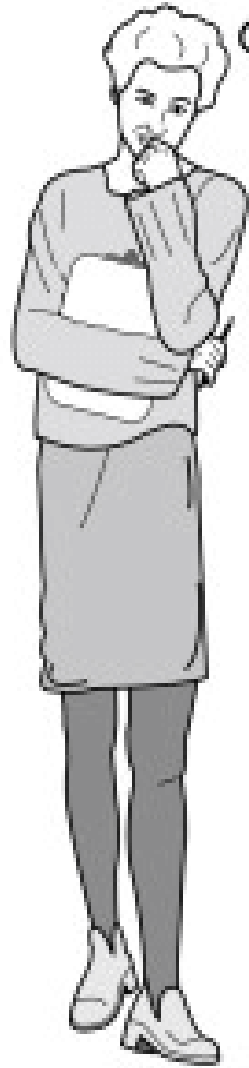


MODULE 4

Association Rule Analysis

Association Rules-Introduction

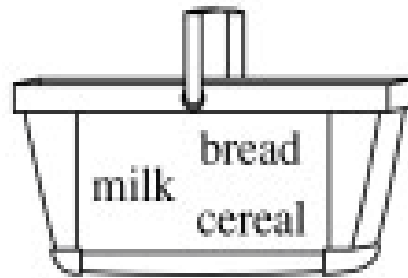
- Association rule mining finds interesting associations and relationships among large sets of data items.
- This rule shows how frequently a itemset occurs in a transaction.
- A typical example is a Market Based Analysis.
- It allows retailers to identify relationships between the items that people buy together frequently.



Market Analyst

Which items are frequently purchased together by customers?

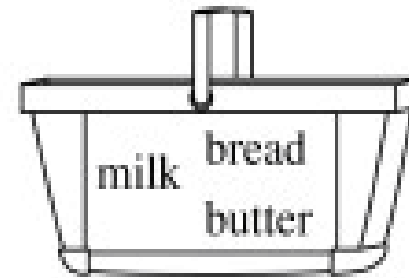
Shopping Baskets



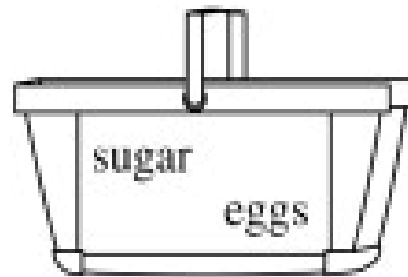
Customer 1



Customer 2



Customer 3



Customer n

- **Association Rule** – An implication expression of the form $X \rightarrow Y$, where X and Y are any 2 itemsets.

- Measures for Rule Creation

1. The **support** for $X \rightarrow Y$ is the probability of both X and Y appearing together, that is $P(X \cup Y)$.

$$\text{support}(A \cup B) = P(A \cup B)$$

1. The **confidence** of $X \rightarrow Y$ is the conditional probability of Y appearing given that X exists. It is written as $P(Y|X)$ and read as P of Y given X .

$$\text{confidence}(A \Rightarrow B) = P(B|A) = \frac{\text{support_count}(A \cup B)}{\text{support_count}(A)}.$$

Example-Find support value of each itemset

Transaction ID	Items Purchased
1	Bread, Cheese, Egg, Juice
2	Bread, Cheese, Juice
3	Bread, Milk, Yogurt
4	Bread, Juice, Milk
5	Cheese, Juice, Milk

Support (item) = Frequency of item/Number of transactions

Answer:

Item	Frequency	Support (in %)
Bread	4	$4/5=80\%$
Cheese	3	$3/5=60\%$
Egg	1	$1/5=20\%$
Juice	4	$4/5=80\%$
Milk	3	$3/5=60\%$
Yogurt	1	$1/5=20\%$

Methods to discover Association rules

In general, association rule mining can be viewed as a two-step process:

1. Find all frequent itemset: By definition, each of these itemset will occur at least as frequently as a predetermined minimum support count
2. Generate strong association rules from the frequent itemset: By definition, these rules must satisfy minimum support and minimum confidence.

Frequent Item Set

- Let T be the transaction database and σ be the user-specified minimum support σ .
- An itemset $X \in A$ is said to be a frequent itemset in T with respect to σ , if

$$\text{Support}(X) \geq \sigma$$

- **Downward Closure Property**: Any subset of a frequent set is a frequent set.
- **Upward Closure Property**: Any superset of an infrequent set is an infrequent set.

Maximal Frequent Set

- A frequent set is a maximal frequent set if it is a frequent set and no superset of this is a frequent set.

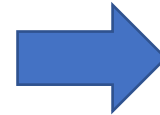
Border Set

An itemset is a border set if it is not a frequent set, but all its proper subsets are frequent sets

(A1, A2, A3, A4, A5, A6, A7, A8, A9). Assume support= 20%.

Sample Database

A1	A2	A3	A4	A5	A6	A7	A8	A9
1	0	0	0	1	1	0	1	0
0	1	0	1	0	0	0	1	0
0	0	0	1	1	0	1	0	0
0	1	1	0	0	0	0	0	0
0	0	0	0	1	1	1	0	0
0	1	1	1	0	0	0	0	0
0	1	0	0	0	1	1	0	1
0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	1	0
0	0	1	0	1	0	1	0	0
0	0	1	0	1	0	1	0	0
0	0	0	0	1	1	0	1	0
0	1	0	1	0	1	1	0	0
1	0	1	0	1	0	1	0	0
0	1	1	0	0	0	0	0	1



Frequent Count for Some Itemsets

X	SUPPORT COUNT
{1}	2
{2}	6
{3}	6
{4}	4
{5}	8
{6}	5
{7}	7
{8}	4
{9}	2
{5, 6}	3
{5, 7}	5
{6, 7}	3
{5, 6, 7}	1

- $\{1\}$ – Not a frequent set
- $\{3\}$ - is a frequent set
- $\{5, 6, 7\}$ - is a border set
- $\{5, 6\}$ - is a maximal frequent set
- $\{2, 4\}$ - is also a maximal frequent set
- But there is no border set having $\{2, 4\}$ as a proper subset

APRIORI ALGORITHM

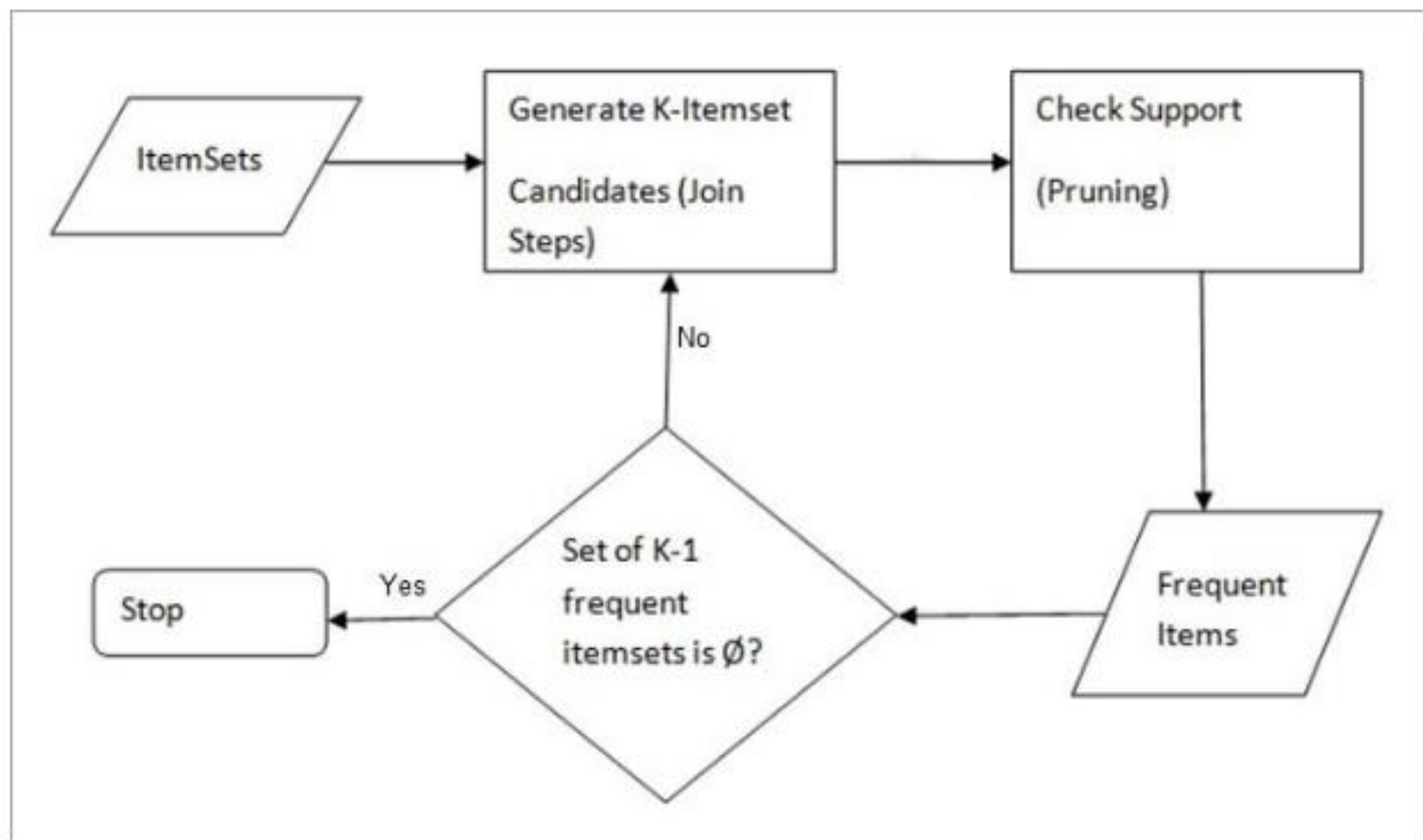
- It is also called the level-wise algorithm
- Apriori algorithm may be simply described by a two step approach:
- Step 1 – discover all frequent (single) items that have support above the minimum support required
- Step 2 – use the set of frequent items to generate the association rules that have high enough confidence level
- The candidate generation process and the pruning process are the most important parts of this algorithm

- The candidate-generation method

- Given L_{k-1} , the set of all frequent (k-1)-itemset, we want to generate a superset of the set of all frequent k-itemset.
- Using this algorithm: from $L3 := \{ \{1, 2, 3\}, \{1, 2, 5\}, \{1, 3, 5\}, \{2, 3, 5\}, \{2, 3, 4\} \}$.
 $C4 := \{ \{1, 2, 3, 5\}, \{2, 3, 4, 5\} \}$
- $\{1, 2, 3, 5\}$ is generated from $\{1, 2, 3\}$ and $\{1, 2, 5\}$.
- Similarly, $\{2, 3, 4, 5\}$ is generated from $\{2, 3, 4\}$ and $\{2, 3, 5\}$.

