FP-Tree

T-61.6020: Popular Algorithms in Data Mining and Machine Learning

Outline

- Problem description
- Motivation for the FP-Tree algorithm
- The FP-Tree algorithm

Problem description

- Transaction database
  - Transactions consist of a set of items *I = {a, b, c, ...}*

TID	Items Bought
1	f, a, c, d, g, i, m, p
2	a, b, c, f, l, m, o
3	b, f, h, j, o
4	b, c, k, s, p
5	a, f, c, e, l, p, m, n

Problem description

- What products are often bought together?
  - (digital camera, memory card, extra battery)
- Problem: Find frequent item sets
  - *frequency ≥ minimum support threshold*
  - Same problem as Apriori

Apriori reminder

- http://www.cs.ualberta.ca/~zaiane/courses/cmput499/slides/Lect10/sld054.htm

Why FP-Tree and not Apriori?

- Apriori works well except when:
  - Lots of frequent patterns
    - Big set of items
    - Low minimum support threshold
  - Long patterns
- Why: Candidate sets become huge
  - 10^4 frequent patterns of length 1 $\rightarrow$ 10^7 length 2 candidates
  - Discovering pattern of length 100 requires at least 2^{100} candidates (nr of subsets)
  - Repeated database scans costly (long patterns)

FP-Tree: Ideas

- Avoid candidate set explosion by:
  1) Compact tree data structure
     - Avoid repeated database scans
  2) Restricted test-only
     - Apriori: restricted generation-and-test
  3) Search divide-and-conquer based
     - Apriori: bottom-up construction

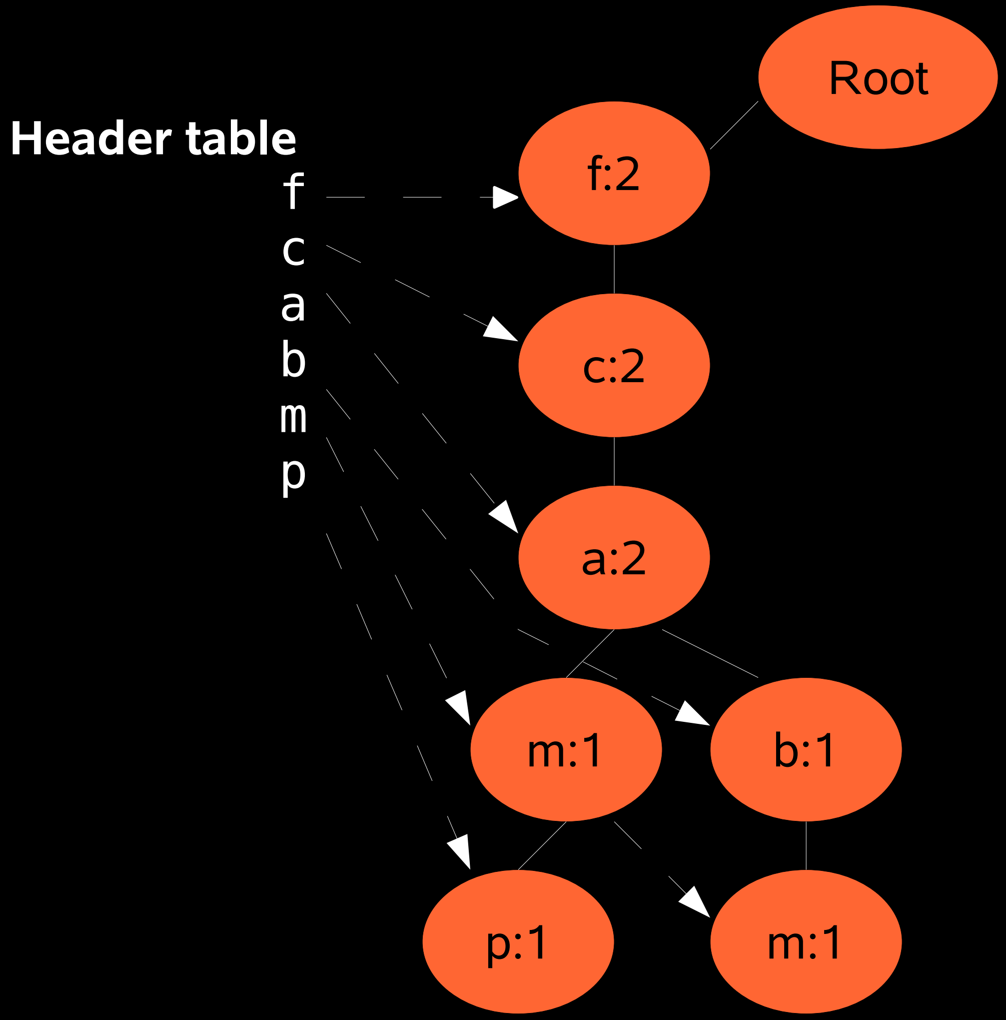
FP-Tree: Algorithm

- Order all items in itemset in frequency descending order (min support = 3)

TID	Items Bought	(Ordered) Frequent Items
1	f, a, c, d, g, i, m, p	f, c, a, m, p
2	a, b, c, f, l, m, o	f, c, a, b, m
3	b, f, h, j, o	f, b
4	b, c, k, s, p	c, b, p
5	a, f, c, e, l, p, m, n	f, c, a, m, p

(f:4, c:4, a:3, b:3, m:3, p:3)

FP-Tree: Data Structure



- Paths represent transactions
- Nodes have counts to track original frequency

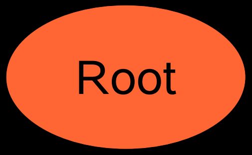
FP-Tree: Construction

- insert_tree([p|P], T)
    - If T has a child n, where n.item = p increment n.count by one
    - else create new node N with n.count = 1
        - Link it up from the header table

    If P is nonempty call insert_tree(P, N)

