BITS Pilani

BITS Pilani
Hyderabad Campus

Dr. Manik Gupta
Assistant Professor
Department of CSIS

innovate achieve lead

BITS Pilani
Hyderabad Campus

Data Mining (CS F415)
Lecture 11 – FP Growth

Tuesday, 4th February 2020

Today’s Agenda

- FP Growth

Introduction

- **Frequent pattern growth,** or simply **FP-growth** adopts a *divide-and-conquer* strategy.
  - First, it compresses the database representing frequent items into a **frequent pattern tree,** or **FP-tree**, which retains the itemset association information.
  - 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,” and mines each database separately. For each “pattern fragment,” only its associated data sets need to be examined.
  - Therefore, this approach may substantially reduce the size of the data sets to be searched, along with the “growth” of patterns being examined.

Steps involved in FP-Growth

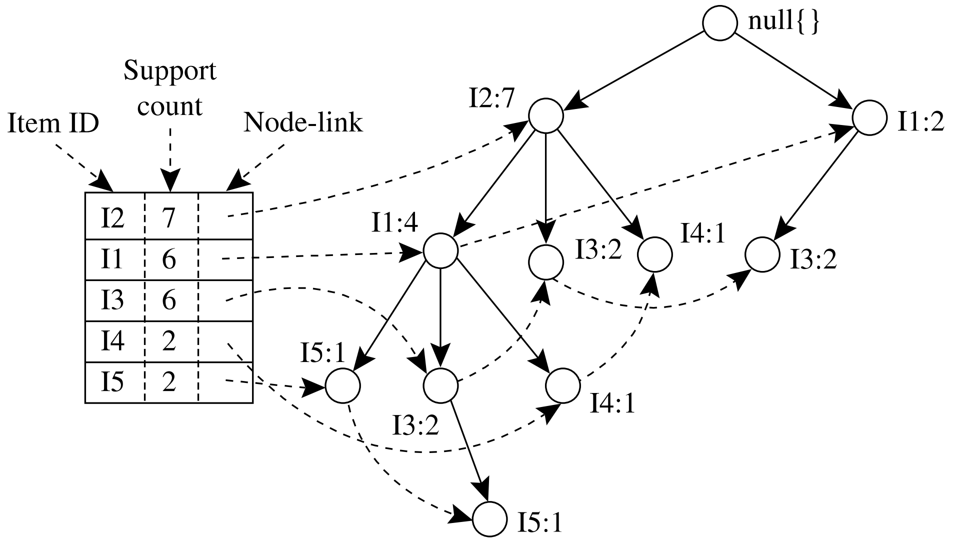
- Use a compressed representation of the database using a FP-tree
- Once an FP-tree has been constructed, it uses a recursive divide-and-conquer approach to mine the frequent itemsets.

Construction of FP-Tree

- Scan data to determine the support count of each item.
  - Infrequent items are discarded, while the frequent items are sorted in decreasing support counts.
- Make a second pass over the data to construct the FP-tree.
  - As the transactions are read, before being processed, their items are sorted according to the above order.

Example

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

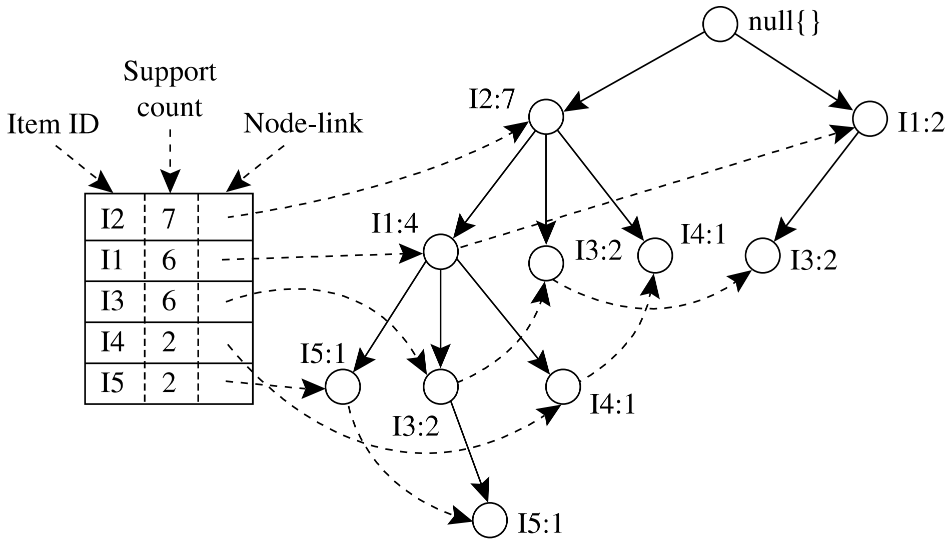


Construction of FP-Tree

- Let the minimum support count be 2. The set of frequent items is sorted in the order of descending support count. This resulting set or *list* is denoted by *L*. Thus, we have *L* = {{I2: 7}, {I1: 6}, {I3: 6}, {I4: 2}, {I5: 2}}.
- First, create the root of the tree, labeled with “null.” Scan database *D* a second time. 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.

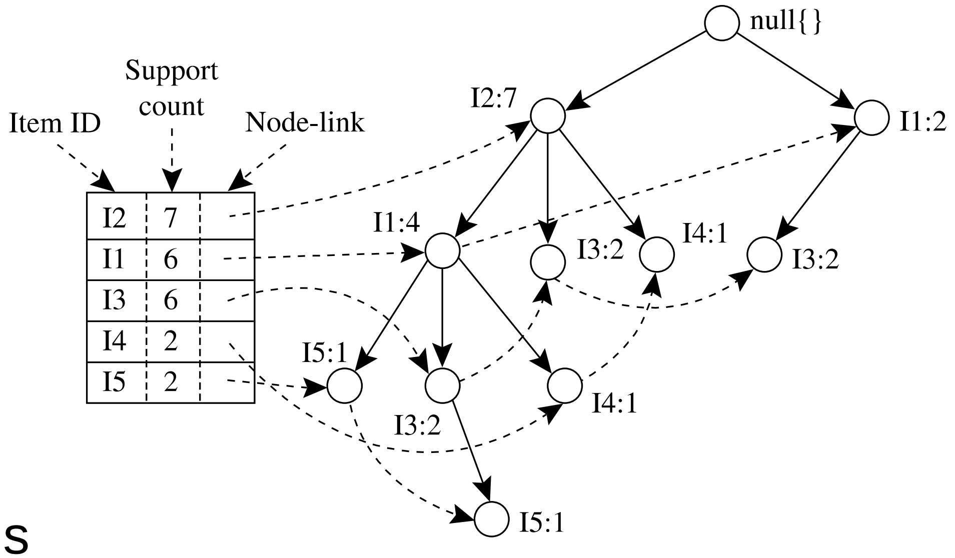
Construction of FP-Tree

- The scan of the first transaction, “T100: I1, I2, I5,” which contains three items (I2, I1, I5 in *L* order), leads to the construction of the first branch of the tree with three nodes, ⟨I2: 1⟩, ⟨I1: 1⟩, and ⟨I5: 1⟩, where I2 is linked as a child to the root, I1 is linked to I2, and I5 is linked to I1.



Construction of FP-Tree

- 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.
- However, this branch would share a common **prefix,** I2, with the existing path for T100.
- Therefore, we instead increment the count of the I2 node by 1, and create a new node, ⟨I4: 1⟩, which is linked as a child to ⟨I2: 2⟩.



Construction of FP-Tree

- 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.

Header Table

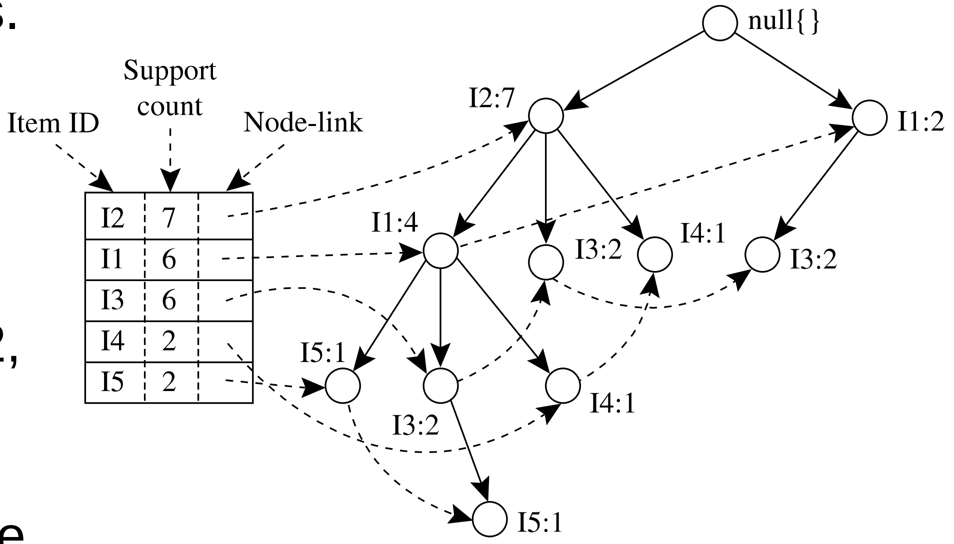
- To facilitate tree traversal, an item header table is built so that each item points to its occurrences in the tree via a chain of **node-links**.
- In this way, the problem of mining frequent patterns in databases is transformed into that of mining the FP-tree.

Mining of the FP-Tree

- Start from each frequent length-1 pattern (as an initial **suffix pattern**), construct its **conditional pattern base** (a “sub-database,” which consists of the set of *prefix paths* in the FP-tree co-occurring with the suffix pattern)
- Next construct its (*conditional*) FP-tree, and perform mining recursively on the tree.
- The pattern growth is achieved by the concatenation of the suffix pattern with the frequent patterns generated from a conditional FP-tree.

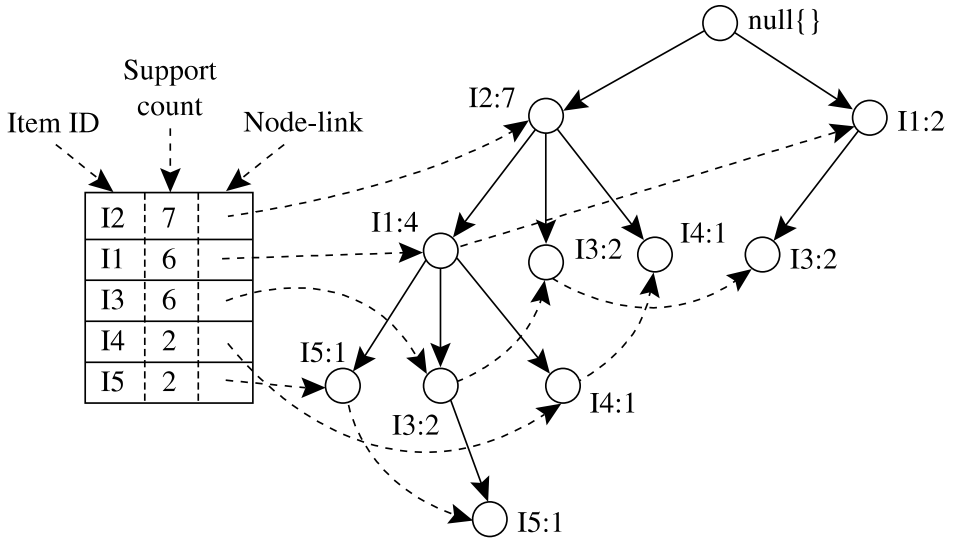
Mining of the FP-Tree

- We first consider I5, which is the last item in *L*, rather than the first.
- I5 occurs in two FP-tree branches. (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***.