- Pruning

- The pruning step eliminates the extensions of (k-1)-itemset which are not found to be frequent, from being considered for counting support.
- For example, from $C4$, the itemset $\{2, 3, 4, 5\}$ is pruned, since all its 3-subsets are not in $L3$



The Apriori Algorithm

- Pseudo-code:

C_k : Candidate itemset of size k

L_k : frequent itemset of size k

$L_1 = \{\text{frequent items}\};$

for ($k = 1; L_k \neq \emptyset; k++$) **do begin**

C_{k+1} = candidates generated from L_k ;

for each transaction t in database **do**

increment the count of all candidates in C_{k+1}
that are contained in t

L_{k+1} = candidates in C_{k+1} with min_support

end

return $\cup_k L_k$;

Example

Find association rules with minimum support of 50% and minimum confidence of 75%.

Transaction ID	Items
100	Bread, Cheese, Eggs, Juice
200	Bread, Cheese, Juice
300	Bread, Milk, Yogurt
400	Bread, Juice, Milk
500	Cheese, Juice, Milk

Example

First find L1. 50% support requires that each frequent item appear in at least three transactions. Therefore L1 is given by:

Item	Frequency
Bread	4
Cheese	3
Juice	4
Milk	3

Example

The candidate 2-itemsets or C2 therefore has six pairs (why?). These pairs and their frequencies are:

Item Pairs	Frequency
(Bread, Cheese)	2
(Bread, Juice)	3
(Bread, Milk)	2
(Cheese, Juice)	3
(Cheese, Milk)	1
(Juice, Milk)	2

Deriving Rules

L2 has only two frequent item pairs {Bread, Juice} and {Cheese, Juice}. After these two frequent pairs, there are no candidate 3-itemsets

Apriori property that all subsets of a frequent itemset must also be frequent

The two frequent pairs lead to the following possible rules:

Bread \rightarrow Juice

Juice \rightarrow Bread

Cheese \rightarrow Juice

Juice \rightarrow Cheese

- The confidence of these rules is obtained by dividing the support for both items in the rule by the support of the item on the left hand side of the rule.
- The confidence of the four rules therefore are

(Bread \rightarrow Juice) ----- $3/4 = 75\%$

$$\text{confidence}(\text{Bread} \rightarrow \text{Juice}) = \frac{\text{support}(\text{Bread} \rightarrow \text{Juice})}{\text{support}(\text{Bread})} = \frac{3}{4}$$

(Juice \rightarrow Bread) ----- $3/4 = 75\%$

(Cheese \rightarrow Juice) ----- $3/3 = 100\%$

(Juice \rightarrow Cheese) ----- $3/4 = 75\%$

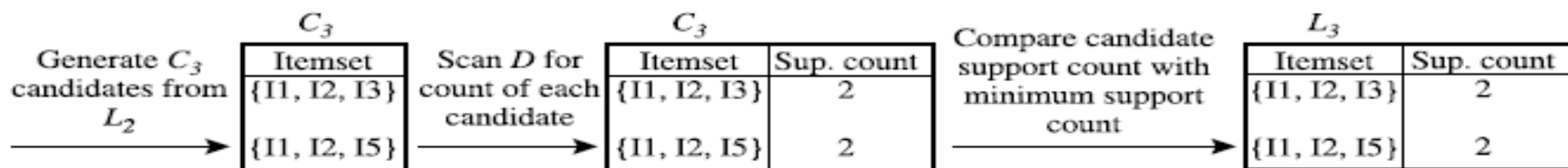
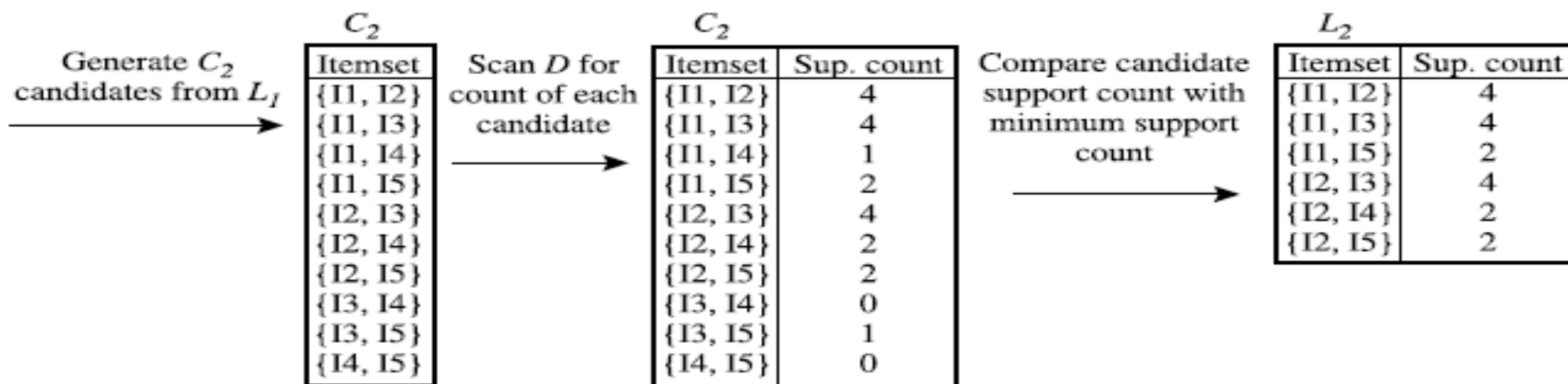
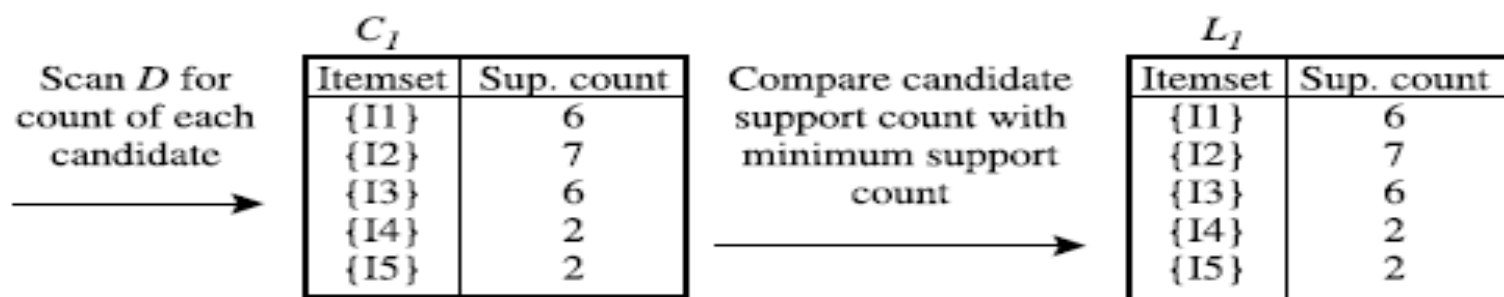
- Since all of them have a minimum 75% confidence, they all qualify.

Problem 2

- Find Association rules from the following data using Apriori algorithm with minimum support count required is 2 and confidence 70%

<i>TID</i>	<i>List of item_IDs</i>
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

Solution



$I1 \wedge I2 \Rightarrow I5,$	$confidence = 2/4 = 50\%$
$I1 \wedge I5 \Rightarrow I2,$	$confidence = 2/2 = 100\%$
$I2 \wedge I5 \Rightarrow I1,$	$confidence = 2/2 = 100\%$
$I1 \Rightarrow I2 \wedge I5,$	$confidence = 2/6 = 33\%$
$I2 \Rightarrow I1 \wedge I5,$	$confidence = 2/7 = 29\%$
$I5 \Rightarrow I1 \wedge I2,$	$confidence = 2/2 = 100\%$

If the minimum confidence threshold is, say, 70%, then only the second, third, and last rules above are output, because these are the only ones generated that are strong.

Partition Algorithm

- Apriori scans the database (set of transactions) several times, in order to compute the supports of candidate frequent k-itemset.
- The Partition Algorithm scans the database only twice.
 - ◁ First scan: generate a set of all potentially large itemset
 - ◁ Second scan: set up counters for each potentially large itemset and compute their actual supports
- During the first scan, a superset of the actual large itemsets is generated. (i.e. false positives may be generated, but no false negatives are generated)

- The Partition Algorithm executes in two phases:
 - ◁ Phase I: the algorithm logically divides the database into a number of non-overlapping partitions. The partitions are considered one at a time and all large itemsets for that partition are generated. At the end of phase I, these large itemsets are merged to generate a set of all potentially large itemsets.
 - ◁ Phase II: the actual supports for these itemsets are generated and the large itemsets are identified.

The partition sizes are chosen such that each partition can be accommodated in the main memory so that the partitions are read only once in each phase.

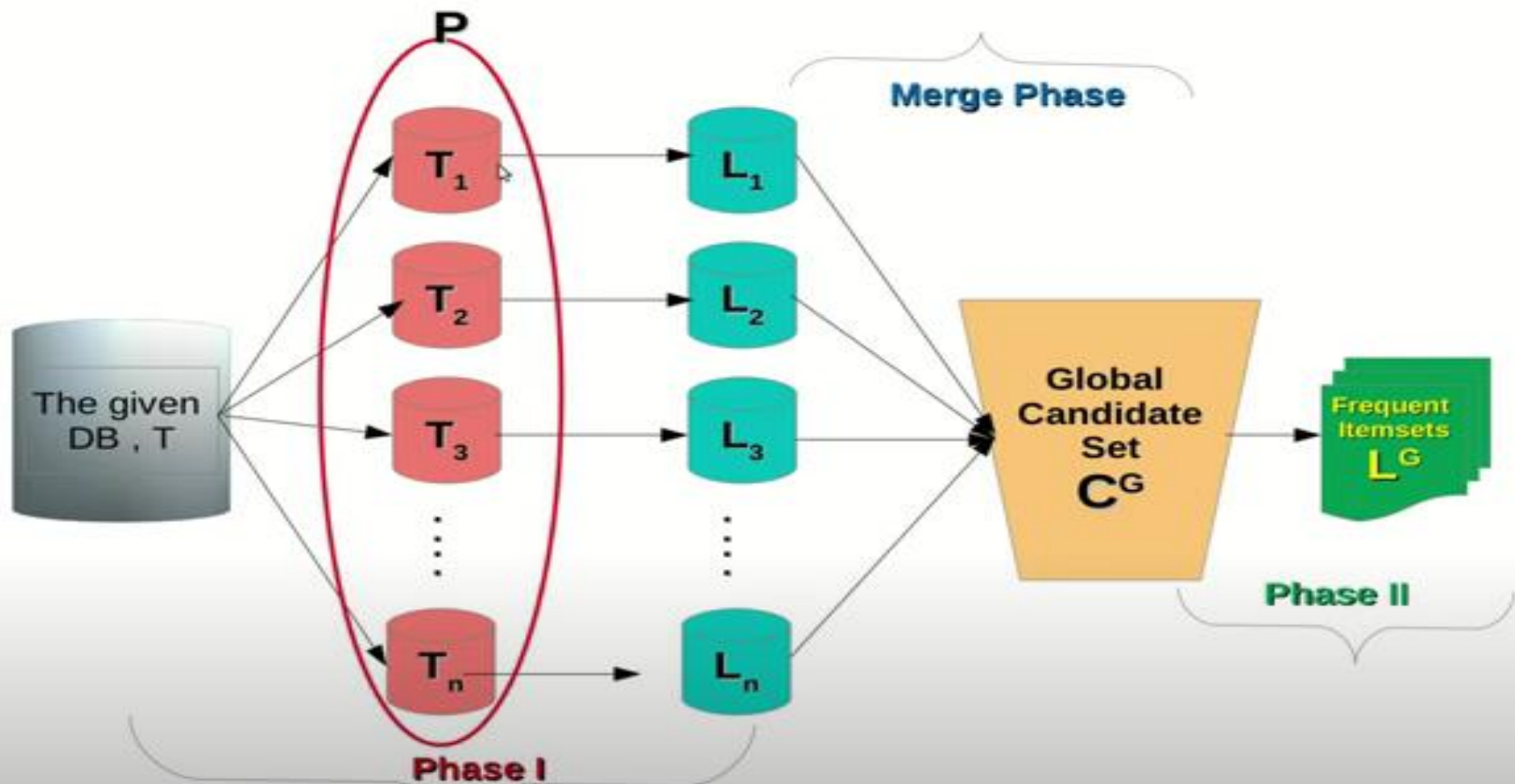
Algorithm

C_k^p	Set of local candidate k -itemsets in partition p
L_k^p	Set of local large k -itemsets in partition p
L^p	Set of all local large itemsets in partition p
C_k^G	Set of global candidate k -itemsets
C^G	Set of all global candidate itemsets
L_k^G	Set of global large k -itemsets

