Mining Frequent Patterns without Candidate Generation

Outline

- Frequent Pattern Mining: Problem statement and an example
- Review of Apriori-like Approaches
- FP-Growth:
  - Overview
  - FP-tree:
    - structure, construction and advantages
  - FP-growth:
    - FP-tree →conditional pattern bases → conditional FP-tree →frequent patterns
- Experiments
- Discussion:
  - Improvement of FP-growth
- Conclusion Remarks

Frequent Pattern Mining: An Example

Given a transaction database DB and a minimum support threshold ξ, find all frequent patterns (item sets) with support no less than ξ.

Input: DB:

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

Minimum support: ξ =3

Output: all frequent patterns, i.e., *f, a*, …, *fa, fac, fam, fm,am*…

Problem Statement: How to **efficiently** find all frequent patterns?

Apriori

- Main Steps of Apriori Algorithm:
  - Use frequent (k – 1)-itemsets (L_{k-1}) to generate candidates of frequent k-itemsets C_k
  - Scan database and count each pattern in C_k, get frequent k-itemsets (L_k) .

Candidate Generation

Candidate Test

- E.g. ,

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

Apriori	*iteration*
C1	*f,a,c,d,g,i,m,p,l,o,h,j,k,s,b,e,n*
L1	*f, a, c, m, b, p*
C2	*fa, fc, fm, fp, ac, am, …bp*
L2	*fa, fc, fm, …*
…	

Performance Bottlenecks of Apriori

- Bottlenecks of *Apriori*: candidate generation
  - Generate huge candidate sets:
    - 10^4 frequent 1-itemset will generate 10^7 candidate 2-itemsets
    - To discover a frequent pattern of size 100, e.g., $\{a_1, a_2, \ldots, a_{100}\}$, one needs to generate $2^{100} \approx 10^{30}$ candidates.
  - Candidate Test incur multiple scans of database: each candidate

Overview of FP-Growth: Ideas

- Compress a large database into a compact, *Frequent-Pattern tree* (FP-tree) structure
  - highly compacted, but complete for frequent pattern mining
  - avoid costly repeated database scans
- Develop an efficient, FP-tree-based frequent pattern mining method (FP-growth)
  - A divide-and-conquer methodology: decompose mining tasks into smaller ones
  - Avoid candidate generation: sub-database test only.

FP-tree:
Construction and Design

Construct FP-tree

Two Steps:

1. Scan the transaction DB for the first time, find frequent items (single item patterns) and order them into a list L in frequency descending order.

   e.g., L={f:4, c:4, a:3, b:3, m:3, p:3}

   In the format of (item-name, support)

2. For each transaction, order its frequent items according to the order in L; Scan DB the second time, construct FP-tree by putting each frequency ordered transaction onto it.