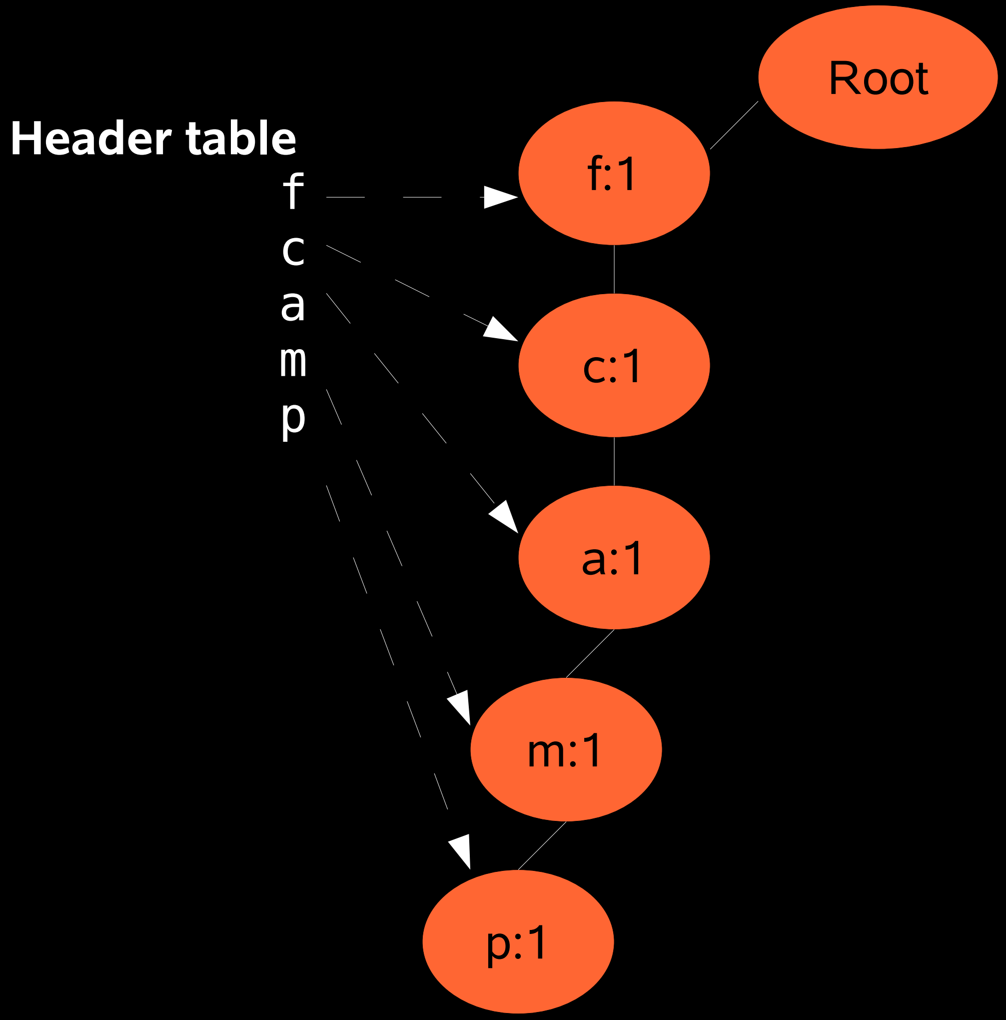
FP-Tree: Construction

- Originally empty

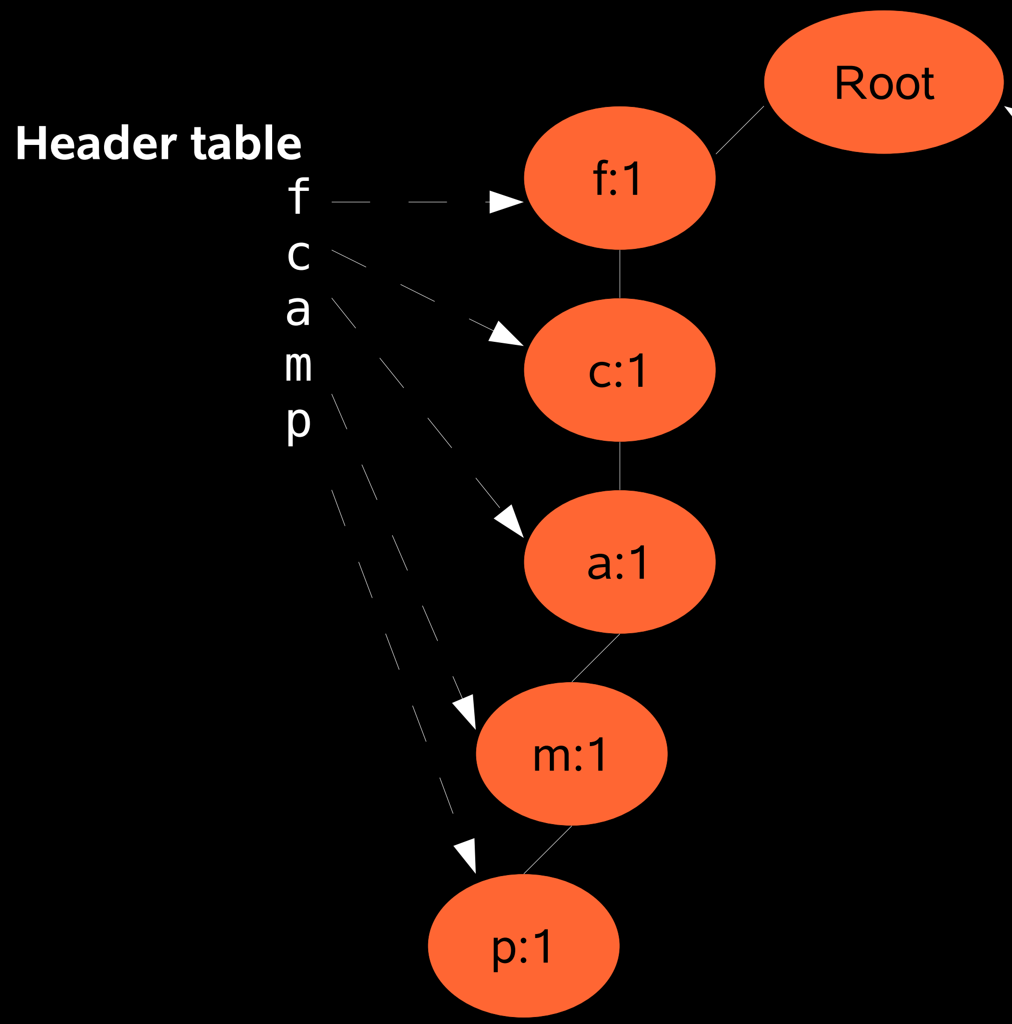


FP-Tree: Construction

- After inserting first transaction (f, c, a, m, p)