Algorithm

```
1)  $P = \text{partition\_database}(\mathcal{D})$ 
2)  $n = \text{Number of partitions}$ 
3) for  $i = 1$  to  $n$  begin // Phase I
4)    $\text{read\_in\_partition}(p_i \in P)$ 
5)    $L^i = \text{gen\_large\_itemsets}(p_i)$ 
6) end
7) for ( $i = 2$ ;  $L_i^j \neq \emptyset$ ,  $j = 1, 2, \dots, n$ ;  $i++$ ) do
8)    $C_i^G = \cup_{j=1,2,\dots,n} L_i^j$  // Merge Phase
10) for  $i = 1$  to  $n$  begin // Phase II
11)    $\text{read\_in\_partition}(p_i \in P)$ 
12)   for all candidates  $c \in C^G$   $\text{gen\_count}(c, p_i)$ 
13) end
14)  $L^G = \{c \in C^G | c.\text{count} \geq \text{minSup}\}$ 
```

Figure 1: Partition Algorithm



Problem

Transaction	Itemset
T1	I1,I5
T2	I2,I4
T3	I4,I5
T4	I2,I3
T5	I5
T6	I2,I3,I4

Solution

Trans.	Itemset	First Scan	Second Scan	Shortlisted
		Support = 20% Min. sup = 1	Support = 20% Min. sup = 2	
T1	I1, I5	I1-1, I2-1, I4-1, I5-1	I1-1 , I2-3	I2-3, I3-2
T2	I2, I4	{I1, I5}-1 {I2, I4}-1	I3-2, I4-3	I4-3, I5-3
T3	I4, I5	I2-1, I3-1, I4-1, I5-1	I5-3, {I1, I5}-1	{I2, I4}-2
T4	I2, I3	{I4, I5}-1, {I2, I3}-1	{I2, I4}-2, {I4, I5}-1	{I2, I3}-2
T5	I5	I2-1, I3-1, I4-1, I5-1	{I2, I3}-2, {I3, I4}-1	
T6	I2, I3, I4	{I2, I3}-1, {I2, I4}-1 {I3, I4}-1 {I2, I3, I4}-1	{I2, I3, I4}-1	

Pincer Search Algorithm

- Apriori algorithm operates in a bottom – up, breadth – first search method.
- The computation starts from the smallest set of frequent item sets and moves upward till it reaches the largest frequent item set
- The number of database passes is equal to the largest size of the frequent item set.
- As a result, the performance decreases.

- pincer – search algorithm is based on bi – directional search, which takes advantages of both the bottom – up as well as the top – down process.
- It attempts to find the frequent item sets in a bottom – up manner but, at the same time, it maintains a list of maximal frequent item sets.

- In this algorithm, in each pass, in addition to counting the supports of the candidate in the bottom – up direction, it also counts the supports of the item sets of some item sets using a top – down approach.
- These are called the Maximal Frequent Candidate Set (MFCS).
- This process helps in pruning the candidate sets very early on in the algorithm.

- MFCS is initialized to contain one itemset, which contains all of the database items.
- MFCS is updated whenever new infrequent itemsets are found
- **Method:**

$$L_0 = \emptyset ; k=1; C_1 = \{ \{i\} \mid i \in I \}; S_0 = \emptyset$$

$$\text{MFCS} = \{ \{1, 2, \dots, n\} \}; \text{MFS} = \emptyset ;$$

do until $C_k = \emptyset$ and $S_{k-1} = \emptyset$

 read the database and count support for C_k & MFCS.

$$\text{MFS} = \text{MFS} \cup \{ \text{frequent itemsets in MFCS} \};$$

$$S_k = \{ \text{infrequent itemsets in } C_k \};$$

call MFCS_gen algorithm if $S_k \neq \emptyset$;

call MFS_pruning procedure;

generate candidates C_{k+1} from C_k ;

if any frequent itemset in C_k is removed from MFS_pruning procedure

call recovery procedure to recover candidates to C_{k+1} .

call MFCS_prune procedure to prune candidates in C_{k+1} .

$k=k+1$;

return MFS

MFCs_gen

for all itemsets $s \in S_k$

for all itemsets $m \in \text{MFCs}$

if s is a subset of m

$\text{MFCs} = \text{MFCs} \setminus \{m\}$

for all items $e \in \text{itemset } s$

if $m \setminus \{e\}$ is not a subset of any itemset in MFCs

$\text{MFCs} = \text{MFCs} \cup \{m \setminus \{e\}\}$

return MFCs

Recovery

for all itemsets $l \in L_k$

for all itemsets $m \in MFS$

if the first $k-1$ items in l are also in m

for i from $j+1$ to $|m|$

$$C_{k+1} = C_{k+1} \cup \{ \{l.item_1, \dots, l.item_k, m.item_i\} \}$$

MFS_prune

for all itemsets c in L_k

if c is a subset of any itemset in the current MFS

delete c from L_k .

MFCS_prune

for all itemsets in C_{k+1}

if c is not a subset of any itemset in the current MFCS

delete c from C_{k+1} ;

Problem

<i>TID</i>	<i>List of item_IDs</i>
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

$C1 = \{\{I1\}, \{I2\}, \{I3\}, \{I4\}, \{I5\}\}$

$MFCS = \{I1, I2, I3, I4, I5\}$

$MFS = \emptyset ;$

→ superset

Pass 1: Database is read to count support as follows:

ITEM	COUNT
I2	7
I1	6
I3	6
I4	2
I5	2

- $\{I1, I2, I3, I4, I5\} - 0$
- So MFCS = $\{I1, I2, I3, I4, I5\}$ & MFS = \emptyset ;
- $L1 = \{\{I1\}, \{I2\}, \{I3\}, \{I4\}, \{I5\}\} \rightarrow \geq \underline{2} \rightarrow L_1 \rightarrow \text{Items with sup-count}$
- $S1 = \emptyset$, we don't need to update MFCS
- $C2 =$

$S1 \rightarrow \text{Infrequent Items}$

C_2

Itemset
$\{I1, I2\}$
$\{I1, I3\}$
$\{I1, I4\}$
$\{I1, I5\}$
$\{I2, I3\}$
$\{I2, I4\}$
$\{I2, I5\}$
$\{I3, I4\}$
$\{I3, I5\}$
$\{I4, I5\}$

- Pass 2: Read database to count support of elements in C2 & MFCS as given below:

Itemset	Sup. count
{I1, I2}	4 ✓
{I1, I3}	4 ✓
{I1, I4}	1
{I1, I5}	2 ✓
{I2, I3}	4 ✓
{I2, I4}	2 ✓
{I2, I5}	2 ✓
{I3, I4}	0
{I3, I5}	1
{I4, I5}	0

c₂ - →
mfcs ↓

Infrequent

- COUNT(MFCS)=0

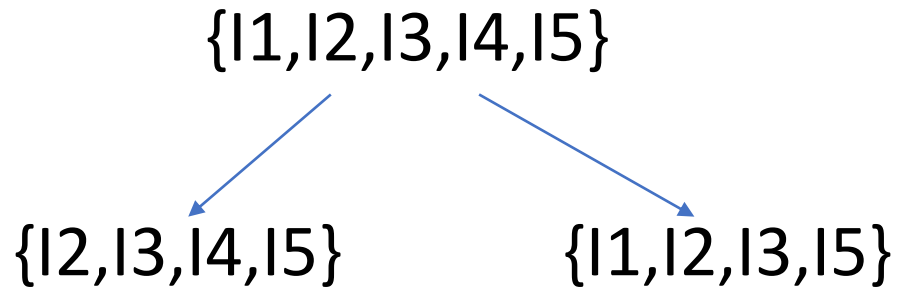
- L2= { {I1,I4},{I1,I3},{I1,I5},
 {I2,I3},{I2,I4},{I2,I5}}