Mining of the FP-Tree

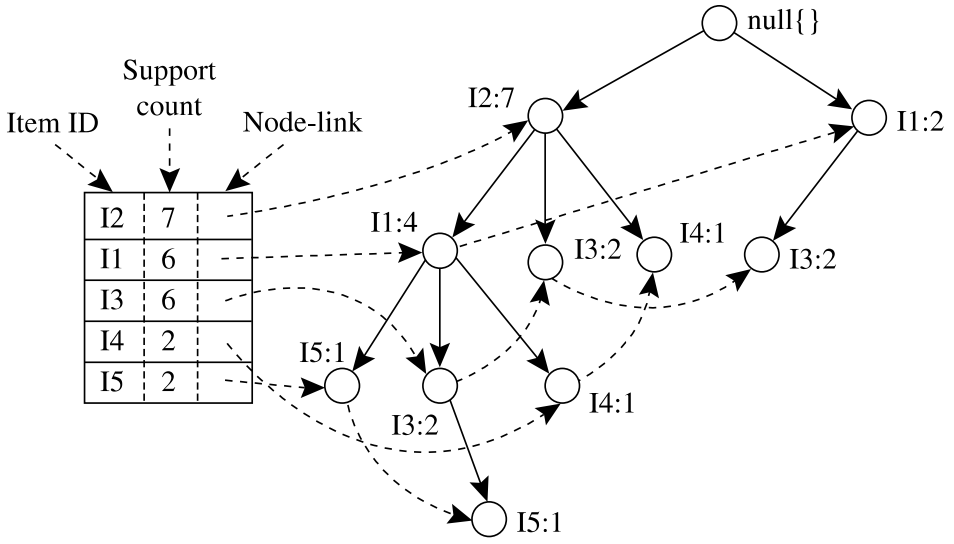
- Using this conditional pattern base as a transaction database, we build an I5-conditional FP-tree, which contains only a single path, ⟨I2: 2, I1: 2⟩
- I3 is not included because its support count of 1 is less than the minimum support count.
- The single path generates all the combinations of frequent patterns: {I2, I5: 2}, {I1, I5: 2}, {I2, I1, I5: 2}.



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

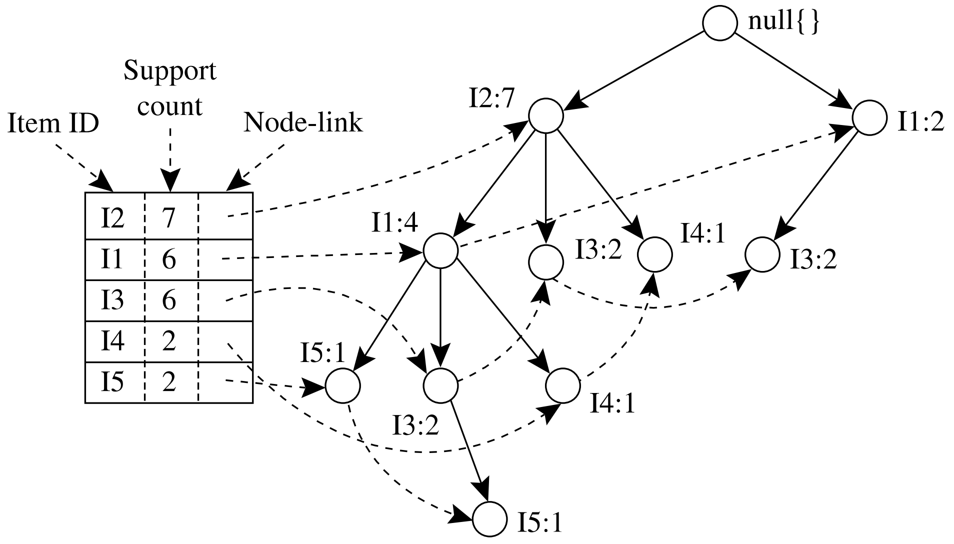
Mining of the FP-Tree

- For I4, its two prefix paths form the conditional pattern base, {{I2 I1: 1}, {I2: 1}}
- Generates a single-node conditional FP-tree, ⟨I2: 2⟩, and derives one frequent pattern, {I2, I4: 2}.



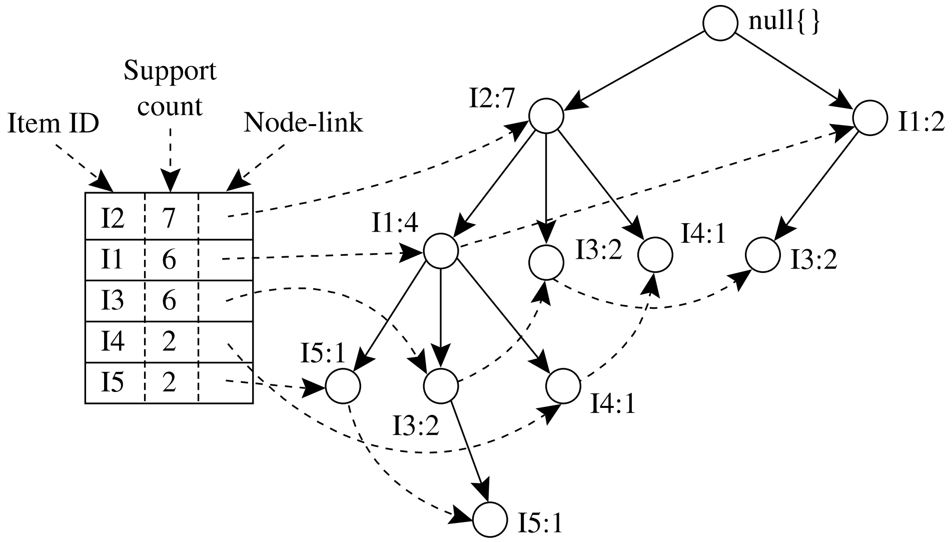
Mining of the FP-Tree

- Similar to the preceding analysis, I3's conditional pattern base is {{I2, I1: 2}, {I2: 2}, {I1: 2}}.
- Its conditional FP-tree has two branches, ⟨I2: 4, I1: 2⟩ and ⟨I1: 2⟩, which generates the set of patterns {{I2, I3: 4}, {I1, I3: 4}, {I2, I1, I3: 2}}.



Mining of the FP-Tree

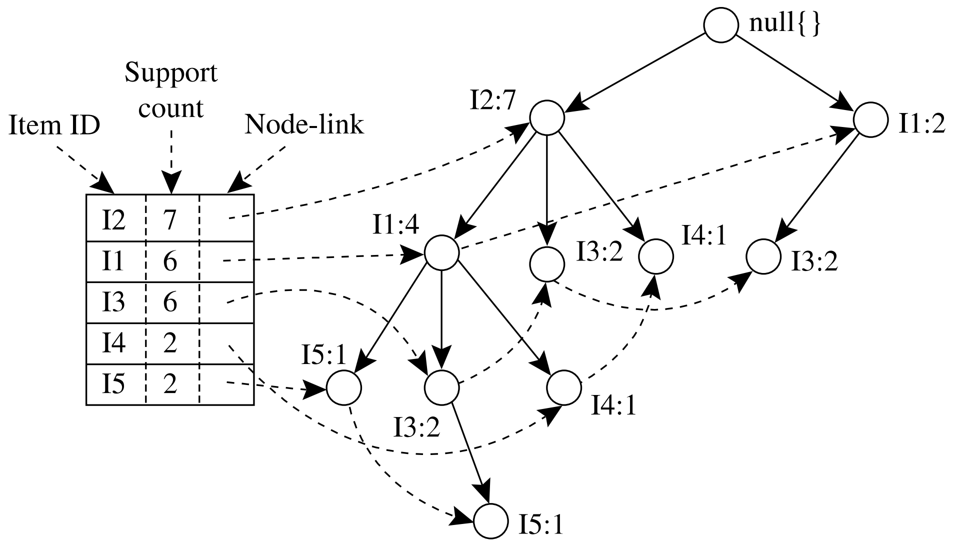
- Finally, I1’s conditional pattern base is {{I2: 4}}
- The FP-tree contains only one node, ⟨I2: 4⟩, which generates one frequent pattern, {I2, I1: 4}.



Mining of 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}



Steps involved in FP-Growth

- Use a compressed representation of the database using a FP-tree
- Once an FP-tree has been constructed, it uses a recursive divide-and-conquer approach to mine the frequent itemsets.

Construction of FP-Tree

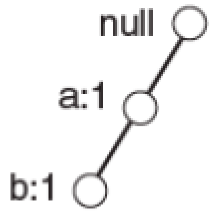
- Scan data to determine the support count of each item.
  - Infrequent items are discarded, while the frequent items are sorted in decreasing support counts.
- Make a second pass over the data to construct the FP-tree.
  - As the transactions are read, before being processed, their items are sorted according to the above order.

Mining of the FP-Tree

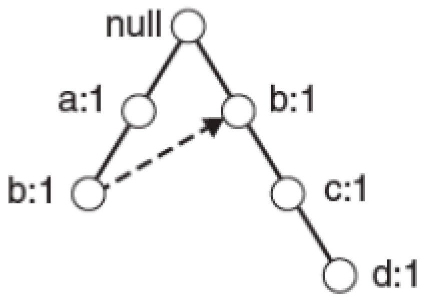
- Start from each frequent length-1 pattern (as an initial **suffix pattern**), construct its **conditional pattern base** (a “sub-database,” which consists of the set of *prefix paths* in the FP-tree co-occurring with the suffix pattern)
- Next construct its (*conditional*) FP-tree, and perform mining recursively on the tree.
- The pattern growth is achieved by the concatenation of the suffix pattern with the frequent patterns generated from a conditional FP-tree.

Construction of FP-Tree - Example

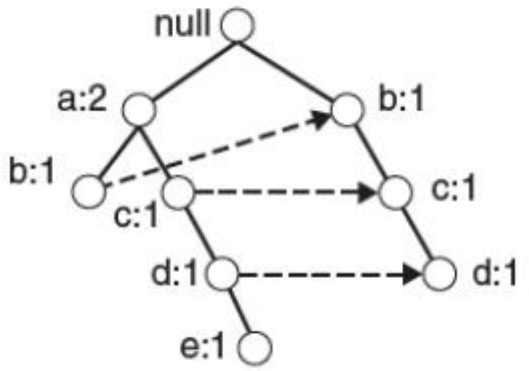
TID	Items
1	{A,B}
2	{B,C,D}
3	{A,C,D,E}
4	{A,D,E}
5	{A,B,C}
6	{A,B,C,D}
7	{A}
8	{A,B,C}
9	{A,B,D}
10	{B,C,E}



