Association Analysis (3)

FP-Tree/FP-Growth Algorithm

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

Building the FP-Tree

1. Scan data to determine the support count of each item.

   Infrequent items are discarded, while the frequent items are sorted in decreasing support counts.

2. Make a second pass over the data to construct the FPtree.

   As the transactions are read, before being processed, their items are sorted according to the above order.

First scan – determine frequent 1-itemsets, then build header

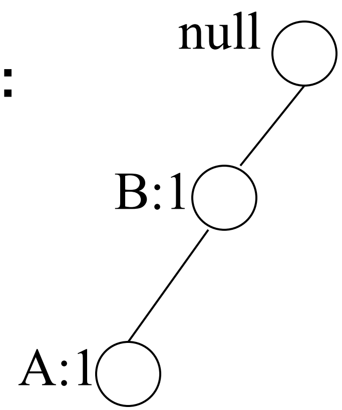
TID	Items
1	{A,B}
2	{B,C,D}
3	{A,C,D,E}
4	{A,D,E}
5	{A,B,C}
6	{A,B,C,D}
7	{B,C}
8	{A,B,C}
9	{A,B,D}
10	{B,C,E}

B	8
A	7
C	7
D	5
E	3

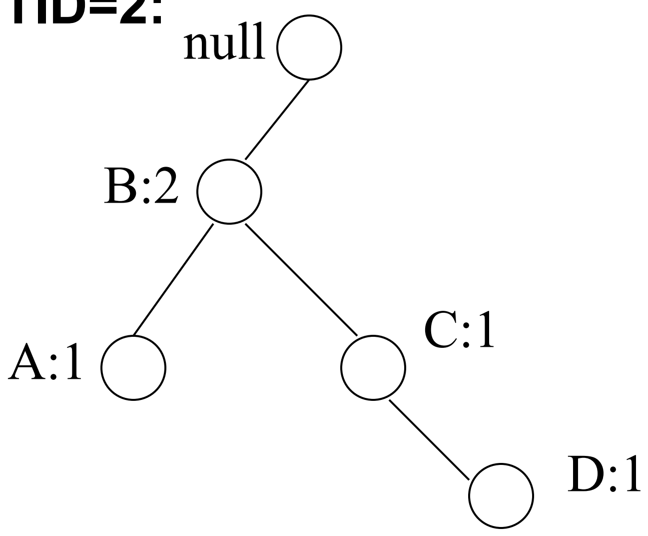
FP-tree construction

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	{B,C}
8	{A,B,C}
9	{A,B,D}
10	{B,C,E}

After reading TID=1:



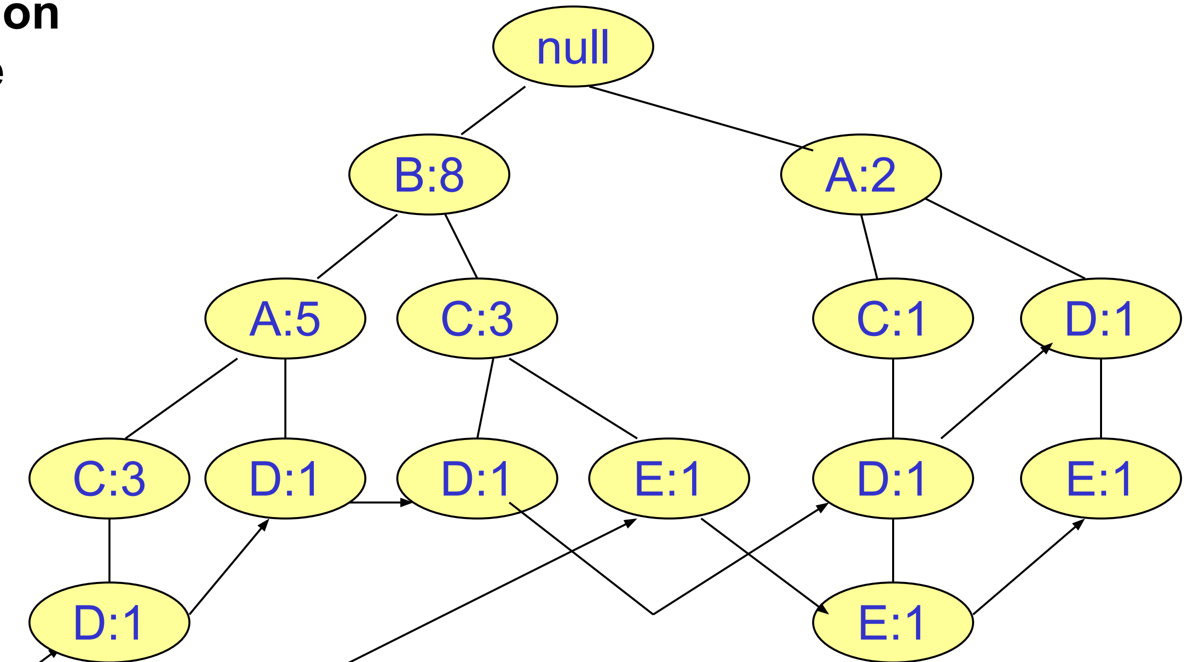
After reading TID=2:



FP-Tree Construction

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	{B,C}
8	{A,B,C}
9	{A,B,D}
10	{B,C,E}

