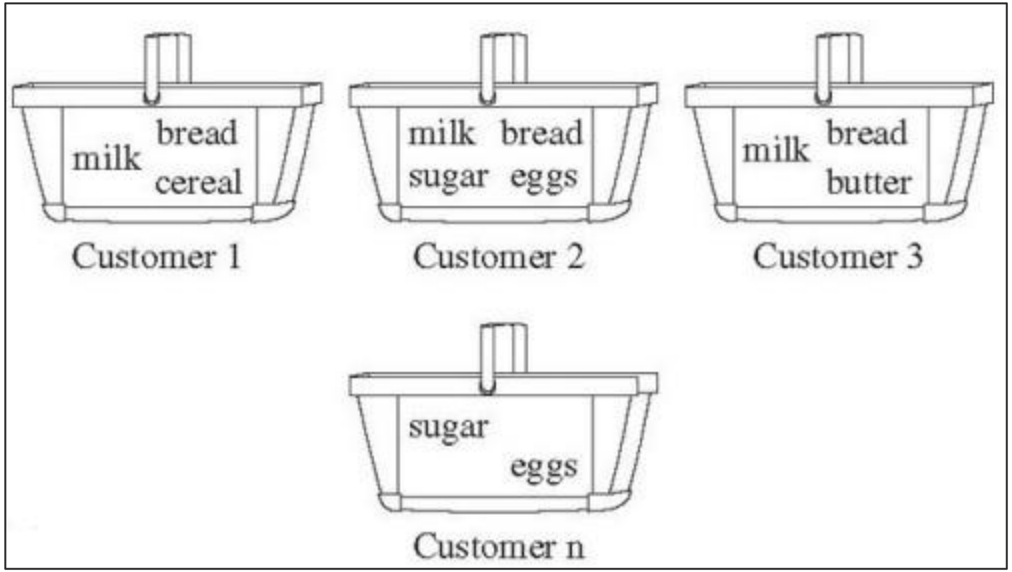
Association Rule Mining

Name: Deepak Yadav
Roll No.: 14075020

Scenario

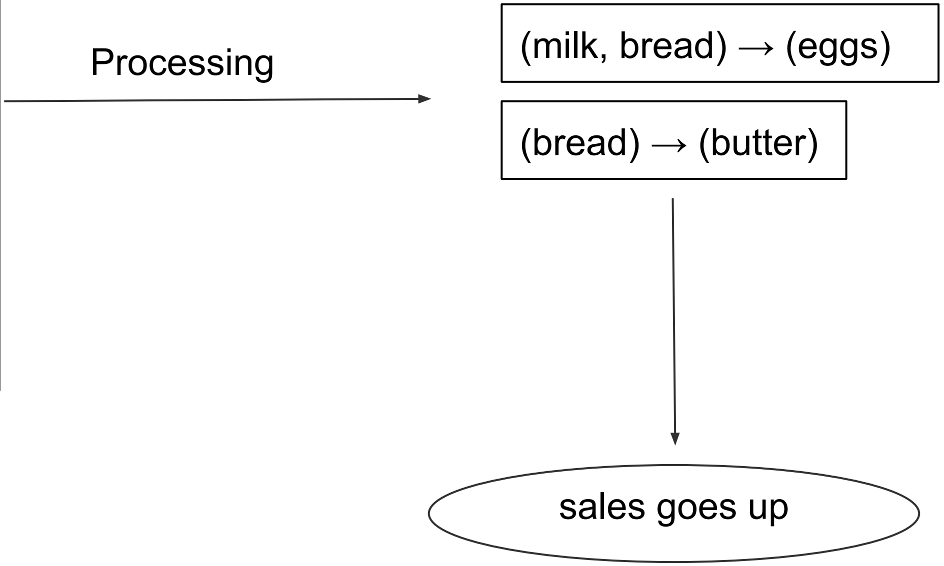


1.	milk, bread, cereal
2.	milk, bread, sugar, eggs
3.	milk, bread, butter
…...	……………………………………..
n.	sugar, eggs

Transaction database

Scenario

	Transaction database
1.	milk, bread, cereal
2.	milk, bread, sugar, eggs
3.	milk, bread, butter
…...	……………………………………..
n.	sugar, eggs



How to find useful patterns?