(i) After reading TID=1



(ii) After reading TID=2



(iii) After reading TID=3

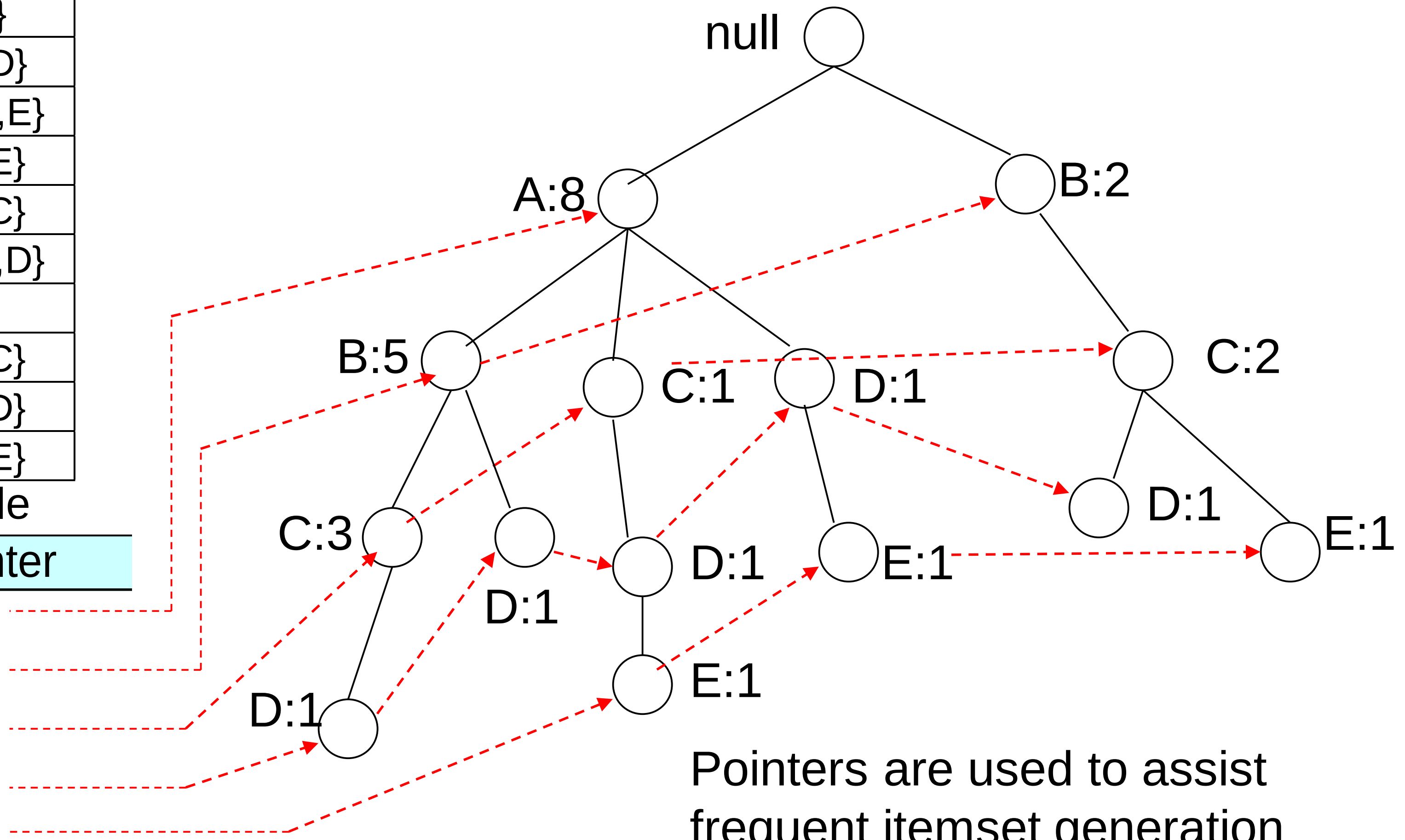
Construction of FP-Tree

Transaction Database

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}

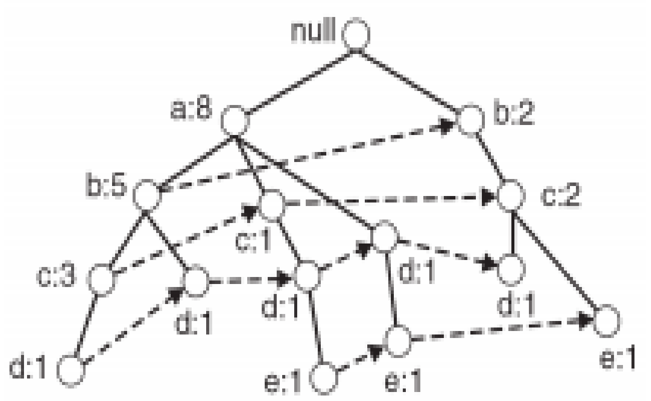
Header table

Item	Pointer
A	
B	
C	
D	
E	

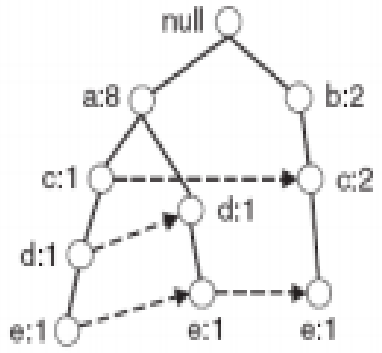


Pointers are used to assist frequent itemset generation

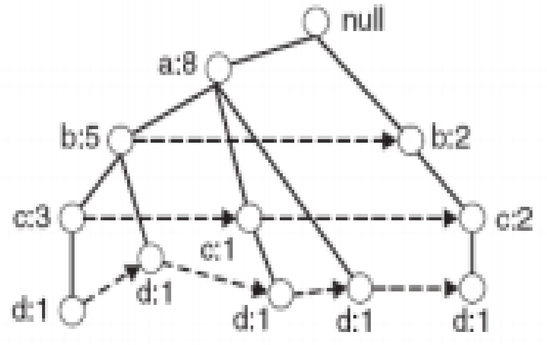
Prefix path sub-trees: Example



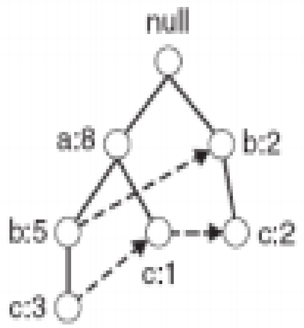
↑ Complete FP-tree



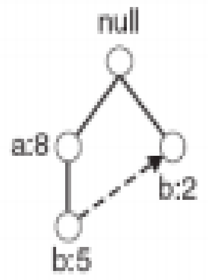
(a) Paths containing node e



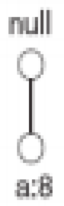
(b) Paths containing node d



(c) Paths containing node c



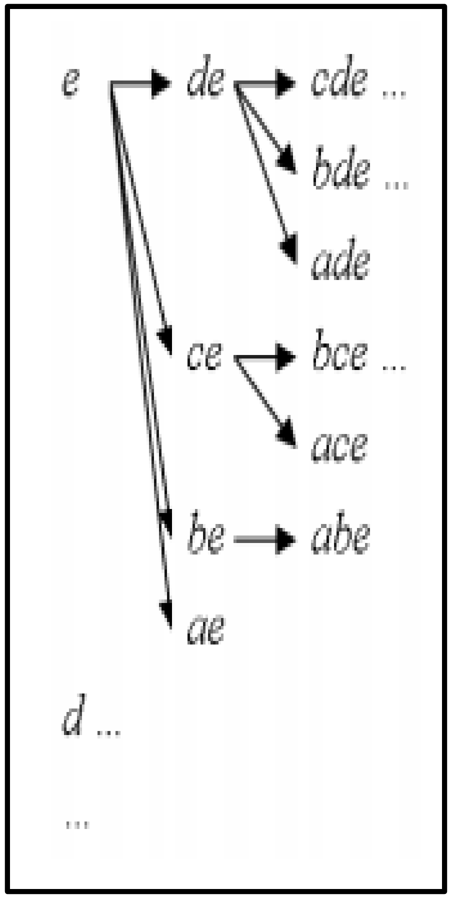
(d) Paths containing node b



(e) Paths containing node a

Frequent Itemset Generation

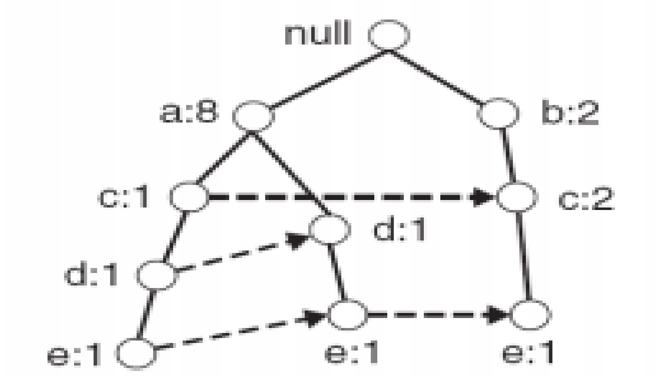
- Each prefix path sub-tree is processed recursively to extract the frequent itemsets. Solutions are then merged.
  - E.g. the prefix path sub-tree for *e* will be used to extract frequent itemsets ending in e, then in de, ce, be and ae, then in cde, bde, cde, etc.
  - Divide and conquer approach



Example

Let $minSup = 2$ and extract all frequent itemsets containing e.

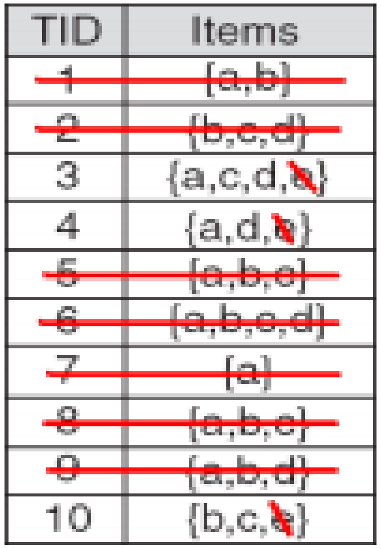
- 1. Obtain the prefix path sub-tree for e:



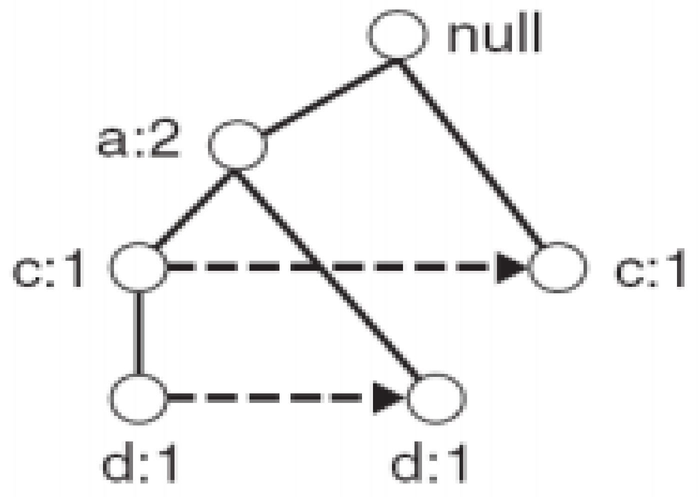
- 2. Check if e is a frequent item by adding the counts along the linked list (dotted line). If so, extract it.
  - Yes, count =3 so $\{e\}$ is extracted as a frequent itemset.
- 3. As e is frequent, find frequent itemsets ending in e. i.e. de, ce, be and ae.
  - i.e. decompose the problem recursively.
  - To do this, we must first to obtain the conditional FP-tree for e.

Conditional FP-Tree

- The FP-Tree that would be built if we only consider transactions containing a particular itemset (and then removing that itemset from all transactions).
- **Example**: FP-Tree conditional on *e*.



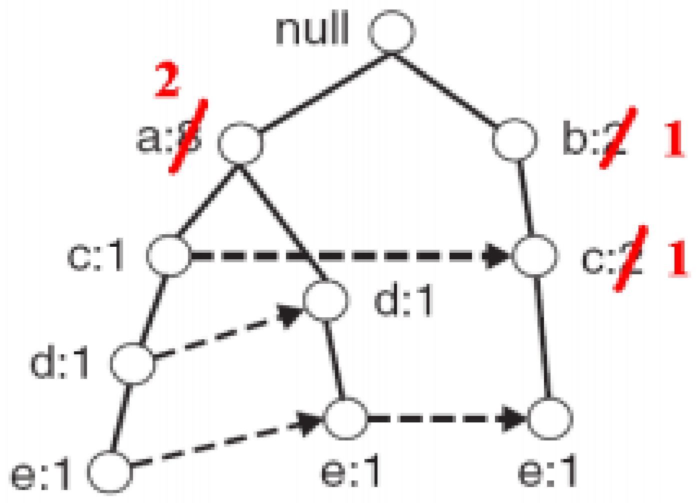
TID	Items
~~1~~	~~{a,b}~~
~~2~~	~~{b,c,d}~~
3	{a,c,d,~~e~~}
4	{a,d,~~e~~}
~~5~~	~~{a,b,c}~~
~~6~~	~~{a,b,c,d}~~
~~7~~	~~{a}~~
~~8~~	~~{a,b,c}~~
~~9~~	~~{a,b,d}~~
10	{b,c,~~e~~}



Conditional FP-Tree

To obtain the *conditional FP-tree* for *e* from the *prefix sub-tree* ending in *e*:

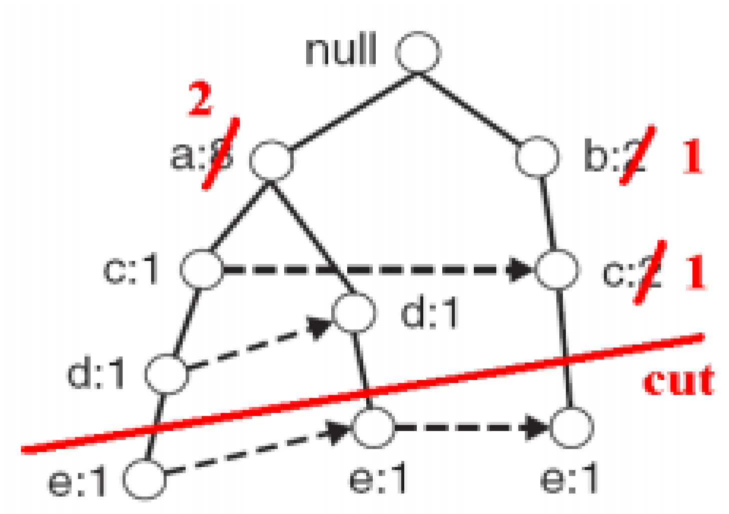
- Update the support counts along the prefix paths (from *e*) to reflect the number of transactions containing *e*.
    - *b* and *c* should be set to 1 and *a* to 2.

