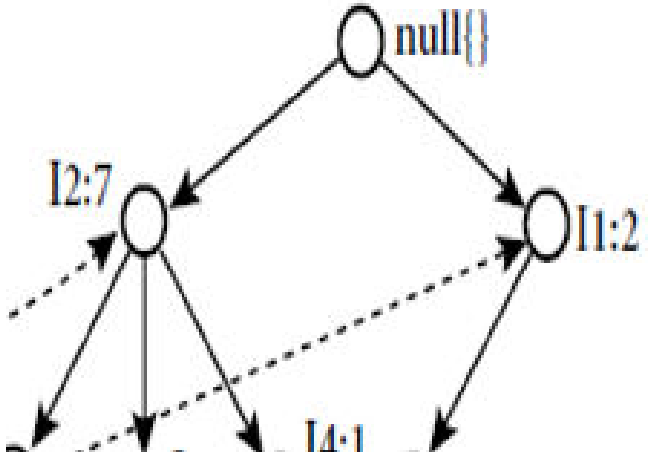
The first scan of the database is the same as Apriori, which derives the set of frequent items (1-itemsets) and their support counts (frequencies).

Itemset	Sup_count
I2	7
I1,	6
I3	6
I4	2
I5	2

Construct an FP-tree

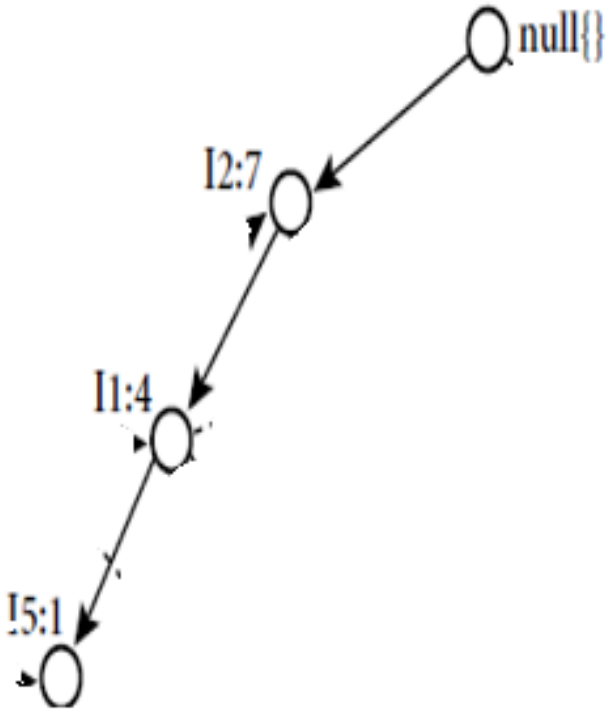
1. Create the root of the tree, labelled with “null.”
2. Scan database *D* a second time
3. The items in each transaction are processed in *L* order (i.e., sorted according to descending support count), and a branch is created for each transaction.





Construct an FP-tree

- The scan of the first transaction, "T100: I_1, I_2, I_5," which contains three items (I_2, I_1, I_5 in L order), leads to the construction of the first branch of the tree with three nodes, I_2: 1, I_1:1, and I_5: 1,