Itemset (I)
- A collection of one or more items.
- Examples: {milk, bread}, {eggs, bread, milk}

Support count (σ)
- Frequency of occurrence of an itemset
- Example: σ({milk, bread}) = 3

Support (s)
- Fraction of transactions that contains an itemset
- $s(I) = \sigma(I) / |T|$
- Example: s({milk, bread}) = 3 / 5

Confidence (c)
- $c(A \rightarrow B) = \sigma(A \cup B) / \sigma(A)$
- c({milk, bread} → {eggs}) = σ({milk, bread, eggs}) / σ({milk, bread})
  = 1 / 3

1.	milk, bread, cereal
2.	milk, bread, sugar, eggs
3.	milk, bread, butter
4.	milk, sugar
5.	sugar, eggs

Useful Patterns

A pattern **A** → **B** is useful if and only if

- **A** and **B** are non-empty itemset
- **A** ∩ **B** = Φ
- s(**A** U **B**) >= Ts (Threshold support) (Ts ∈ [0, 1])
- c(**A** → **B**) >= Tc (Threshold confidence) (Tc ∈ [0, 1])

Useful Patterns

Given $Ts = 0.5$ and $Tc = 0.6$. Is ({milk} → {bread}) a useful pattern?

$s(\{milk, bread\}) = 3 / 5 = 0.6$ ($>= Ts$)

$c(\{milk\} \rightarrow \{bread\}) = \sigma(\{milk, bread\}) / \sigma(\{milk\}) = 3 / 4 = 0.75$ ($>= Tc$)

Hence, ({milk} → {bread}) is a useful pattern

1.	milk, bread, cereal
2.	milk, bread, sugar, eggs
3.	milk, bread, butter
4.	milk, sugar
5.	sugar, eggs

Problem Statement

Given a transaction database(**T**), threshold support(**Ts**) and threshold confidence(**Tc**), our objective is to find all association rules(useful patterns).

1.	milk, bread, cereal
2.	milk, bread, sugar, eggs
3.	milk, bread, butter
…… ..	………………………… ………..
n.	sugar, eggs

Association Rule Mining →

(milk, bread) → (eggs)

(bread) → (butter)

Frequent Itemset

An itemset **I** is said to be frequent iff s(**I**) >= Ts

1.	milk, bread, cereal
2.	milk, bread, sugar, eggs
3.	milk, bread, butter
4.	milk, sugar
5.	sugar, eggs

