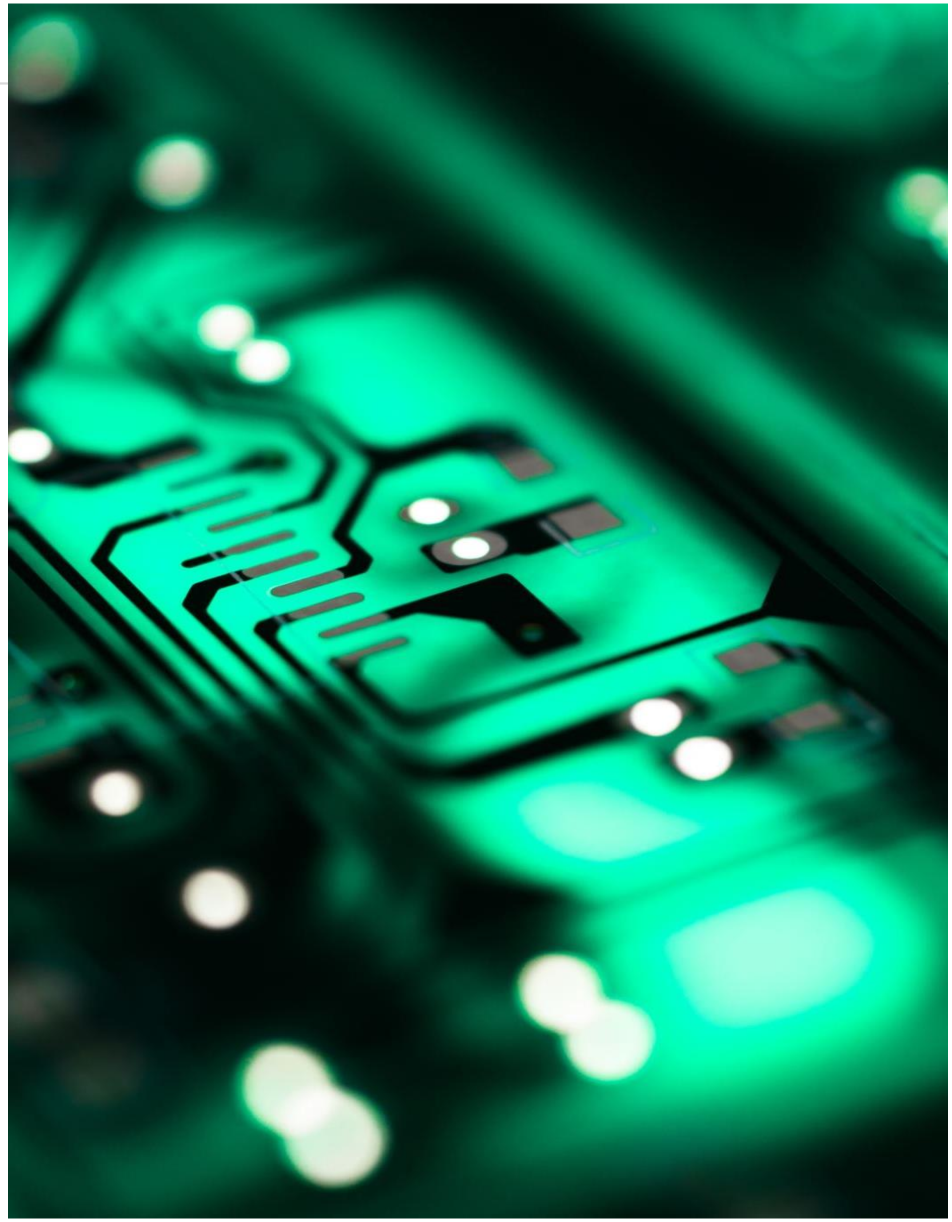
Introduction
Artificial Intelligence



Mining Frequent Patterns without Candidate Generation (FP Tree)

Dr. Muhammad Shuaib Qureshi

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

Candidate Generation

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) .

E.g. ,

Candidate Test

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

Disadvantages of Apriori-like Approach

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

Question

What are the main drawbacks of Apriori –like approaches and explain why ?