FP-tree Example: step 1

Step 1: Scan DB for the first time to generate L

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

L

Item	*frequency*
f	*4*
c	*4*
a	*3*
b	*3*
m	*3*
p	*3*

By-Product of First Scan of Database

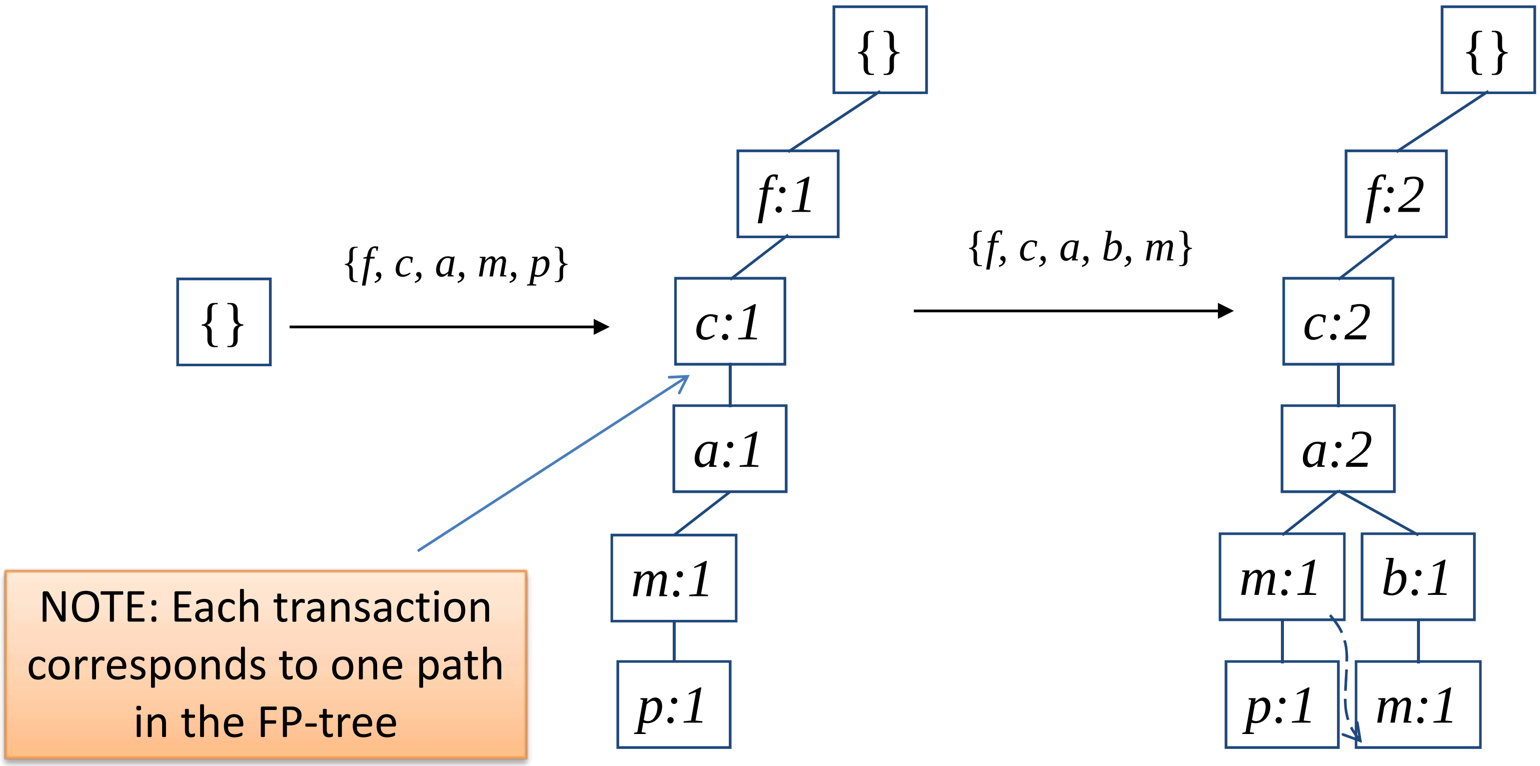
FP-tree Example: step 2

Step 2: scan the DB for the second time, order frequent items in each transaction

TID	*Items bought*	*(ordered) frequent items*
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*}	{*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*}