- S2= { {I1,I4}, {I3, I4}, {I3, I5}, {I4,I5}}

S2 not empty , update MFCS

- MFCS = {I1, I2, I3, I4, I5}
- S2 = { {I1, I4}, {I3, I4}, {I3, I5}, {I4, I5} }

Delete {I1, I4} from MFCS one at a time

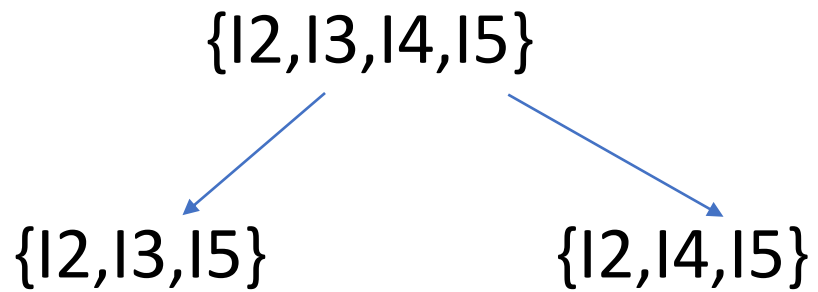


New MFCS = { {I2, I3, I4, I5}, {I1, I2, I3, I5} }

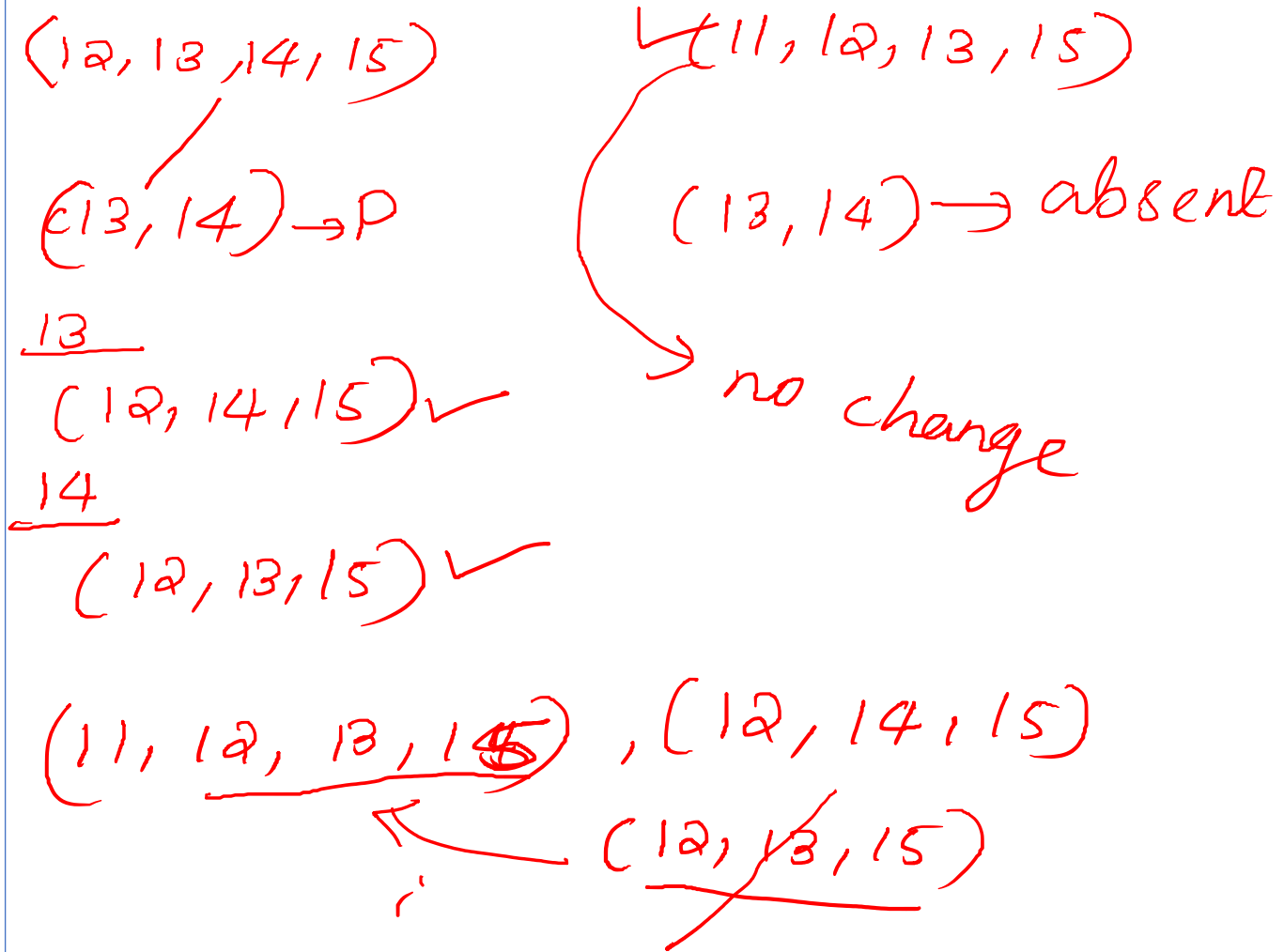
✓ I1, I2, I3, I4, I5
 check(I1, I4) } I1
 mfcs → (I2, I3, I4, I5) (I1, I2, I3, I5)

- New MFCS = $\{ \{12,13,14,15\}, \{11,12,13,15\} \}$ ✓
- $S2 = \{ \{11,14\}, \{13, 14\}, \{13, 15\}, \{14,15\} \}$

Delete $\{13, 14\}$ from each item in MFCS one at a time

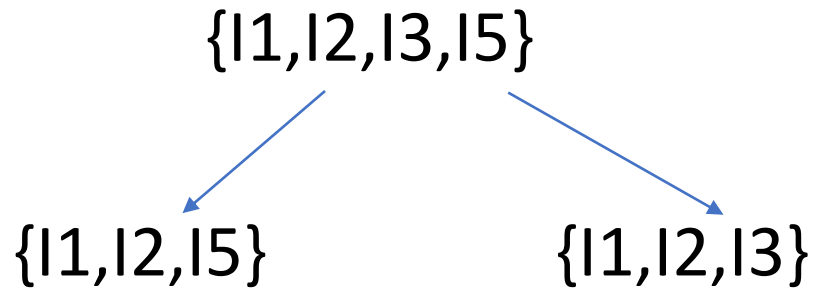


- $\{13,14\}$ not a subset of $\{11, 12,13,15\}$ so no change
- New MFCS = $\{ \{12,13,15\}, \{12,14,15\}, \{11,12,13,15\} \}$
- = $\{ \{12,14,15\}, \{11,12,13,15\} \}$



- New MFCS = $\{\{I2, I4, I5\}, \{I1, I2, I3, I5\}\}$
- $S2 = \{\{I1, I4\}, \{I3, I4\}, \{I3, I5\}, \{I4, I5\}\}$

Delete $\{I3, I5\}$ from each item in MFCS one at a time



- $\{I3, I5\}$ not a subset of $\{I2, I4, I5\}$ so no change
- New MFCS = $\{\{I1, I2, I5\}, \{I1, I2, I3\}, \{I2, I4, I5\}\}$ ✓

mfcs $(I1, I2, I3, I5)$

$(I3, I5)$

$\frac{I3}{(I1, I2, I5)} \checkmark$

$\frac{I5}{(I1, I2, I3)} \checkmark$

mfcs = $\{\{I1, I2, I5\}, \{I1, I2, I3\}, \{I2, I4, I5\}\}$

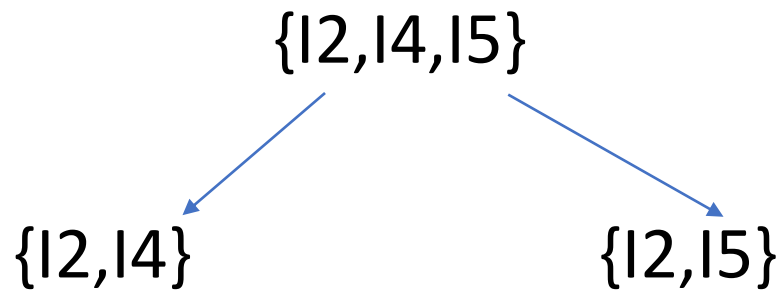
✓ $\{I2, I4, I5\}$

$(I3, I5) \rightarrow ab$
no change

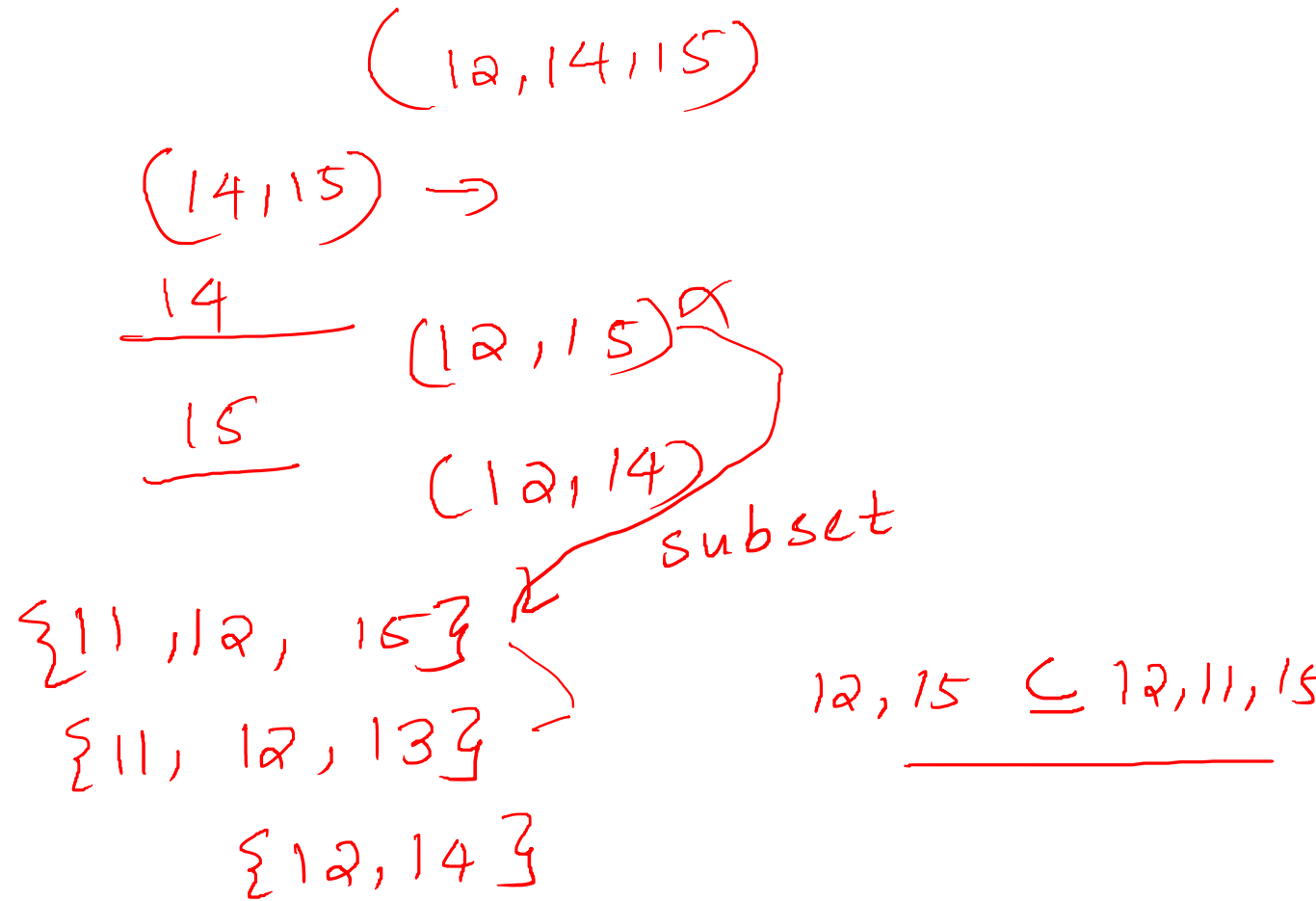
- New MFCS = { {I1,I2,I5}, {I1,I2,I3}, {I2,I4,I5}} ✓ ✓ab ✓ab

- S2 = { {I1,I4}, {I3, I4}, {I3, I5}, {I4,I5}}

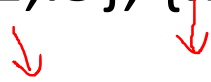
Delete {I4, I5} from each item in MFCS one at a time



- {I3,I5} not a subset of {I1,I2,I5}, {I1,I2,I3} so no change
- New MFCS = { {I1,I2,I5}, {I1,I2,I3}, {I2,I4}} ✓



- Final MFCS = { {I1,I2,I5}, {I1,I2,I3}, {I2,I4} }



- Final Output : Largest sets of three items.

Frequent Pattern \rightarrow {I1, I2, I5}
{I1, I2, I3}

Frequent Pattern Growth Algorithm

- This algorithm is an improvement to the Apriori method.
- A frequent pattern is generated without the need for candidate generation.
- FP growth algorithm represents the database in the form of a tree called a frequent pattern tree or FP tree.
- This tree structure will maintain the association between the itemsets.