A:

The main disadvantages of Apriori-like approaches are:

1. It is costly to generate the candidate sets;
2. It incurs multiple scan of the database.

The reason is that: Apriori is based on the following heuristic/down-closure property: if any length *k* patterns is not frequent in the database, any length *(k+1)* super-pattern can never be frequent.

The two steps in Apriori are candidate generation and test. If the 1-itemsets is huge in the database, then the generation for successive item-sets would be quite costly and thus the test.

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

Minimum support: ξ =3

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

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*}

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

By-Product of First Scan of Database

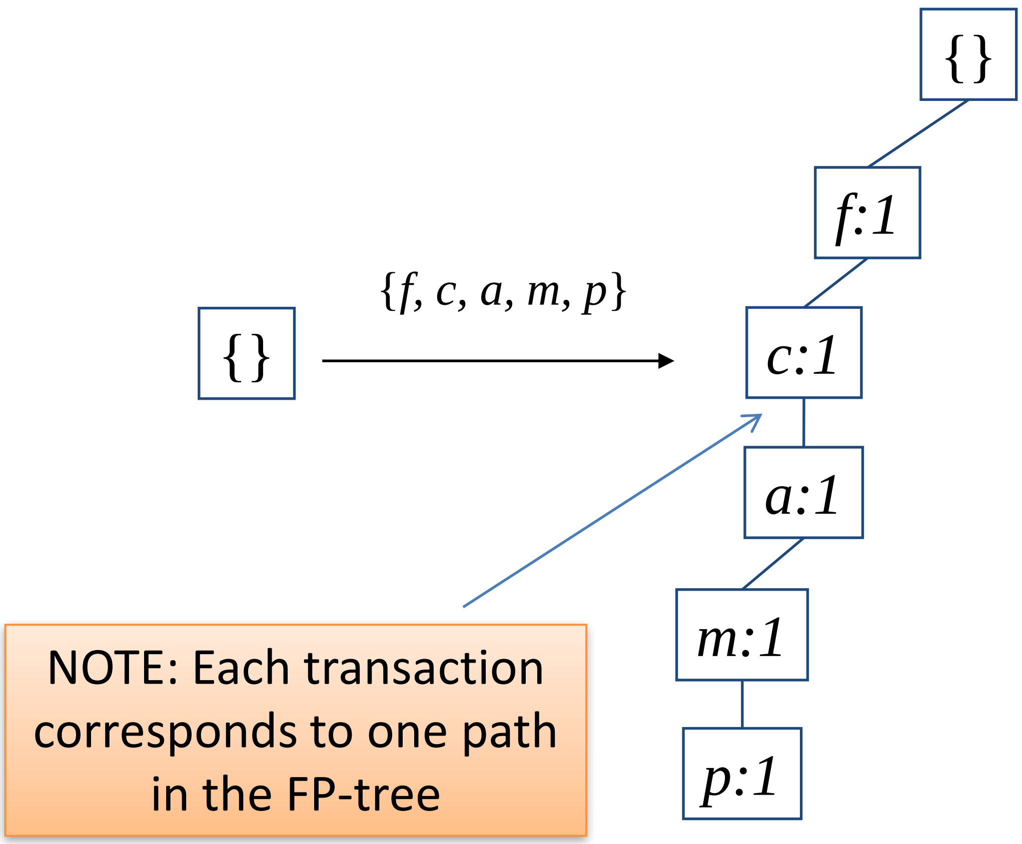
FP-tree Example: step 2

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*}

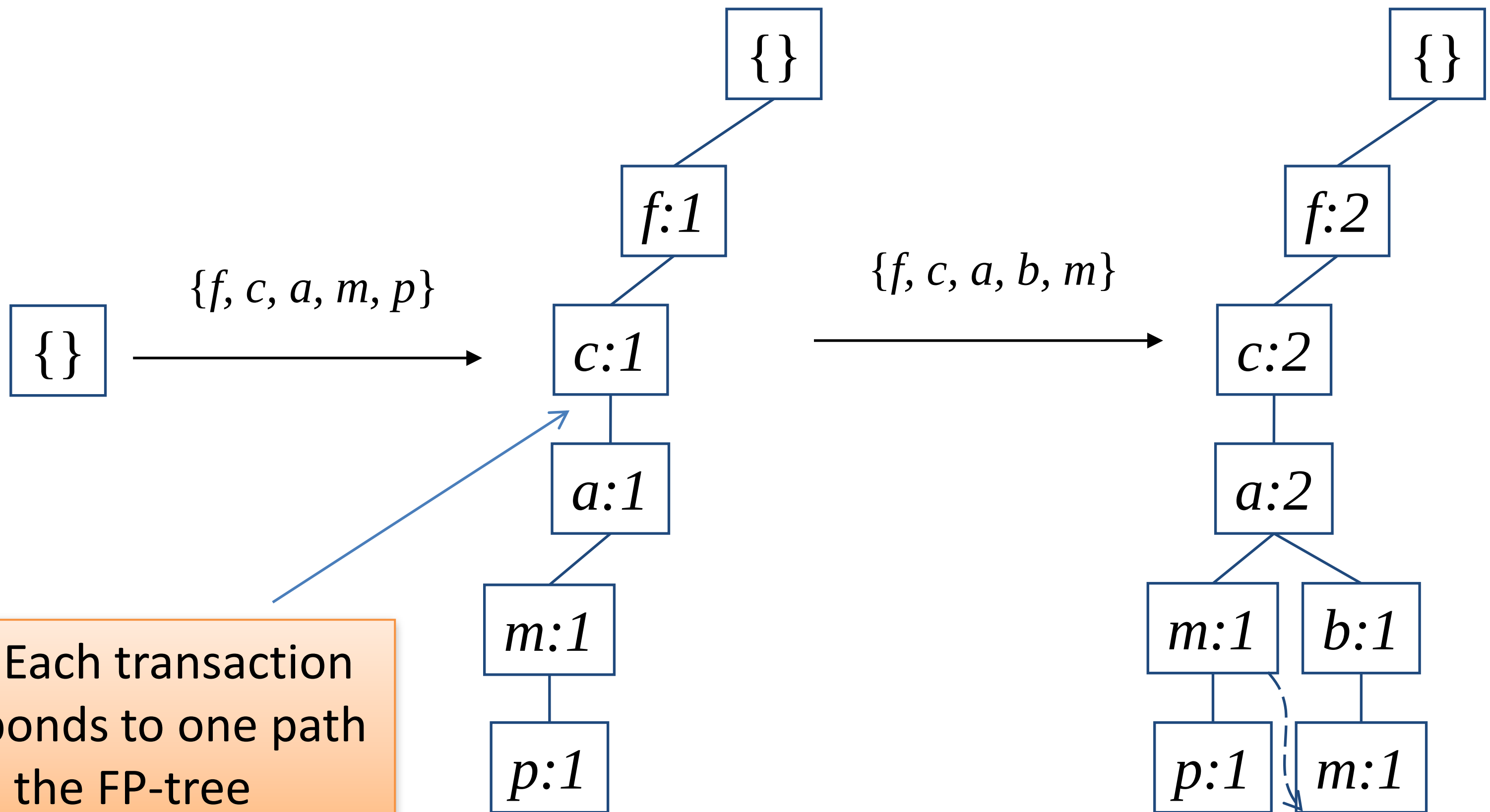
FP-tree Example: step 2