Ts = 0.6
min_support_count = 0.6*|T| = 0.6 * 5 = 3

{milk, bread} is frequent, since, $\sigma(\{milk, bread\}) = 3$

{sugar} is frequent, since, $\sigma(\{sugar\}) = 3$

{milk, butter} is not frequent, since, $\sigma(\{milk, butter\}) = 1$

{milk} is frequent, since, $\sigma(\{milk\}) = 4$

Association Rule Mining

Association Rule Mining is done in 2 steps

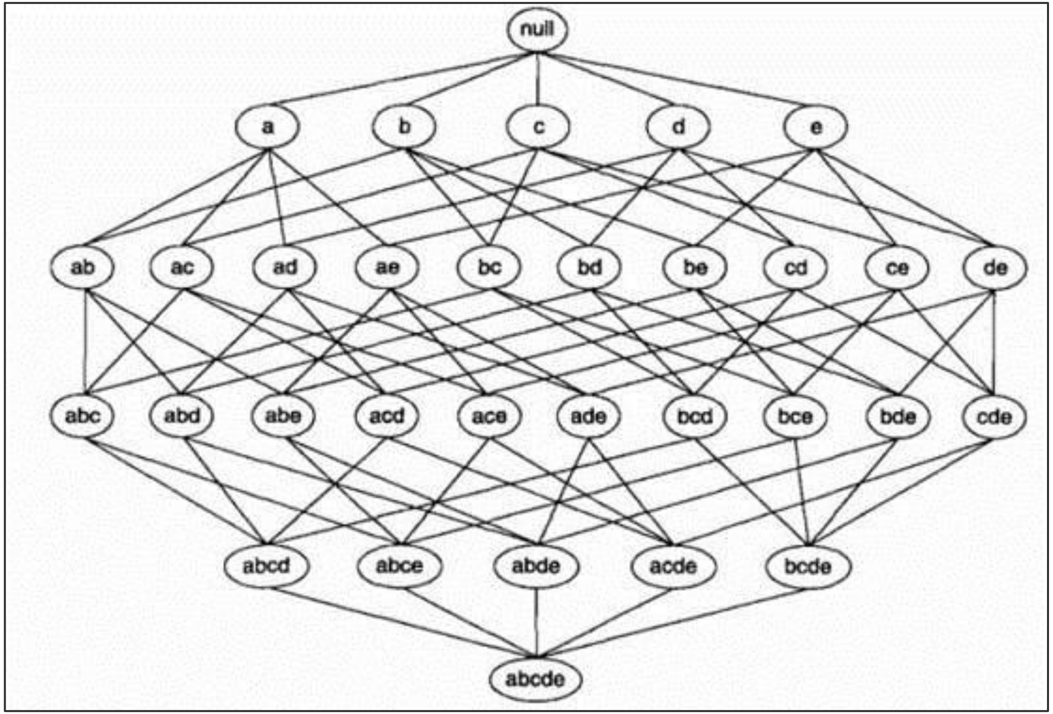
1) Find all frequent itemsets

2) Generate association rules from the frequent itemsets

Step 1: Find all frequent itemsets

1) Brute Force approach
2) Apriori algorithm
3) FP-Growth algorithm

Brute Force Approach

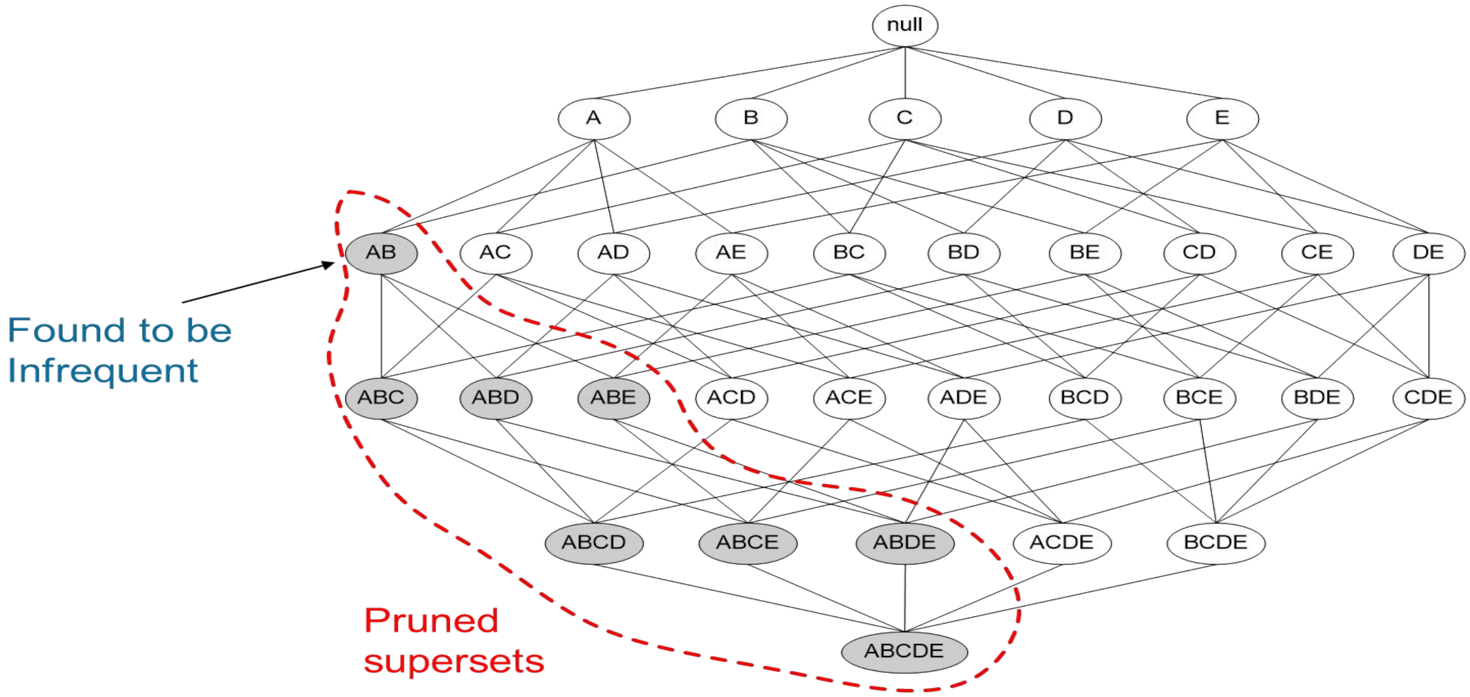


- Given P items, there are 2^P possible candidate itemsets
- Count the support of each candidate by scanning the database
- Complexity ~ O(2^P * Q)
  - Q is the cost of scanning the database

Expensive approach

Apriori Algorithm

The Apriori Principle: If an itemset is frequent, then all of its subsets must also be frequent. Conversely, if an itemset is infrequent, then all of its supersets must be infrequent, too.



Apriori Algorithm

L(k) : set of frequent itemsets of length k

C(k) : set of candidate itemsets (may or may not be frequent) of length k

Important observations