Construct an FP-tree

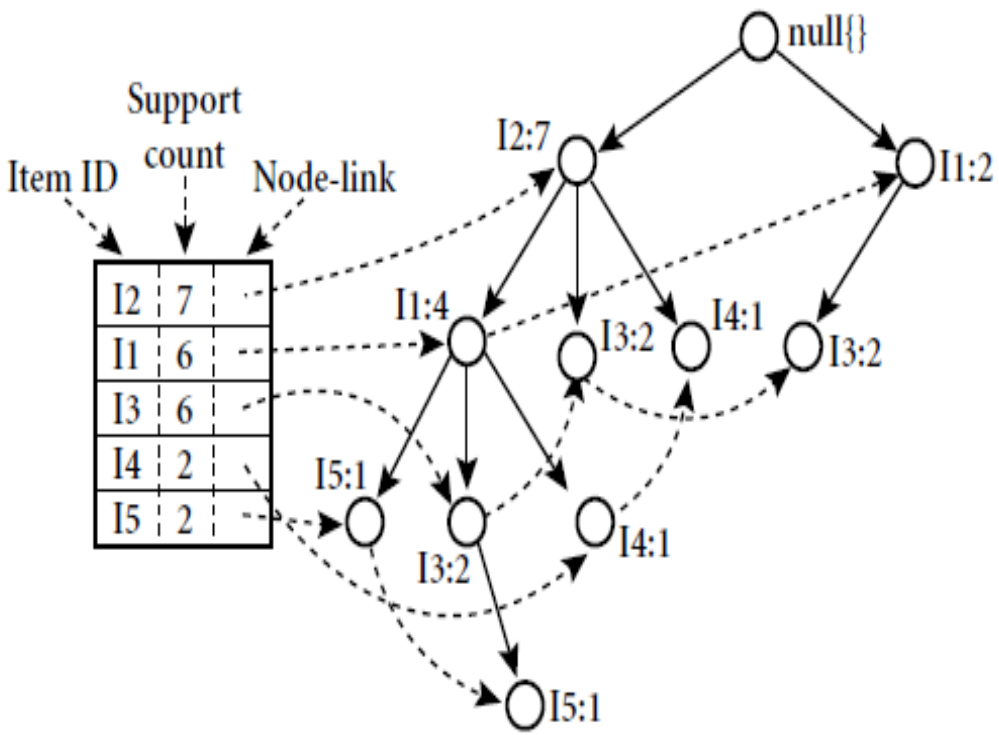
- The scan of the first transaction, "T100: I_1, I_2, I_5," which contains three items (I_2, I_1, I_5 in L order), leads to the construction of the first branch of the tree with three nodes, I_2: 1, I_1:1, and I_5: 1, where I_2 is linked as a child of the root, I_1 is linked to I_2, and I_5 is linked to I_1.

The second transaction, T200, contains the items **I2** and **I4** in L order, which would result in a branch where **I2** is linked to the root and **I4** is linked to **I2**.

This branch would share a common prefix, **I2**, with the existing path for T100.

Increment the count of the I2 node by 1, and create a new node, I4: 1,which is linked as a child of I2: 2.

When considering the branch to be added for a transaction, the count of each node along a common prefix is incremented by 1, and nodes for the items following 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.
With the associated node-links the problem of mining frequent patterns in databases is transformed to that of mining the FP-tree.

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}

Start from each frequent length-1 pattern (as an initial suffix pattern), construct its conditional pattern base (a "subdatabase,"which consists of the set of *prefix paths* in the FP-tree co-occurring with the suffix pattern), then construct its (*conditional*) FP-tree, and perform mining recursively on such a tree.

The pattern growth is achieved by the concatenation of the suffix pattern with the frequent patterns generated from a conditional FP-tree.

Mining the FP-tree by creating conditional (sub-)pattern bases.

First consider I5, which is the last itemin L, rather than the first.

I5 occurs in two branches of the FP-tree (The occurrences of I5 can easily be found by following its chain of node-links.)

The paths formed by these branches are I2, I1, I5: 1 and I2, I1, I3, I5: 1.

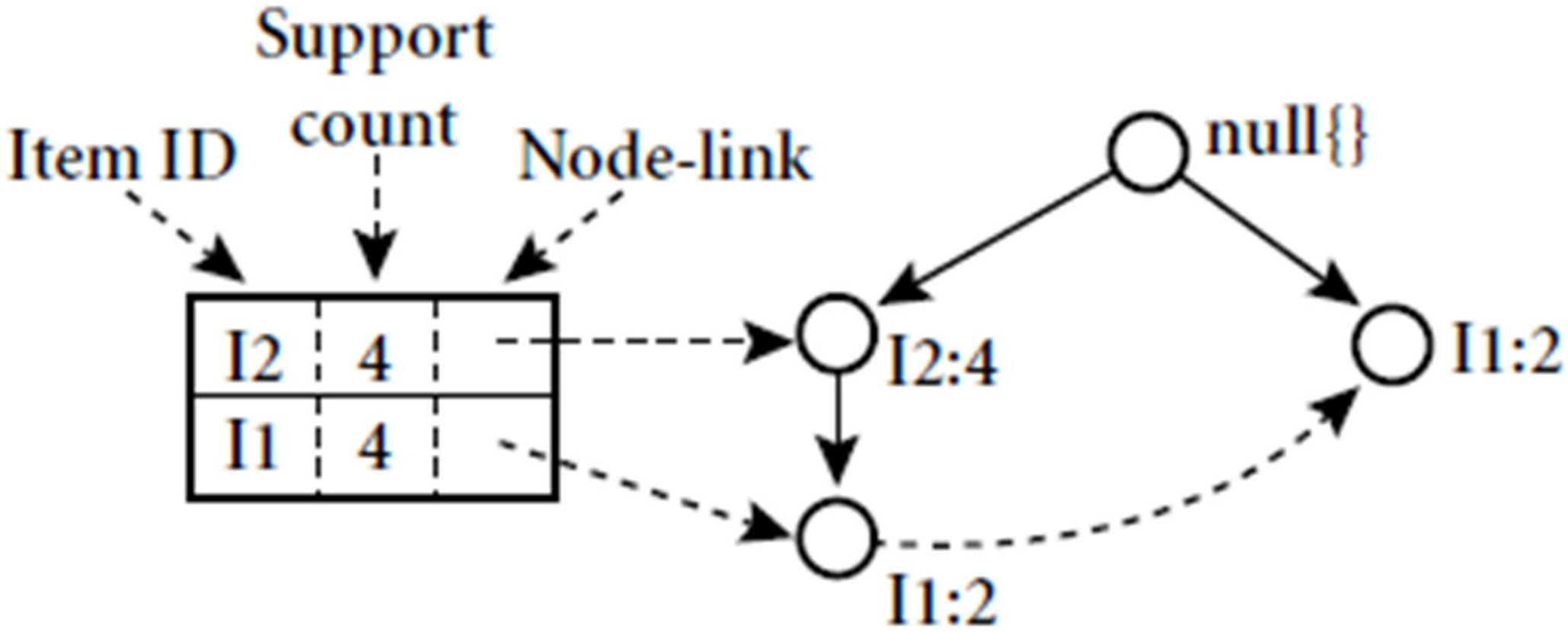
Therefore, considering I5 as a suffix, its corresponding two prefix paths are I2, I1: 1 and I2, I1, I3: 1, which form its conditional pattern base.

