Mining Frequent Patterns without Candidate Generation

- Terminology
- Apriori-like Algorithms
  - Generate-and-Test
  - Cost Bottleneck
- FP-Tree and FP-Growth Algorithm
  - FP-Tree: Frequent Pattern Tree
  - FP-Growth: Mining frequent patterns with FP-Tree

Terminology

- Item set
  - A set of items: $I = \{a_1, a_2, \ldots\ldots, a_m\}$
- Transaction database
  - $DB = <T_1, T_2, \ldots\ldots, T_n>$
- Pattern
  - A set of items: A
- Support
  - The number of transactions containing A in DB
- Frequent pattern
  - A's support $\geq$ *minimum support threshold* ξ
- Frequent Pattern Mining Problem
  - The problem of finding the complete set of frequent patterns

Apriori-like Algorithms

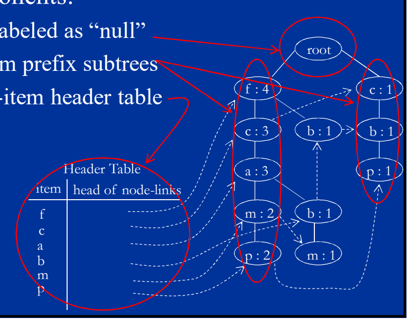
- Algorithm
  - Anti-Monotone Heuristic
    - If any length k pattern is not in the database, its length $(k+1)$ super-pattern can never be frequent
  - Generating candidate set
  - Testing candidate set
- Two non-trivial costs: (Bottleneck)
  - Candidate sets are huge. (They are pruned already but still increase exponentially with stage number k).
  - Repeated scan the database and test the candidate set by pattern matching.

FP-Tree and FP-Growth Algorithm

- FP-Tree: Frequent Pattern Tree
  - Compact presentation of the DB without information loss.
  - Easy to traverse, can quickly find out patterns associated with a certain item.
  - Well-ordered by item frequency.
- FP-Growth Algorithm
  - Start mining from length-1 patterns
  - Recursively do the following
    - Constructs its conditional FP-tree
    - Concatenate patterns from conditional FP-tree with suffix
  - Divide-and-Conquer mining technique

FP-Tree Definition

- Three components:
  - One root: labeled as “null”
  - A set of item prefix subtrees
  - A frequent-item header table



FP-Tree Definition (cont.)

- Each node in the item prefix subtree consists of three fields:
  - item-name
  - node-link
  - count
- Each entry in the frequent-item header table consists of two fields:
  - item-name
  - head of node-link

Example 1: FP-Tree Construction

- The transaction database used (fist two column only):

TID	Items Bought	(Ordered) Frequent Items
100	*f,a,c,d,g,i,m,p*	*f a c m p* – ***f c a m p***
200	*a,b,c,f,l,m,o*	*a b c f m* – ***f c a b m***
300	*b,f,h,j,o*	*b f* - ***f b***
400	*b,c,k,s,p*	*b c p* – ***c b p***
500	*a,f,c,e,l,p,m,n*	*a f c p m* – ***f c a m p***

a - 3
b - 3
c - 4
d - 1
e - 1
f - 4
g - 1
h - 1
i - 1
j - 1
k - 1
l - 2
m - 3
n -1
o - 2
p - 3
s -1

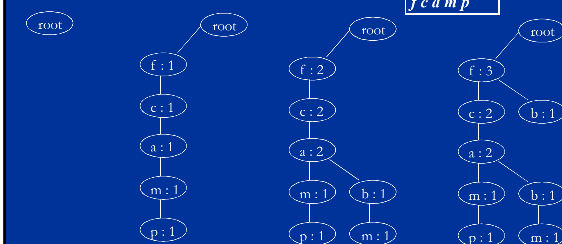
minimum support threshold ξ= 3

f c a b m p
a b c f m p
f c a b m p

Example 1 (cont.)

f c a m p
f c a b m
f b
c b p
f c a m p

The building process of the tree



Create root

After trans 1
(f,c,a,m,p)

After trans 2
(f,c,a,b,m)

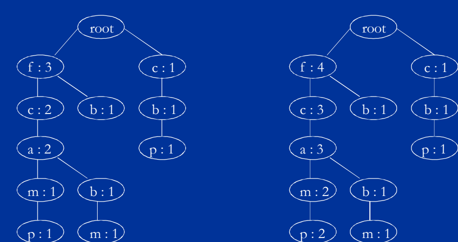
After trans 3
(f,b)

Example 1 (cont.)