- If a k-itemset is frequent then all its subset of length k-1 will be frequent
- C(k) can be obtained from L(k-1)
- L(k) can be obtained from C(k)

How C(k) can be obtained from L(k-1)?

C(k) is generated by joining L(k-1) with itself

Join $L_{k\text{-}1}$ *p* with $L_{k\text{-}1}$q, as follows:

insert into C_k

select $p.\text{item}_1, p.\text{item}_2, \ldots, p.\text{item}_{k\text{-}1}, q.\text{item}_{k\text{-}1}$

from $L_{k\text{-}1}$ *p*, $L_{k\text{-}1}$q

where $p.\text{item}_1 = q.\text{item}_1, \ldots\ p.\text{item}_{k\text{-}2} = q.\text{item}_{k\text{-}2}, p.\text{item}_{k\text{-}1} < q.\text{item}_{k\text{-}1}$

How C(k) can be obtained from L(k-1)?

Join L_{k-1} *p* with L_{k-1}q, as follows:
insert into C_k
select *p*.item$_1$, *p*.item$_2$, . . . , *p*.item$_{k-1}$, *q*.item$_{k-1}$
from L_{k-1} *p*, L_{k-1}q
where *p*.item$_1$ = *q*.item$_1$, . . . *p*.item$_{k-2}$ = *q*.item$_{k-2}$, *p*.item$_{k-1}$ < *q*.item$_{k-1}$

- L(3) = {{a, b, c}, {a, b, d}, {a, c, d}, {a, c, e}}
- {a, b, c} and {a, b, d} can be merged to obtain {a, b, c, d} which is then added to C(4)
- {a, b, d} and {a, c, d} cannot be merged
- {a, c, d} and {a, c, e} can be merged. {a, c, d, e} is added to C(4)
- C(4) = {{a, b, c, d}, {a, c, d, e}}

How L(k) can be obtained from C(k)?