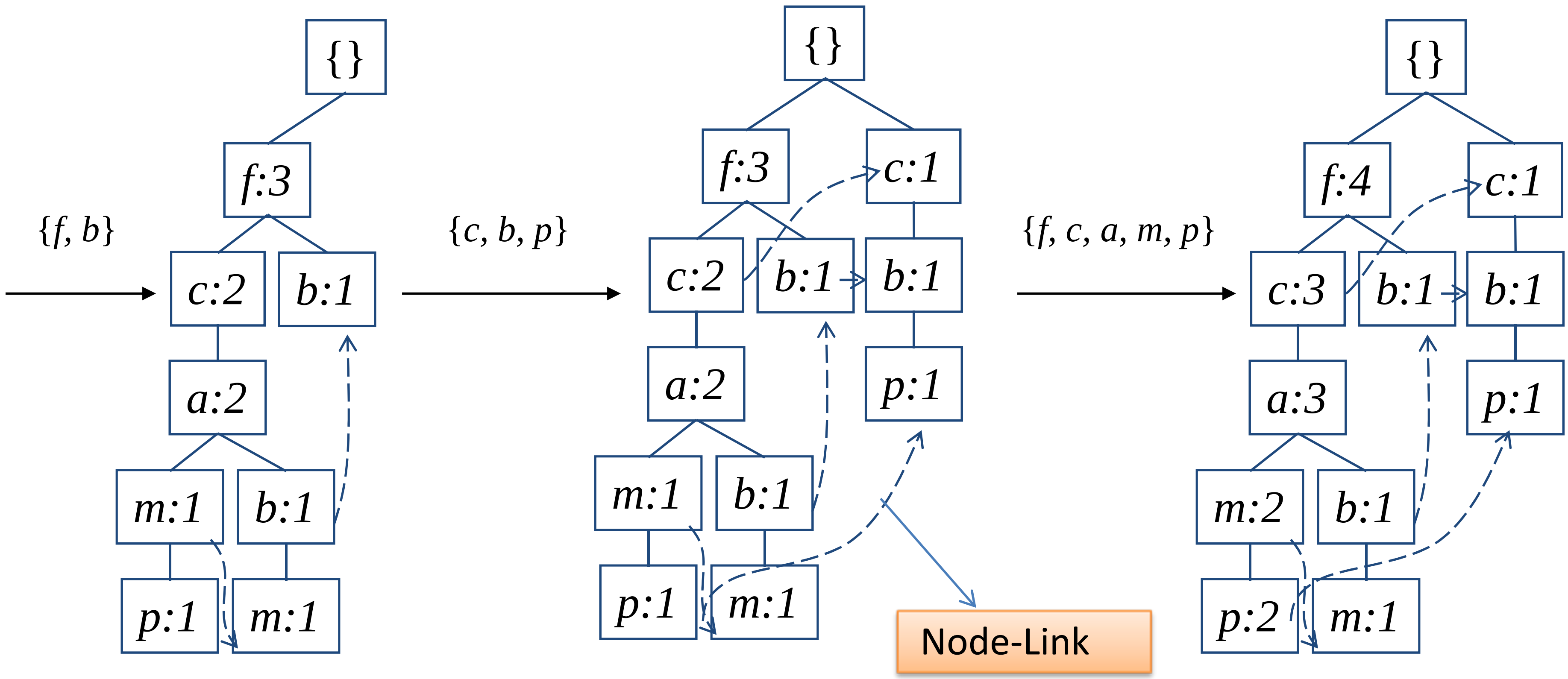
FP-tree Example: step 2

Step 2: construct FP-tree



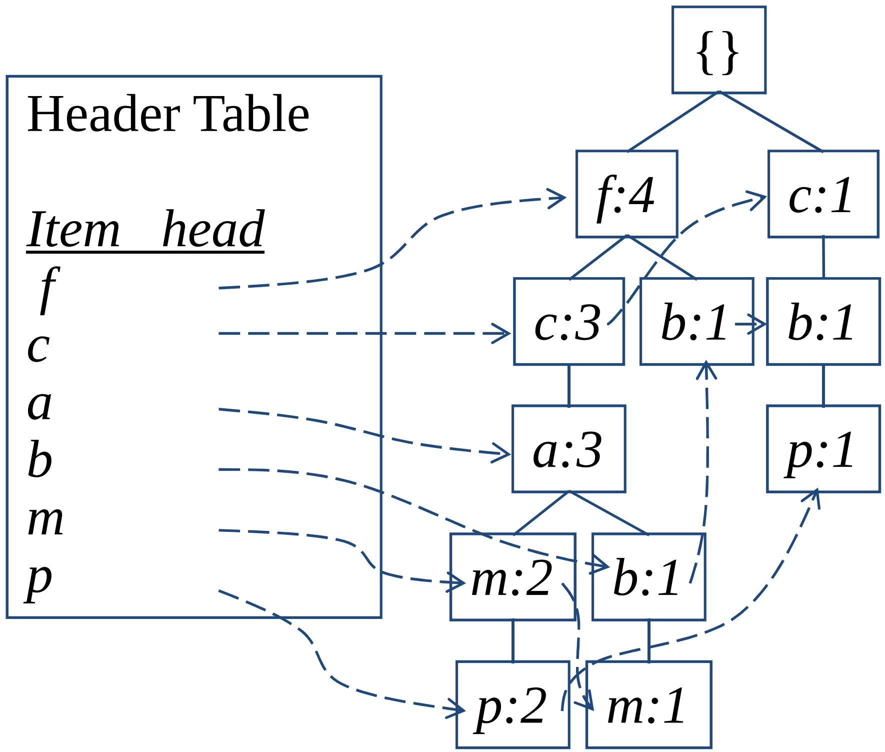
FP-tree Example: step 2

Step 2: construct FP-tree



Construction Example

Final FP-tree



FP-Tree Definition

- FP-tree is **a frequent pattern tree** . Formally, FP-tree is a tree structure defined below:

  1. One root labeled as “null", a set of *item prefix sub-trees* as the children of the root, and a *frequent-item header table*.

  2. Each node in *the item prefix sub-trees* has three fields:
     - item-name : register which item this node represents,
     - count, the number of transactions represented by the portion of the path reaching this node,
     - node-link that links to the next node in the FP-tree carrying the same item-name, or null if there is none.

  3. Each entry in the *frequent-item header table* has two fields,
     - item-name, and
     - head of node-link that points to the first node in the FP-tree carrying the item-name.

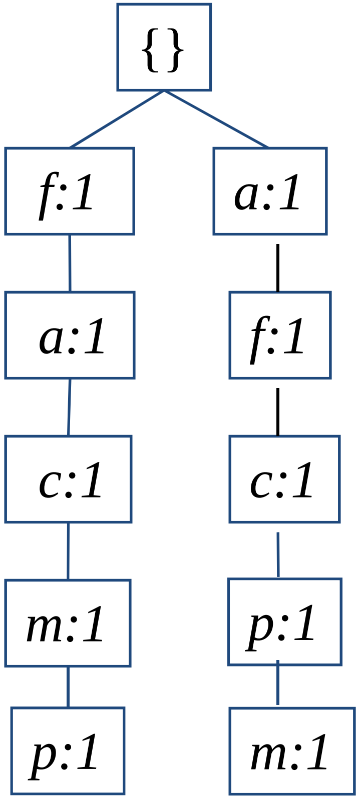
Advantages of the FP-tree Structure

- The most significant advantage of the FP-tree
  - Scan the DB only twice and twice only.
- Completeness:
  - the FP-tree contains all the information related to mining frequent patterns (given the min-support threshold). Why?
- Compactness:
  - The size of the tree is bounded by the occurrences of frequent items
  - The height of the tree is bounded by the maximum number of items in a transaction

Questions?