Transaction Database



Header table

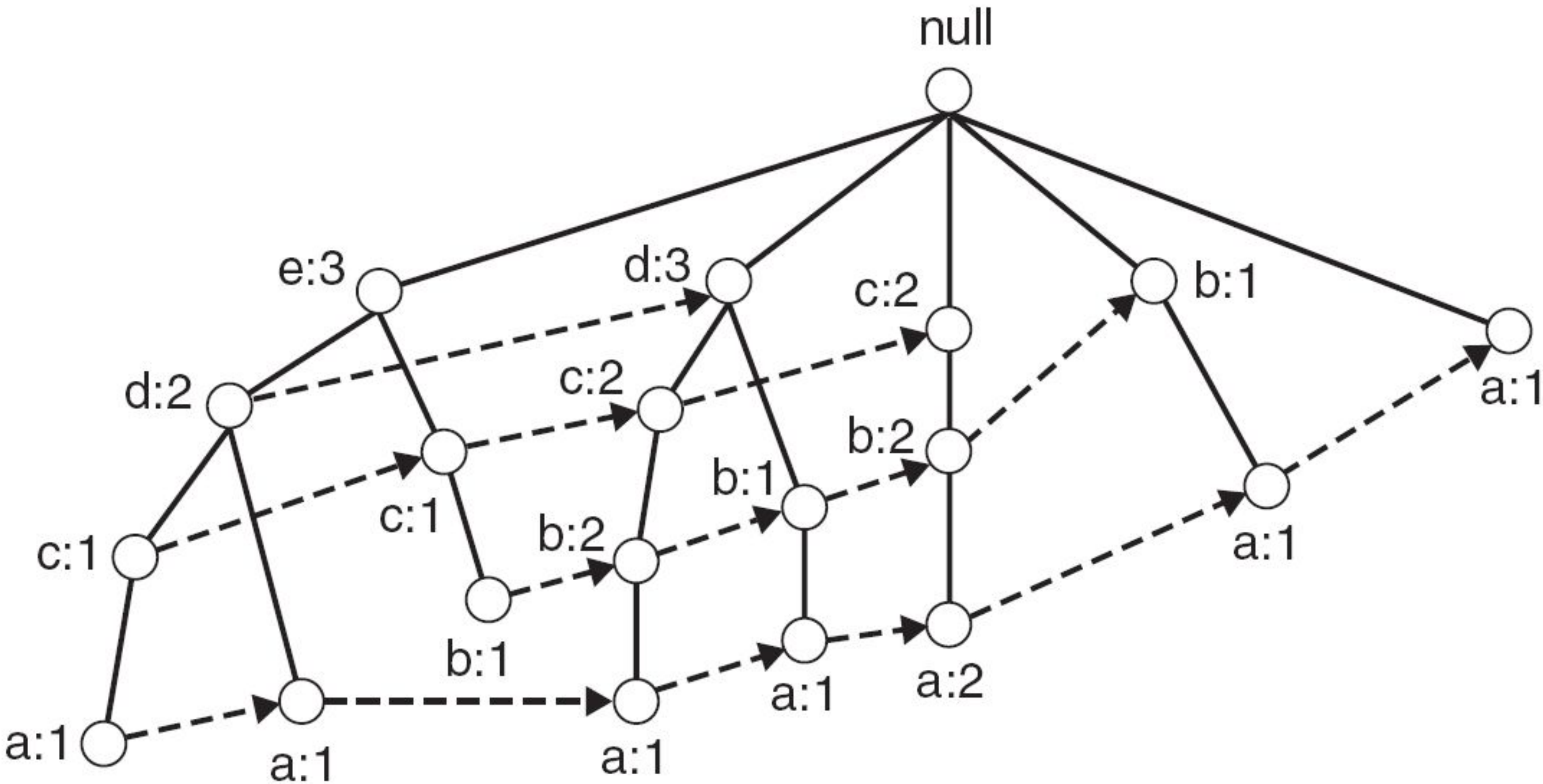
Item	Pointer
B	8
A	7
C	7
D	5
E	3

Chain pointers help in quickly finding all the paths of the tree containing some given item.

FP-Tree size

- The size of an FPtree is typically smaller than the size of the uncompressed data because many transactions often share a few items in common.
- **Bestcase** scenario:
  - All transactions have the same set of items, and the FPtree contains only a single branch of nodes.
- **Worstcase** scenario:
  - Every transaction has a unique set of items.
  - As none of the transactions have any items in common, the size of the FP tree is effectively the same as the size of the original data.

- The size of an FPtree also depends on how the items are ordered.
  - If the ordering scheme in the preceding example is reversed,
    - i.e., from lowest to highest support item, the resulting FPtree probably is denser (shown in next slide).
    - Not always though…ordering is just a heuristic.



An FPtree representation for the data set with a different item ordering scheme.

FP-Growth (I)

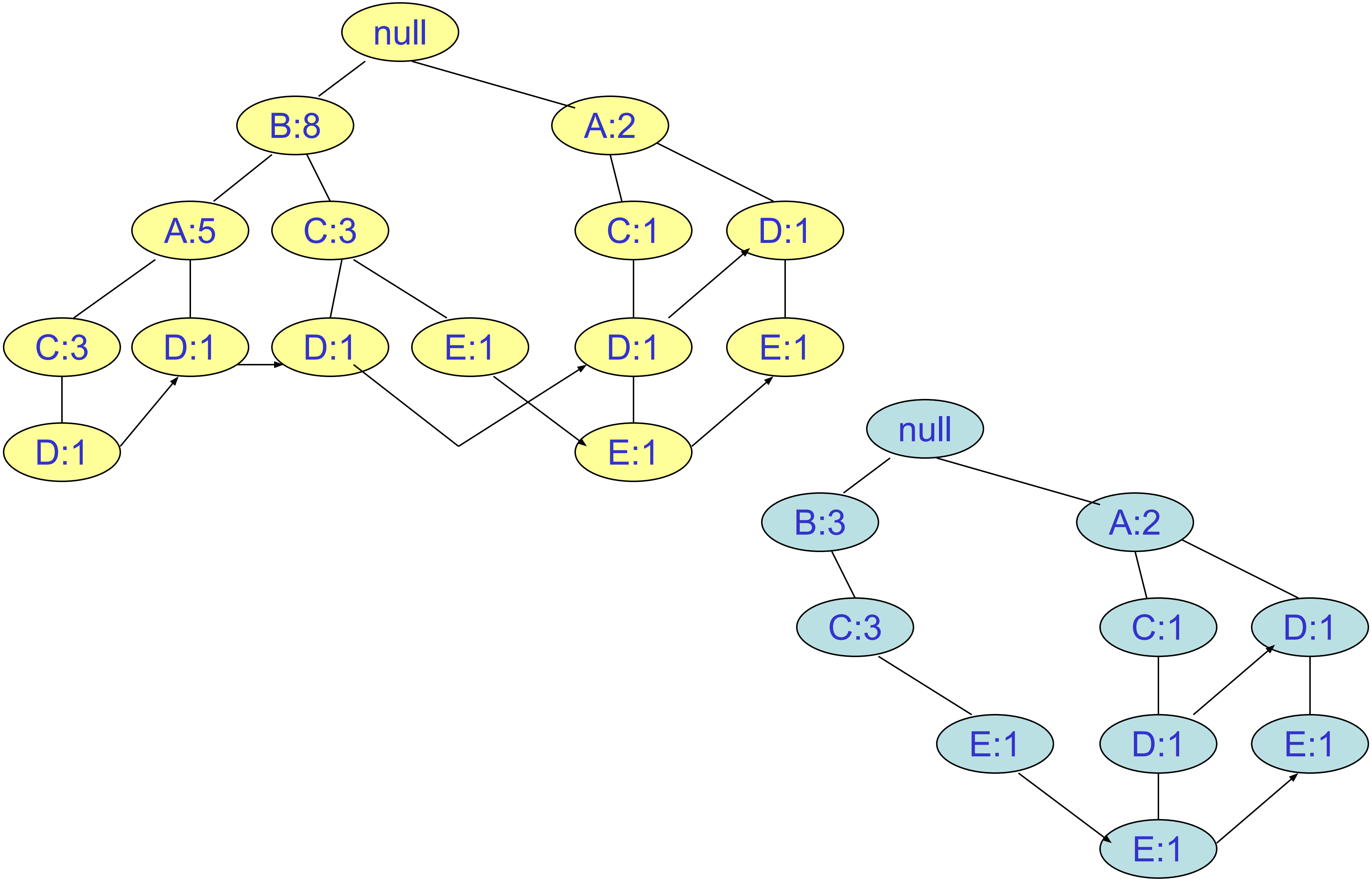
- FPgrowth generates frequent itemsets from an FPtree by exploring the tree in a bottomup fashion.

- Given the example tree, the algorithm looks for frequent itemsets ending in **E** first, followed by **D**, **C**, **A**, and finally, **B**.

- Since every transaction is mapped onto a path in the FPtree, we can derive the frequent itemsets ending with a particular item, say, **E**, by examining only the paths containing node **E**.

- These paths can be accessed rapidly using the pointers associated with node **E**.

Paths containing node E



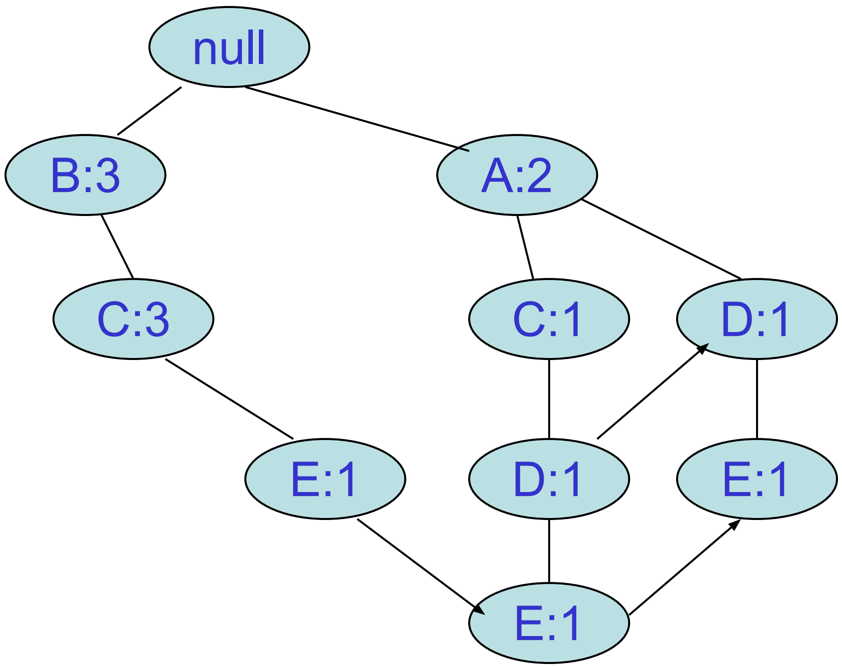
Conditional FP-Tree for E

- We now need to build a conditional FP-Tree for E, which is the tree of itemsets ending in E.