1. Scan the transaction database to determine the support for each candidate itemset in C_k
2. Save the frequent itemsets in L_k

Apriori algorithm

Pass 1

1. Generate the candidate itemsets in C_1
2. Save the frequent itemsets in L_1

Pass *k*

1. Generate the candidate itemsets in C_k from the frequent
2. itemsets in L_{k-1}
    1. Join L_{k-1} p with L_{k-1}q, as follows:
    2. **insert into** C_k
    3. **select** p.item$_1$, p.item$_2$, . . . , p.item$_{k-1}$, q.item$_{k-1}$
    4. **from** L_{k-1} p, L_{k-1}q
    5. **where** p.item$_1$ = q.item$_1$, . . . p.item$_{k-2}$ = q.item$_{k-2}$, p.item$_{k-1}$ < q.item$_{k-1}$
3. Scan the transaction database to determine the support for each candidate itemset in C_k
4. Save the frequent itemsets in L_k

T =

Transaction No.	Items
T1	1, 2, 3, 4, 5, 6
T2	7, 2, 3, 4, 5, 6
T3	1, 8, 4, 5
T4	1, 9, 0, 4, 6
T5	0, 9, 2, 4, 5

min_support_count = 3

Apriori algorithm

Pass 1

1. Generate the candidate itemsets in C_1
2. Save the frequent itemsets in L_1

Pass *k*

1. Generate the candidate itemsets in C_k from the frequent
2. itemsets in L_{k-1}
    1. Join L_{k-1} p with L_{k-1}q, as follows:
    2. **insert into** C_k
    3. **select** p.item$_1$, p.item$_2$, . . . , p.item$_{k-1}$, q.item$_{k-1}$
    4. **from** L_{k-1} p, L_{k-1}q
    5. **where** p.item$_1$ = q.item$_1$, . . . p.item$_{k-2}$ = q.item$_{k-2}$, p.item$_{k-1}$ < q.item$_{k-1}$
3. Scan the transaction database to determine the support for each candidate itemset in C_k
4. Save the frequent itemsets in L_k

Transaction No.	Items
T1	1, 2, 3, 4, 5, 6
T2	7, 2, 3, 4, 5, 6
T3	1, 8, 4, 5
T4	1, 9, 0, 4, 6
T5	0, 9, 2, 4, 5

C(1) =

Item	Occurrence / Frequency
1	3
2	3
3	2
4	5
5	4
6	3
7	1
8	1
9	2
0	2

Apriori algorithm

Pass 1

1. Generate the candidate itemsets in C_1
2. Save the frequent itemsets in L_1

Pass *k*

1. Generate the candidate itemsets in C_k from the frequent
2. itemsets in L_{k-1}
    1. Join L_{k-1} p with L_{k-1}q, as follows:
    2. **insert into** C_k
    3. **select** $p.\text{item}_1, p.\text{item}_2, \ldots, p.\text{item}_{k-1}, q.\text{item}_{k-1}$
    4. **from** L_{k-1} p, L_{k-1}q
    5. **where** $p.\text{item}_1 = q.\text{item}_1, \ldots p.\text{item}_{k-2} = q.\text{item}_{k-2}, p.\text{item}_{k-1} < q.\text{item}_{k-1}$
3. Scan the transaction database to determine the support for each candidate itemset in C_k
4. Save the frequent itemsets in L_k

C(1) =

Item	Occurrence / Frequency
1	3
2	3
3	2
4	5
5	4
6	3
7	1
8	1
9	2
0	2

L(1) =

Item	Occurrence / Frequency
1	3
2	3
4	5
5	4
6	3

Apriori algorithm

Pass 1

1. Generate the candidate itemsets in C_1
2. Save the frequent itemsets in L_1

Pass *k*

1. Generate the candidate itemsets in C_k from the frequent
2. itemsets in L_{k-1}
    1. Join L_{k-1} p with L_{k-1}q, as follows:
    2. **insert into** C_k
    3. **select** p.item$_1$, p.item$_2$, . . . , p.item$_{k-1}$, q.item$_{k-1}$
    4. **from** L_{k-1} p, L_{k-1}q
    5. **where** p.item$_1$ = q.item$_1$, . . . p.item$_{k-2}$ = q.item$_{k-2}$, p.item$_{k-1}$ < q.item$_{k-1}$
3. Scan the transaction database to determine the support for each candidate itemset in C_k
4. Save the frequent itemsets in L_k

