FP-growth Algorithm

Efficient Mining of Frequent Itemsets

What is FP-growth?

- FP-growth (Frequent Pattern Growth) is a data mining algorithm used for mining frequent itemsets without generating candidate sets.
- **Purpose**: Overcomes limitations of the Apriori algorithm by using a divide-and-conquer strategy.
- **How It Works**:
  - Compresses the database into an FP-tree structure.
  - Divides into smaller conditional databases for recursive mining.

Why Not Apriori?

- Problems with Apriori:
  - Generates Too Many Candidate Sets: For 10,000 frequent 1-itemsets, Apriori generates over 10 million candidate 2-itemsets.
  - Multiple Database Scans: Apriori repeatedly scans the entire database, which is time-consuming and inefficient.
- FP-growth Solution:
  - No candidate generation.
  - Compresses data into a compact structure for efficient mining.

Why Use Pattern Growth for Mining Frequent Patterns?

- **Apriori**: A **breadth-first search** mining algorithm.
  - First, it finds the complete set of frequent k-itemsets.
  - Then, it generates candidates for frequent (k+1)-itemsets.
  - The database is scanned again to identify the true frequent (k+1)-itemsets.
- **Motivation for a new approach**:
  - Can we develop a more efficient **depth-first search** mining algorithm?
  - Can we confine the search for frequent itemsets to only the transactions containing a specific frequent itemset (denoted by ρ)?
- This idea leads to the **frequent pattern growth approach: FP-growth**.

FPGrowth (J. Han, J. Pei, Y. Yin, "Mining Frequent Patterns without Candidate Generation," SIGMOD 2000)

FPGrowth Step 1: Building the FP-tree

- **Goal**: Compress the database into an FP-tree for mining.
- **Steps**:
  - **1.1 First Scan**: Identify frequent items and their support counts.
    - Sort items in descending order of frequency into a list **L**.
  - **1.2 Second Scan**: Build the FP-tree by inserting sorted transactions.
    - For each transaction, create a branch of the tree.
    - Share common paths for transactions that share items.

Step 1.1 First Scan: From Transactional DB to Ordered Frequent Itemlist

Example: A Sample Transactional Database

TID	Items in the Transaction
100	{f, a, c, d, g, i, m, p}
200	{a, b, c, f, l, m, o}
300	{b, f, h, j, o, w}
400	{b, c, k, s, p}
500	{a, f, c, e, l, p, m, n}

Let min_support = 3

- Scan DB once, find single item frequent pattern: f:4, a:3, c:4, b:3, m:3, p:3
- Sort frequent items in frequency descending order, f-list F-list = f-c-a-b-m-p
- Scan DB again, use the ordered frequent itemlist for each transaction to construct an FP-tree

TID	Items in the Transaction	Ordered, frequent itemlist
100	{f, a, c, d, g, i, m, p}	f, c, a, m, p
200	{a, b, c, f, l, m, o}	f, c, a, b, m
300	{b, f, h, j, o, w}	f, b
400	{b, c, k, s, p}	c, b, p
500	{a, f, c, e, l, p, m, n}	f, c, a, m, p

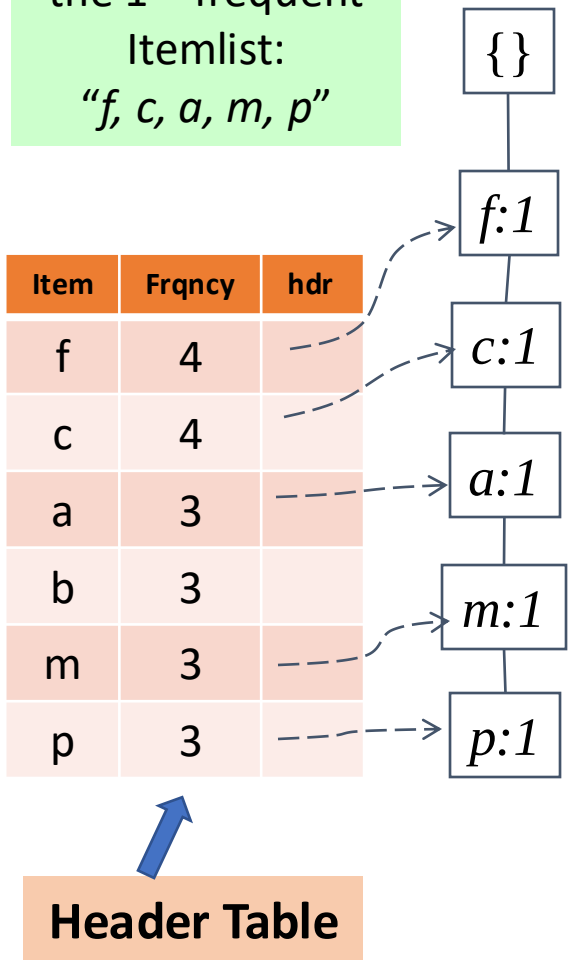
Step 1.2 Second Scan: Construct FP-tree from Transaction DB