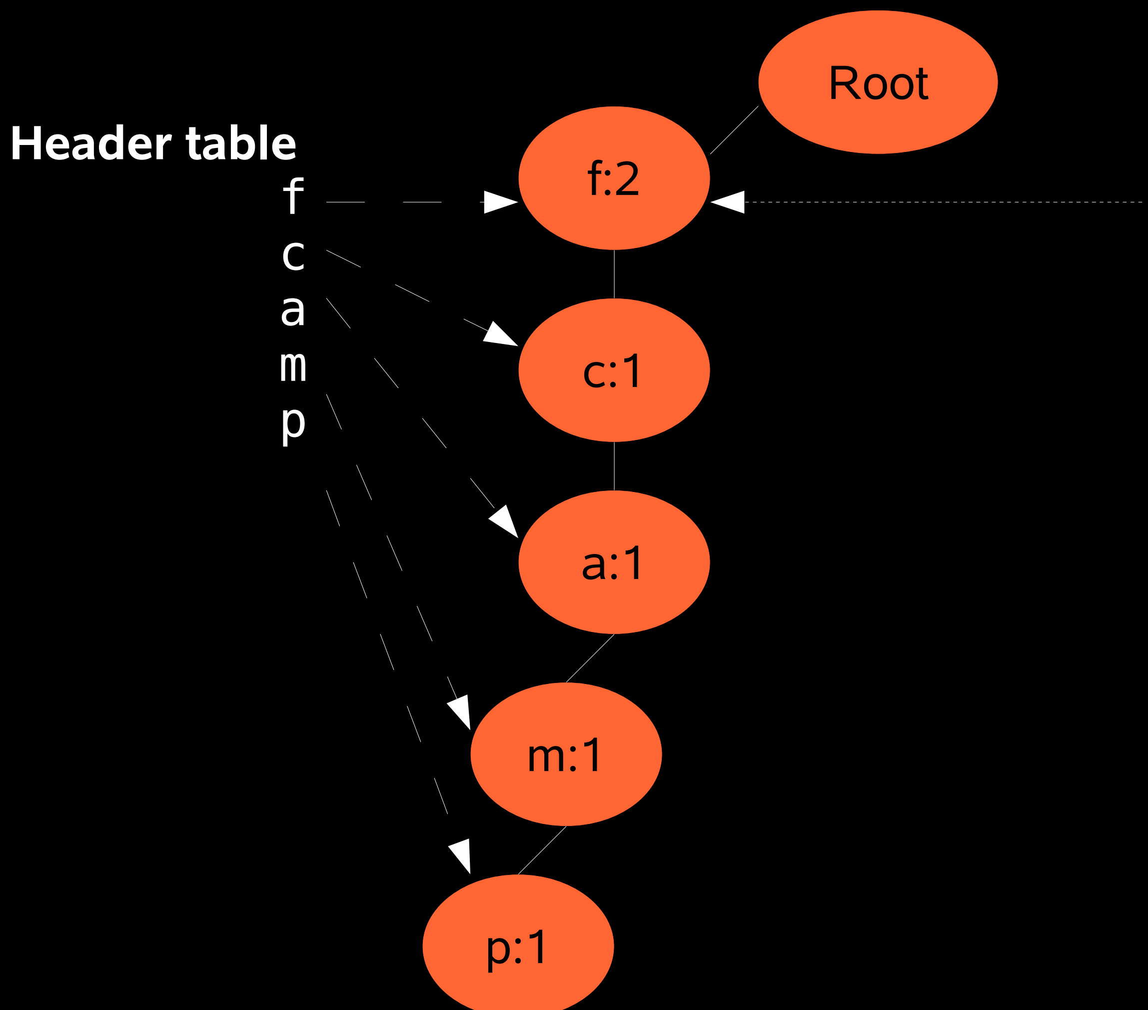
FP-Tree: Construction



P: (f, c, a, b, m)

T:

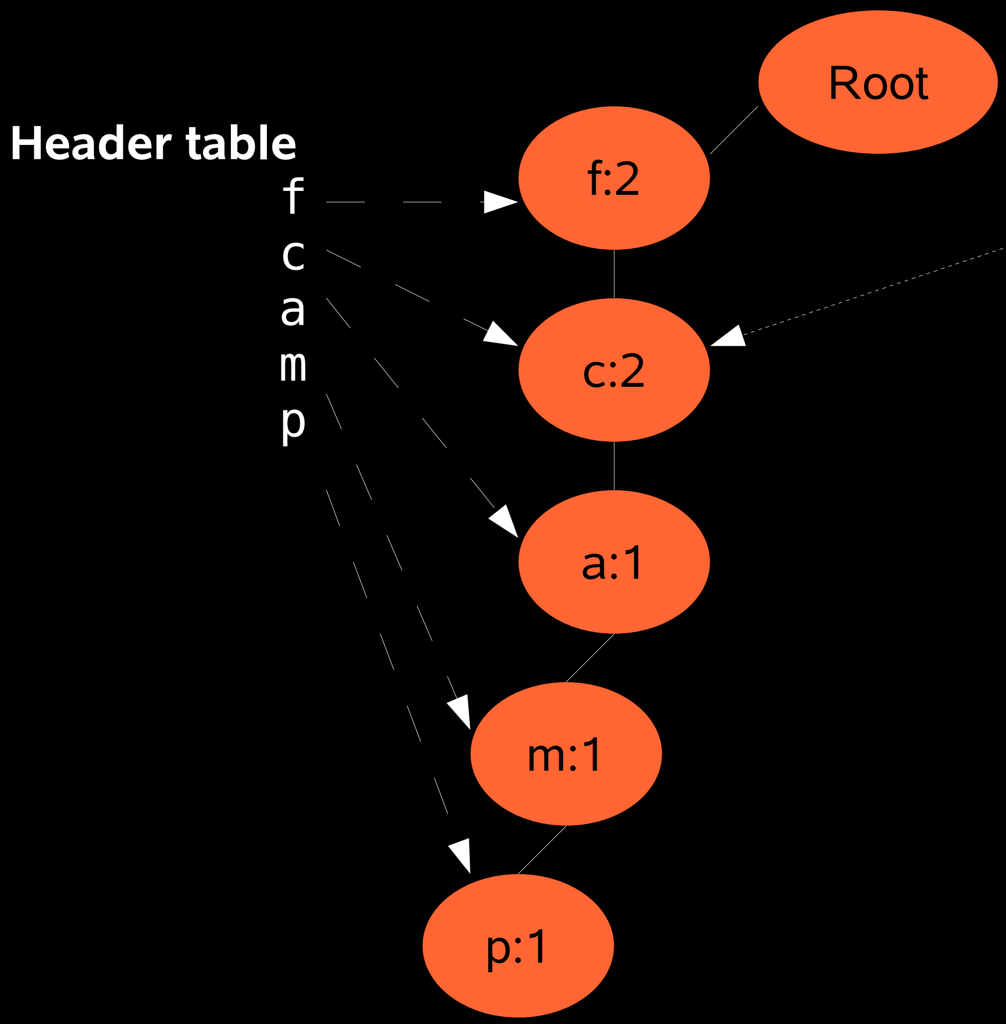
FP-Tree: Construction



P: (c, a, b, m)

T:

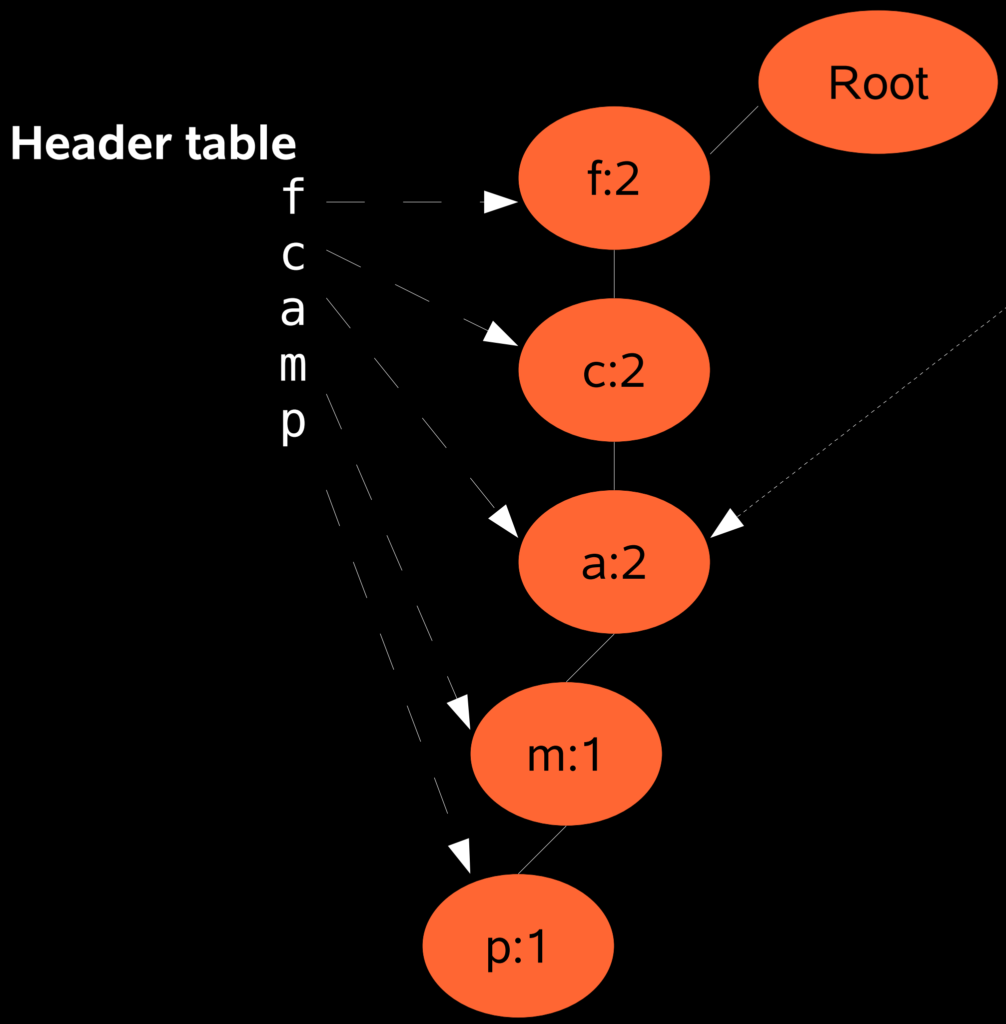
FP-Tree: Construction



P: (a, b, m)

T:

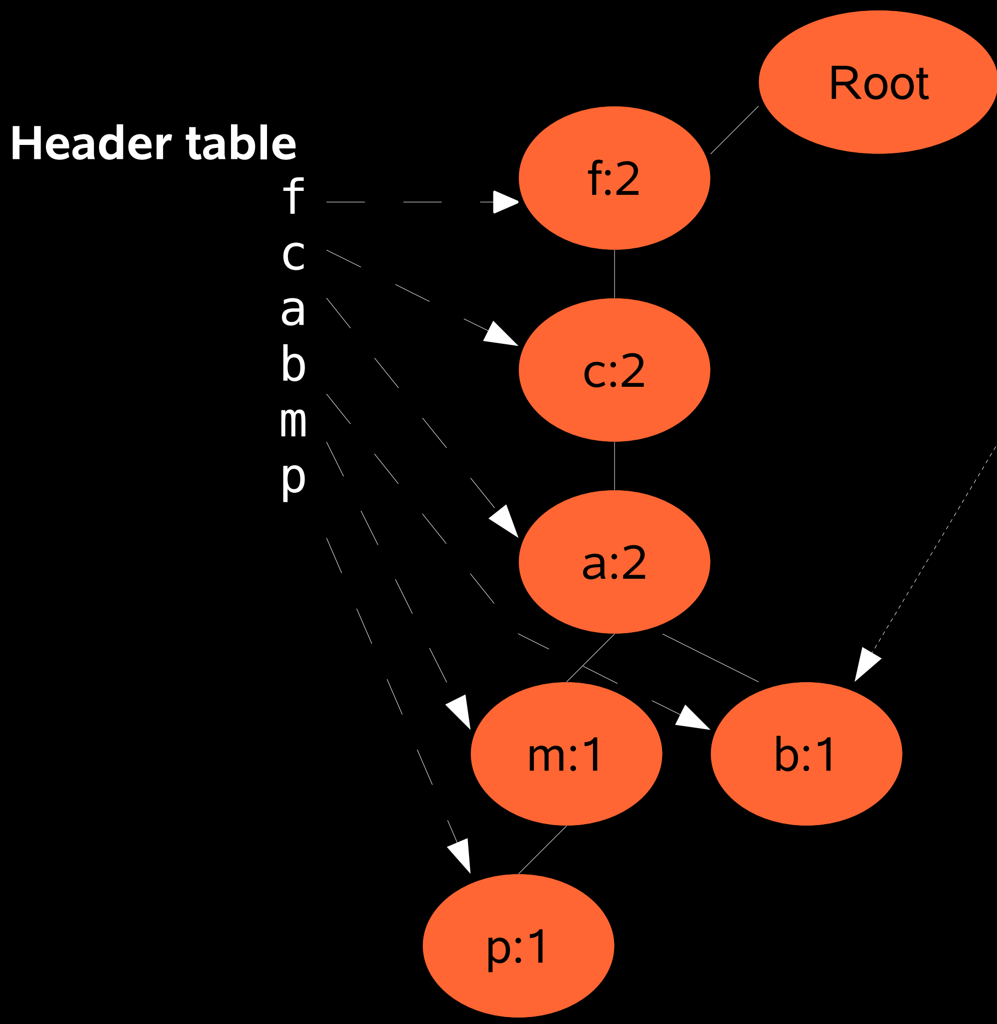
FP-Tree: Construction



P: (b, m)

T:

FP-Tree: Construction

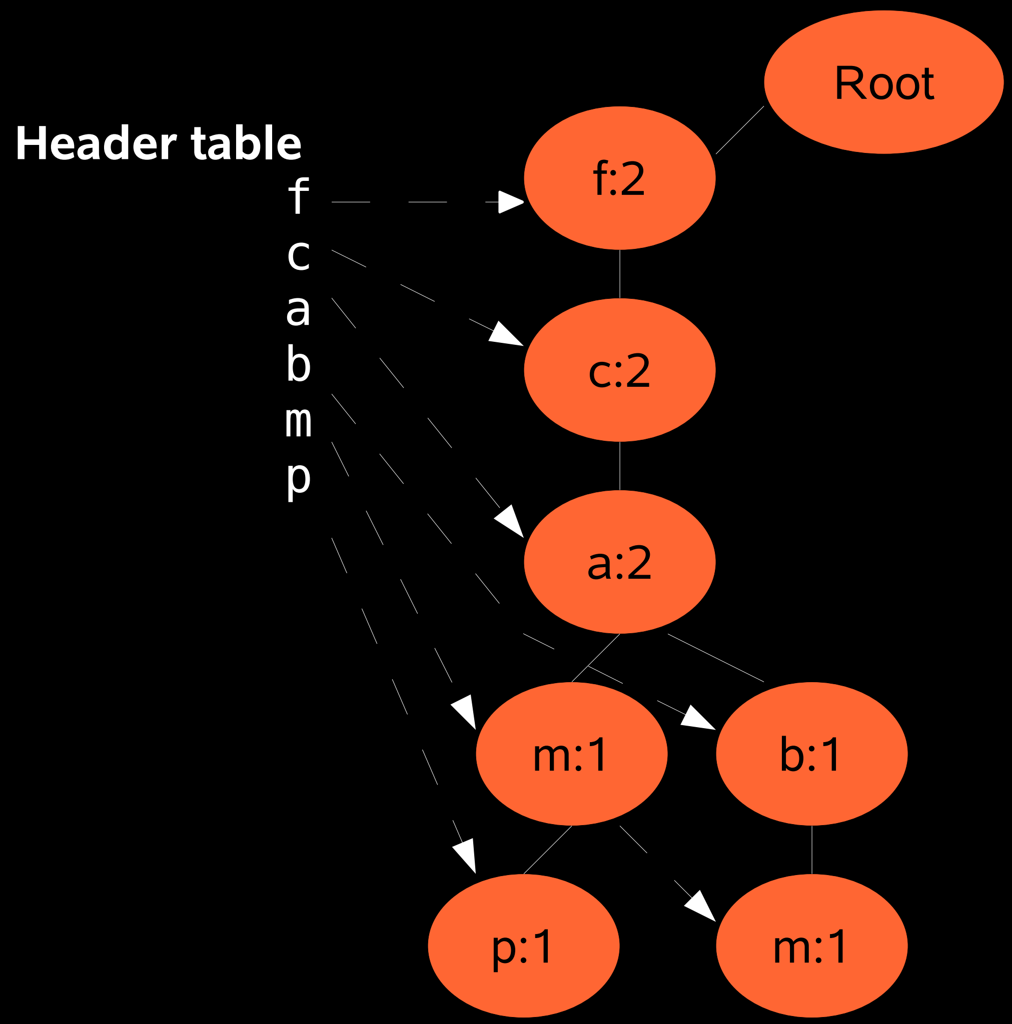


P: (m)

T:

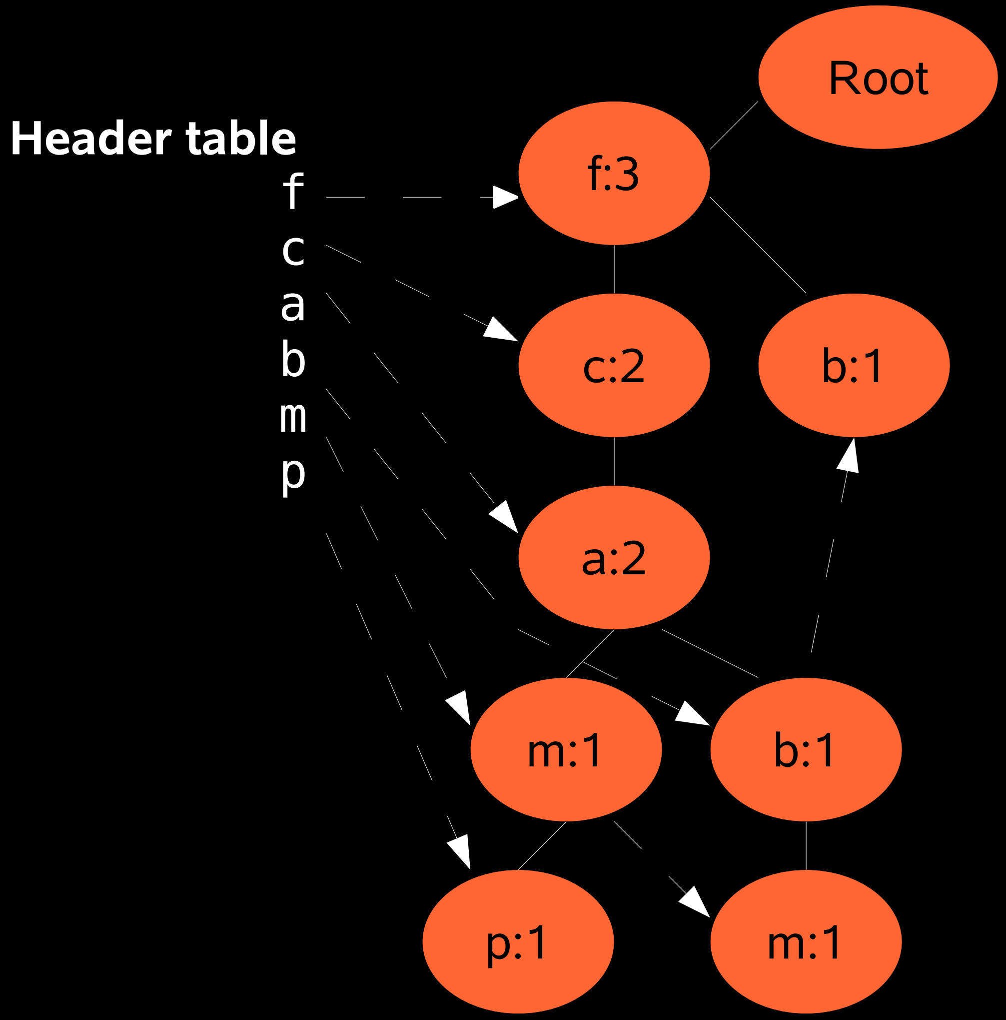
FP-Tree: Construction

- Second insertion complete



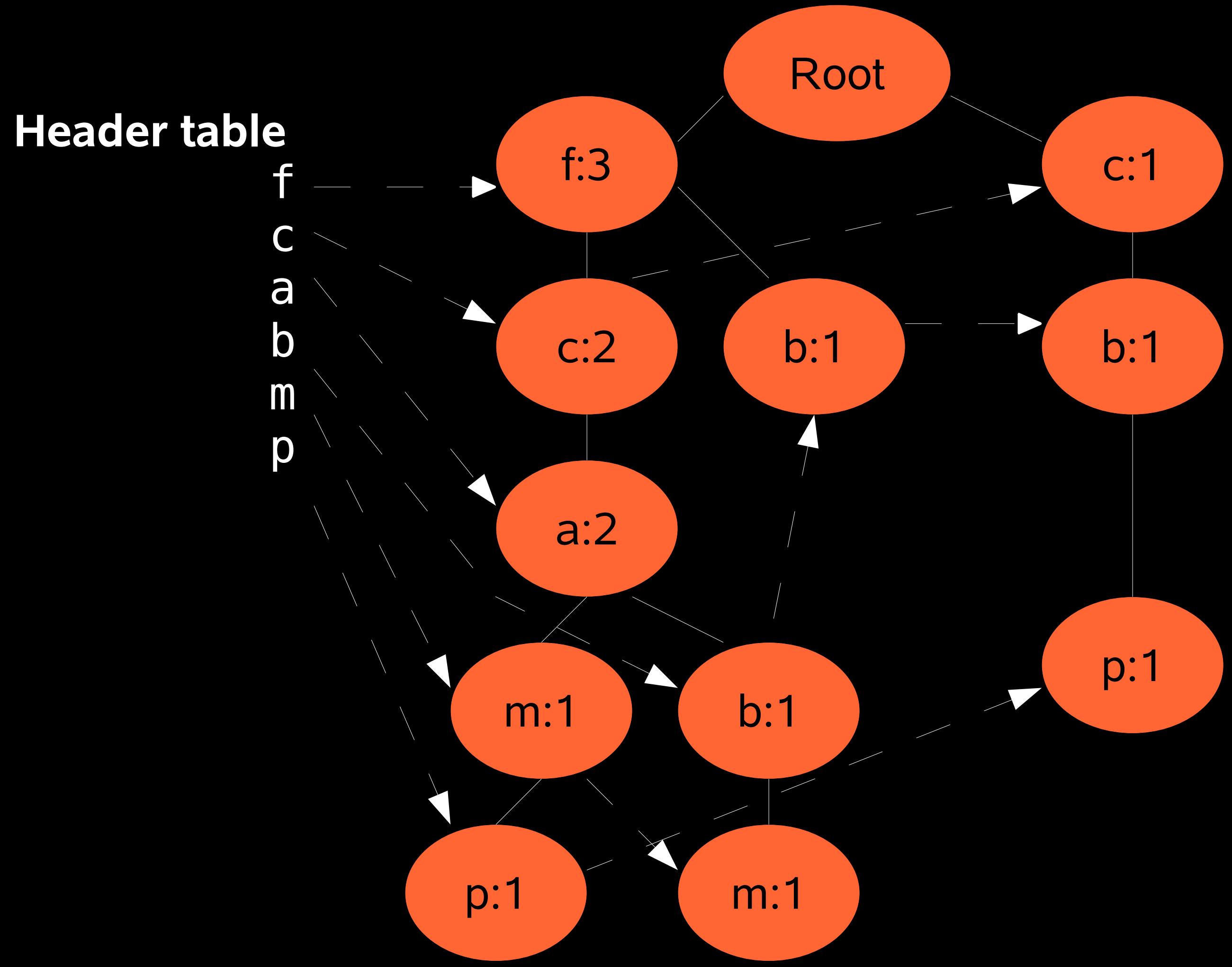
FP-Tree: Construction

- After insertion of third transaction (f, b)



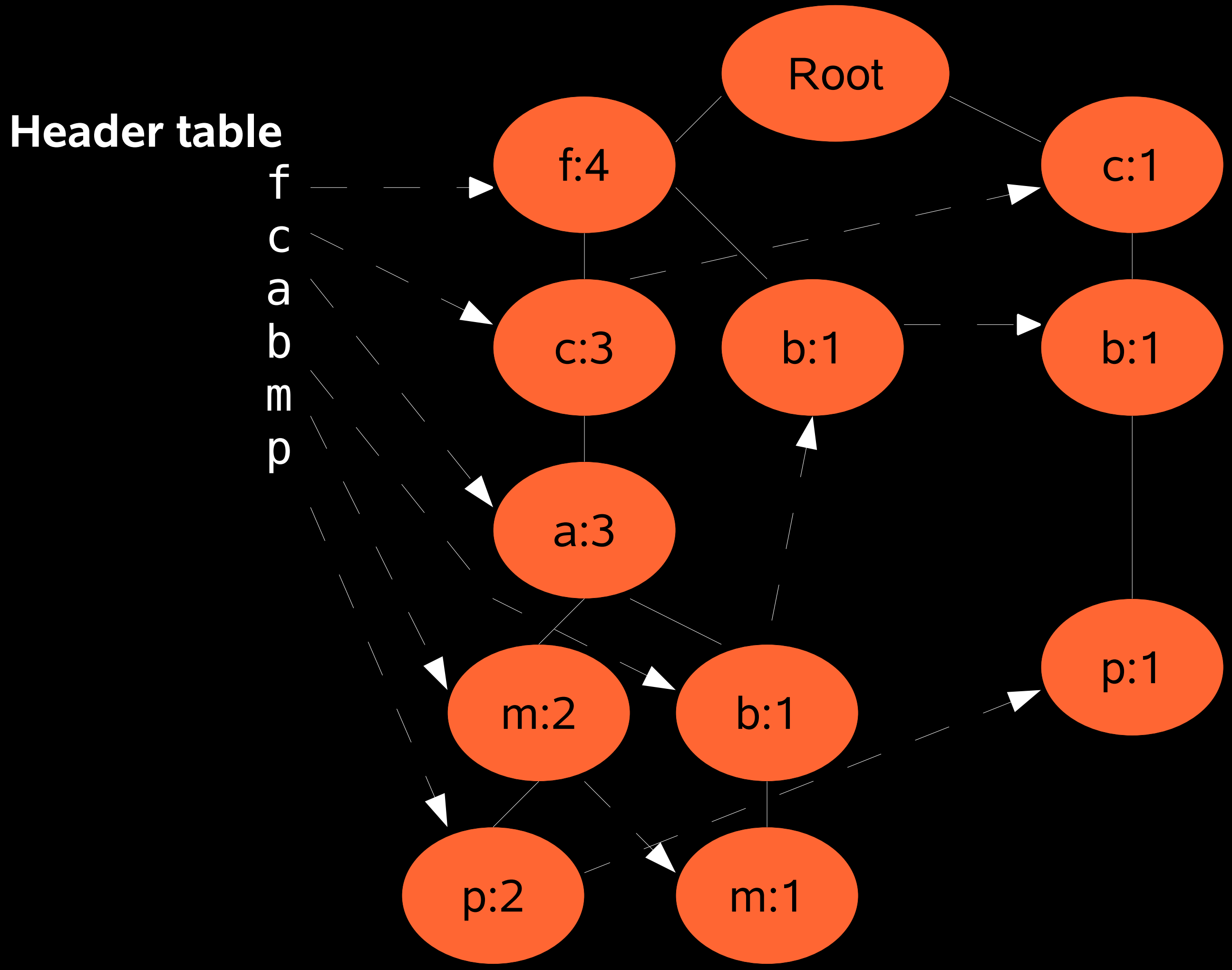
FP-Tree: Construction

- After insertion of fourth transaction (c, b, p)



FP-Tree: Construction

- After insertion of fifth transaction (f, c, a, m, p)