- **FP Tree**

- Frequent Pattern Tree is a tree-like structure that is made with the initial itemsets of the database.
- The purpose of the FP tree is to mine the most frequent pattern.
- Each node of the FP tree represents an item of the itemset.
- The root node represents null while the lower nodes represent the itemsets.
- The association of the nodes with the lower nodes that is the itemsets with the other itemsets are maintained while forming the tree.

Problem 1

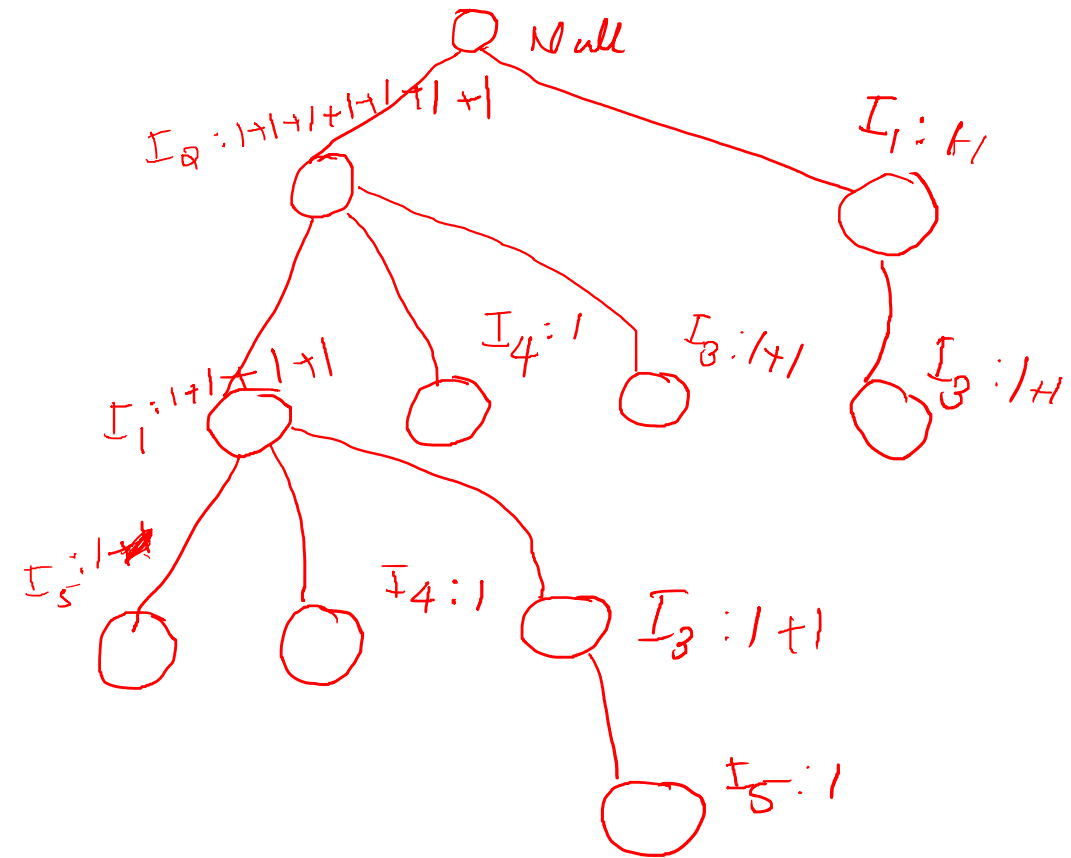
<i>TID</i>	<i>List of item_IDs</i>
T100	I1, I2, I5
T200	I2, I4
T300	I2, I3
T400	I1, I2, I4
T500	I1, I3
T600	I2, I3
T700	I1, I3
T800	I1, I2, I3, I5
T900	I1, I2, I3

Solution

ITEM	COUNT
I2	7
I1	6
I3	6
I4	2
I5	2

Eliminate those item whose support count below minimum support value
Here support_count = 2

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



Item	Conditional Pattern Base	Conditional FP Tree(with min support_count>=2)	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,I1:1}, {I2: 1} }	<I2 : 2>	{ I2, I4 : 2}
I3	{ {I2, I1 : 2}, {I2 :2} } - left subtree {I1 : 2} – right subtree	<I2:4, I1 :2 > <I1: 2>	{I2, I3 :4} {I1, I3:4} {I2, I1, I3 : 2}
I1	{ {I2: 4} }	< I2 : 4 >	{I2, I1: 4}

Problem : 2

- Support threshold=50%, Confidence= 60%

Transaction	List of items
T1	I1,I2,I3
T2	I2,I3,I4
T3	I4,I5
T4	I1,I2,I4
T5	I1,I2,I3,I5
T6	I1,I2,I3,I4

Solution:

Support threshold=50% $\Rightarrow 0.5 \times 6 = 3 \Rightarrow \text{min_sup} = 3$

1. Count of each item

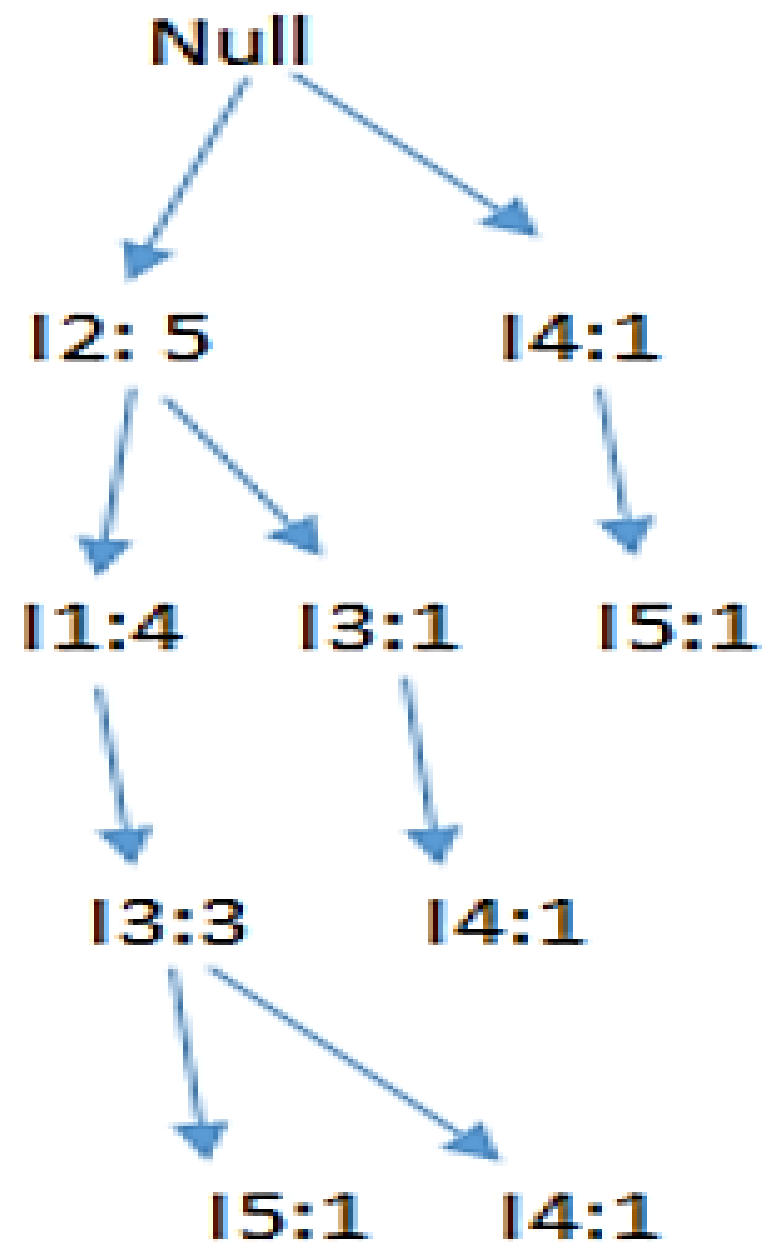
Table 2

Item	Count
I1	4
I2	5
I3	4
I4	4
I5	2

2. Sort the itemset in descending order.

Table 3

Item	Count
I2	5
I1	4
I3	4
I4	4



Item	Conditional Pattern Base	Conditional FP-tree	Frequent Patterns Generated
I4	{I2,I1,I3:1},{I2,I3:1}	{I2:2, I3:2}	{I2,I4:2},{I3,I4:2}, {I2,I3,I4:2}
I3	{I2,I1:3},{I2:1}	{I2:4, I1:3}	{I2,I3:4}, {I1:I3:3}, {I2,I1,I3:3}
I1	{I2:4}	{I2:4}	{I2,I1:4}

Advantages Of FP Growth Algorithm

- 1.This algorithm needs to scan the database only twice when compared to Apriori which scans the transactions for each iteration.
- 2.The pairing of items is not done in this algorithm and this makes it faster.
- 3.The database is stored in a compact version in memory.
- 4.It is efficient and scalable for mining both long and short frequent patterns.

Disadvantages Of FP-Growth Algorithm

1. FP Tree is more difficult to build than Apriori.
2. It may be expensive.
3. When the database is large, the algorithm may not fit in the shared memory

FP Growth vs Apriori

FP Growth

Pattern Generation

FP growth generates pattern by constructing a FP tree

Candidate Generation

There is no candidate generation

Process

The process is faster as compared to Apriori. The runtime of process increases linearly with increase in number of itemsets.

Memory Usage

A compact version of database is saved

Apriori

Apriori generates pattern by pairing the items into singletons, pairs and triplets.

Apriori uses candidate generation





The process is comparatively slower than FP Growth, the runtime increases exponentially with increase in number of itemsets

The candidates combinations are saved in memory

Dynamic Itemset Counting Algorithm

- Alternative to Apriori Itemset Generation
- Itemsets are dynamically added and deleted as transactions are read
- Relies on the fact that for an itemset to be frequent, all of its subsets must also be frequent, so we only examine those itemsets whose subsets are all frequent

Itemsets are marked in four different ways as they are counted:

- **Solid box:**  confirmed frequent itemset - an itemset we have finished counting and exceeds the support threshold *minsupp*
- **Solid Circle:**  confirmed infrequent itemset - we have finished counting and it is below *minsupp*
- **Dashed box:**  suspected frequent itemset - an itemset we are still counting that exceeds *minsupp*
- **Dashed Circle:**  suspected infrequent itemset - an itemset we are still counting that is below *minsupp*

DIC Algorithm

1. Mark the empty itemset with a solid square. Mark all the 1-itemsets with dashed circles. Leave all other itemsets unmarked.
2. While any dashed itemsets remain:
 1. Read M transactions (if we reach the end of the transaction file, continue from the beginning). For each transaction, increment the respective counters for the itemsets that appear in the transaction and are marked with dashes.
 2. If a dashed circle's count exceeds *minsupp*, turn it into a dashed square. If any immediate superset of it has all of its subsets as solid or dashed squares, add a new counter for it and make it a dashed circle.
 3. Once a dashed itemset has been counted through all the transactions, make it solid and stop counting it.