Its conditional FP-tree contains only a single path, I2: 2, I1: 2; I3 is not included because its support count is less than the minimum support count.

The single path generates all tof 1he combinations of frequent patterns: I2, I5: 2, I1, I5: 2, I2, I1, I5: 2.

For I4, its two prefix paths form the conditional pattern base, I2, I1: 1, I2: 1, which generates a single-node conditional FP-tree, I2: 2, and derives one frequent

The conditional FP-tree associated with the conditional node I_3.



The conditional FP-tree associated with the conditional node I_3.

- The FP-growth method transforms the problem of finding long frequent patterns to searching for shorter ones recursively and then concatenating the suffix.
- It uses the least frequent items as a suffix, offering good selectivity. The method substantially reduces the search costs.
- When the database is large, it is sometimes unrealistic to construct a main memory based FP-tree.
- An interesting alternative is to first partition the database into a set of projected databases, and then construct an FP-tree and mine it in each projected database.
- Such a process can be recursively applied to any projected database if its FP-tree still cannot fit in main memory.

Mining Frequent Itemsets Using Vertical Data Format

- Both the Apriori and FP-growth methods mine frequent patterns from a set of transactions in *TID-itemset* format (that is, $TID : itemset$), where *TID* is a transaction-id and *itemset* is the set of items bought in transaction *TI*
- This data format is known as horizontal data format
- Data can also be presented in *item-TID set* format (that is, $item : TID\ set$g), where *item* is an item name, and *TID set* is the set of transaction identifiers containing the item.
- This format is known as vertical data format.

Vertical data format of the transaction data set D

itemset	*TID_set*
I1	{T100, T400, T500, T700, T800, T900}
I2	{T100, T200, T300, T400, T600, T800, T900}
I3	{T300, T500, T600, T700, T800, T900}
I4	{T200, T400}
I5	{T100, T800}

Mining Frequent Itemsets Using Vertical Data Format

Mining can be performed on this data set by intersecting the TID sets of every pair of frequent single items.

Minimum support count is 2

Every single item is frequent in the Table,

10 intersections are performed

in total,which lead to 8 nonempty 2-itemsets

itemset	*TID_set*
{I1, I2}	{T100, T400, T800, T900}
{I1, I3}	{T500, T700, T800, T900}
{I1, I4}	{T400}
{I1, I5}	{T100, T800}
{I2, I3}	{T300, T600, T800, T900}
{I2, I4}	{T200, T400}
{I2, I5}	{T100, T800}
{I3, I5}	{T800}

Mining Frequent Itemsets Using Vertical Data Format

- Based on the Apriori property, a given 3-itemset is a candidate 3-itemset only if every one of its 2-itemset subsets is frequent.
- The candidate generation process here will generate only two 3-itemsets:

itemset	*TID_set*
{I1, I2, I3}	{T800, T900}
{I1, I2, I5}	{T100, T800}