Step 2: construct FP-tree



FP-tree Example: step 2

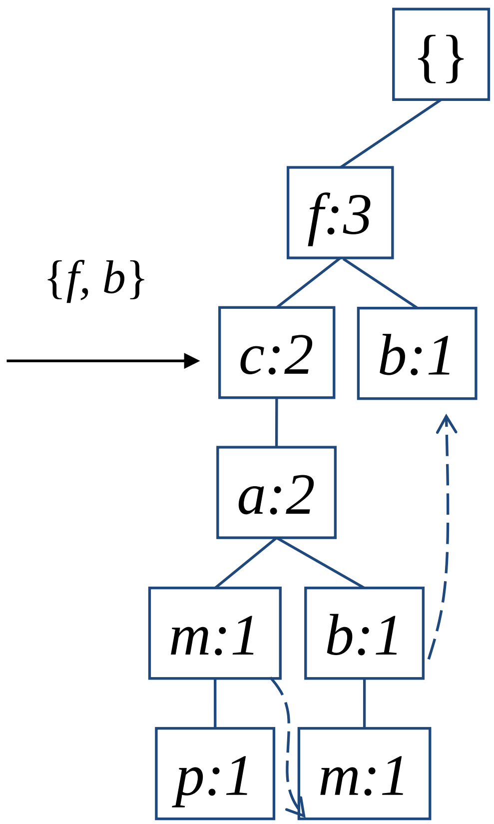
Step 2: construct FP-tree



NOTE: Each transaction corresponds to one path in the FP-tree

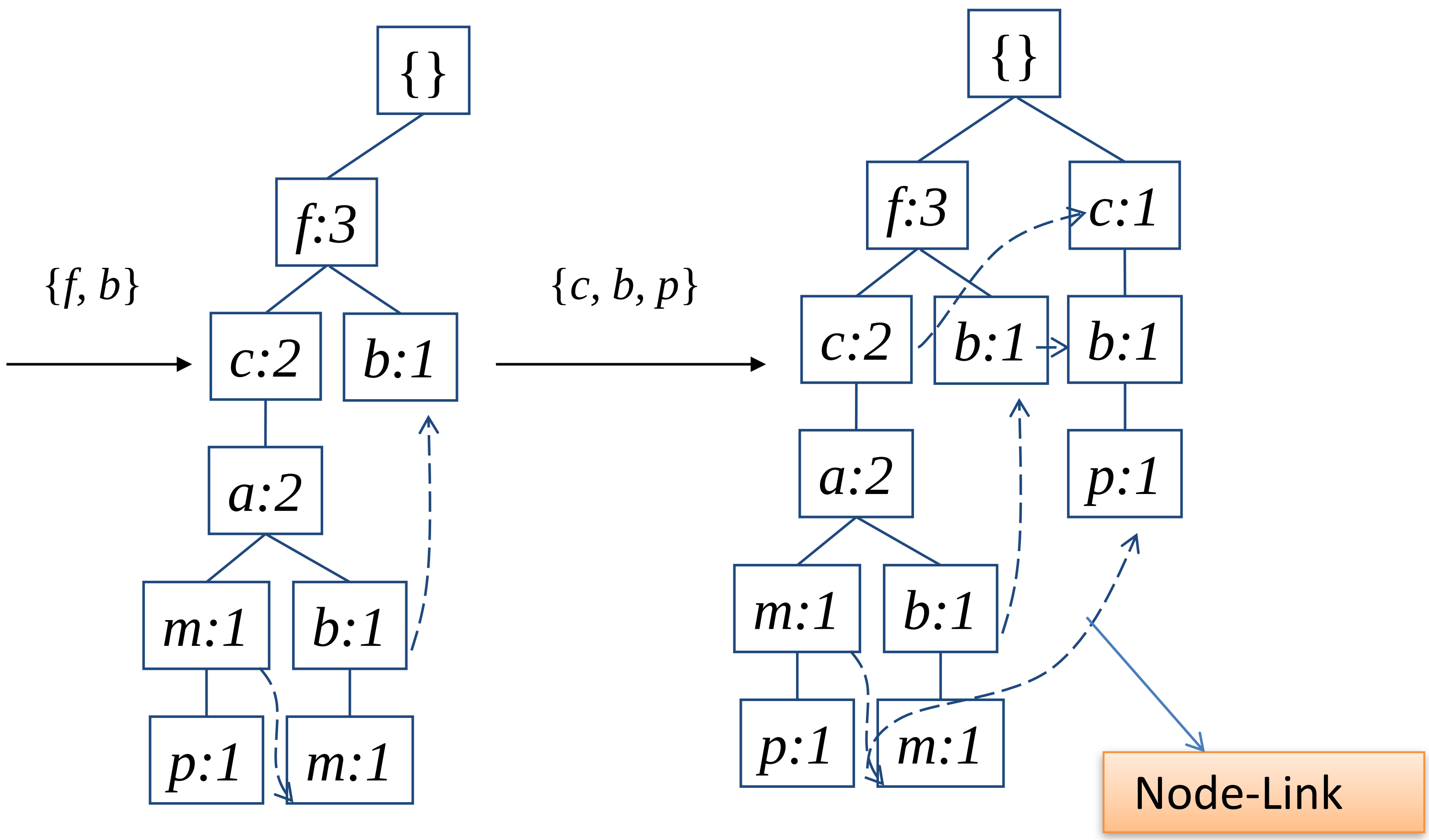
FP-tree Example: step 2

Step 2: construct FP-tree



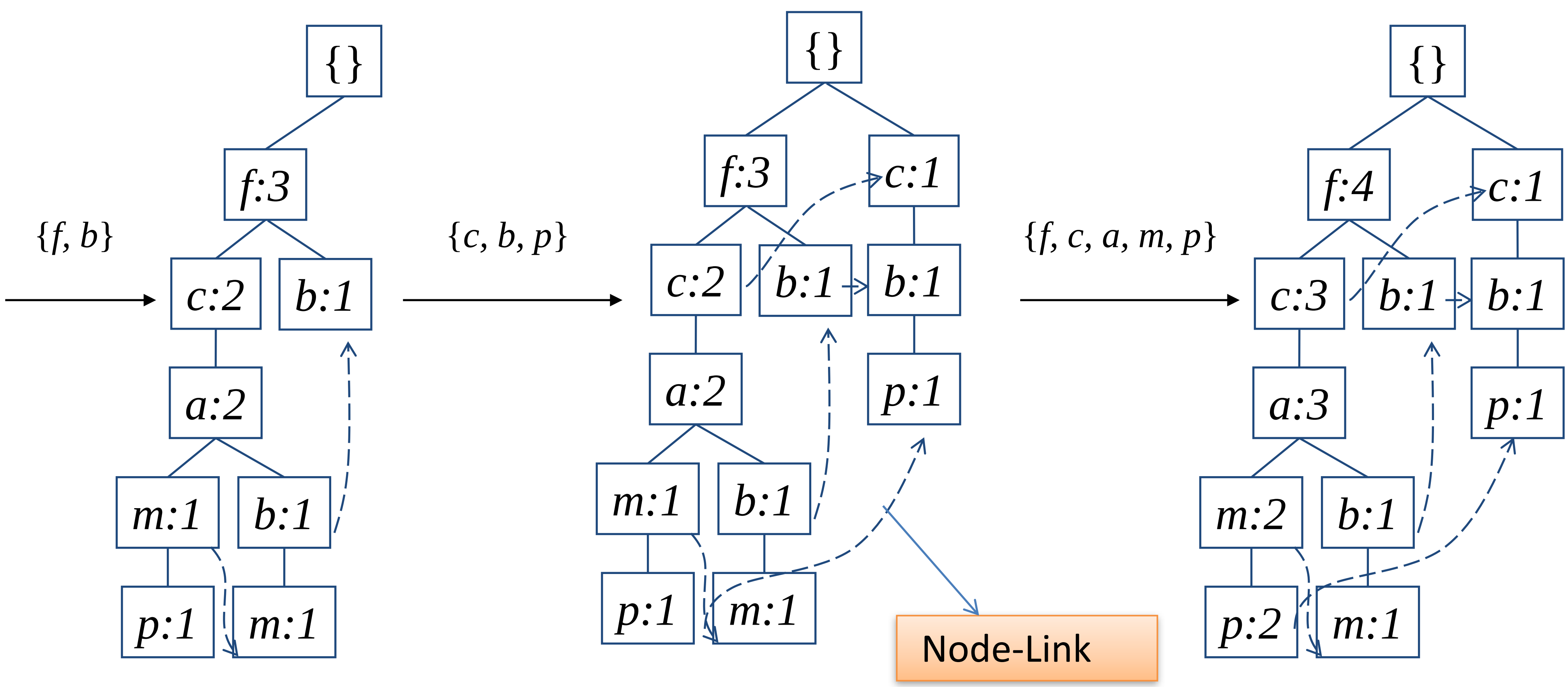
FP-tree Example: step 2