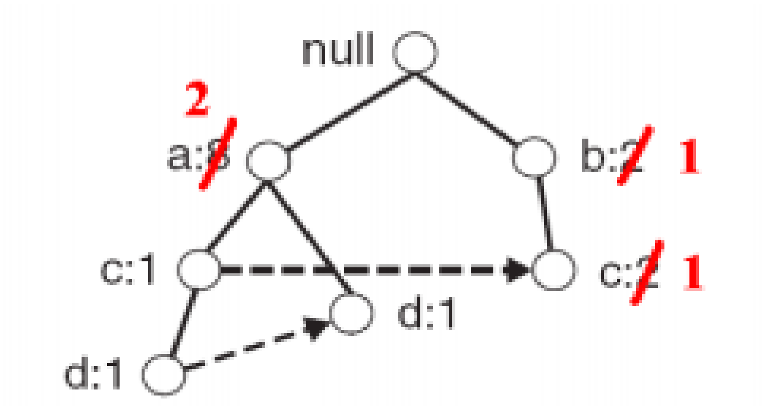


Conditional FP-Tree

To obtain the *conditional FP-tree* for *e* from the *prefix sub-tree* ending in *e*:

- Remove the nodes containing *e* – information about node *e* is no longer needed because of the previous step

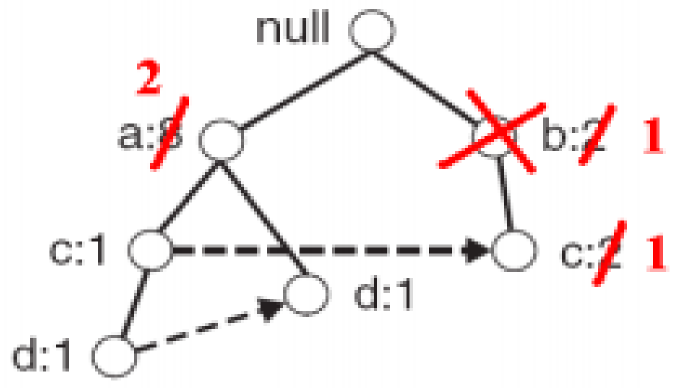


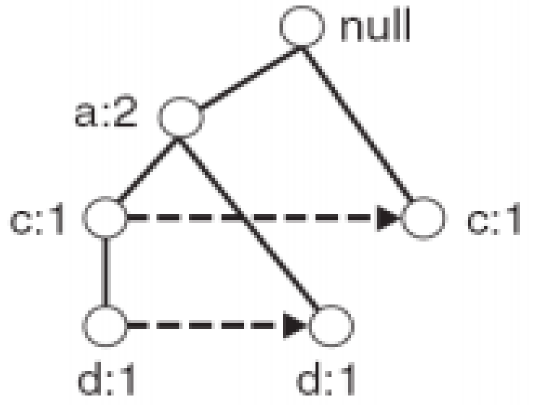


Conditional FP-Tree

To obtain the *conditional FP-tree* for *e* from the *prefix sub-tree* ending in *e*:

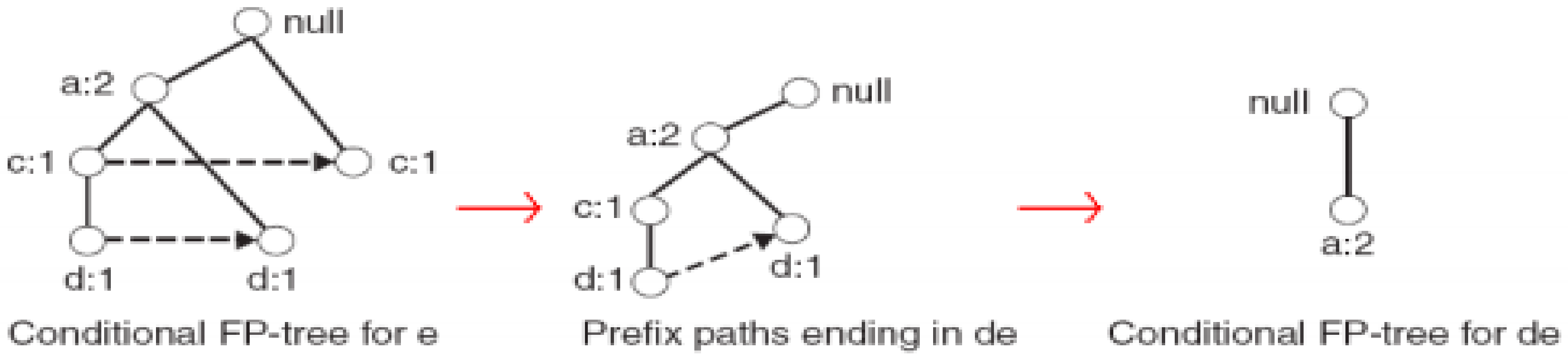
- Remove infrequent items (nodes) from the prefix paths
- E.g. *b* has a support of 1 (note this really means *be* has a support of 1). i.e. there is only 1 transaction containing *b and e* so *be* is infrequent – can remove b.





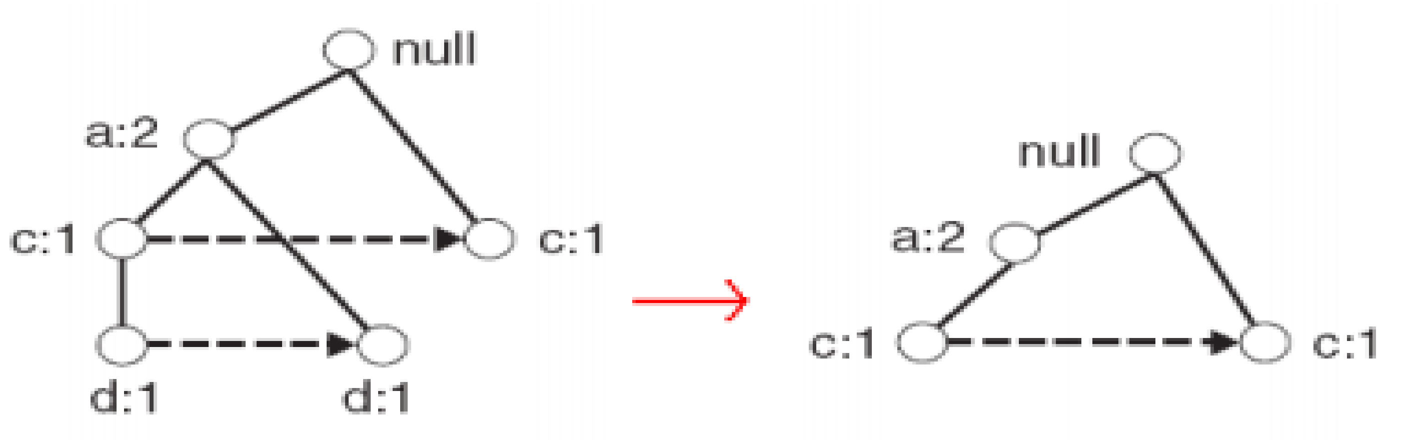
Conditional FP-Tree

- 4. Use the the conditional FP-tree for *e* to find frequent itemsets ending in *de*, *ce* and *ae*
  - Note that *be* is not considered as *b* is not in the conditional FP-tree for *e*.
  - For each of them (e.g. *de*), find the prefix paths from the conditional tree for *e*, extract frequent itemsets, generate conditional FP-tree, etc... (recursive)
  - **Example:** $e \longrightarrow de \longrightarrow ade$ ($\{d, e\}$,$\{a, d, e\}$ are found to be frequent)


null
a:2
c:1
c:1
d:1
d:1
Conditional FP-tree for e
null
a:2
c:1
d:1
d:1
Prefix paths ending in de
null
a:2
Conditional FP-tree for de

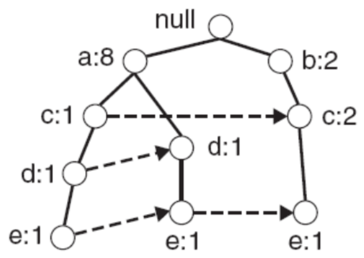
Conditional FP-Tree

- 4. Use the the conditional FP-tree for *e* to find frequent itemsets ending in *de*, *ce* and *ae*
    - **Example:** $e \longrightarrow ce$ ($\{c, e\}$ is found to be frequent)

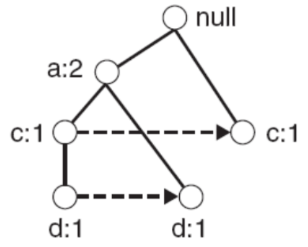


Conditional FP-tree for e Prefix paths ending in ce

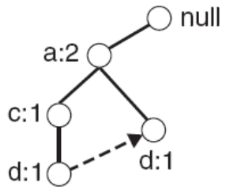
- etc... (*ae*, then do the whole thing for *b*, ... etc)



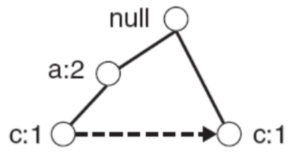
(a) Prefix paths ending in e



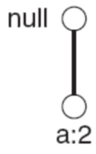
(b) Conditional FP-tree for e



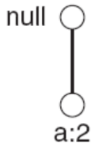
(c) Prefix paths ending in de



(e) Prefix paths ending in ce



(f) Prefix paths ending in ae



(d) Conditional FP-tree for de

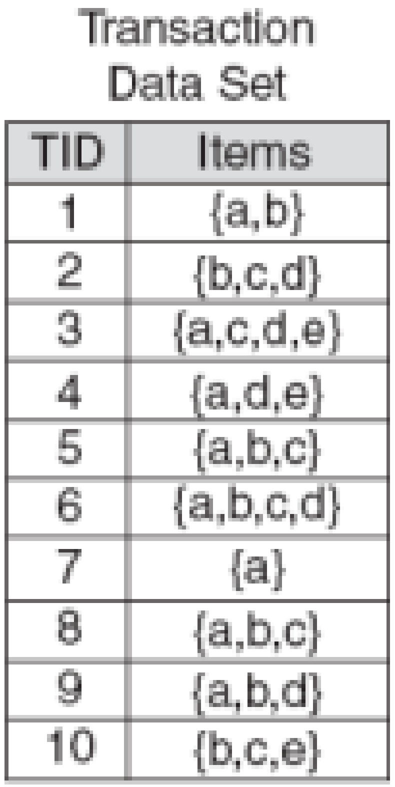
Conditional FP-tree for **ce**

Conditional FP-tree for **ae**

Conditional FP-tree for **ade**

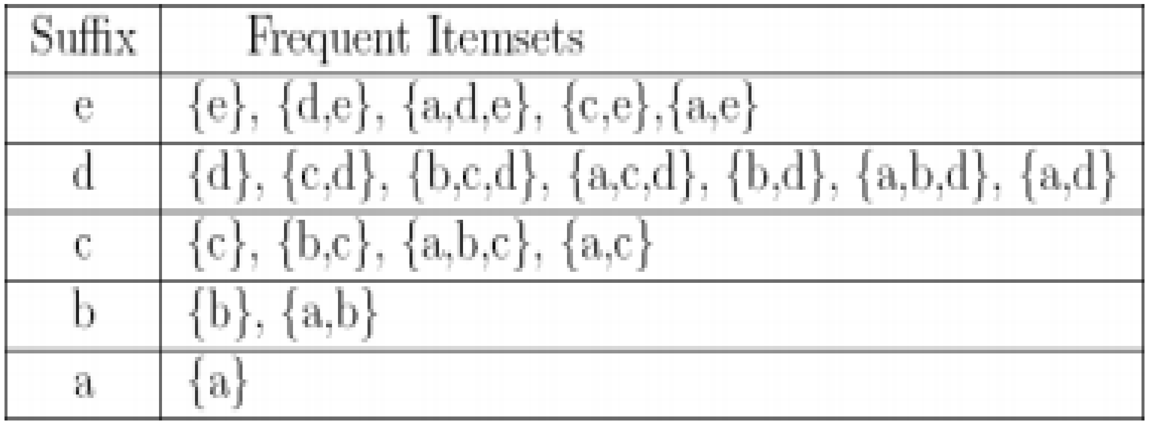
Result

- Frequent itemsets found (ordered by suffix 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}

FP-Growth Algorithm

Algorithm: **FP_growth.** Mine frequent itemsets using an FP-tree by pattern fragment growth.

Input:

- *D*, a transaction database;
- *min_sup*, the minimum support count threshold.

Output: The complete set of frequent patterns.

Method:

1. The FP-tree is constructed in the following steps:

(a) Scan the transaction database *D* once. Collect *F*, the set of frequent items, and their support counts. Sort *F* in support count descending order as *L*, the *list* of frequent items.

(b) Create the root of an FP-tree, and label it as "null." For each transaction *Trans* in *D* do the following.
Select and sort the frequent items in *Trans* according to the order of *L*. Let the sorted frequent item list in *Trans* be $[p|P]$, where p is the first element and P is the remaining list. Call insert_tree($[p|P]$, T), which is performed as follows. If T has a child N such that $N.item\text{-}name = p.item\text{-}name$, then increment N's count by 1; else create a new node N, and let its count be 1, its parent link be linked to T, and its node-link to the nodes with the same *item-name* via the node-link structure. If P is nonempty, call insert_tree(P, N) recursively.