FP-Tree: Properties

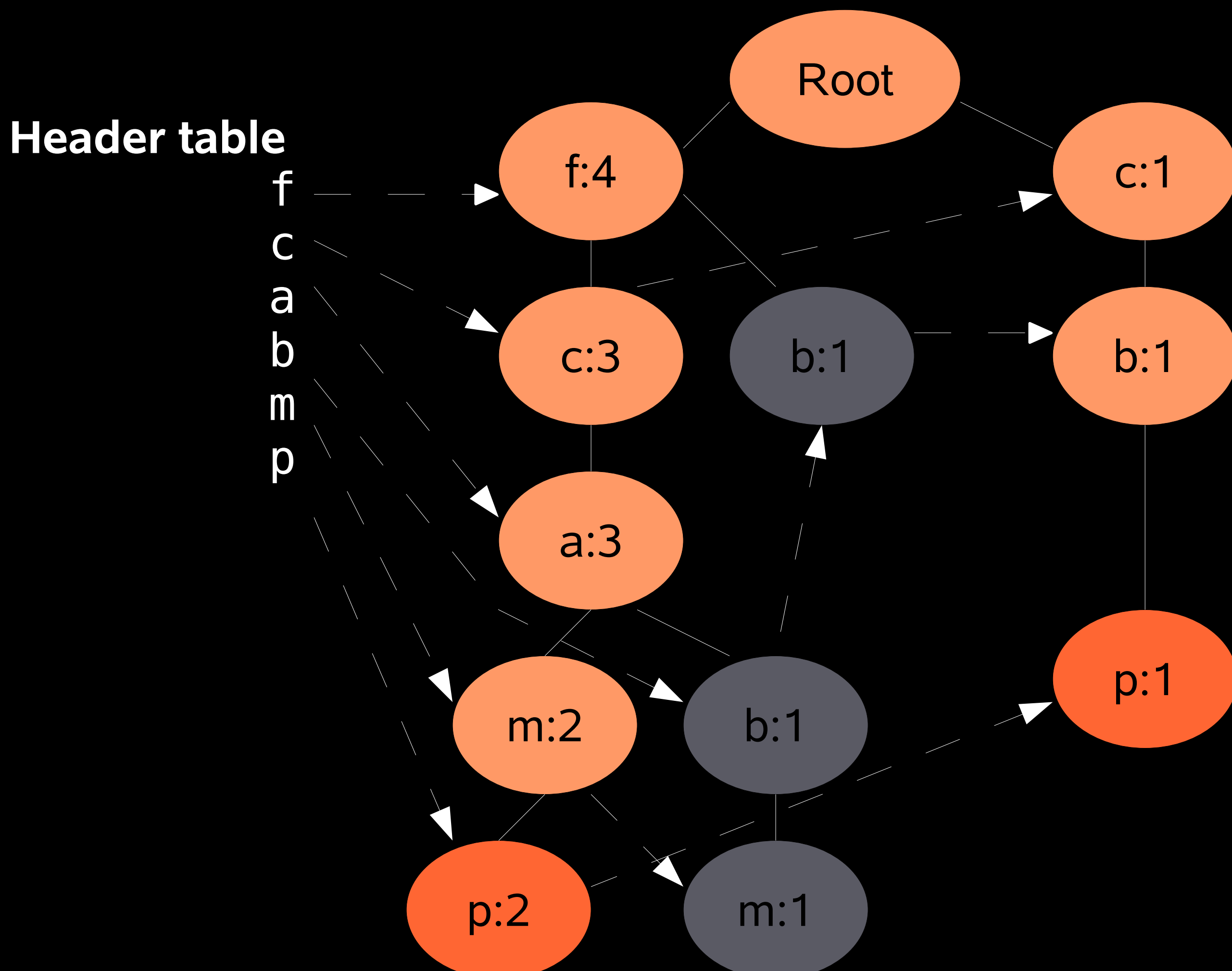
- The FP-Tree contains everything from the database we need to know for mining frequent patterns
- The size of the FP-tree is ≤ Occurrence of frequent patterns in database

Mining Frequent Patterns

- How do we get find all frequent patterns from the FP-Tree?
  - Intuitively:
    1) Find all frequent patterns containing one of the items
    2) Then find all frequent patterns containing the next item but NOT containing the previous one
    3) Repeat 2) until we're out of items

Finding all patterns with 'p'

- Starting from the bottom of the header table



Generate (p:3)

'p' exists in paths:

(f:4, c:3, a:3, m:2, p:2) and (c:1, b:1, p:1) process these further

Paths with 'p'

- We got (f:4, c:3, a:3, m:2, p:2) and (c:1, b:1, p:1)
- The transactions containing 'p' have p.count
- We get (f:2, c:2, a:2, m:2, p:2) and (c:1, b:1, p:1)
  - Since we know that 'p' is part of these we can drop 'p'

Conditional Pattern Base

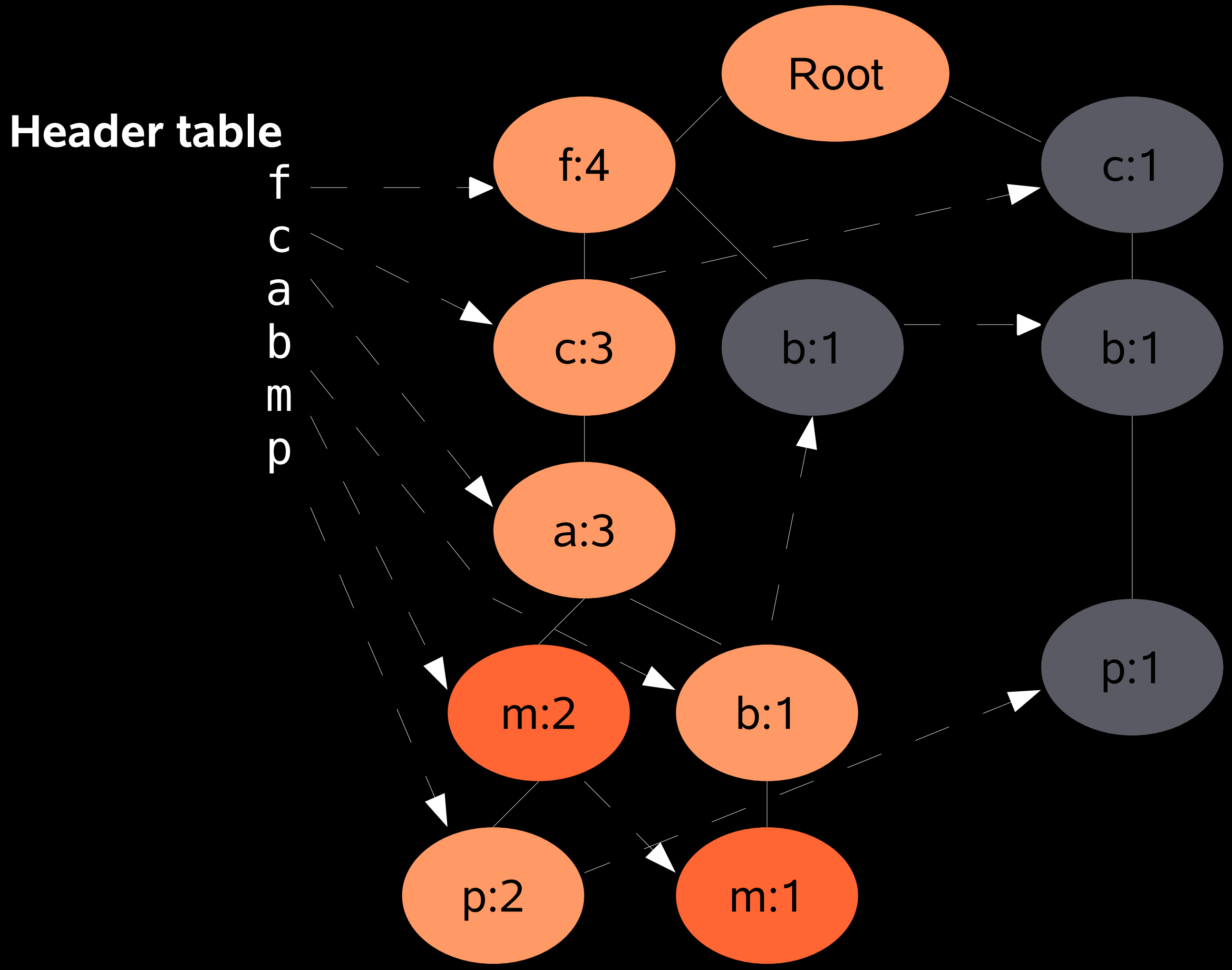
- We get paths (p dropped):
  - (f:2, c:2, a:2, m:2) and (c:1, b:1)
- Called *conditional pattern base (CPB)*
  - Contains transactions in which 'p' occurs
  - To find all frequent patterns containing 'p' we need to find all frequent patterns in the CPB and add 'p' to them
  - We can do this by constructing a new FP-Tree for the CPB