Step 2: construct FP-tree



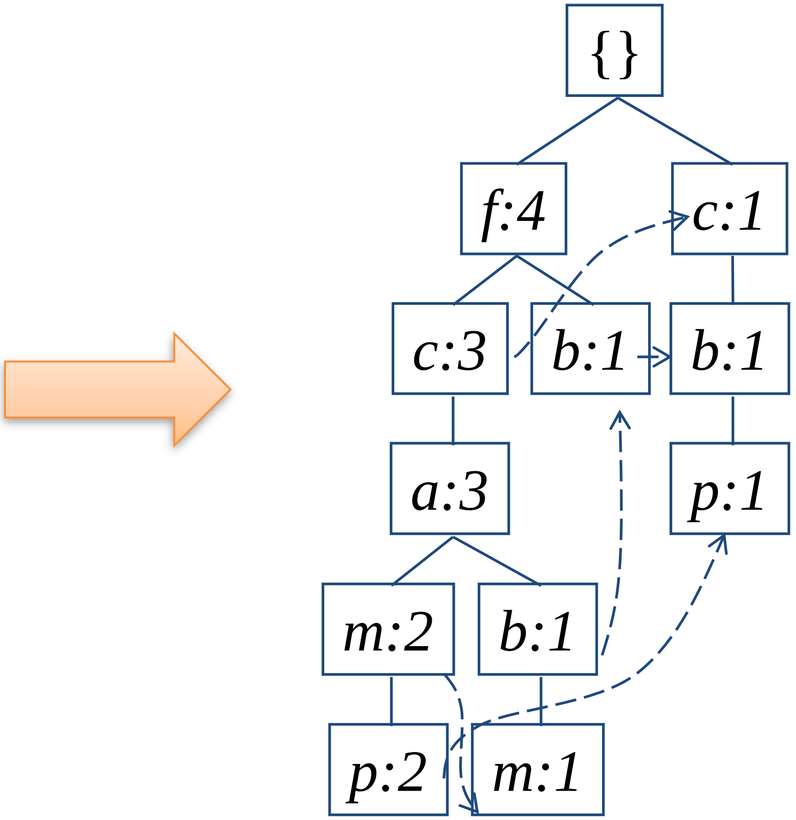
FP-tree Example: step 2

Step 2: construct FP-tree



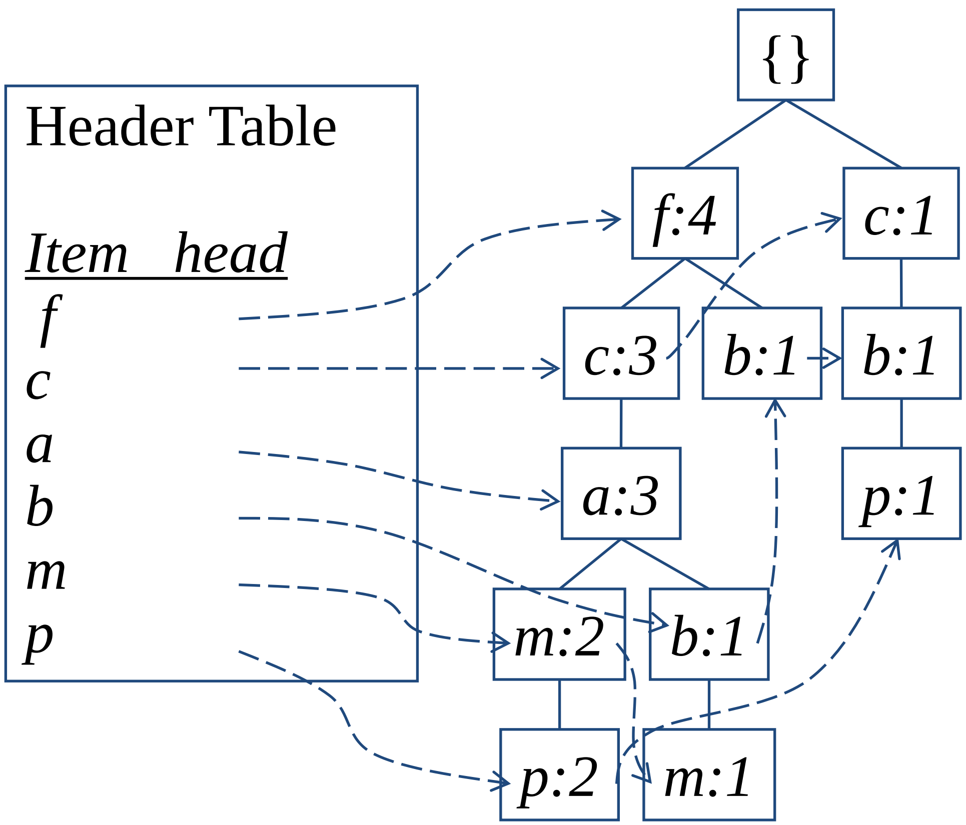
Items bought

{f, a, c, d, g, i, m, p}
{a, b, c, f, l, m, o}
{b, f, h, j, o}
{b, c, k, s, p}
{a, f, c, e, l, p, m, n}



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).