- It is not the tree obtained in previous slide as result of deleting nodes from the original tree.

- Why? Because the order of the items change.
  - In this example, C has a higher count than B.

Conditional FP-Tree for E

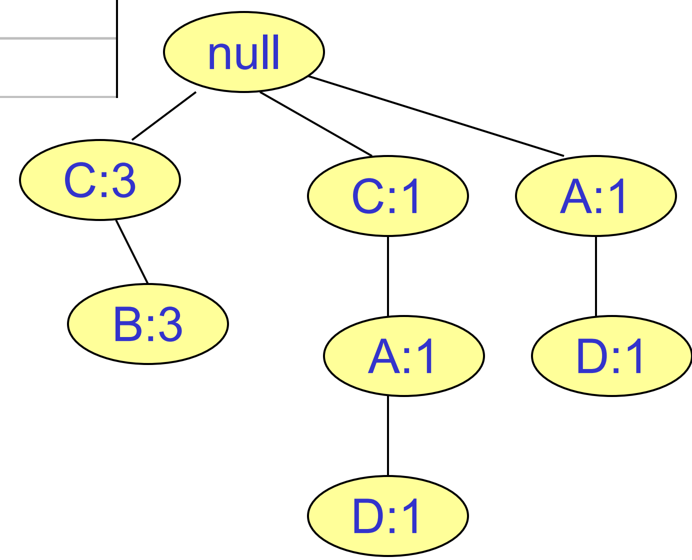


Header table

Item	Pointer
C	4
B	3
A	2
D	2

The new header

The conditional FP-Tree for E



The set of paths containing E.

Insert each path (after truncating E) into a new tree.

Adding up the counts for D we get 2, so {E,D} is frequent itemset.

We continue recursively.
Base of recursion: When the tree has a single path only.

FP-Tree Another Example

Transactions
A B C E F O
A C G
E I
A C D E G
A C E G L
E J
A B C E F P
A C D
A C E G M
A C E G N

Freq. 1-Itemsets.
Supp. Count ≥2

A:8	
C:8	
E:8	
G:5	
B:2	
D:2	
F:2	

Transactions with items sorted based on frequencies, and ignoring the infrequent items.