FP-Growth Algorithm

2. The FP-tree is mined by calling FP_growth(*FP_tree*, *null*), which is implemented as follows.

```
procedure FP_growth(Tree, $\alpha$)
(1)   if Tree contains a single path P then
(2)       for each combination (denoted as $\beta$) of the nodes in the path P
(3)           generate pattern $\beta \cup \alpha$ with $support\_count = minimum\ support\ count\ of\ nodes\ in\ \beta$;
(4)   else for each $a_i$ in the header of Tree {
(5)       generate pattern $\beta = a_i \cup \alpha$ with $support\_count = a_i.support\_count$;
(6)       construct $\beta$'s conditional pattern base and then $\beta$'s conditional FP_tree $Tree_\beta$;
(7)       if $Tree_\beta \neq \emptyset$ then
(8)           call FP_growth($Tree_\beta$, $\beta$); }
```

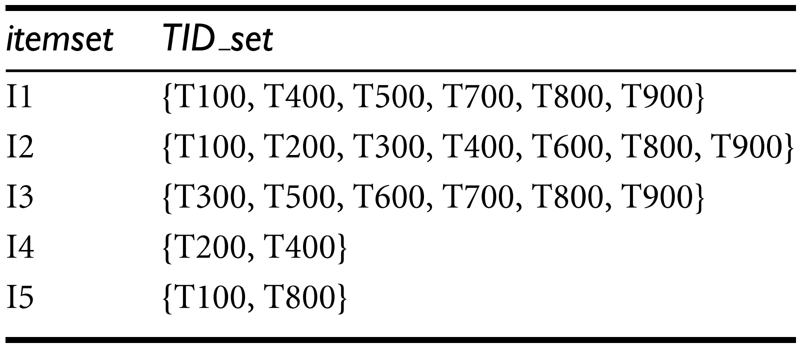
FP-Growth Advantages and Disadvantages

- 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

Mining Frequent Itemsets Using the Vertical Data Format

- Data can be presented in *item-TID set* format (i.e., {*item* : *TID set*}), where *item* is an item name, and *TID set* is the set of transaction identifiers containing the item. This is known as the **vertical data format**.

The Vertical Data Format of the Transaction Data



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 the Vertical Data Format

- Mining can be performed on this data set by **intersecting the TID sets of every pair of frequent single items**. The minimum support count is 2.
- Because every single item is frequent, there are 10 intersections performed in total, which lead to eight nonempty 2-itemsets.
- Notice that because the itemsets {I1, I4} and {I3, I5} each contain only one transaction, they do not belong to the set of frequent 2-itemsets.

2-Itemsets in Vertical Data Format

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 the 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: {I1, I2, I3} and {I1, I2, I5}.
  - By intersecting the TID sets of any two corresponding 2-itemsets, it derives only two frequent 3-itemsets: {I1, I2, I3: 2} and {I1, I2, I5: 2}.

3-Itemsets in Vertical Data Format

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

Mining Frequent Itemsets Using the Vertical Data Format

- First, we transform the horizontally formatted data into the vertical format by scanning the data set once. The support count of an itemset is simply the length of the TID set of the itemset.
- Starting with *k* = 1, the frequent *k*-itemsets can be used to construct the candidate (*k*+1)-itemsets based on the Apriori property.
- The computation is done by intersection of the TID sets of the frequent *k*-itemsets to compute the TID sets of the corresponding (*k*+1)-itemsets.
- This process repeats, with *k* incremented by 1 each time, until no frequent itemsets or candidate itemsets can be found.

Advantages

- What are the potential advantages?
  - One of the merit of this method is that there is no need to scan the database to find the support of (k+1)-itemsets (for $k \geq 1$).
  - This is because the TID set of each k-itemset carries the complete information required for counting such support.

Disadvantages

- What are potential disadvantages?
  - The TID sets can be quite long, taking substantial memory space as well as computation time for intersecting the long sets.

Pattern Evaluation

- Association rule algorithms can produce large number of rules
- Interestingness measures can be used to prune/rank the patterns
  - In the original formulation, support & confidence are the only measures used

Computing Interestingness Measure

- Given X → Y or {X,Y}, information needed to compute interestingness can be obtained from a contingency table

A contingency table can be used to tabulate the frequency counts

	Y	$\overline{Y}$	
X	f_{11}	f_{10}	f_{1+}
$\overline{X}$	f_{01}	f_{00}	f_{o+}
	f_{+1}	f_{+0}	N

f_{11}: Support of X and Y
f_{10}: Support of X and $\overline{Y}$
f_{01}: Support of $\overline{X}$ and Y
f_{00}: Support of $\overline{X}$ and $\overline{Y}$
f_{1+}: Row sum represents support for X

f_{+1}: Column sum represents support for Y

Drawback of Confidence

Customers	Tea	Coffee	…
C1	0	1	…
C2	1	0	…
C3	1	1	…
C4	1	0	…
…			

	Coffee	$\overline{\text{Coffee}}$	
Tea	150	50	200
$\overline{\text{Tea}}$	650	150	800
	800	200	1000

Association Rule: Tea $\rightarrow$ Coffee

Confidence = 150/200 = 0.75

Confidence > 50%, meaning people who drink tea are more likely to drink coffee than not drink coffee

So rule seems reasonable..

Drawback of Confidence

	Coffee	$\overline{\text{Coffee}}$	
Tea	150	50	200
$\overline{\text{Tea}}$	650	150	800
	800	200	1000

Association Rule: Tea → Coffee

Confidence= 15/20 = 0.75

But P(Coffee) = 0.8, i.e fraction of people who drink coffee, regardless of whether they drink tea.

Thus, knowing that a person is a tea drinker actually decreases her probability of being a coffee drinker from 80% to 75%.

The rule is therefore misleading despite its high confidence value.

What’s the pitfall of confidence measure?

Ignores the support of the itemset in the rule consequent!

Introduce new metric called lift

$$lift(B,C) = \frac{c(B \rightarrow C)}{s(C)} = \frac{s(B \cup C)}{s(B) \times s(C)}$$

Exercise

Create a FP tree for the given transactions:

A C E B F

A C G

E

A C E G D

A C E G

E

A C E B F

A C D

A C E G

A C E G

FP-Tree after reading 1st transaction

A C E B F

A C G

E

A C E G D

A C E G

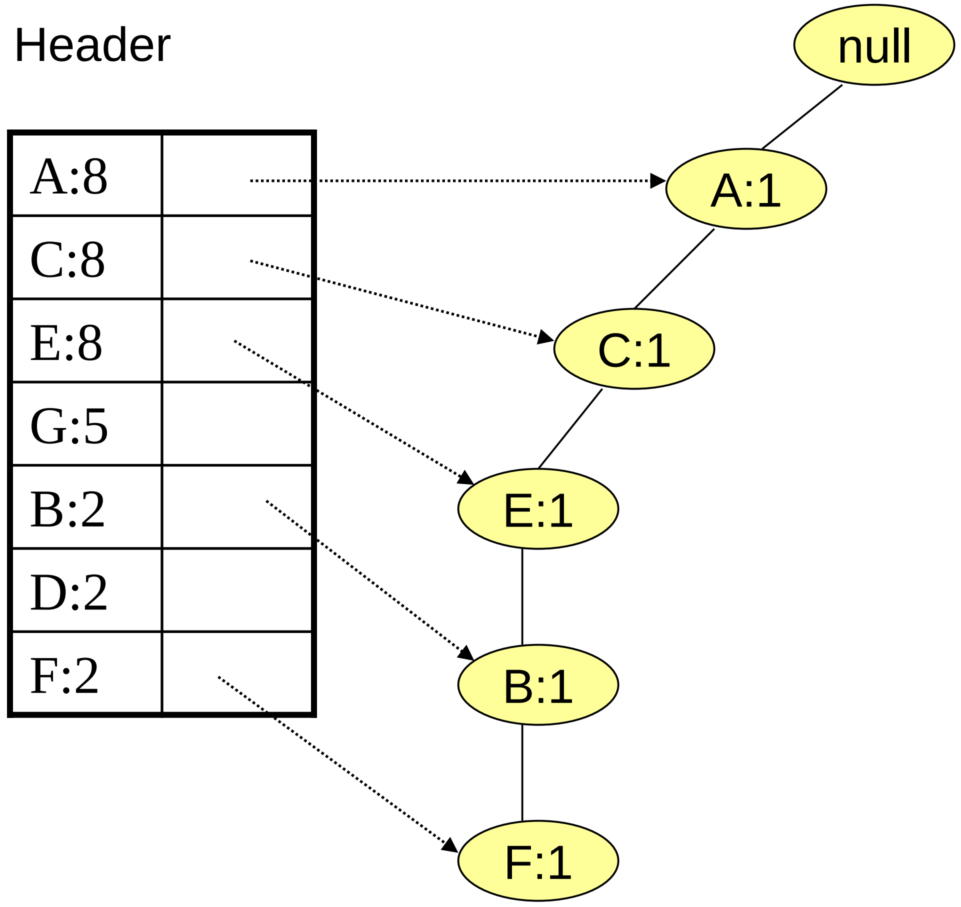
E

A C E B F

A C D

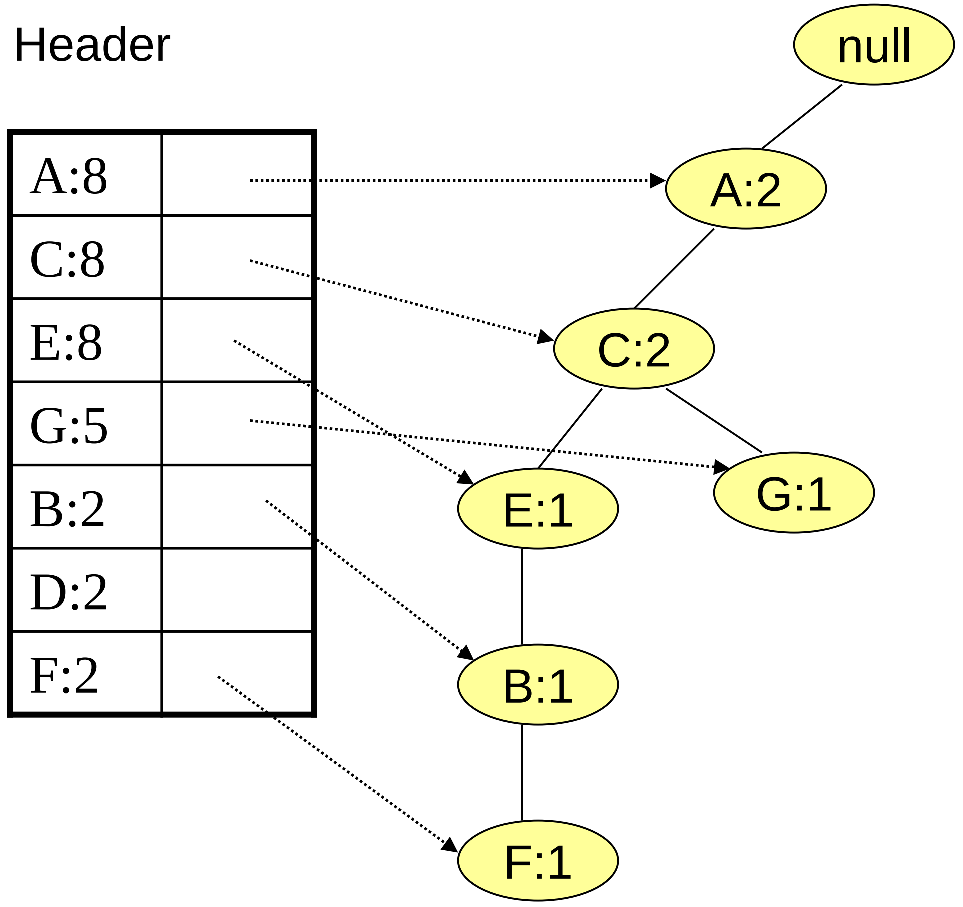
A C E G

A C E G



FP-Tree after reading 2nd transaction

A C E B F
A C G
E
A C E G D
A C E G
E
A C E B F
A C D
A C E G
A C E G



FP-Tree after reading 3rd transaction

A C E B F

A C G

E

A C E G D

A C E G

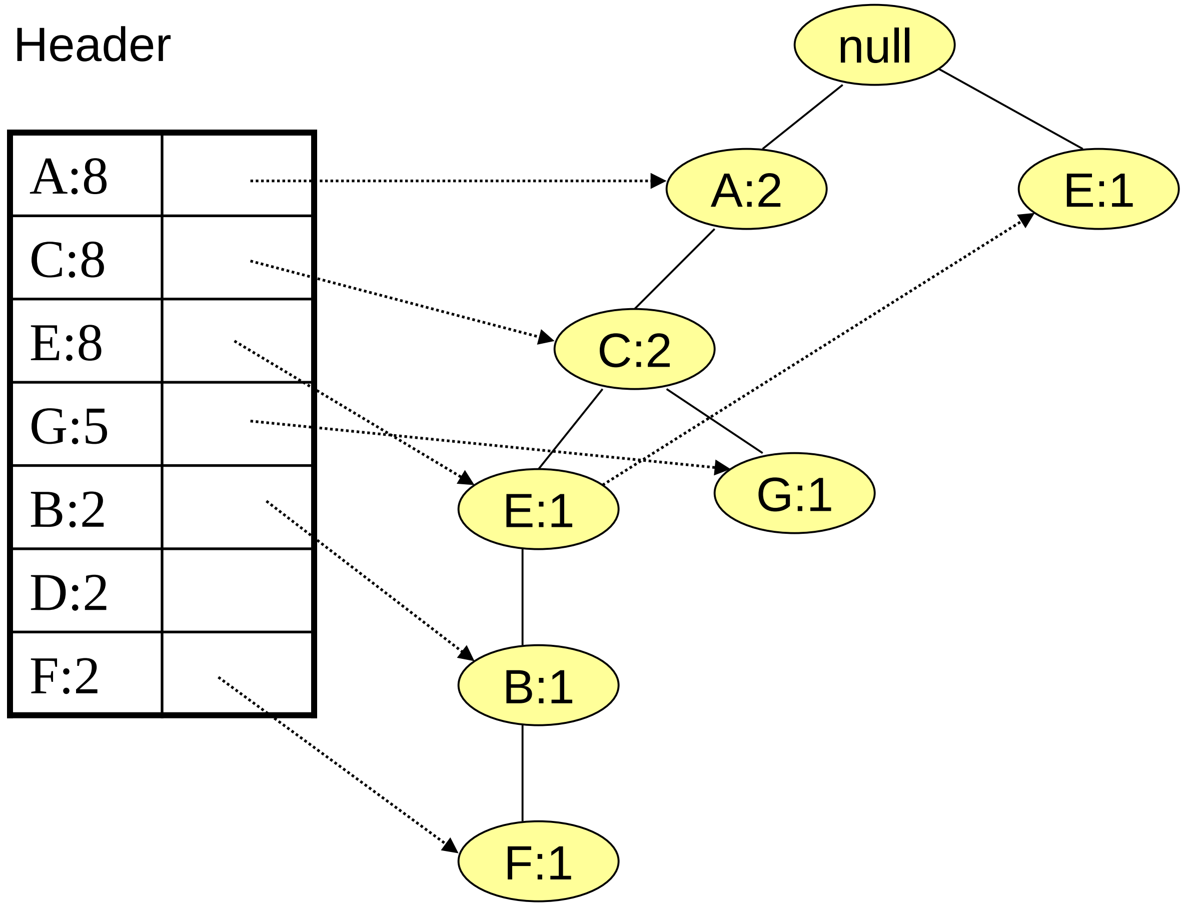
E

A C E B F

A C D

A C E G

A C E G



FP-Tree after reading 4th transaction

A C E B F

A C G

E

A C E G D

A C E G

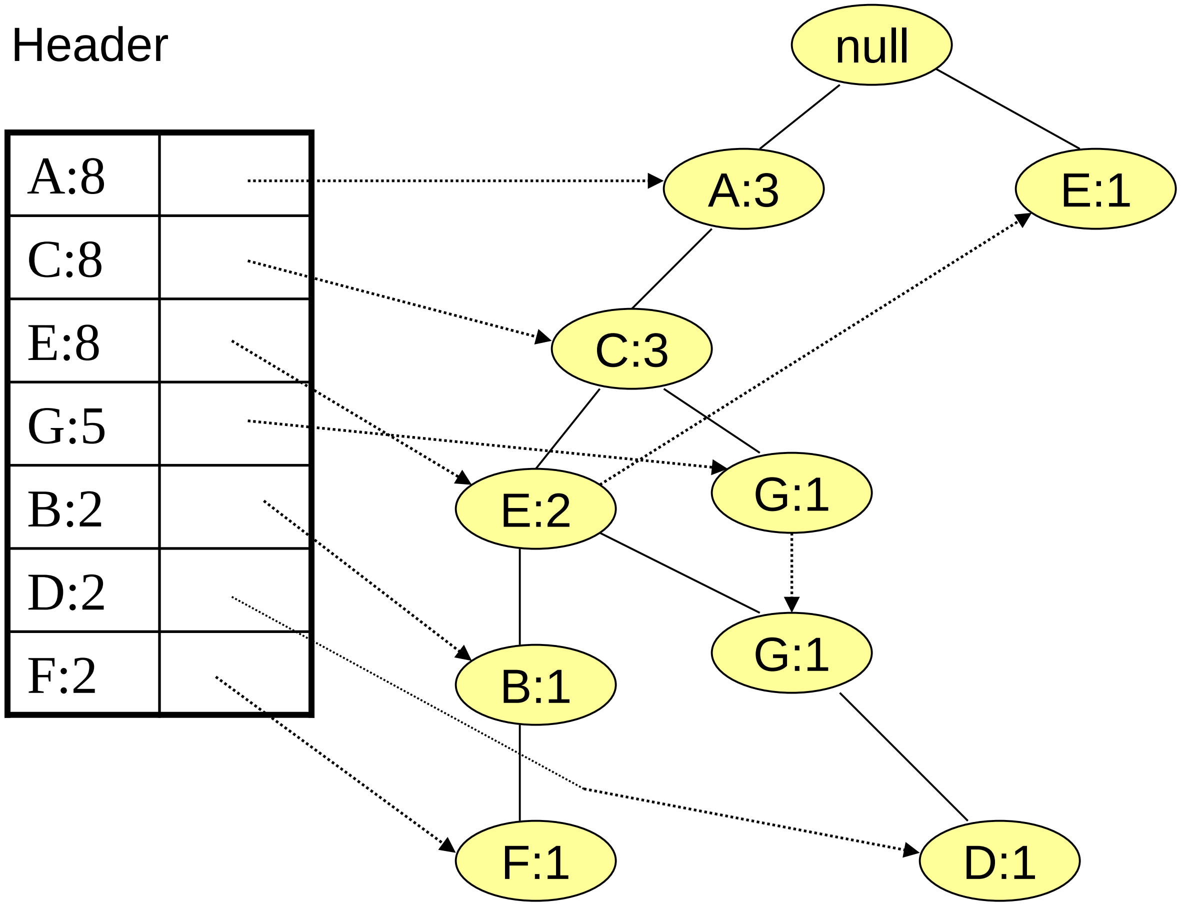
E

A C E B F

A C D

A C E G

A C E G



FP-Tree after reading 5th transaction

A C E B F

A C G

E

A C E G D

A C E G

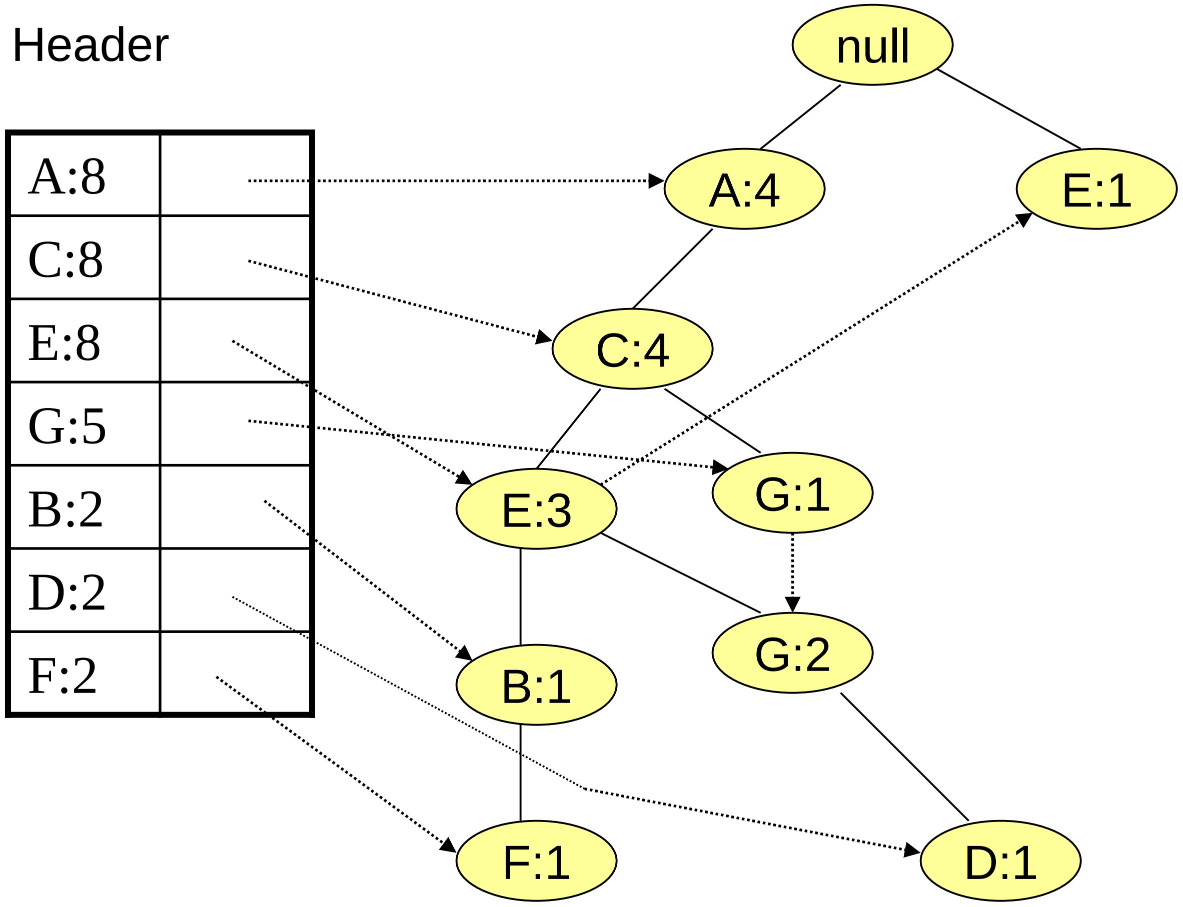
E

A C E B F

A C D

A C E G

A C E G