Finding all patterns with 'p'

- We again filter away all items < minimum support threshold
  - (f:2, c:2, a:2, m:2), (c:1, b:1) => (c:3)
- We generate (cp:3)
  - Support value is taken from the sub-tree
  - Frequent patterns thus far: (p:3, cp:3)

Patterns with 'm' but not 'p'

- Find 'm' from header table



Generate (m:3)

'm' exists in paths:

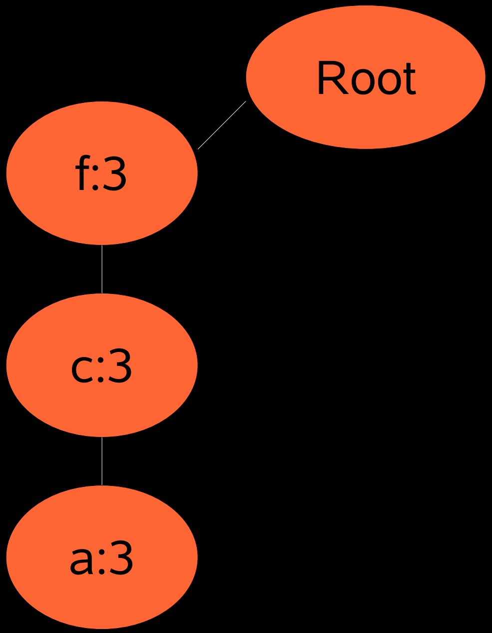
(f:4, c:3, a:3, m:2, p:2) and (f:4, c:3, a:3, b:1, m:1)

Patterns with 'm' but not 'p'

- Conditional Pattern Base:
  - (f:4, c:3, a:3, m:2, p:2) → (f:2, c:2, a:2)
  - (f:4, c:3, a:3, b:1, m:1) → (f:1, c:1, a:1, b:1)
  - Note: Only prefix considered
    - Systematic way of avoiding considering 'p'

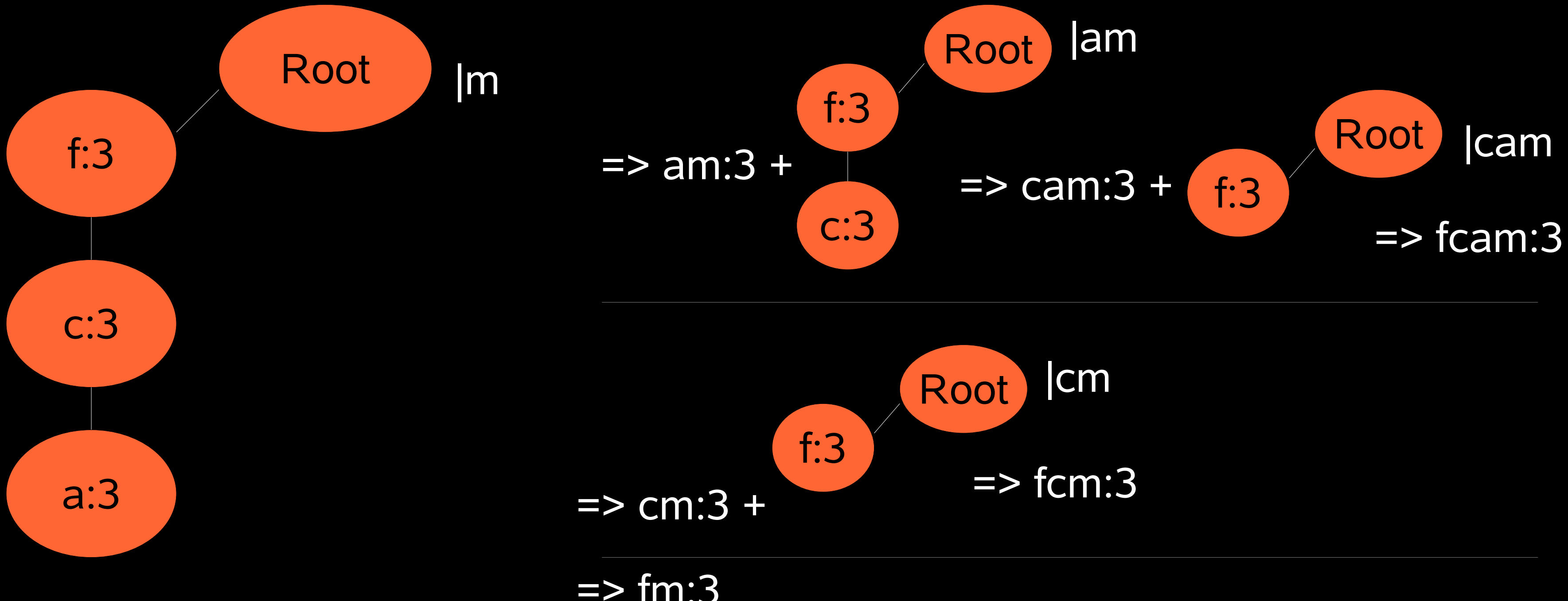
Patterns with 'm' but not 'p'

- Build FP-Tree from (f:2, c:2, a:2) and (f:1, c:1, a:1, b:1)
  - Initial filtering removes b:1
  - Resulting tree:



Conditional trees for 'm'

- Apply FP-Tree algorithm recursively to the new tree given 'm'
  - What this means is that to all frequent patterns found in this tree we add 'm'



Mining algorithm

```
FP-Growth(Tree, α)

for each(a_i in the header of Tree) do {
     β := a_i U α
     generate(β with support = a_i.support)
     construct β's conditional base pattern
     and β's conditional FP-Tree Tree_β

     if Tree_β≠Ø
     then call FP-growth(Tree_β, β)
```

Initially call:
FP-Growth(*Tree, null)*

Conclusions

- FP-Tree is an efficient algorithm for finding frequent patterns in transaction databases
- A compact tree structure is used
- Mining based on the tree structure is significantly more efficient than Apriori

References

Jiawei Han, Jian Pei, Yiwen Yin: *Mining Frequent Patterns without Candidate Generation* In Proceedings of the 2000 ACM SIGMOD international Conference on Management of Data (Dallas, Texas, United States, May 15 - 18, 2000). SIGMOD '00. ACM Press, New York, NY, 1-12.

Attributions

Cover image: http://flickr.com/photos/beccag/68389502/ (Creative Commons)