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

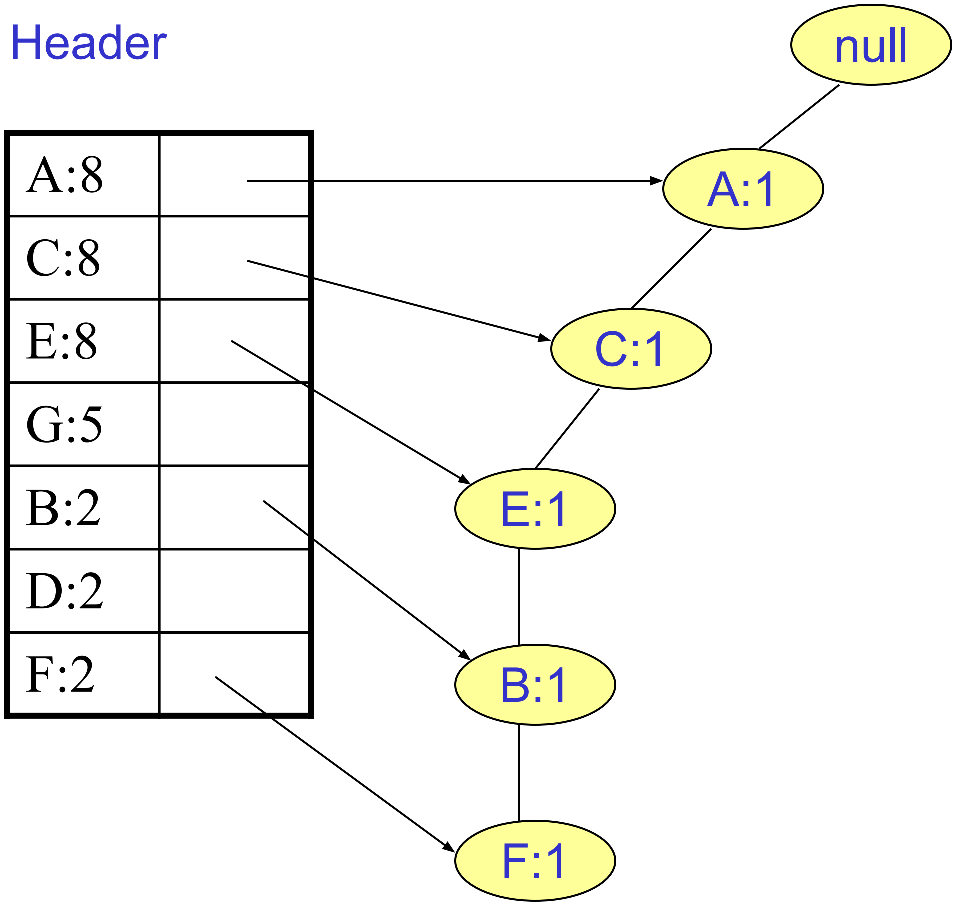
E

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

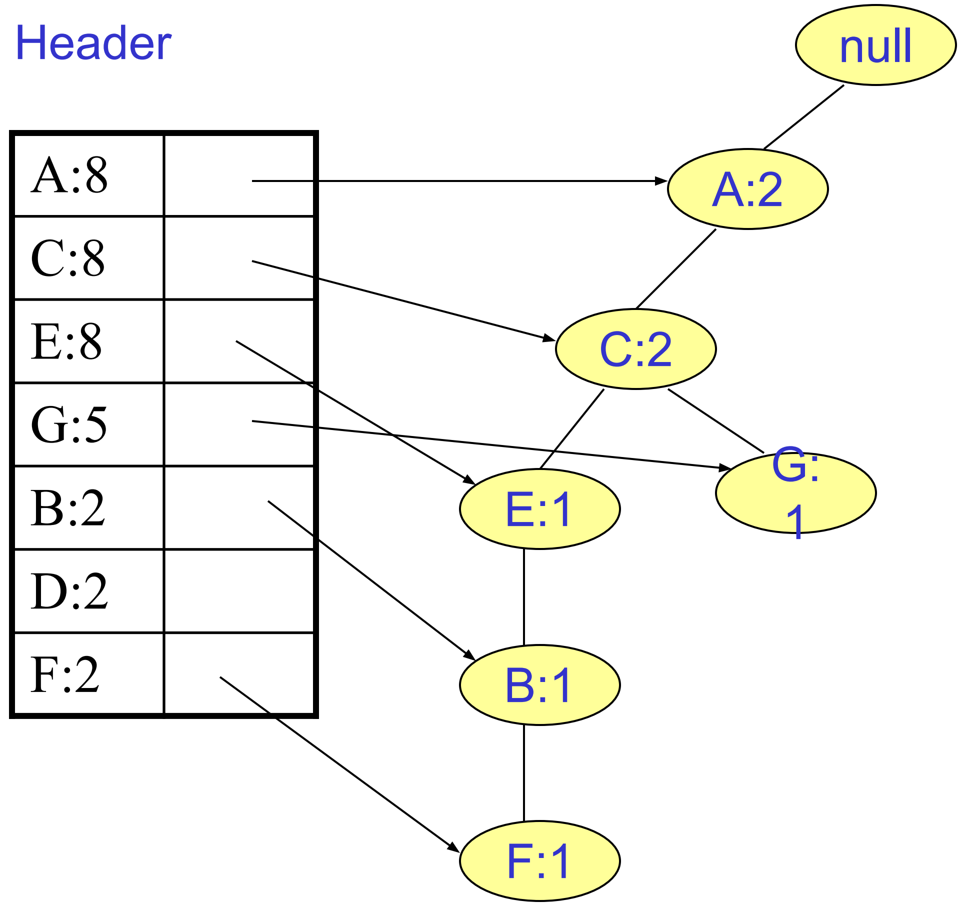
E

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

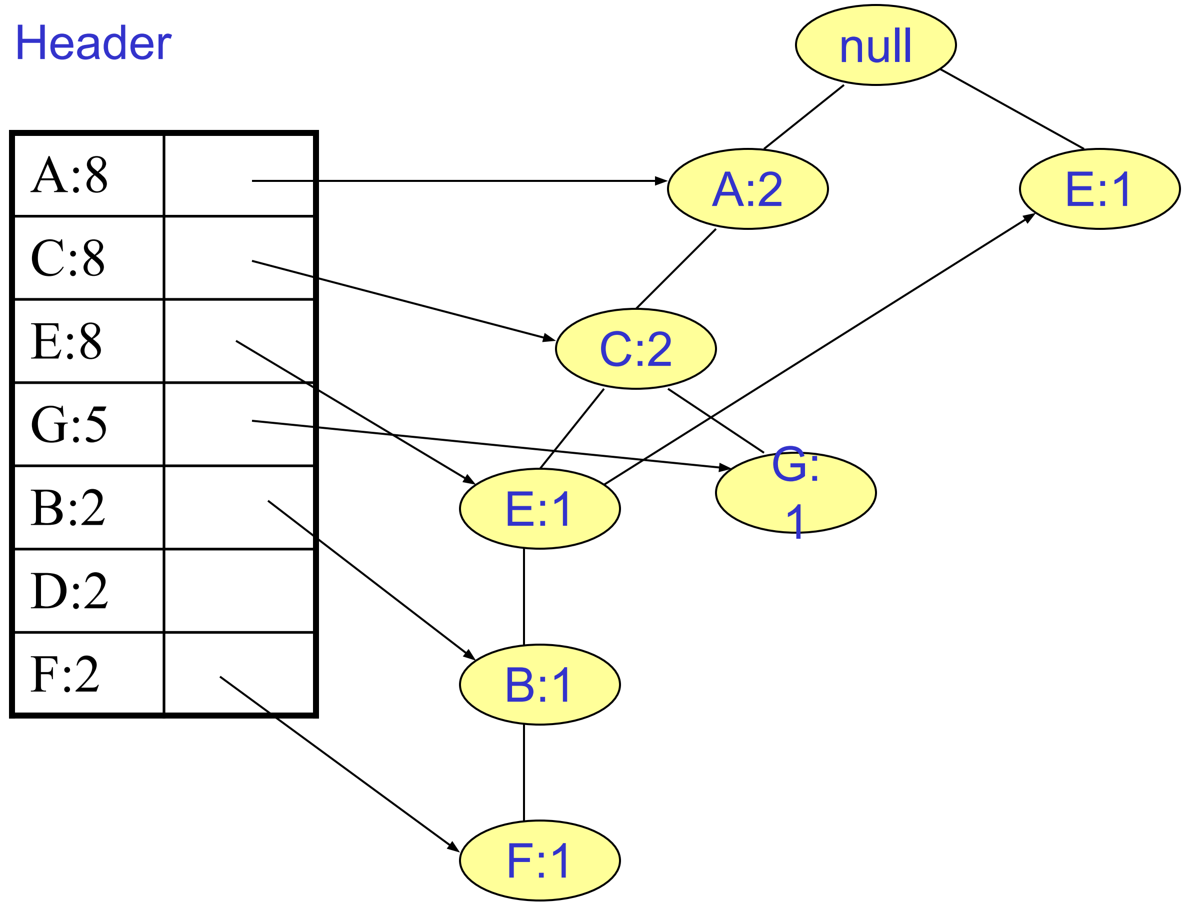
E

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

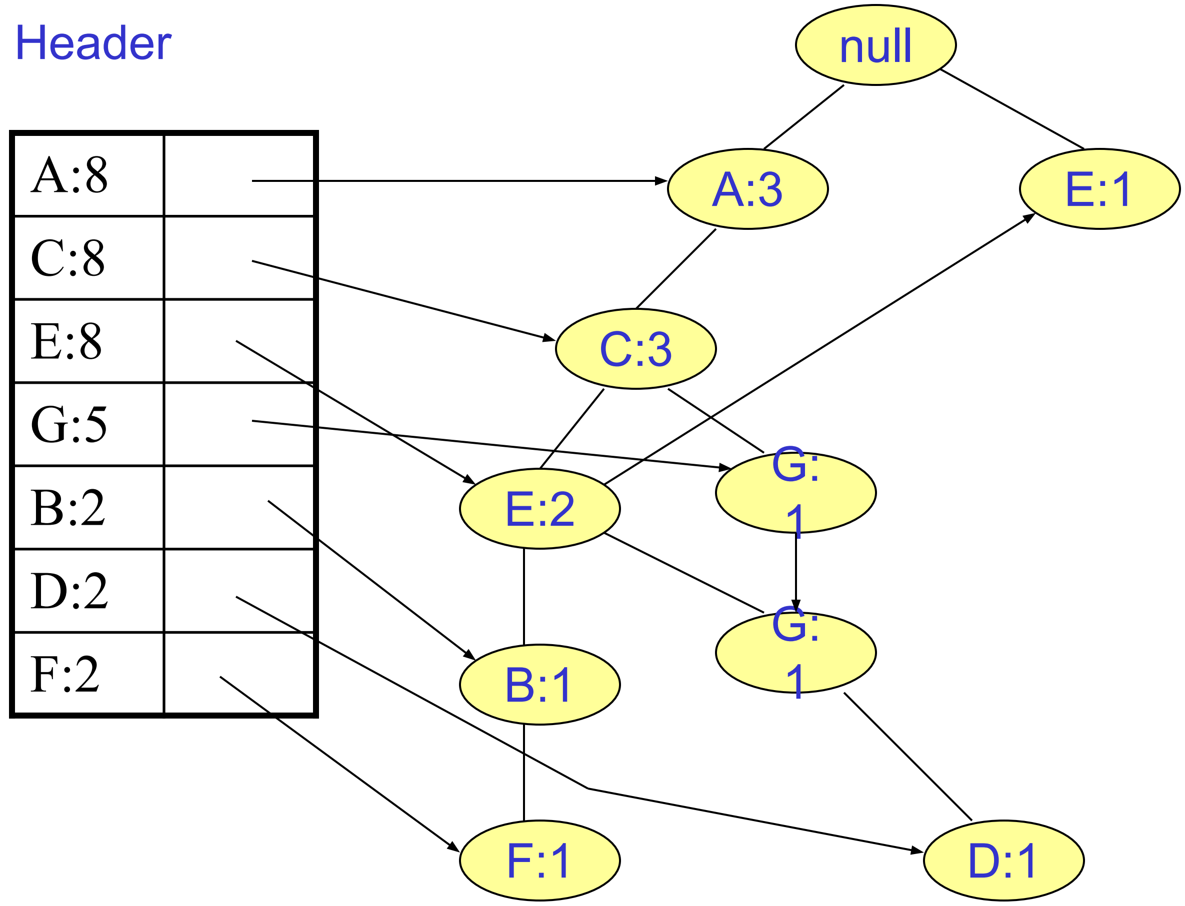
E

A C E B F

A C D

A C E G