TID	Ordered, frequent itemlist
100	f, c, a, m, p
200	f, c, a, b, m
300	f, b
400	c, b, p
500	f, c, a, m, p

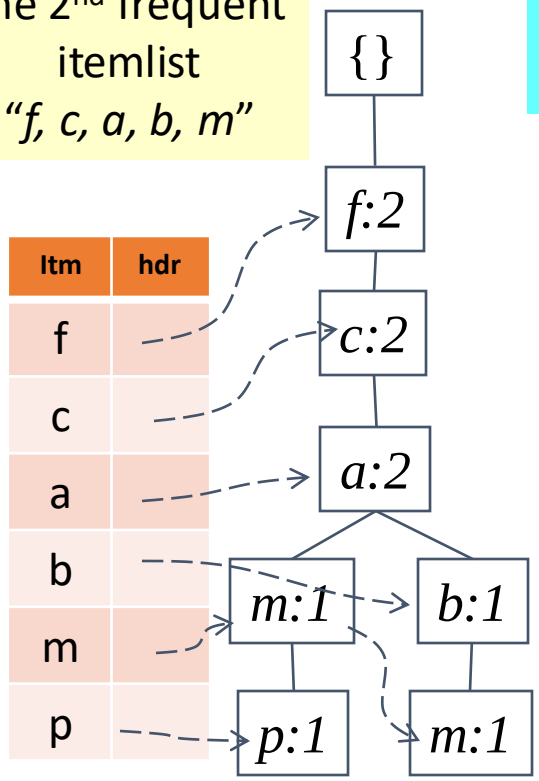
FP-Tree Construction:

For each transaction, insert the ordered frequent itemlist into an FP-tree, with shared sub-branches merged, counts accumulated

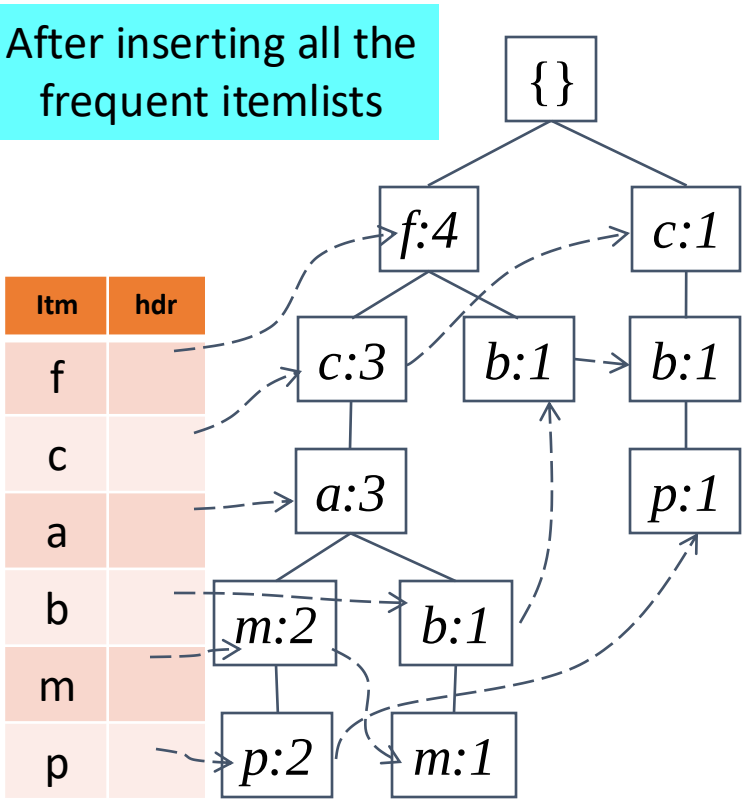
After inserting the 1st frequent Itemlist: *"f, c, a, m, p"*