- Why descending order?
- Example 1:

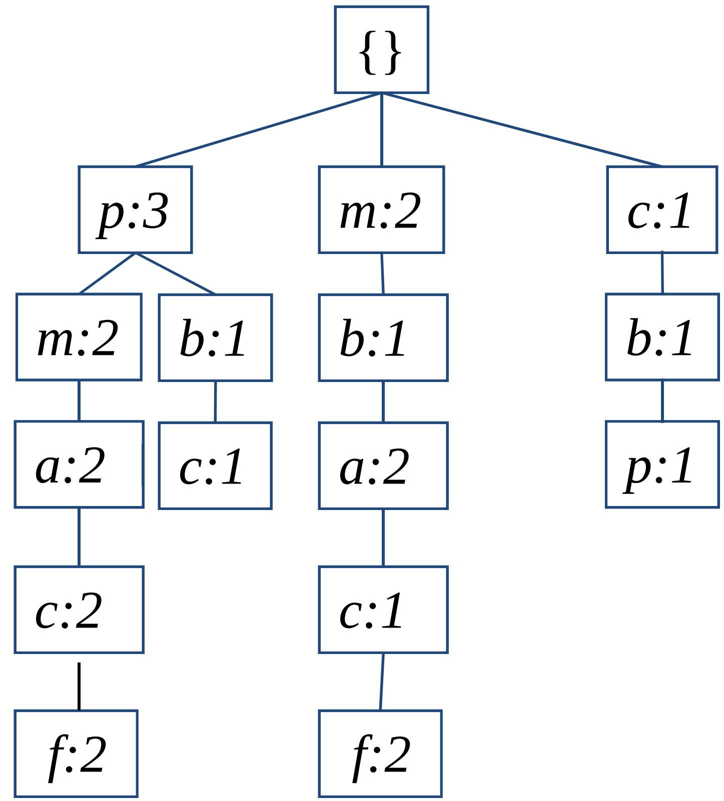
TID	*(unordered) frequent items*
100	{*f, a, c, m, p*}
500	{*a, f, c, p, m*}



Questions?

- Example 2:

TID	*(ascended) frequent items*
100	{*p*, *m*, *a*, *c*, *f*}
200	{*m*, *b*, *a*, *c*, *f*}
300	{*b*, *f*}
400	{*p*, *b*, *c*}
500	{*p*, *m*, *a*, *c*, *f*}



This tree is larger than FP-tree, because in FP-tree, more frequent items have a higher position, which makes branches less

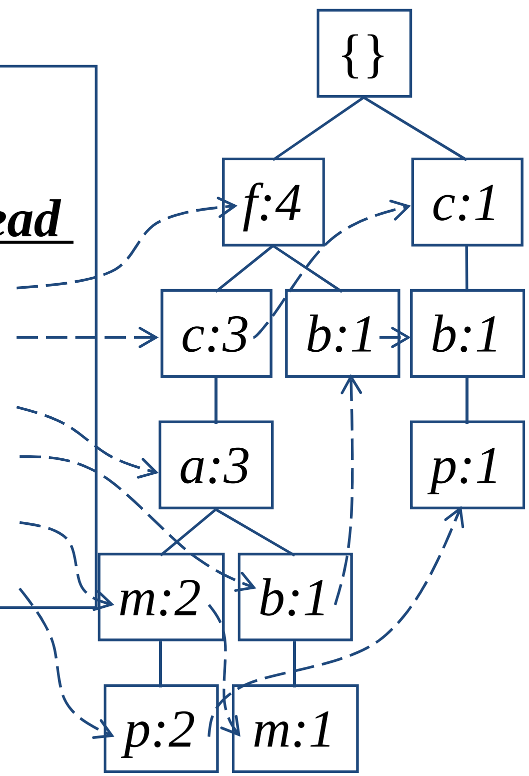
Construct FP-tree from a Transaction Database

TID	*Items bought*	*(ordered) frequent items*
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*}**

min_support = 3

Header Table

Item	*frequency*	*head*
f	*4*	
c	*4*	
a	*3*	
b	*3*	
m	*3*	
p	*3*	



1. Scan DB once, find frequent 1-itemset (single item pattern)
2. Sort frequent items in frequency descending order, f-list
3. Scan DB again, construct FP-tree

F-list = f-c-a-b-m-p

Partition Patterns and Databases

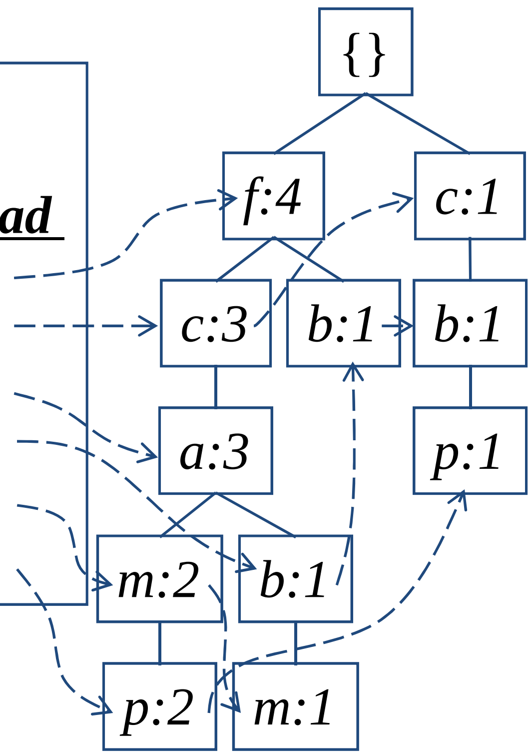
- Frequent patterns can be partitioned into subsets according to f-list
  - F-list = f-c-a-b-m-p
  - Patterns containing p
  - Patterns having m but no p
  - …
  - Patterns having c but no a nor b, m, p
  - Pattern f
- Completeness and non-redundency