FP-Tree after reading 6th transaction

A C E B F

A C G

E

A C E G D

A C E G

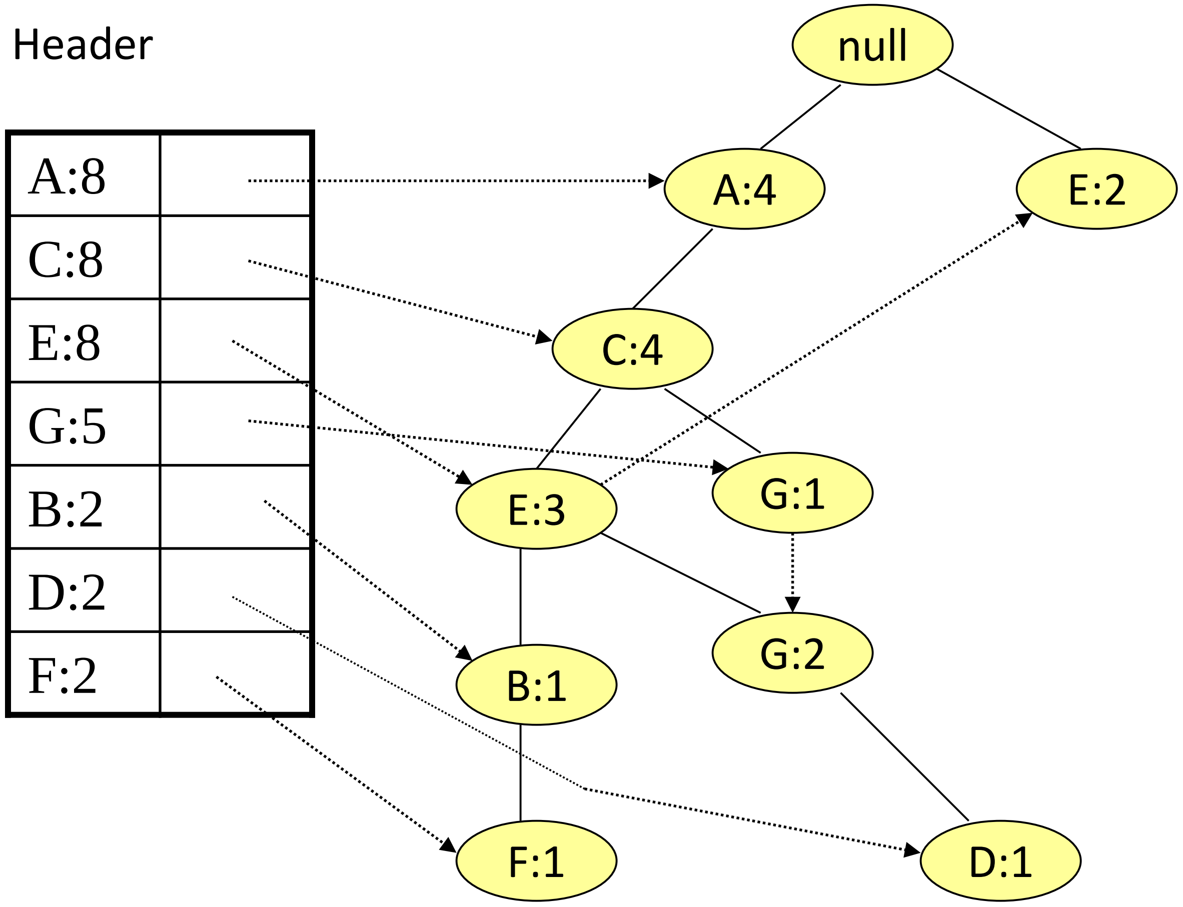
E

A C E B F

A C D

A C E G

A C E G



FP-Tree after reading 7th transaction

A C E B F

A C G

E

A C E G D

A C E G

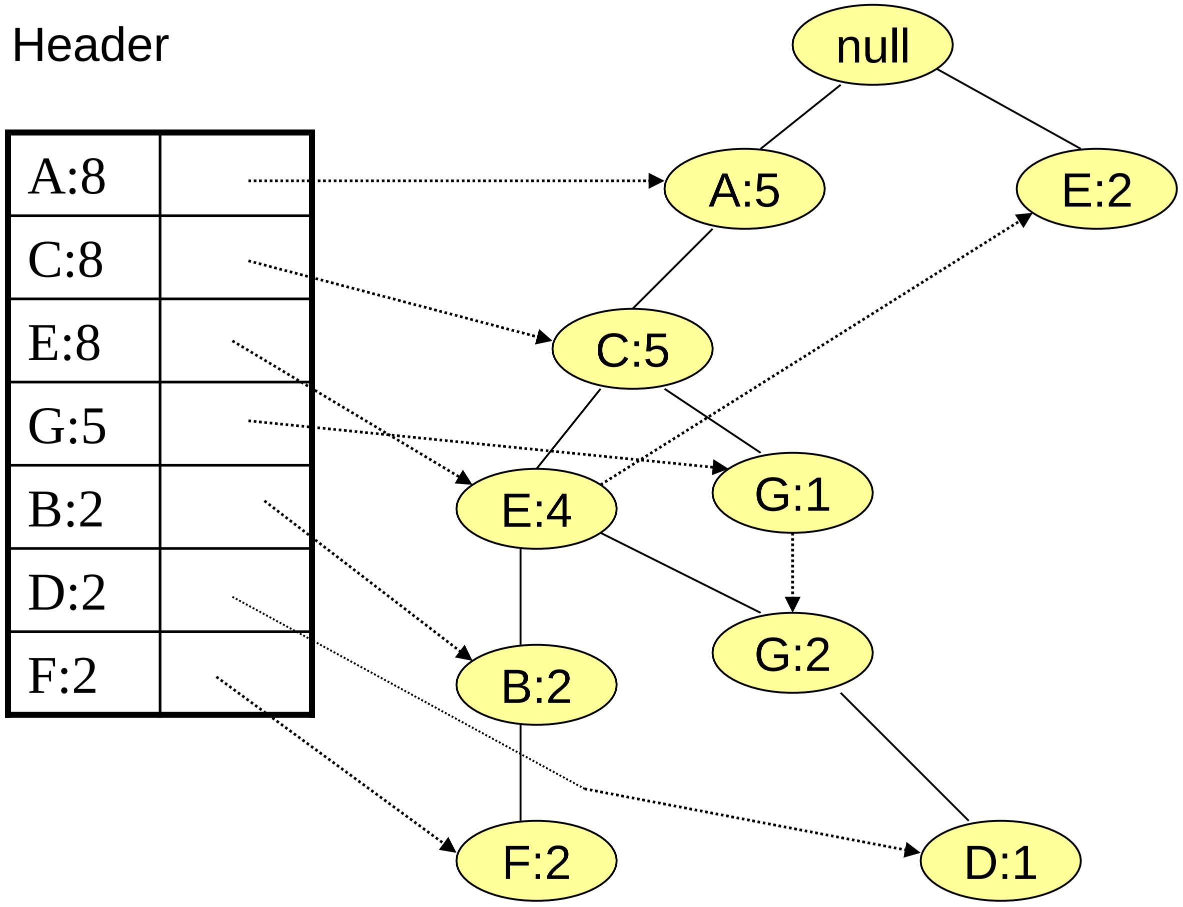
E

A C E B F

A C D

A C E G

A C E G



FP-Tree after reading 8th transaction

A C E B F

A C G

E

A C E G D

A C E G

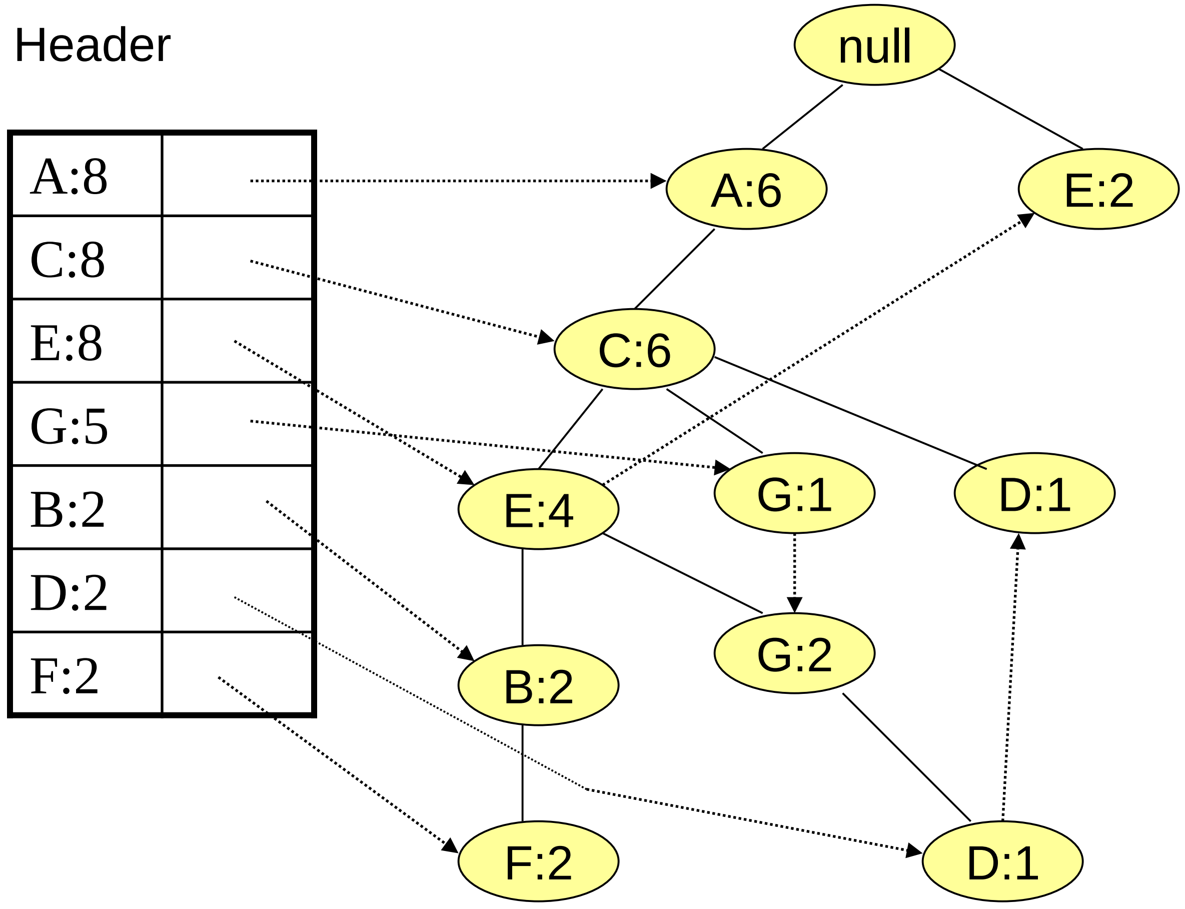
E

A C E B F

A C D

A C E G

A C E G



FP-Tree after reading 9th transaction

A C E B F

A C G

E

A C E G D

A C E G

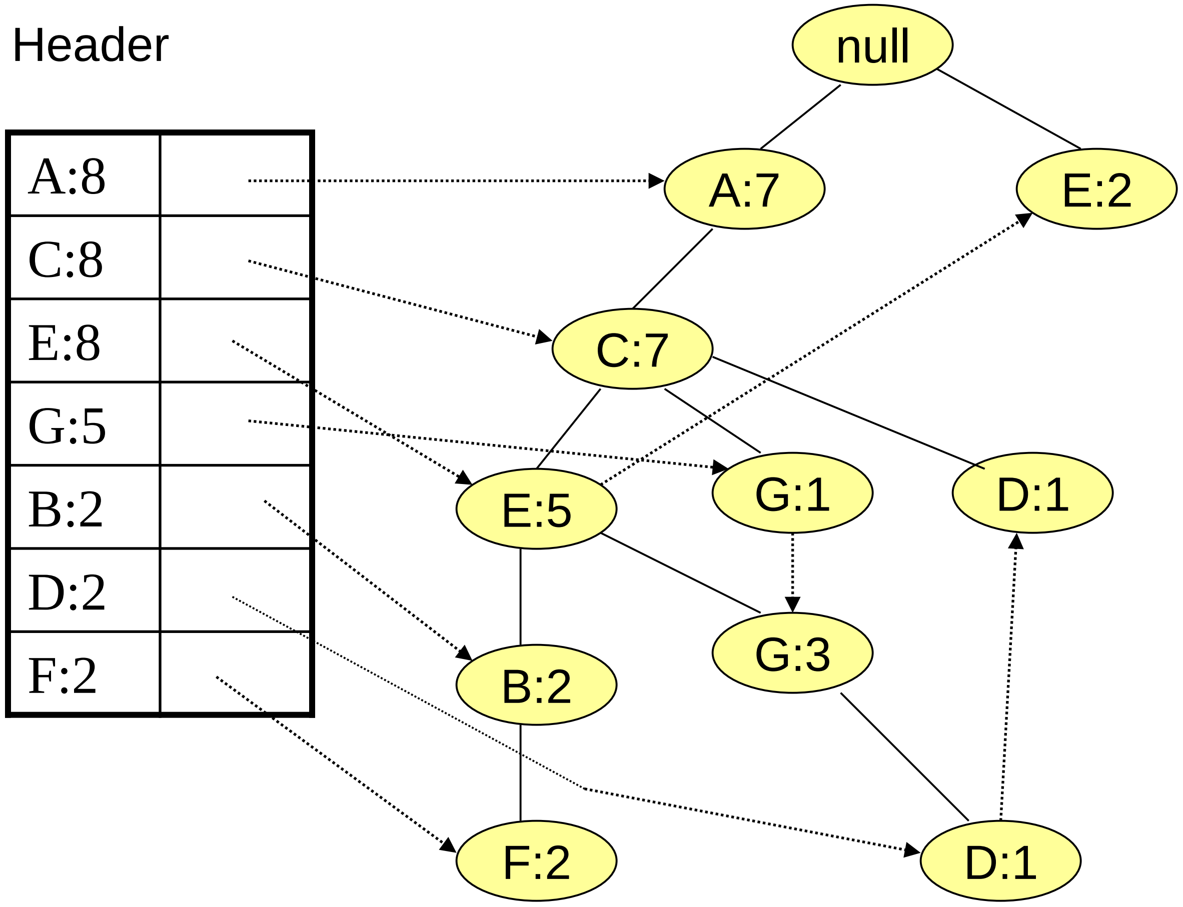
E

A C E B F

A C D

A C E G

A C E G



FP-Tree after reading 10th transaction

A C E B F

A C G

E

A C E G D

A C E G

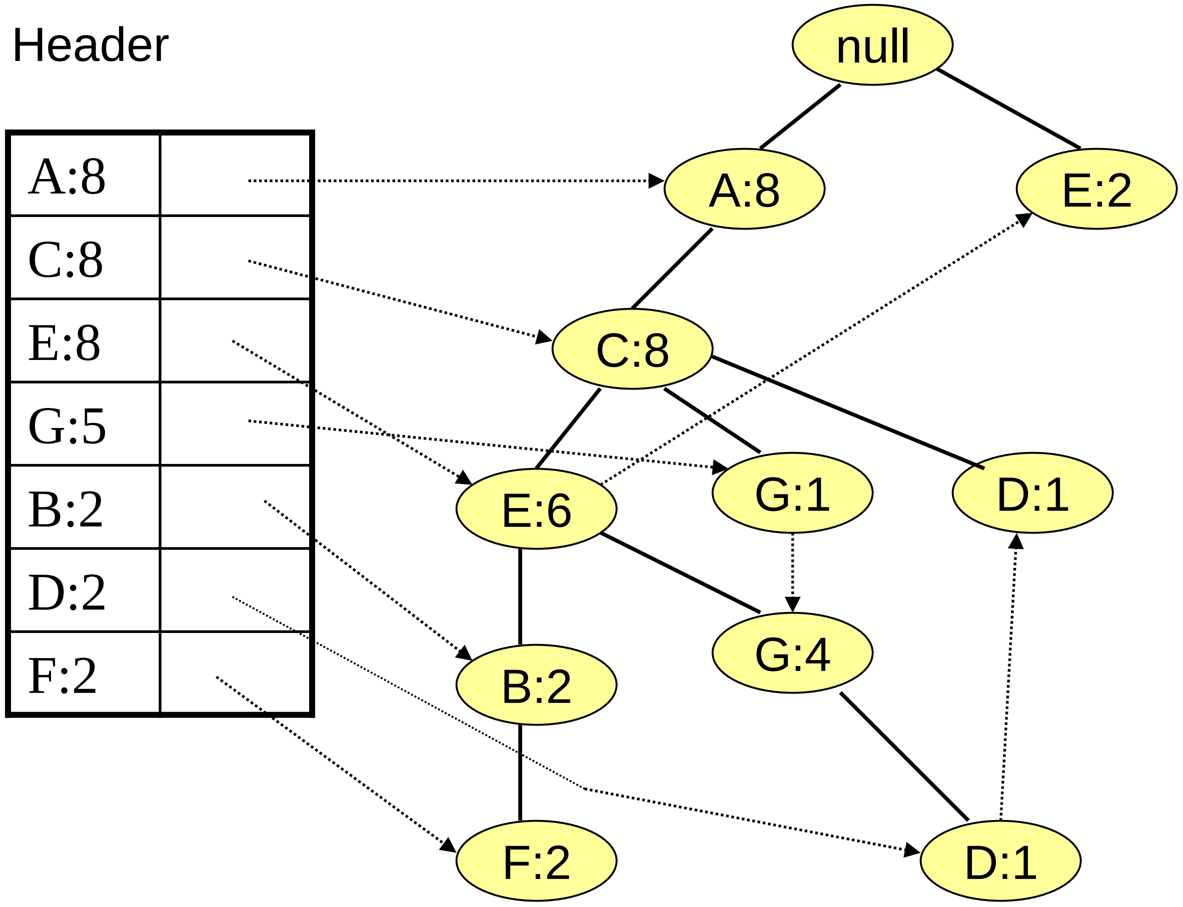
E

A C E B F

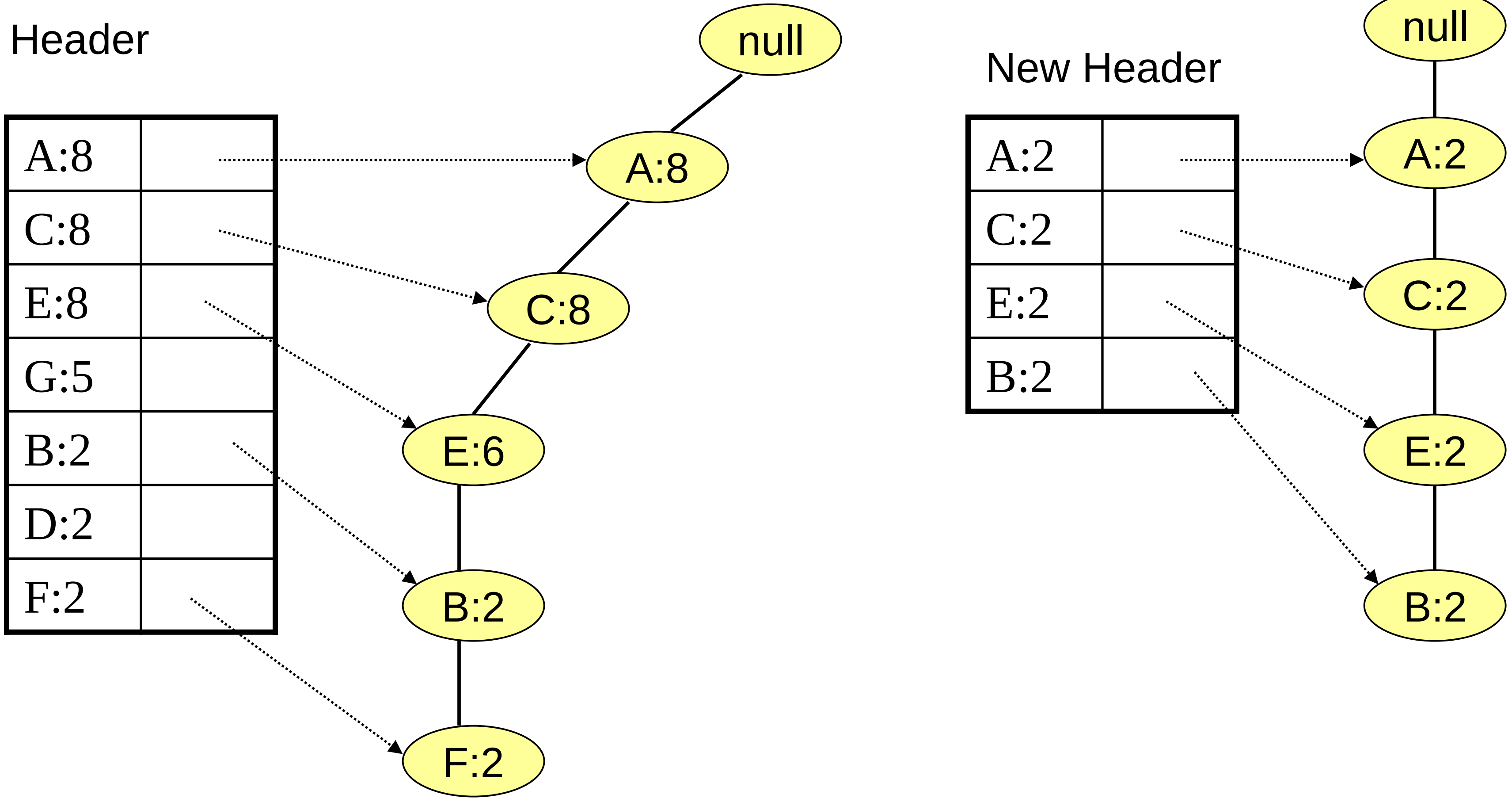
A C D

A C E G

A C E G



Conditional FP-Tree for F

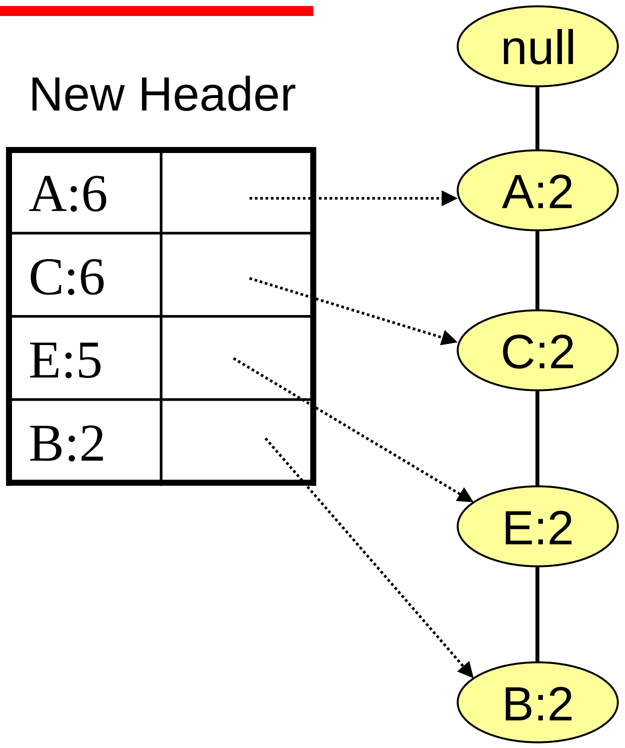