A C E G



FP-Tree after reading 5^{th} transaction

A C E B F

A C G

E

A C E G D

A C E G

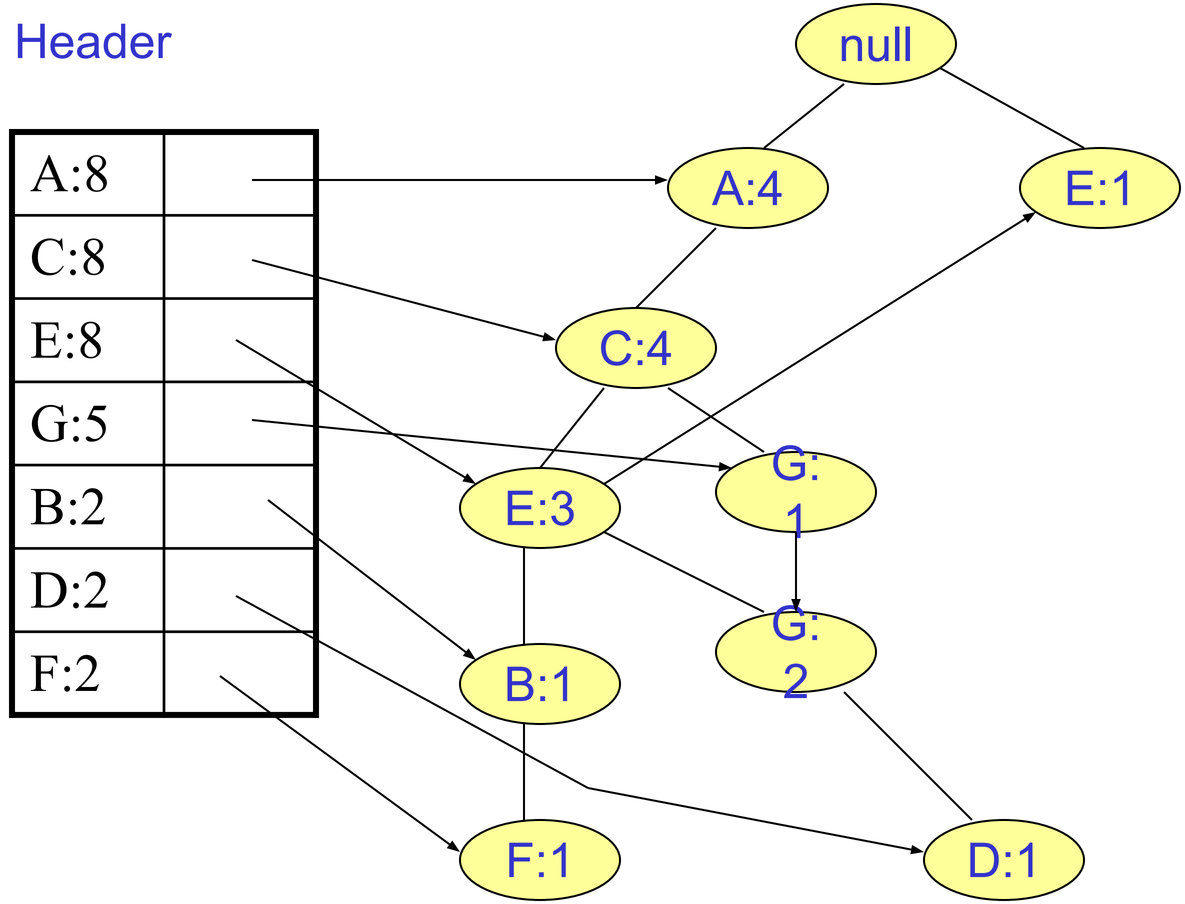
E

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

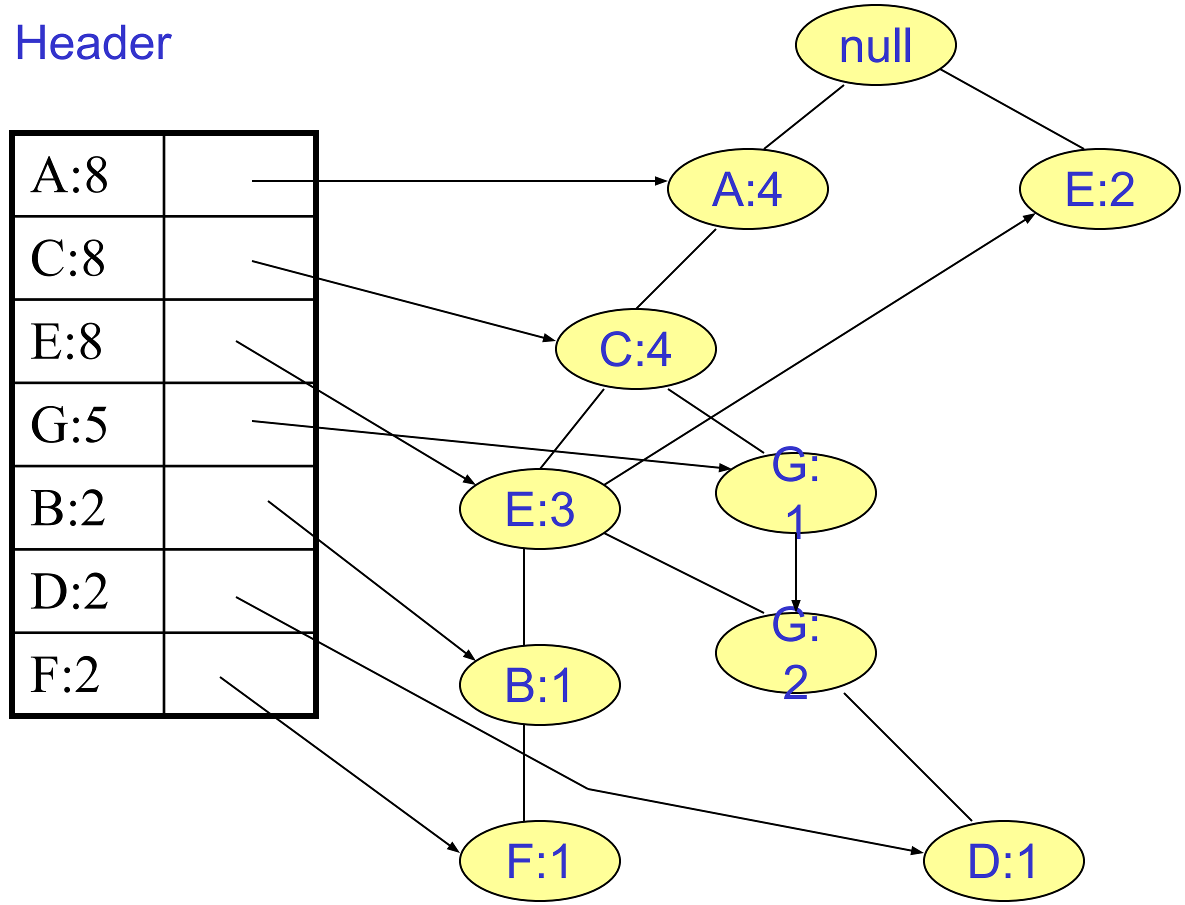
E

A C E B F

A C D

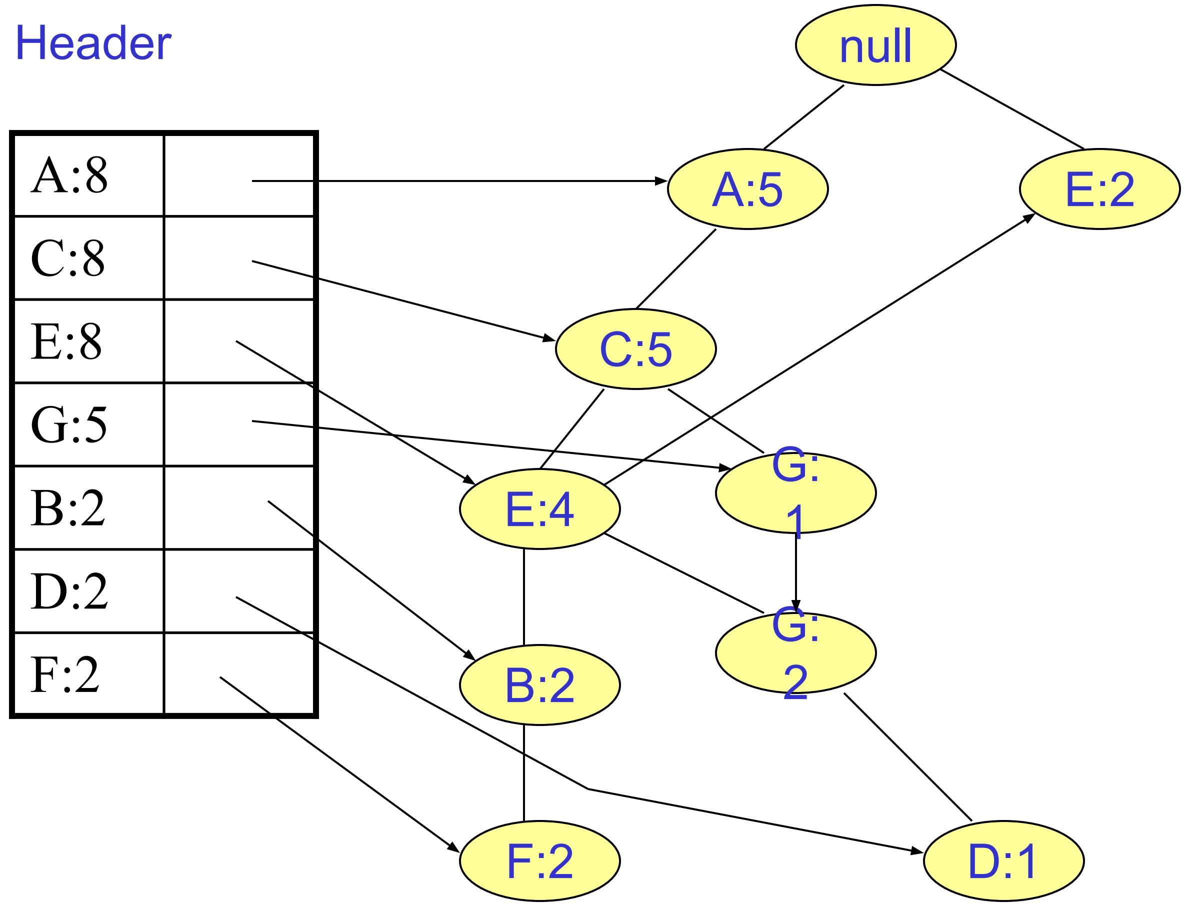
A C E G

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

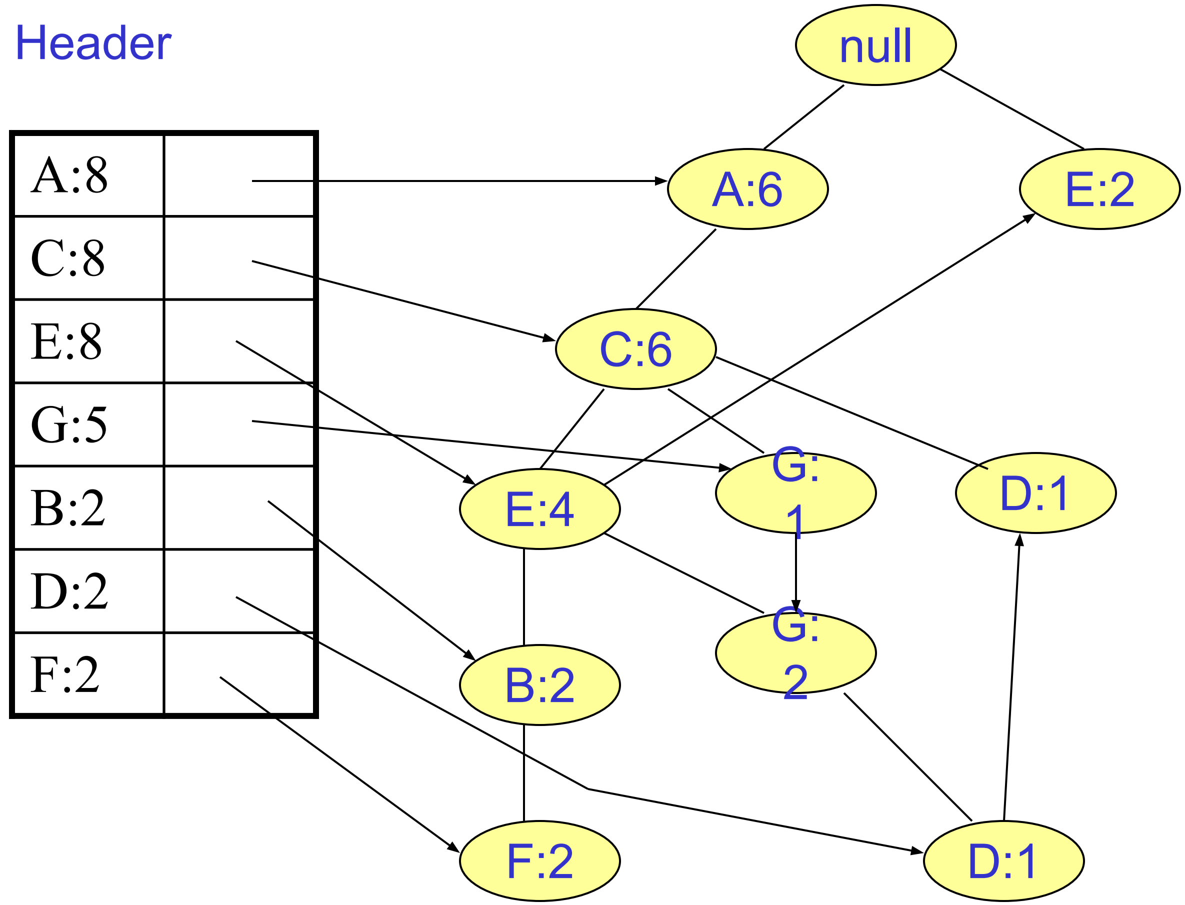
E

A C E B F