Find Patterns Having P From P-conditional Database

- Starting at the frequent item header table in the FP-tree
- Traverse the FP-tree by following the link of each frequent item *p*
- Accumulate all of *transformed prefix paths* of item *p* to form *p*'s conditional pattern base

Header Table

Item	***frequency***	***head***
f	*4*	
c	*4*	
a	*3*	
b	*3*	
m	*3*	
p	*3*	

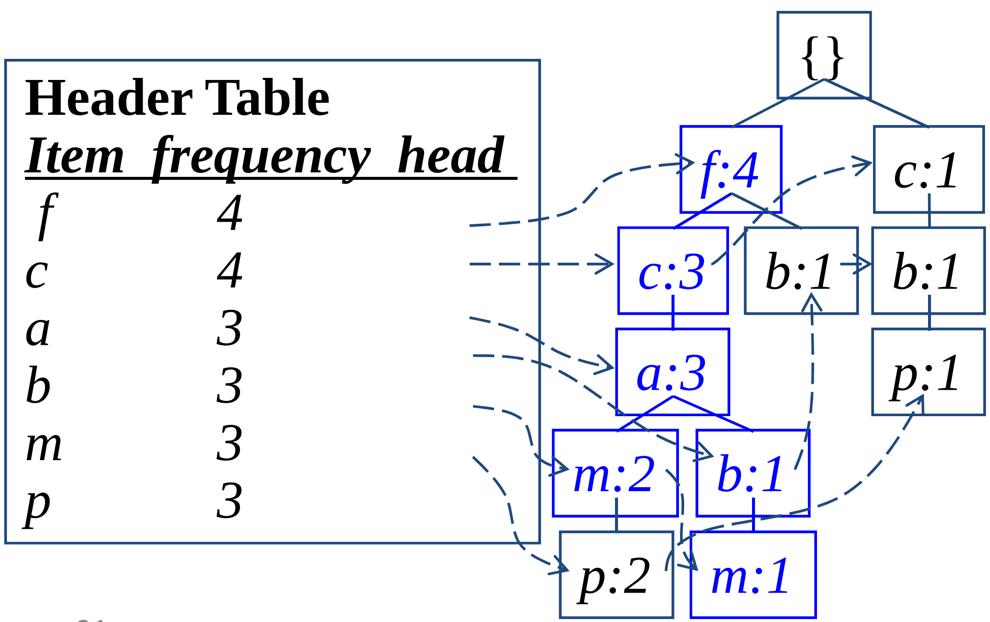


Conditional **pattern bases**

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

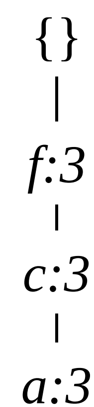
From Conditional Pattern-bases to Conditional FP-trees

- For each pattern-base
  - Accumulate the count for each item in the base
  - Construct the FP-tree for the frequent items of the pattern base



m-conditional **pattern base:**

fca:2, fcab:1



m-conditional **FP-tree**

All frequent patterns relate to *m*

m,

fm, cm, am,

fcm, fam, cam,

fcam

Recursion: Mining Each Conditional FP-tree

{}
|
f:3
|
c:3
|
a:3

m-conditional **FP-tree**

Cond. pattern base of “am”: (fc:3)

{}
|
f:3
|
c:3

am-conditional **FP-tree**

Cond. pattern base of “cm”: (f:3)

{}
|
f:3

cm-conditional **FP-tree**

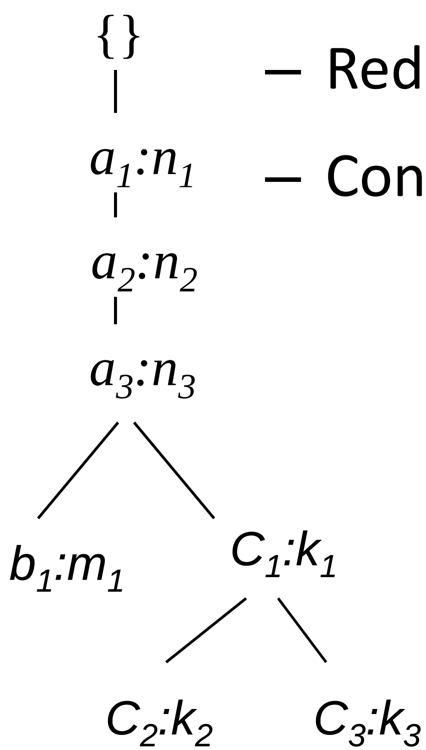
Cond. pattern base of “cam”: (f:3)