- First Scan: //count and sort
  - count the frequencies of each item
  - collect length-1 frequent items, then sort them in support descending order into *L*, *frequent item list*.
    L = {(*f*:4), (*c*:4), (*a*:3), (*b*:3), (*m*:3), (*p*:3)}

Example 1 (cont.)

The building process of the tree (cont.)



After trans 4
(c,b,p)

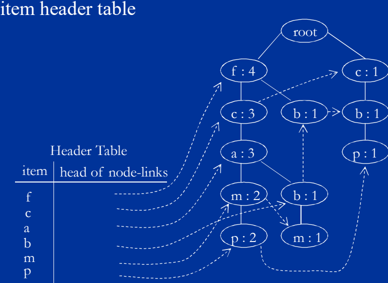
After trans 5
(f,c,a,m,p)

Example 1 (cont.)

- Second Scan://create the tree and header table
  - create the root, label it as "*null*"
  - for each transaction *Trans*, do
    - select and sort the frequent items in *Trans*
    - increase nodes count or create new nodes
      If prefix nodes already exist, increase their counts by 1;
      If no prefix nodes, create it and set count to 1.
  - build the item header table
    - nodes with the same item-name are linked in sequence via node-links

Example 1 (cont.)

Build the item header table



FP-Tree Properties

- Completeness
  - Each transaction that contains frequent pattern is mapped to a path.
  - Prefix sharing does not cause path ambiguity, as only path starts from root represents a transaction.
- Compactness
  - Number of nodes bounded by overall occurrence of frequent items.
  - Height of tree bounded by maximal number of frequent items in any transaction.

FP-growth(*Tree*, α)

```
Procedure FP-growth(Tree, α)
{
  if (Tree contains only a single path P)
    {  for each combination β of the nodes in P
          {  generate pattern β Uα;
             support = min(sup of all nodes in β)
          }
    }
  else // Tree contains more than one path
    {  for each a_i in the header of Tree
          {  generate pattern β= a_i Uα;
             β.support = a_i.support;
             construct β's conditional pattern base;
             construct β's conditional FP-tree Tree_β;
             if (Tree_β ≠ Φ)
                FP-growth(Tree_β , β);
          }
    }
}
```

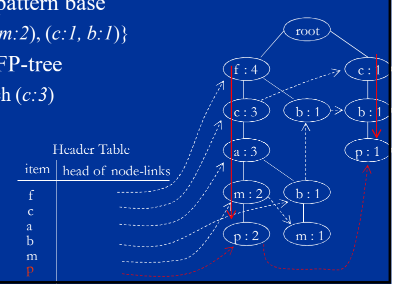
FP-Tree Properties (cont.)

- Traversal Friendly (for mining task)
  - 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.
  - This property is important for divide-and-conquer. It assures the soundness and completeness of problem reduction.

Example 2

- Start from the bottom of the header table: node *p*
- Two paths
- *p*'s conditional pattern base transformed prefix path
  - {(*f:2, c:2, a:2, m:2*), (*c:1, b:1*)}
- *p*'s conditional FP-tree
  - Only one branch (*c:3*)

    ➔pattern *(cp:3)*
- Patterns:
  - *(p:3)*
  - *(cp:3)*



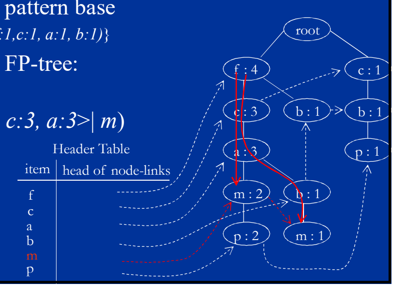
FP-Growth Algorithm

- Functionality:
  - Mining frequent patterns using FP-Tree generated before
- Input:
  - FP-tree constructed earlier
  - minimum support threshold ξ
- Output:
  - The complete set of frequent patterns
- Main algorithm:
  - Call FP-growth(*FP-tree*, *null*)

m fm cm am fcm fam cam fcam

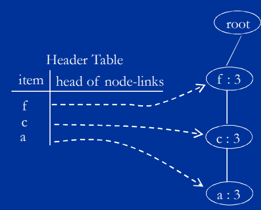
Example 2 (cont.)