L(1) =

Item	Occurrence / Frequency
1	3
2	3
4	5
5	4
6	3

C(2) =

ItemPairs	Occurrence / Frequency
12	1
14	3
15	2
16	2
24	3
25	3
26	2
45	4
46	3
56	2

Apriori algorithm

Pass 1

1. Generate the candidate itemsets in C_1
2. Save the frequent itemsets in L_1

Pass *k*

1. Generate the candidate itemsets in C_k from the frequent
2. itemsets in L_{k-1}
    1. Join $L_{k-1}\ p$ with $L_{k-1}q$, as follows:
    2. **insert into** C_k
    3. **select** $p.\text{item}_1$, $p.\text{item}_2$, . . . , $p.\text{item}_{k-1}$, $q.\text{item}_{k-1}$
    4. **from** $L_{k-1}\ p$, $L_{k-1}q$
    5. **where** $p.\text{item}_1 = q.\text{item}_1$, . . . $p.\text{item}_{k-2} = q.\text{item}_{k-2}$, $p.\text{item}_{k-1} < q.\text{item}_{k-1}$
3. Scan the transaction database to determine the support for each candidate itemset in C_k
4. Save the frequent itemsets in L_k

C(2) =

ItemPairs	Occurrence / Frequency
12	1
14	3
15	2
16	2
24	3
25	3
26	2
45	4
46	3
56	2

L(2) =

ItemPairs	Occurrence / Frequency
14	3
24	3
25	3
45	4
46	3

Apriori algorithm

Pass 1

1. Generate the candidate itemsets in C_1
2. Save the frequent itemsets in L_1

Pass *k*

1. Generate the candidate itemsets in C_k from the frequent
2. itemsets in L_{k-1}
    1. Join L_{k-1} p with L_{k-1}q, as follows:
    2. **insert into** C_k
    3. **select** p.item$_1$, p.item$_2$, . . . , p.item$_{k-1}$, q.item$_{k-1}$
    4. **from** L_{k-1} p, L_{k-1}q
    5. **where** p.item$_1$ = q.item$_1$, . . . p.item$_{k-2}$ = q.item$_{k-2}$, p.item$_{k-1}$ < q.item$_{k-1}$
3. Scan the transaction database to determine the support for each candidate itemset in C_k
4. Save the frequent itemsets in L_k

L(2) =

ItemPairs	Occurrence / Frequency
14	3
24	3
25	3
45	4
46	3

C(3) =

ItemTriples	Occurrence / Frequency
245	3
456	2

L(3) =

ItemTriples	Occurrence / Frequency
245	3

Apriori algorithm

Pass 1

1. Generate the candidate itemsets in C_1
2. Save the frequent itemsets in L_1

Pass *k*

1. Generate the candidate itemsets in C_k from the frequent
2. itemsets in L_{k-1}
    1. Join L_{k-1} p with L_{k-1}q, as follows:
    2. **insert into** C_k
    3. **select** p.item$_1$, p.item$_2$, . . . , p.item$_{k-1}$, q.item$_{k-1}$
    4. **from** L_{k-1} p, L_{k-1}q
    5. **where** p.item$_1$ = q.item$_1$, . . . p.item$_{k-2}$ = q.item$_{k-2}$, p.item$_{k-1}$ < q.item$_{k-1}$
3. Scan the transaction database to determine the support for each candidate itemset in C_k
4. Save the frequent itemsets in L_k

Thus, the frequent itemsets are

- L(1) = {{1}, {2}, {4}, {5}, {6}}
- L(2) = {{1, 4}, {2, 4}, {2, 5}, {4, 5}, {4, 6}}
- L(3) = {{2, 4, 5}}

All frequent itemsets = L(1) U L(2) U L(3)