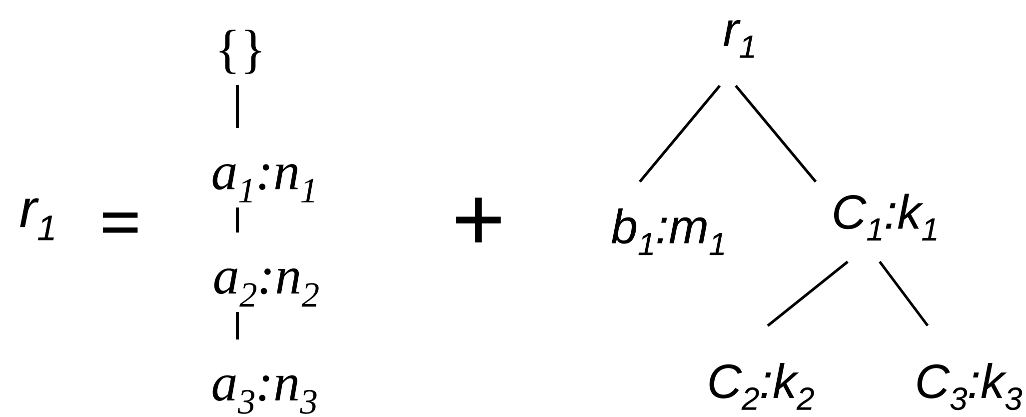
{}
|
f:3

cam-conditional **FP-tree**

A Special Case: Single Prefix Path in FP-tree

- Suppose a (conditional) FP-tree T has a shared single prefix-path P
- Mining can be decomposed into two parts
  - Reduction of the single prefix path into one node
  - Concatenation of the mining results of the two parts





Benefits of the FP-tree Structure

- Completeness
  - Preserve complete information for frequent pattern mining
  - Never break a long pattern of any transaction
- Compactness
  - Reduce irrelevant info—infrequent items are gone
  - Items in frequency descending order: the more frequently occurring, the more likely to be shared
  - Never be larger than the original database (not count node-links and the *count* field)

The Frequent Pattern Growth Mining Method

- Idea: Frequent pattern growth
  - Recursively grow frequent patterns by pattern and database partition
- Method
  - For each frequent item, construct its conditional pattern-base, and then its conditional FP-tree
  - Repeat the process on each newly created conditional FP-tree
  - Until the resulting FP-tree is empty, or it contains only one path—single path will generate all the combinations of its sub-paths, each of which is a frequent pattern

FP-growth:

Mining Frequent Patterns Using FP-tree

Mining Frequent Patterns Using FP-tree

- General idea (divide-and-conquer)

  Recursively grow frequent patterns using the FP-tree: looking for shorter ones recursively and then concatenating the suffix:
  - For each frequent item, construct its conditional pattern base, and then its conditional FP-tree;
  - Repeat the process on each newly created conditional FP-tree until the resulting FP-tree is empty, or it contains only one path (single path will generate all the combinations of its sub-paths, each of which is a frequent pattern)

3 Major Steps

Starting the processing from the end of list L:

Step 1:

Construct **conditional pattern base** for each item in the header table

Step 2

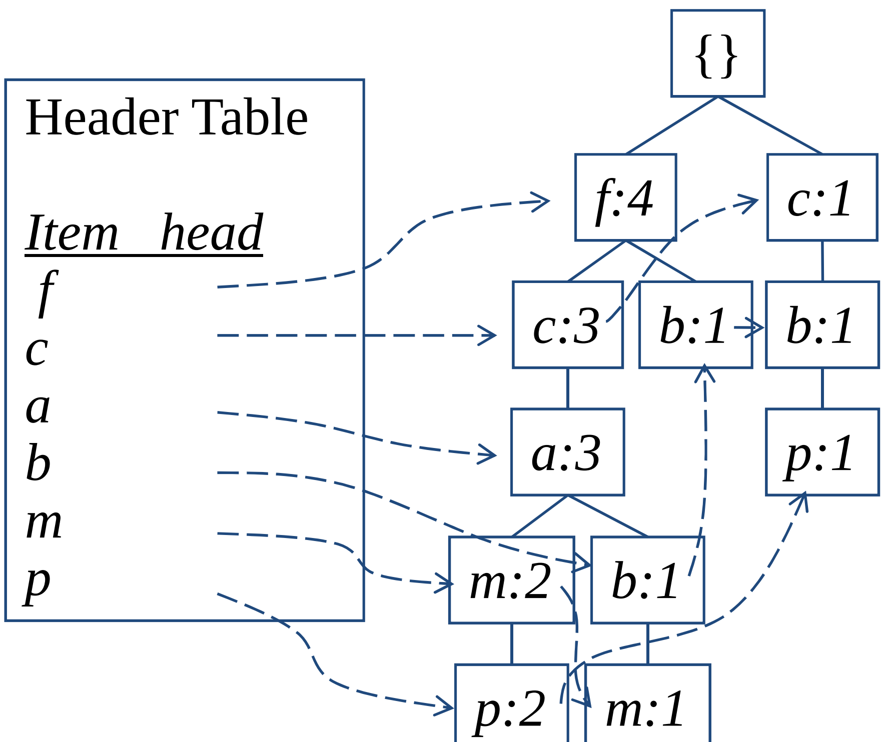
Construct **conditional FP-tree** from each conditional pattern base