A C D

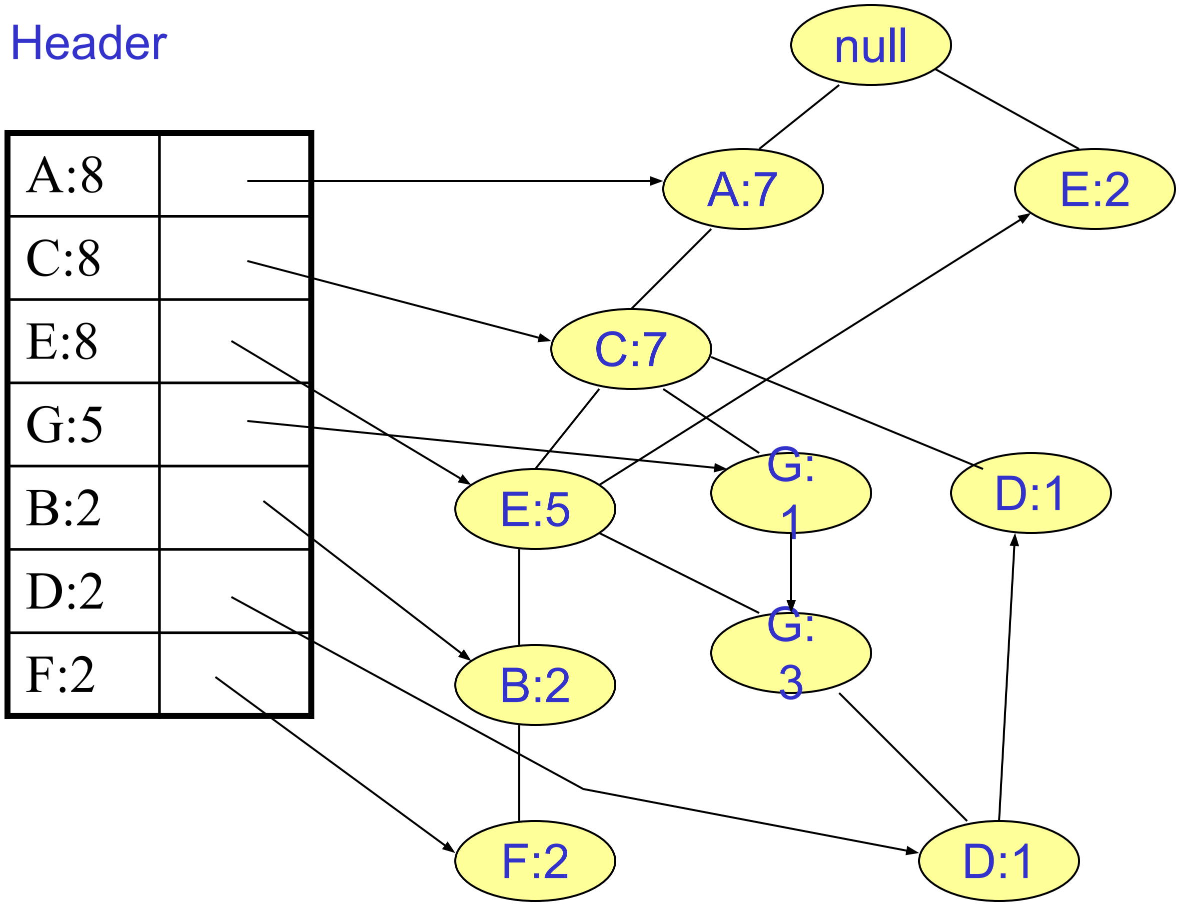
A C E G

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

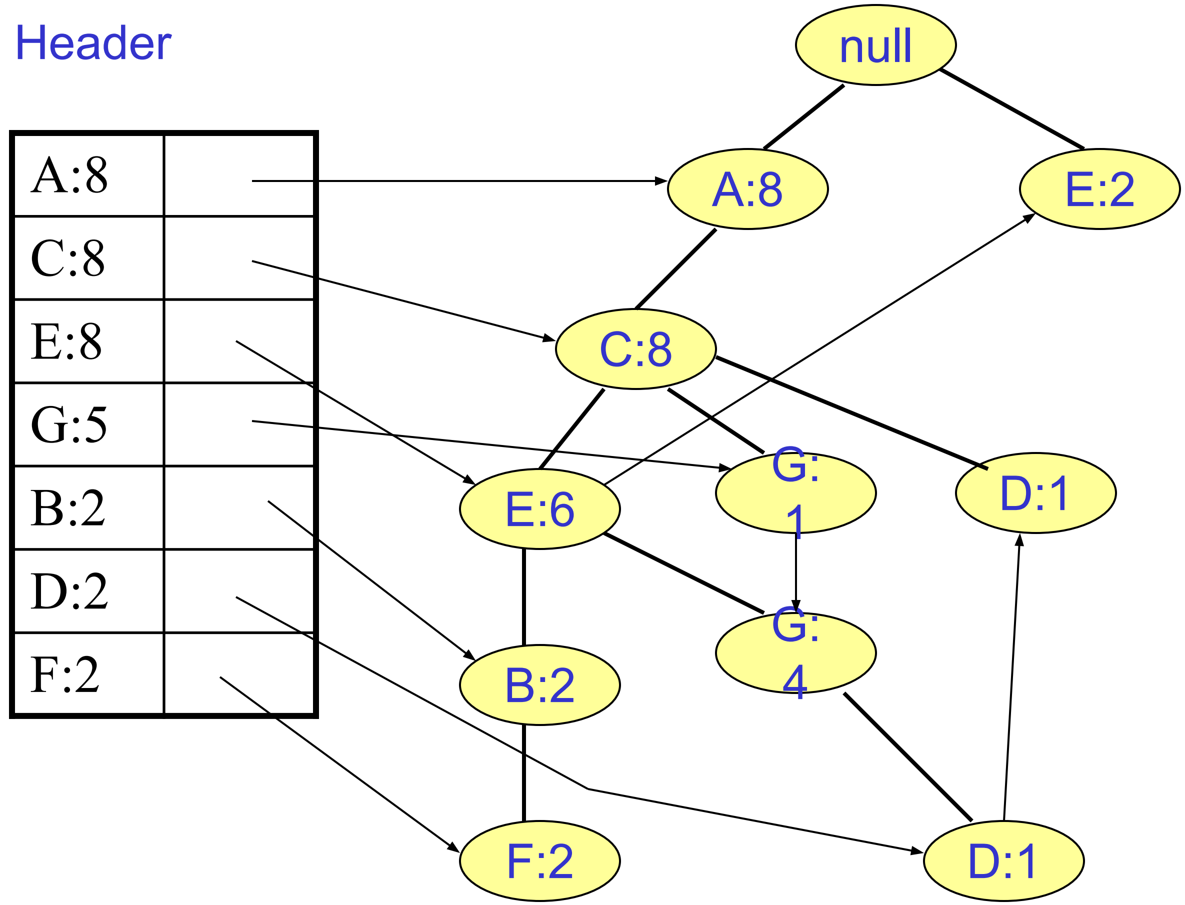
E

A C E B F

A C D

A C E G

A C E G



Conditional FP-Trees

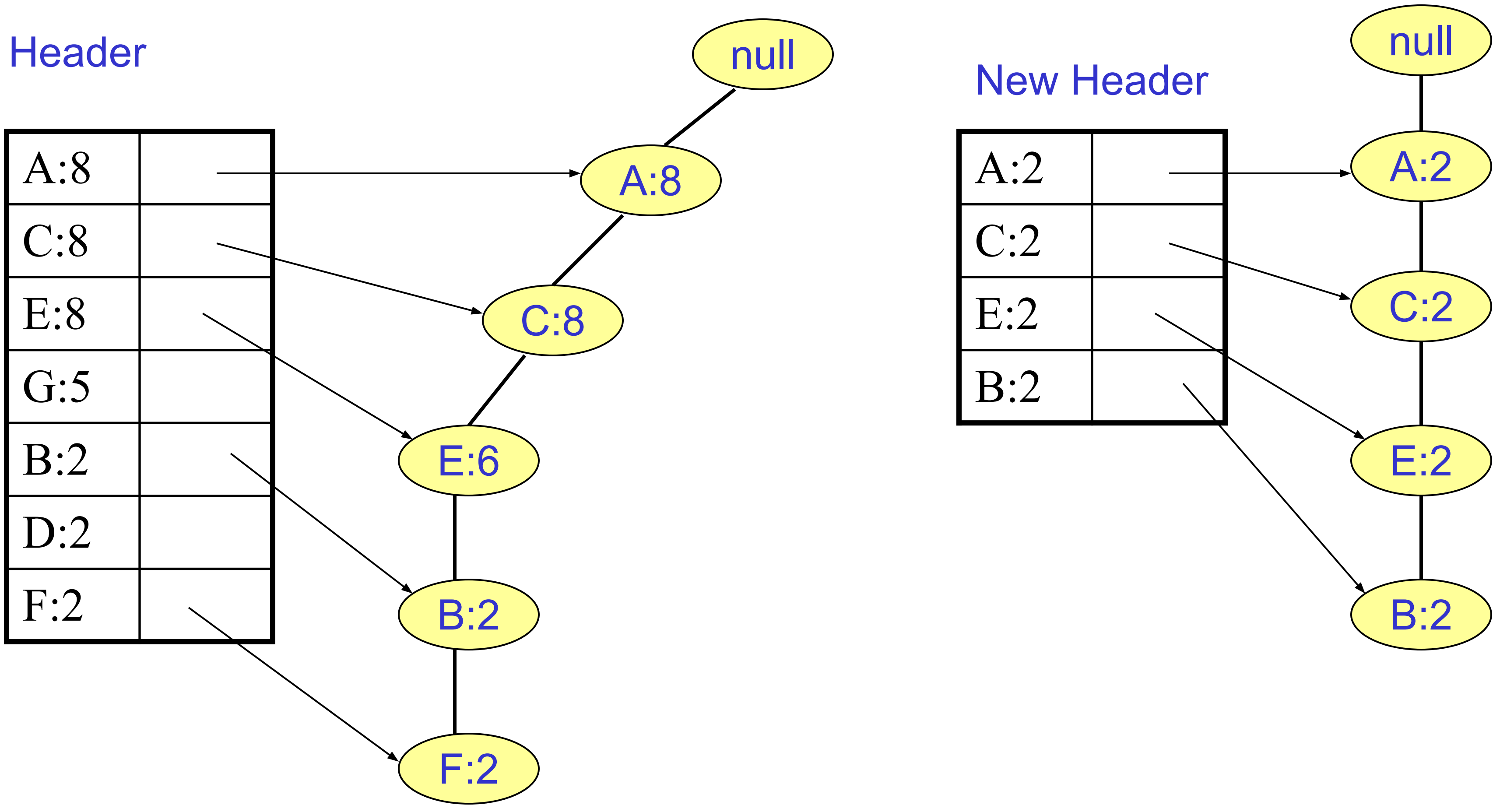
Build the conditional FP-Tree for each of the items.
For this:

1. Find the paths containing on focus item. With those paths we build the conditional FP-Tree for the item.