- **Itemset lattices:** An itemset lattice contains all of the possible itemsets for a transaction database. Each itemset in the lattice points to all of its supersets. When represented graphically, a itemset lattice can help us to understand the concepts behind the DIC algorithm

Problem

- Example: *minsupp* = **25%** (support count= $0.25 \times 4 = 1$) and **M** = **2**.

<i>TID</i>	<i>A</i>	<i>B</i>	<i>C</i>
T1	1	1	0
T2	1	0	0
T3	0	1	1
T4	0	0	0

Transaction Database

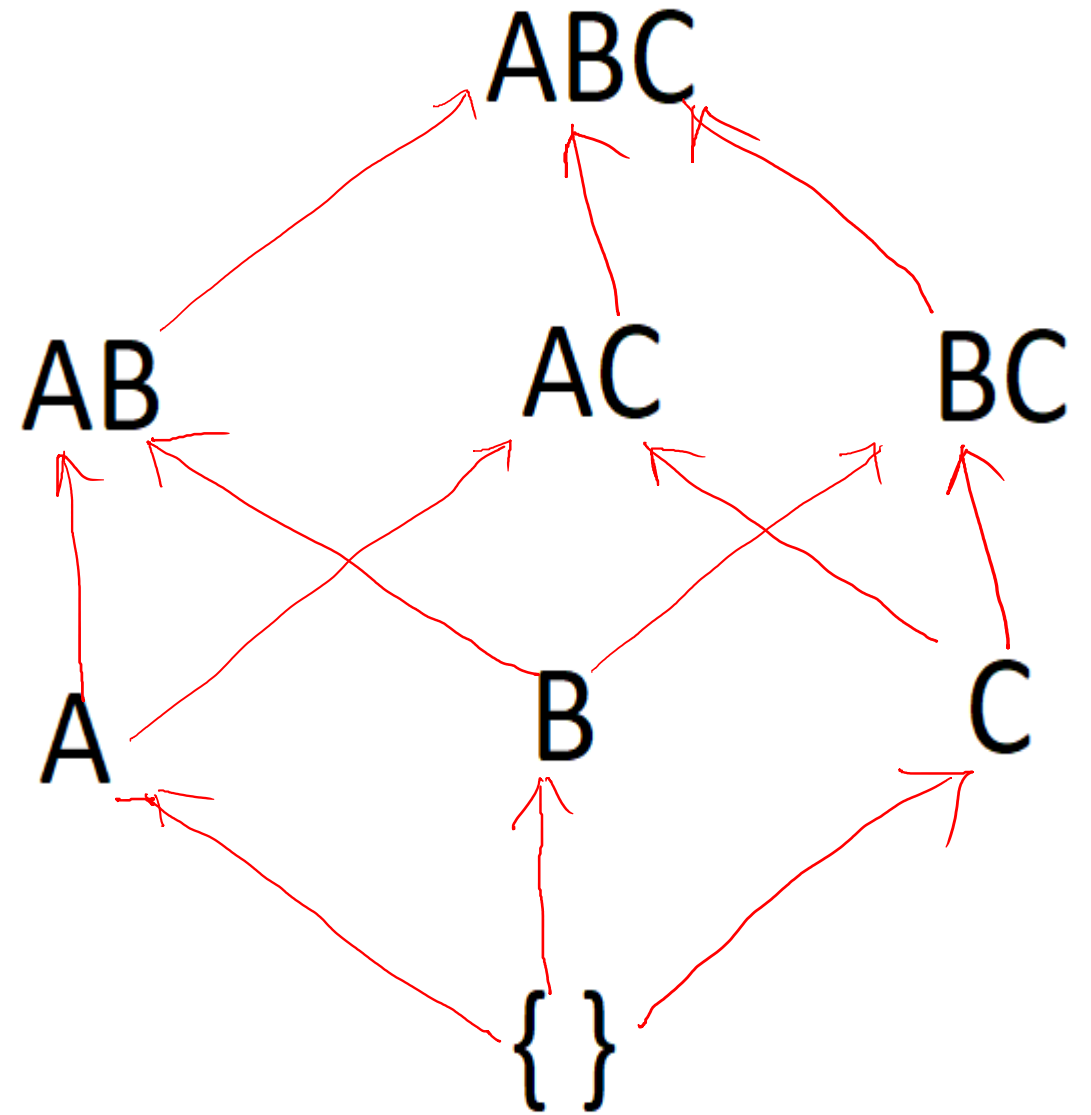
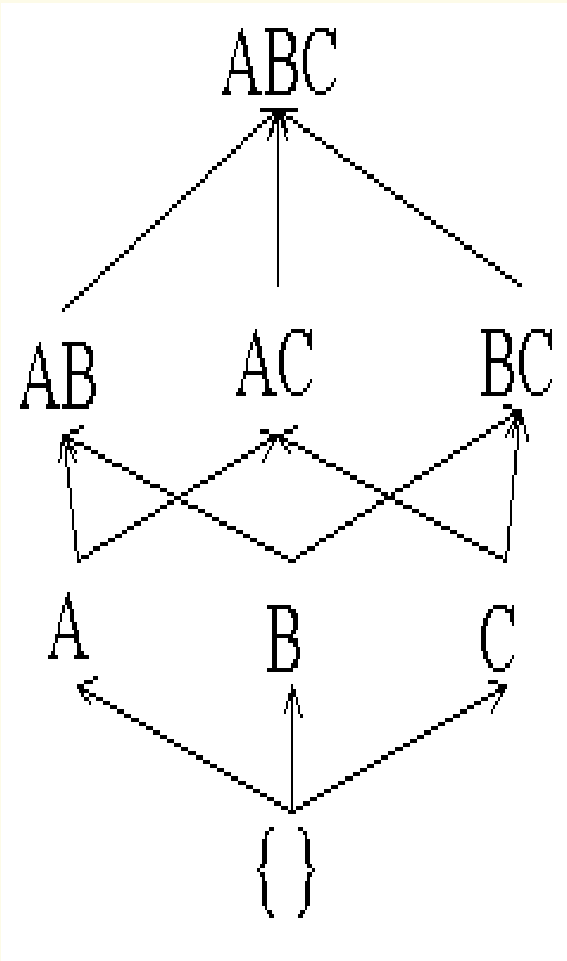
A B

A

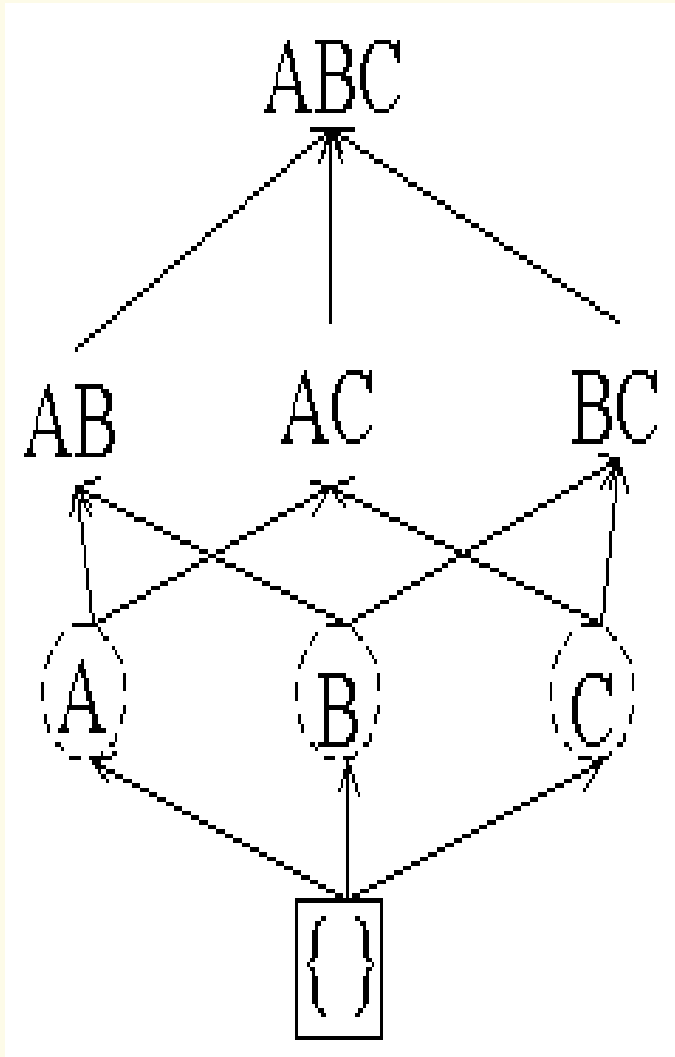
B C

—

Itemset lattice for the above transaction database:



Itemset lattice before any transactions are read:



Counters: $A=0$, $B=0$, $C=0$

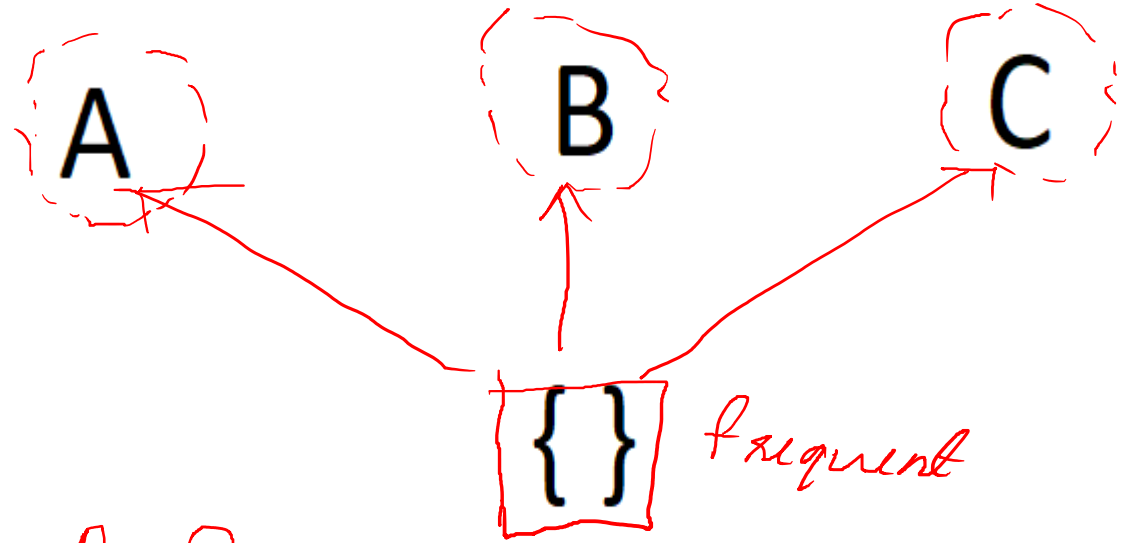


ABC

AB

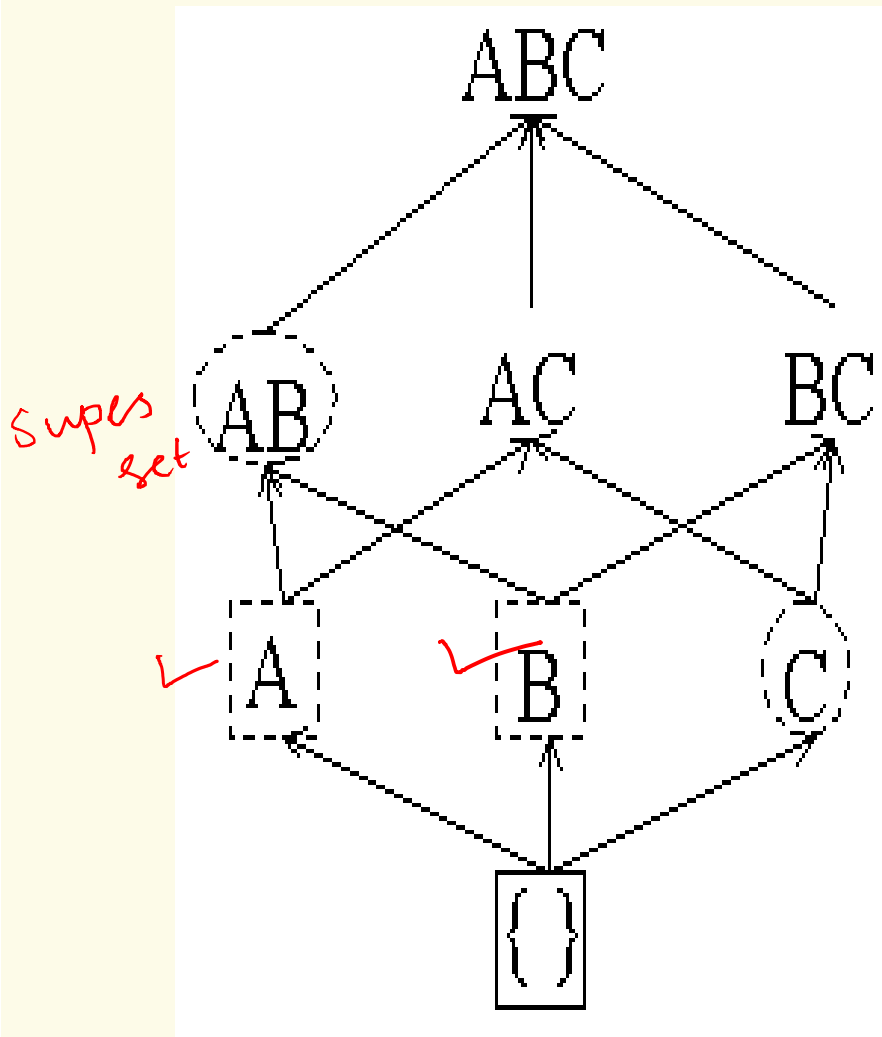
AC

BC



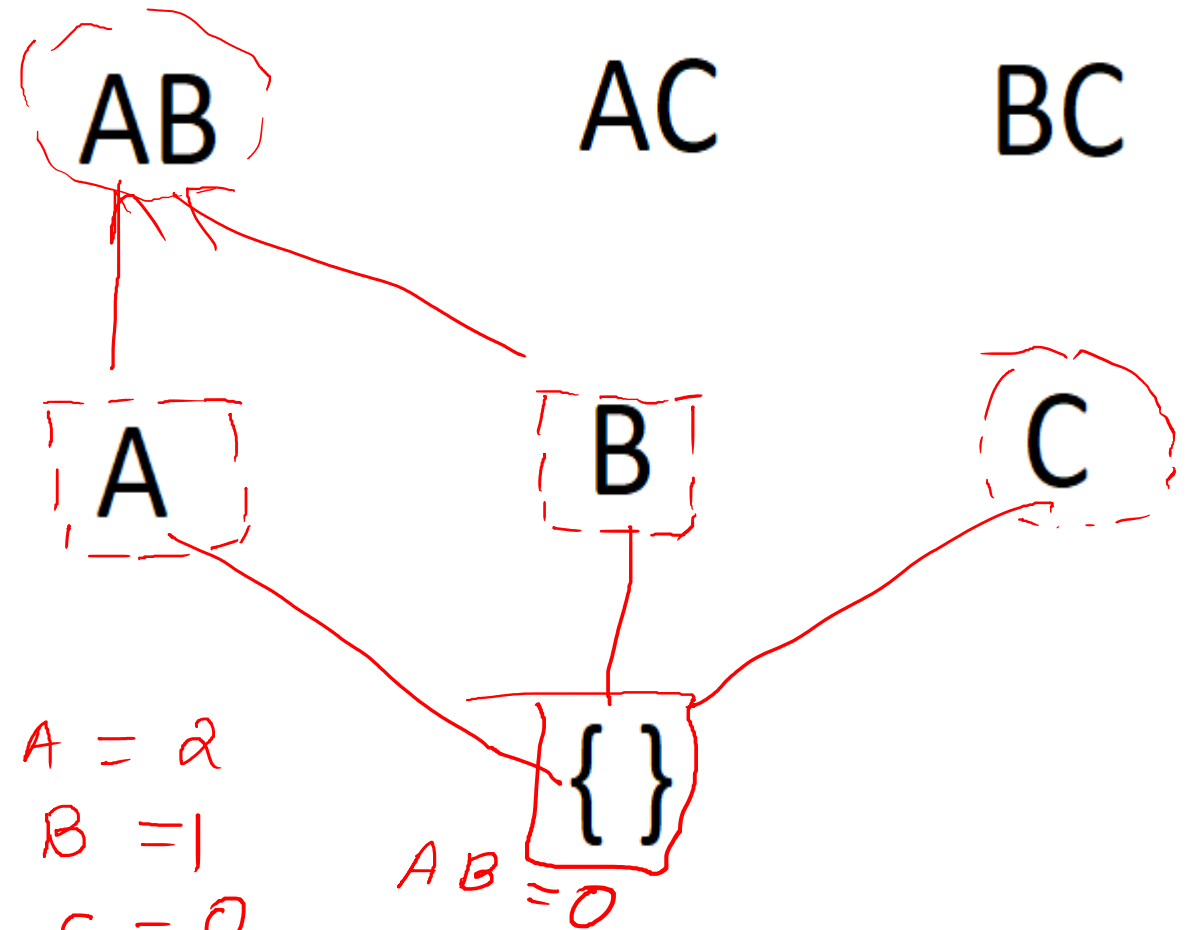
$A=0$
 $B=0$ $C=0$

After M transactions are read:

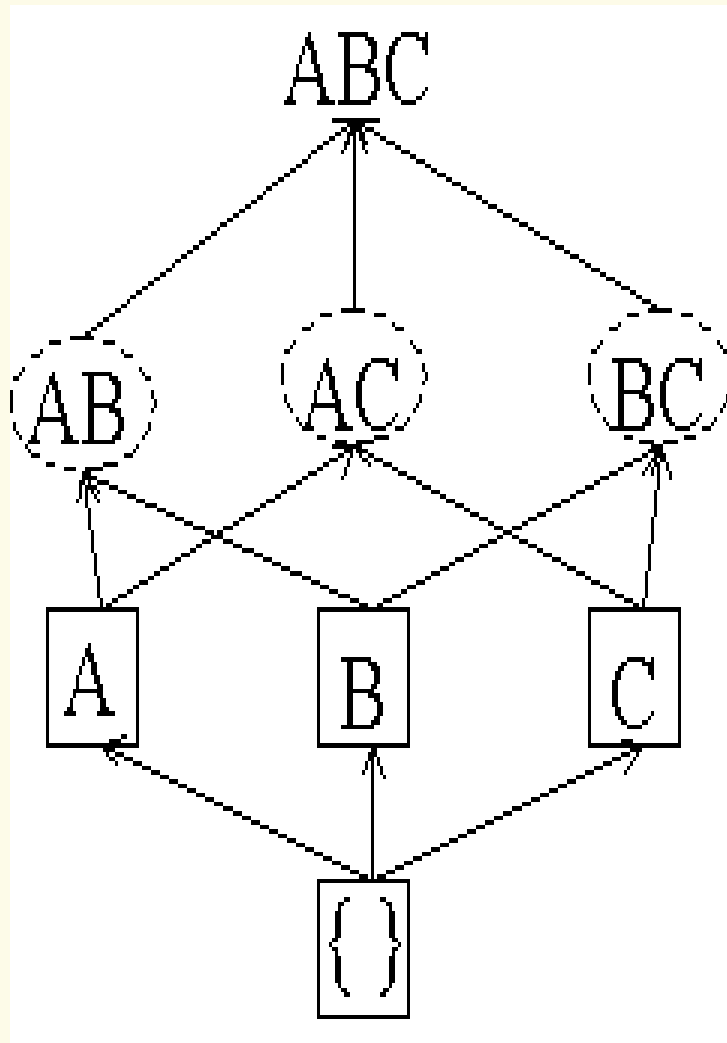


Counters: $A = 2$, $B = 1$, $C = 0$, $AB = 0$

ABC

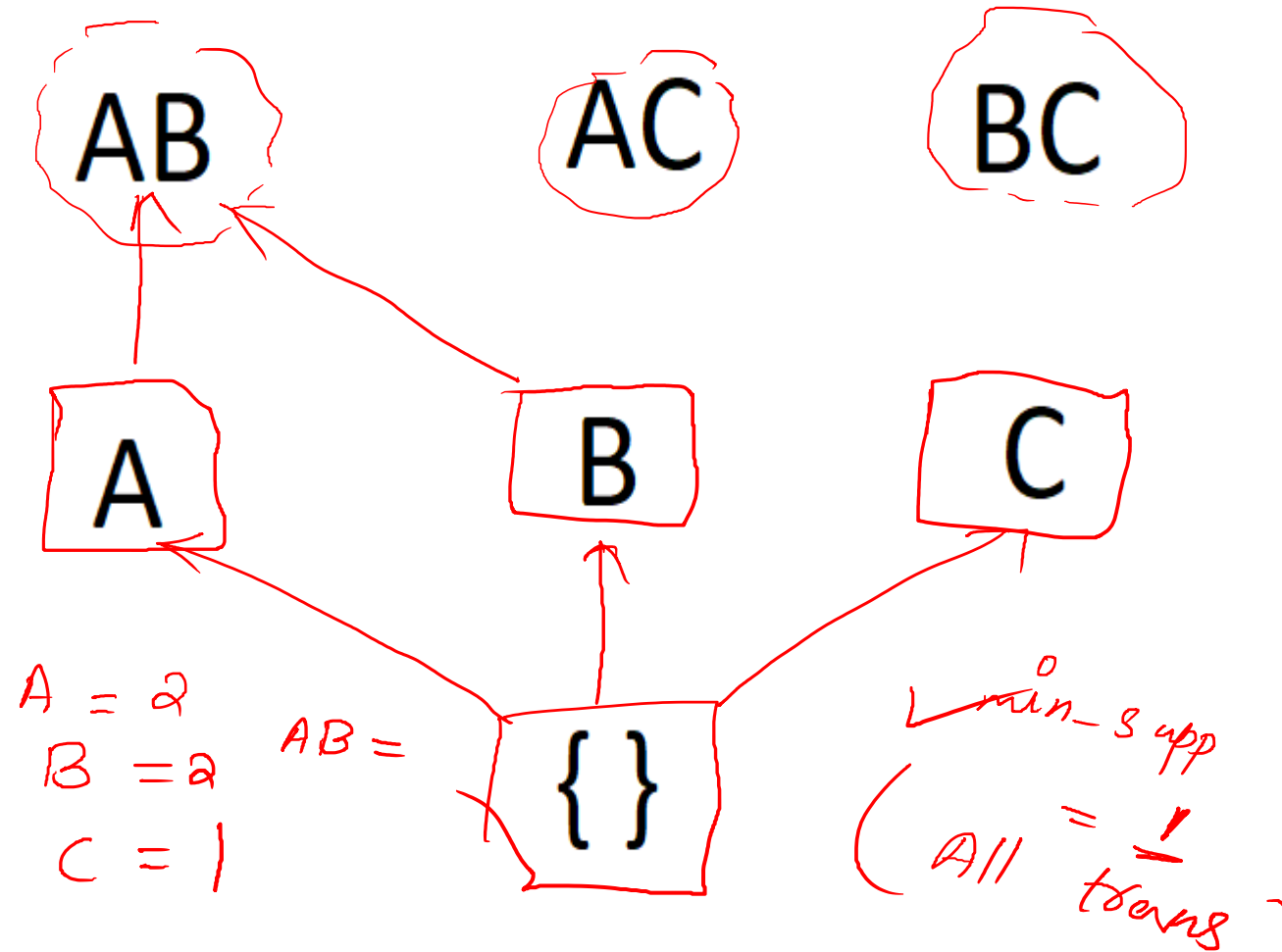


After 2M transactions are read:

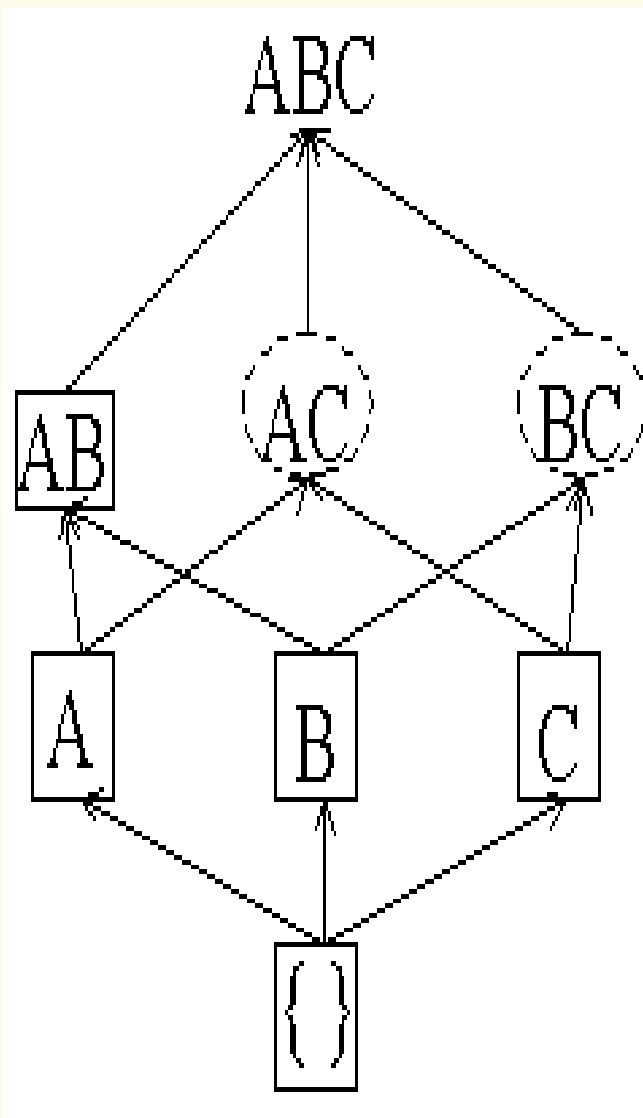


Counters: $A = 2$, $B = 2$, $C = 1$, $AB = 0$, $AC = 0$, $BC = 0$

ABC



After 3M transactions read:



Counters: A = 2, B = 2, C = 1, AB = 1, AC = 0, BC = 0

ABC

AB

AC

BC

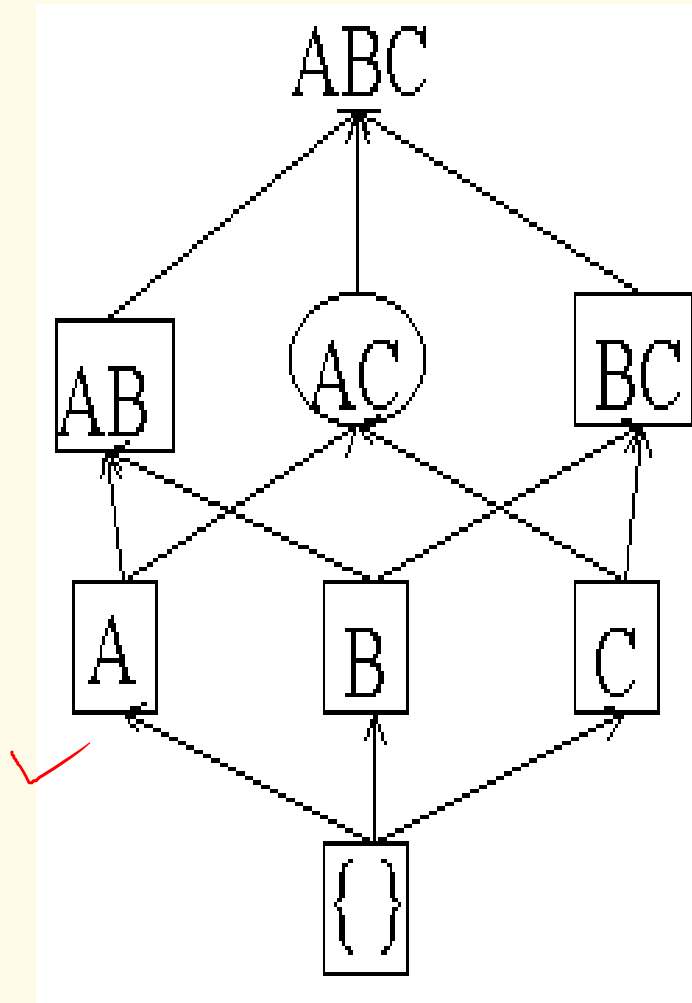
A

B

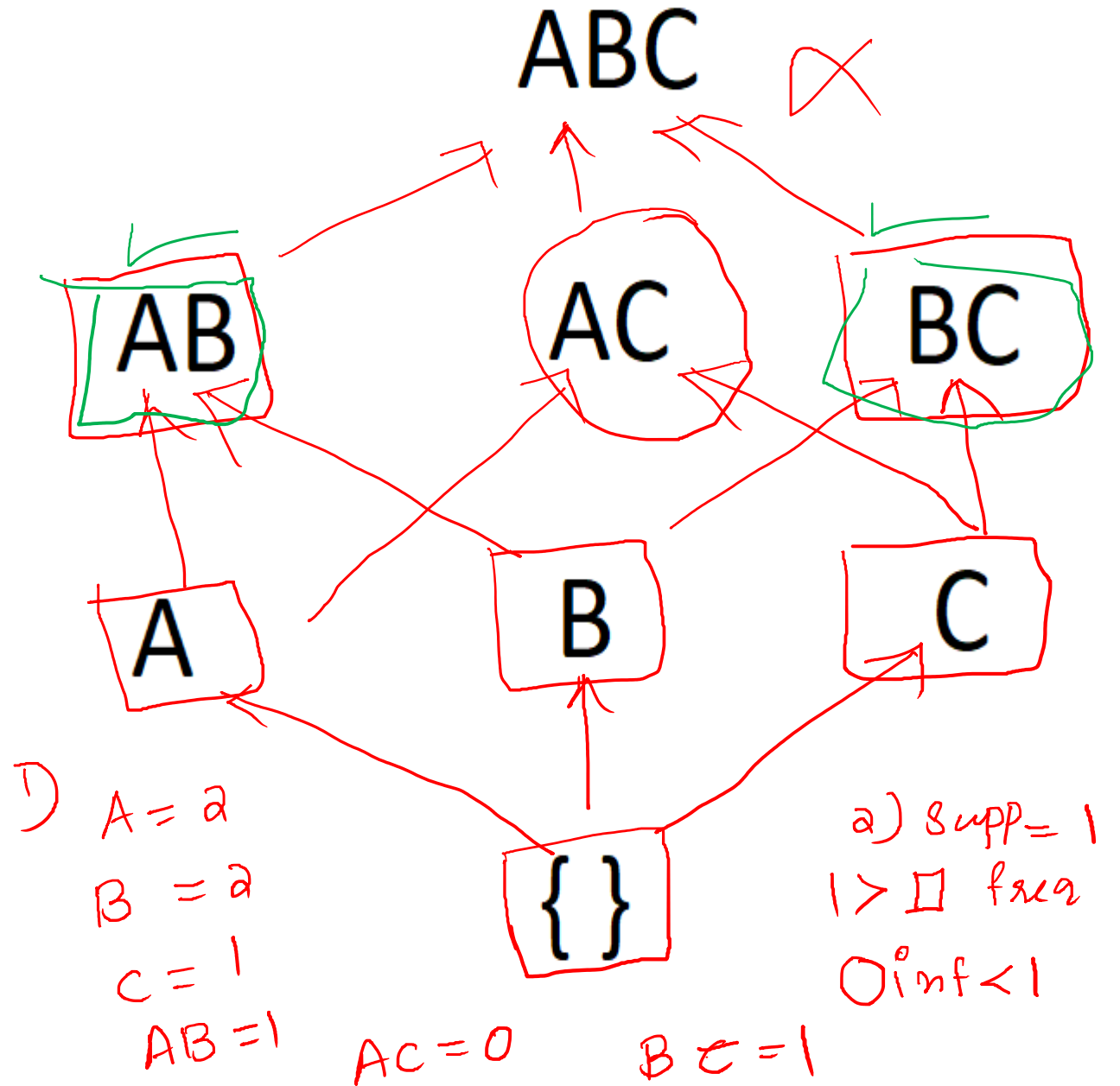
C

{ }

After 4M transactions read:



Counters: $A=2, B=2, C=1, AB=1, AC=0, BC=1$



```

SS =  $\emptyset$  // solid square (frequent)
SC =  $\emptyset$  // solid circle (infrequent)
DS =  $\emptyset$  // dashed square (suspected frequent)
DC = { all 1-itemsets } // dashed circle (suspected infrequent)
while (DS != 0) or (DC != 0) do begin
    read M transactions from database into T
    forall transactions t ∈ T do begin
        //increment the respective counters of the itemsets marked with dash
        for each itemset c in DS or DC do begin
            if ( c ∈ t ) then
                c.counter++ ;
        for each itemset c in DC
            if ( c.counter ≥ threshold ) then
                move c from DC to DS ;
            if ( any immediate superset sc of c has all of its subsets in SS or DS ) then
                add a new itemset sc in DC ;
        end
        for each itemset c in DS
            if ( c has been counted through all transactions ) then
                move it into SS ;
        for each itemset c in DC
            if ( c has been counted through all transactions ) then
                move it into SC ;
        end
    end
end

Answer = { c ∈ SS } ;

```