Step 3

Recursively mine conditional FP-trees and grow frequent patterns obtained so far. If the conditional FP-tree contains a single path, simply enumerate all the patterns

Step 1: Construct Conditional Pattern Base

- Starting at the bottom of frequent-item header table in the FP-tree
- Traverse the FP-tree by following the link of each frequent item
- Accumulate all of **transformed prefix paths** of that item to form a **conditional pattern base**



Conditional **pattern bases**

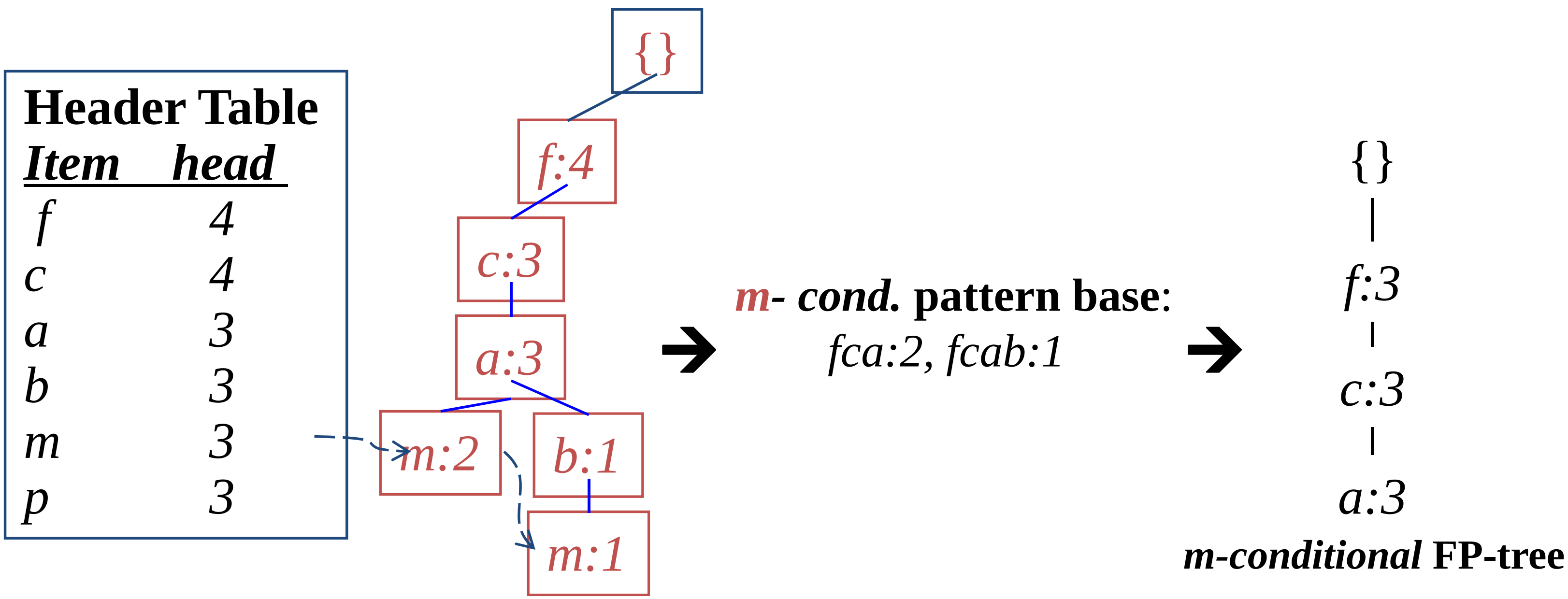
item	***cond. pattern base***
p	***fcam:2, cb:1***
m	***fca:2, fcab:1***
b	***fca:1, f:1, c:1***
a	***fc:3***
c	***f:3***
f	***{ }***