After inserting the 2nd frequent itemlist *"f, c, a, b, m"*



After inserting all the frequent itemlists



FPGrowth Step 2: Mining the FP-tree

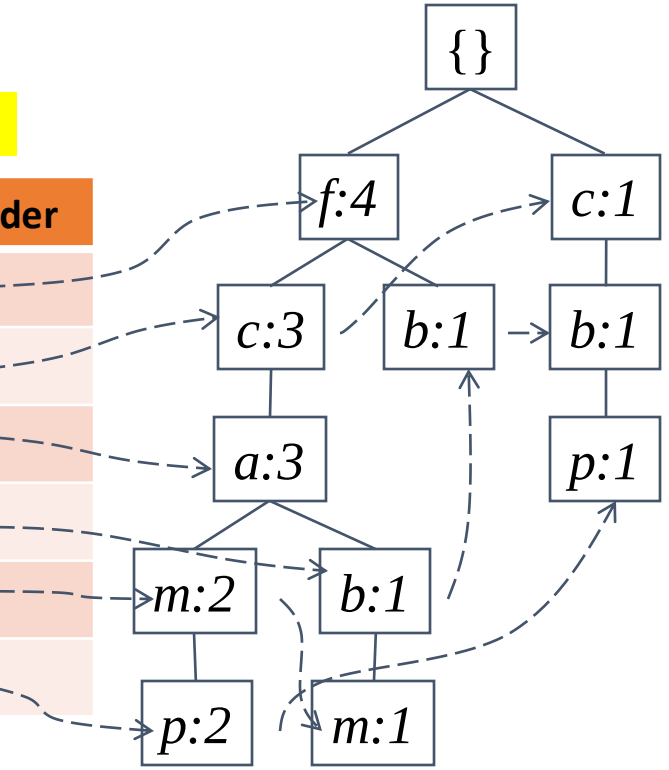
- **Goal**: Extract frequent itemsets from the FP-tree.
- **Steps**:
  - 2.1 Start from the Least Frequent Item
  - 2.2 Construct Conditional Pattern Base: Find all paths in the FP-tree that contain the item
  - 2.3 Create Conditional FP-tree: Use the conditional pattern base to build a smaller FP-tree.
  - 2.4 Recursive Mining: Repeat the process for each item

Mining FP-Tree: Divide and Conquer Based on Patterns and Data

- Pattern mining can be partitioned according to current patterns
  - Patterns containing *p*: *p*'s conditional database: *fcam:2, cb:1*
    - *p*'s conditional database (i.e., the database under the condition that *p* exists):
      - *transformed prefix paths* of item *p*
  - Patterns having m but no p: m's conditional database: *fca:2, fcab:1*
  - …… ……

min_support = 3

Item	Frequency	Header
f	4	
c	4	
a	3	
b	3	
m	3	
p	3	



Conditional database of each pattern

Item	***Conditional database***
c	***f:3***
a	***fc:3***
b	***fca:1, f:1, c:1***
m	***fca:2, fcab:1***
p	***fcam:2, cb:1***

Mine Each Conditional Database Recursively

min_support = 3

Conditional Data Bases

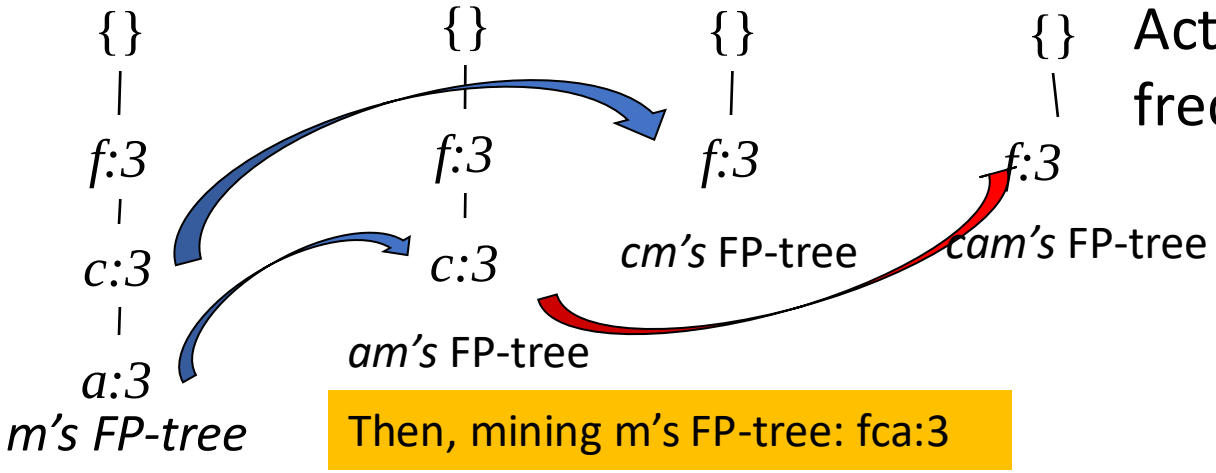
item	*cond. data base*
c	*f:3*
a	*fc:3*
b	*fca:1, f:1, c:1*
m	*fca:2, fcab:1*
p	*fcam:2, cb:1*

- For each conditional database
  - Mine single-item patterns
  - Construct its FP-tree & mine it

p's conditional DB: ***fcam:2, cb:1 → c: 3***

m's conditional DB: ***fca:2, fcab:1 → fca: 3***

b's conditional DB: ***fca:1, f:1, c:1 → ϕ***



Then, mining m's FP-tree: fca:3

Actually, for single branch FP-tree, all the frequent patterns can be generated in one shot

m: 3

fm: 3, cm: 3, am: 3

fcm: 3, fam:3, cam: 3

fcam: 3