There is only a single path containing F

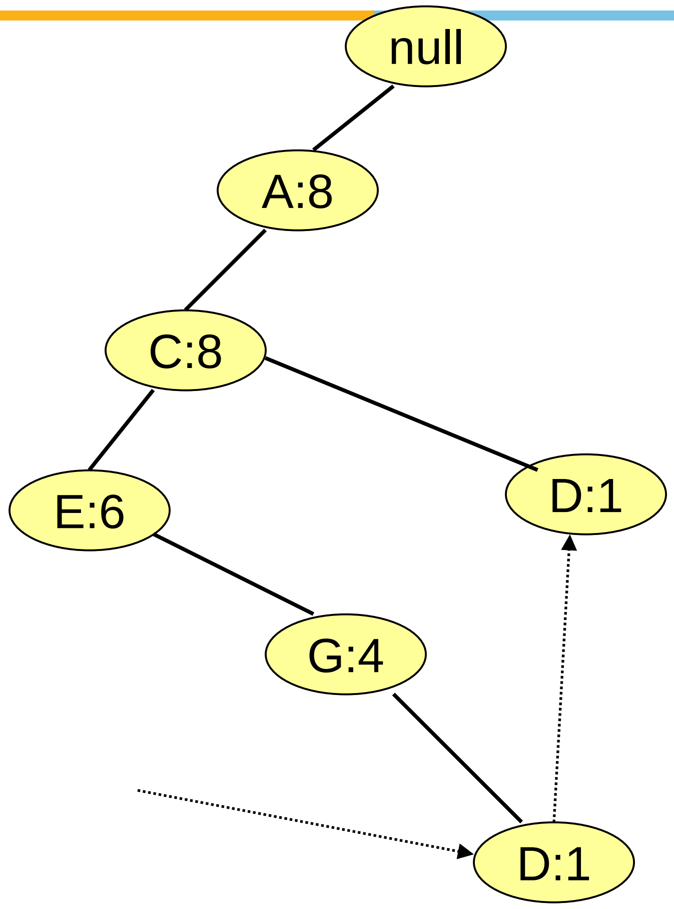
Recursion

- We continue recursively on the conditional FP-Tree for F.
- However, when the tree is just a single path it is the base case for the recursion.
- So, we just produce all the subsets of the items on this path merged with F.

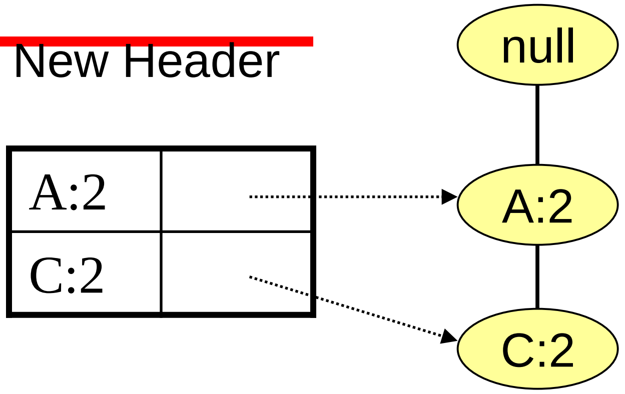
{F} {A,F} {C,F} {E,F} {B,F} {A,C,F}, …, {A,C,E,F}



Conditional FP-Tree for D



Paths containing D after updating the counts



The other items are removed as infrequent.

The tree is just a single path; it is the base case for the recursion.

So, we just produce all the subsets of the items on this path merged with D.

{D} {A,D} {C,D} {A,C,D}

Exercise: Complete the example.

Thanks!

Next Lecture:

- Advanced concepts in Association Analysis

Readings:

- Chapter 6 - Tan & Kumar
- Chapter 6 - Han & Kamber