Compactness:

The size of the tree is bounded by the occurrences of frequent items

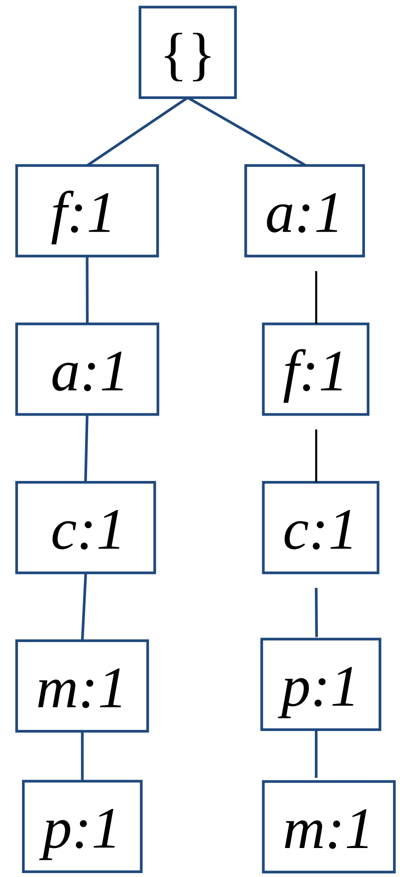
The height of the tree is bounded by the maximum number of items in a transaction

Questions?

Why descending order?

Example 1:

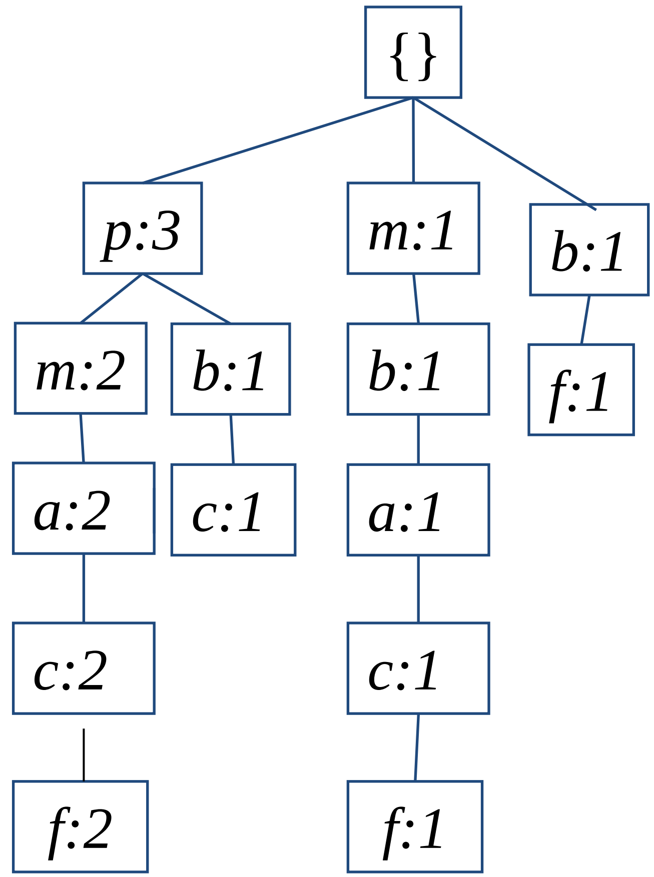
TID	*(unordered) frequent items*
100	{*f, a, c, m, p*}
500	{*a, f, c, p, m*}



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

Activity:

Use FP Tree algorithm to construct FP Tree. **Minimum Support = 2**

TID	ITEMS
1	A B C D
2	B C E
3	A B C E
4	B E F
5	A B F D
6	B C D E F