2. Read again the tree to determine the new counts of the items along those paths. Build a new header.

3. Insert the paths in the conditional FP-Tree according to the new order.

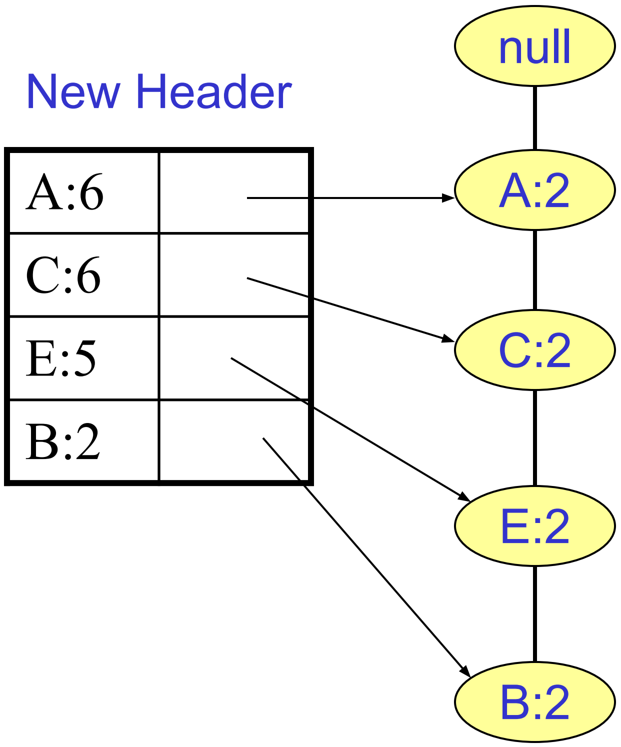
Conditional FP-Tree for F



There is only a single path containing F

Recursion

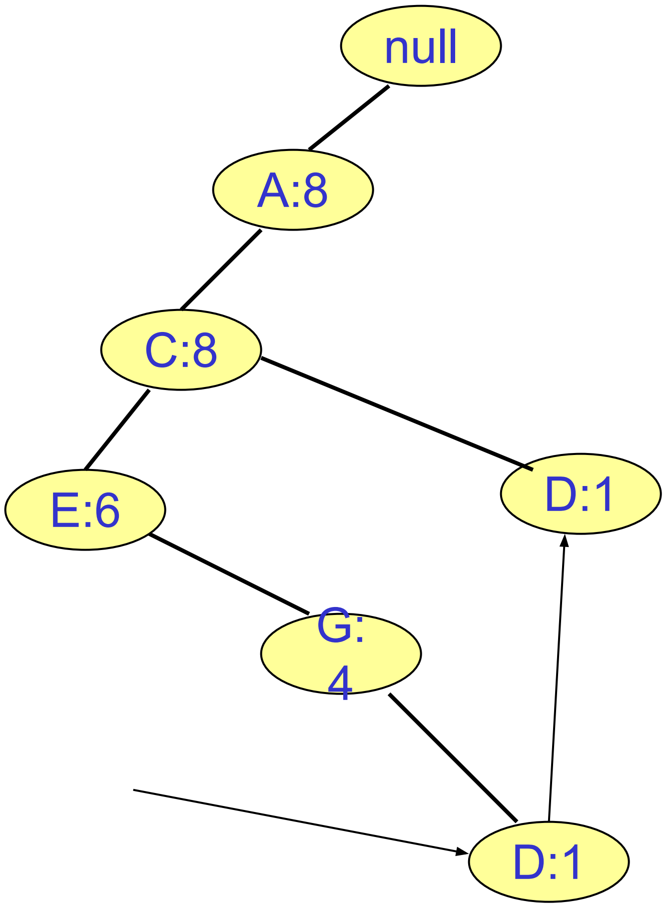
- We continue recursively on the conditional FP-Tree for F.
- However, when the tree is just a single path it is the base case for the recursion.
- So, we just produce all the subsets of the items on this path merged with F.

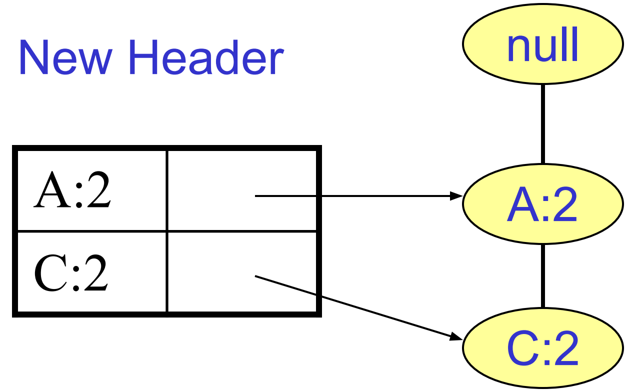


{F} {A,F} {C,F} {E,F} {B,F} {A,C,F}, …,

{A,C,E,F}

Conditional FP-Tree for D





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}

Paths containing D after updating the counts

Exercise: Complete the example.