- Continue with node *m*
- Two paths
- *m*'s conditional pattern base
  - {(*f:2, c:2, a:2*), (*f:1,c:1, a:1, b:1*)}
- *m*'s conditional FP-tree:
  - (*f:3, c:3, a:3*)
- Call mine(<*f:3, c:3, a:3*>| *m*)
- Patterns:
  - *(m:3)*



mine(<(*f:3, c:3, a:3*>| *m*)

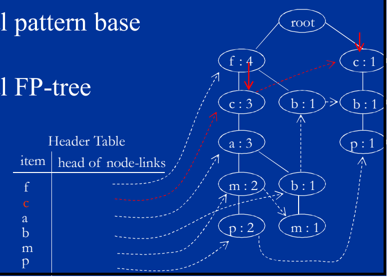
- node *a*:
  - *(am:3)*
  - call mine(<*f:3, c:3*>|*am*)
    - *(cam:3)*
    - call(<*f:3*)|*cam*)
      - *(fcam:3)*
    - *(fam:3)*
- node *c*:
  - *(cm:3)*
  - call mine(<*f:3*>|*cm*)
    - *(fcm:3)*
- node *f*:
  - *(fm:3)*
- All the patterns: *(m:3, am:3, cm:3, fm:3, cam:3, fam:3, fcm:3, fcam:3)*
- Conclusion: *A single path FP-Tree can be mined by outputting all the combination of the items in the path.*



conditional FP-tree of "m"

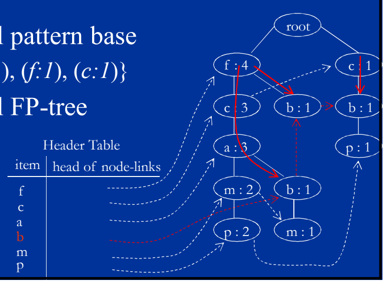
Example 2 (cont.)

- Continue with node *c*
- Two paths
- *c*'s conditional pattern base
  - {(*f:3*)}
- *c*'s conditional FP-tree
  - {(*f:3*)}
- Patterns:
  - *(c:4)*
  - *(fc:3)*



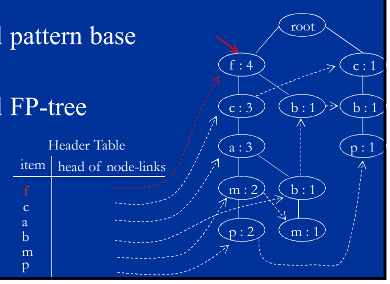
Example 2 (cont.)

- Continue with node *b*
- Three paths
- *b*'s conditional pattern base
  - {(*f:1, c:1, a:1*), (*f:1*), (*c:1*)}
- *b*'s conditional FP-tree
  - *Φ*
- Patterns:
  - *(b:3)*



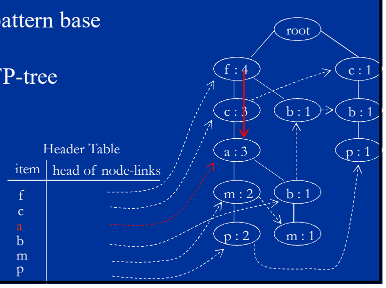
Example 2 (cont.)

- Continue with node *f*
- One path
- *f*'s conditional pattern base
  - *Φ*
- *f*'s conditional FP-tree
  - *Φ*
- Patterns:
  - *(f:4)*



Example 2 (cont.)

- Continue with node *a*
- One path
- *a*'s conditional pattern base
  - {(*f:3, c:3*)}
- *a*'s conditional FP-tree
  - {(*f:3, c:3*)}
- Patterns:
  - *(a:3)*
  - *(ca:3)*
  - *(fa:3)*
  - *(fca:3)*



Example 2 (cont.)

- Final results:

item	conditional pattern base	conditional FP-tree
p	{(f:2, c:2, a:2, m:2), (c:1, b:1)}	{(c:3)}| p – **p, cp**
m	{(f:4, c:3, a:3, m:2), (f:4, c:3, a:3, b:1, m:1)}	{(f:3, c:3, a:3)}| m **m fm cm am fcm cam fam fcam**
b	{(f:4, c:3, a:3, b:1), (f:4, b:1), (c:1, b:1)}	Φ **b**
a	{(f;3, c:3)}	{(f:3, c:3}| a **a fa ca fca**
c	{(f:3)}	{(f:3)}| c **c fc**
f	Φ	Φ **f**