Properties of FP-Tree

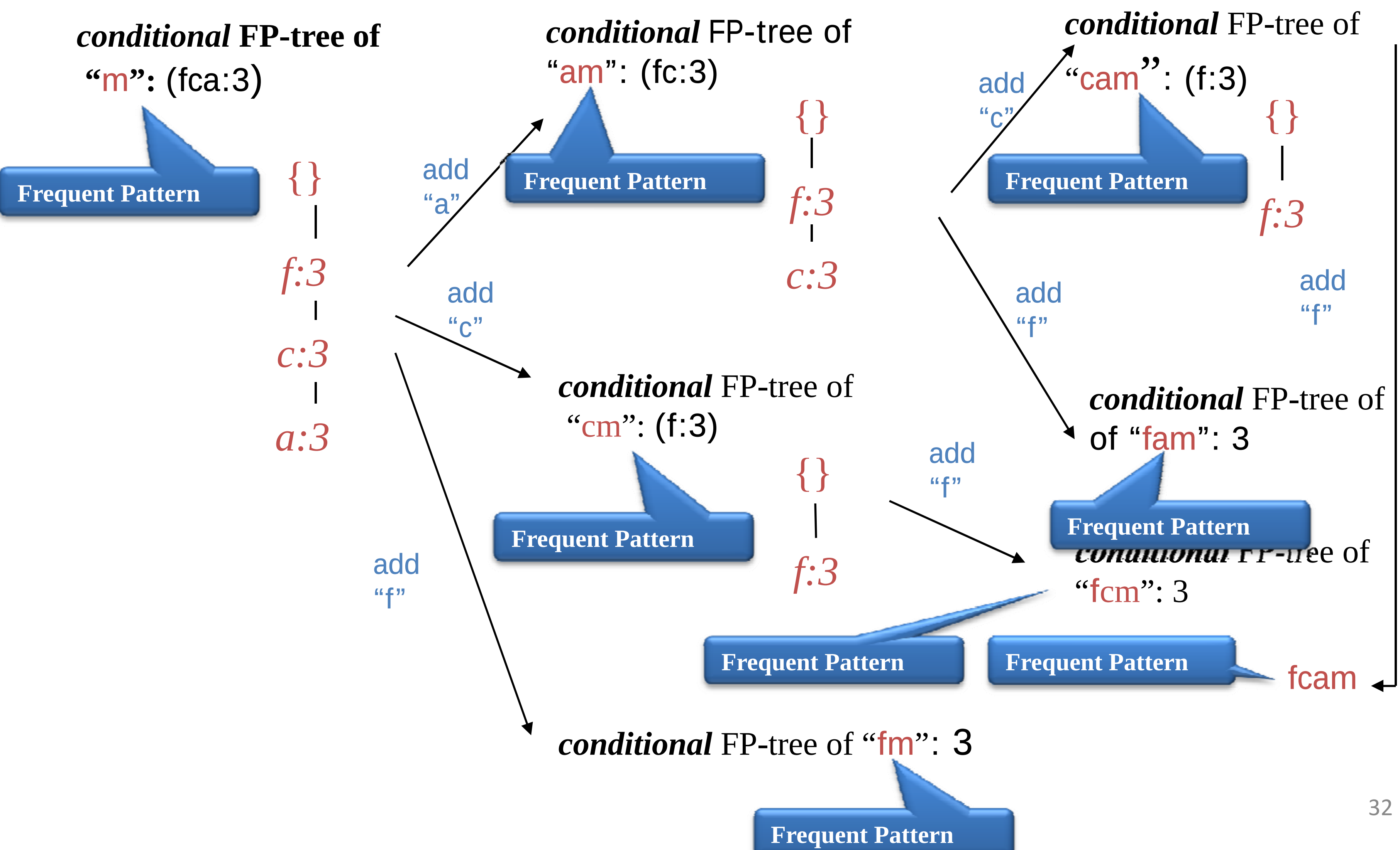
- Node-link property
  - For any frequent item a_i, all the possible frequent patterns that contain a_i can be obtained by following a_i's node-links, starting from a_i's head in the FP-tree header.
- Prefix path property
  - To calculate the frequent patterns for a node a_i in a path P, only the prefix sub-path of a_i in P need to be accumulated, and its frequency count should carry the same count as node a_i.

Step 2: Construct Conditional FP-tree

- For each pattern base
  - Accumulate the count for each item in the base
  - Construct the conditional FP-tree for the frequent items of the pattern base



Step 3: Recursively mine the conditional FP-tree



Principles of FP-Growth

- Pattern growth property
  - Let α be a frequent itemset in DB, B be α's conditional pattern base, and β be an itemset in B. Then $\alpha \cup \beta$ is a frequent itemset in DB iff β is frequent in B.
- Is "*fcabm* " a frequent pattern?
  - "fcab" is a branch of m's conditional pattern base
  - "b" is **NOT** frequent in transactions containing "fcab "
  - "bm" is **NOT** a frequent itemset.

Conditional Pattern Bases and Conditional FP-Tree

Item	Conditional pattern base	Conditional FP-tree
p	{(fcam:2), (cb:1)}	{(c:3)}|p
m	{(fca:2), (fcab:1)}	{(f:3, c:3, a:3)}|m
b	{(fca:1), (f:1), (c:1)}	Empty
a	{(fc:3)}	{(f:3, c:3)}|a
c	{(f:3)}	{(f:3)}|c
f	Empty	Empty

order of L

Single FP-tree Path Generation

- Suppose an FP-tree T has a single path P. The complete set of frequent pattern of T can be generated by enumeration of all the combinations of the sub-paths of P

{}
|
f:3
|
c:3
|
a:3

m-conditional FP-tree

→

All frequent patterns concerning *m*:
combination of {f, c, a} and *m*

m,

fm, *cm*, *am*,

fcm, *fam*, *cam*,

fcam

Summary of FP-Growth Algorithm

- Mining frequent patterns can be viewed as first mining 1-itemset and progressively growing each 1-itemset by mining on its conditional pattern base recursively

- Transform a frequent k-itemset mining problem into a sequence of k frequent 1-itemset mining problems via a set of conditional pattern bases

Efficiency Analysis

Facts: usually

1. FP-tree is much smaller than the size of the DB
2. Pattern base is smaller than original FP-tree
3. Conditional FP-tree is smaller than pattern base

→ mining process works on a set of usually much smaller pattern bases and conditional FP-trees

→ Divide-and-conquer and dramatic scale of shrinking