Is Apriori fast enough? - Performance Bottlenecks

- The core of the Apriori algorithm
  - Use frequent (k – 1)-itemsets to generate candidate frequent k-itemsets

- The bottleneck of Apriori: **candidate generation**
  - 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., {a1 , a2 , …, a100}, one needs to generate 2^100 ≈ 10^30 candidates.
  - Multiple scans of database
    - Needs (n +1) scans, where n is the length of the longest pattern

FP-Growth Algorithm (No candidate itemset generation)

FP-Growth allows frequent itemset discovery without candidate itemset generation. Two step approach:

1. Build a compact data structure called the FP-tree
    a. Built using 2 passes over the database

2. Extracts frequent itemsets directly from the FP-tree

FP-Tree Construction

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*

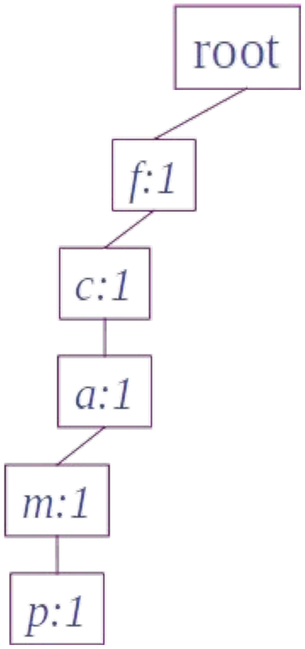
(ordered) frequent items
{f, c, a, m, p}
{f, c, a, b, m}
{f, b}
{c, b, p}
{f, c, a, m, p}

min_support_count = 3

FP-Tree Construction

(ordered) frequent items
{f, c, a, m, p}
{f, c, a, b, m}
{f, b}
{c, b, p}
{f, c, a, m, p}

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



FP-Tree Construction

(ordered) frequent items

{f, c, a, m, p}

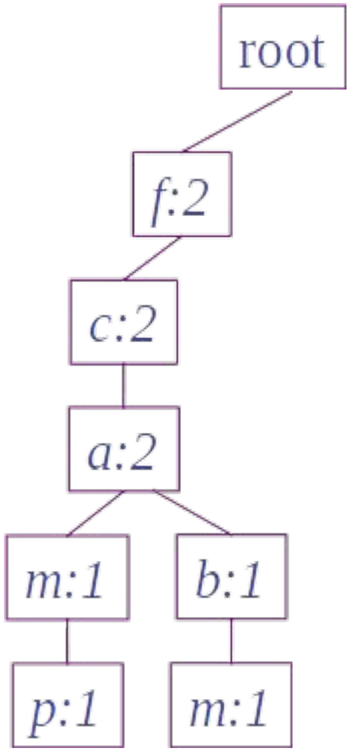
{f, c, a, b, m}

{f, b}

{c, b, p}

{f, c, a, m, p}

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



FP-Tree Construction

(ordered) frequent items

{f, c, a, m, p}

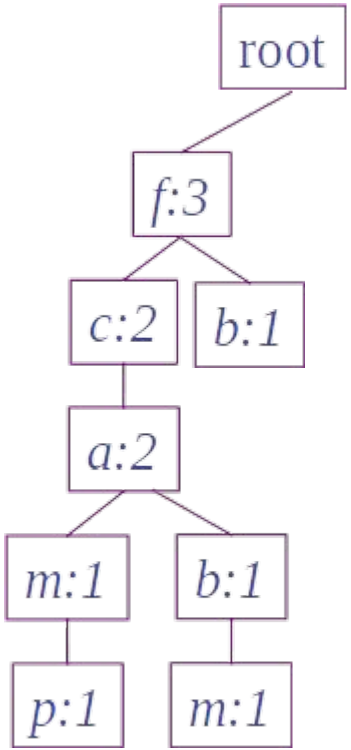
{f, c, a, b, m}

{f, b}

{c, b, p}

{f, c, a, m, p}

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

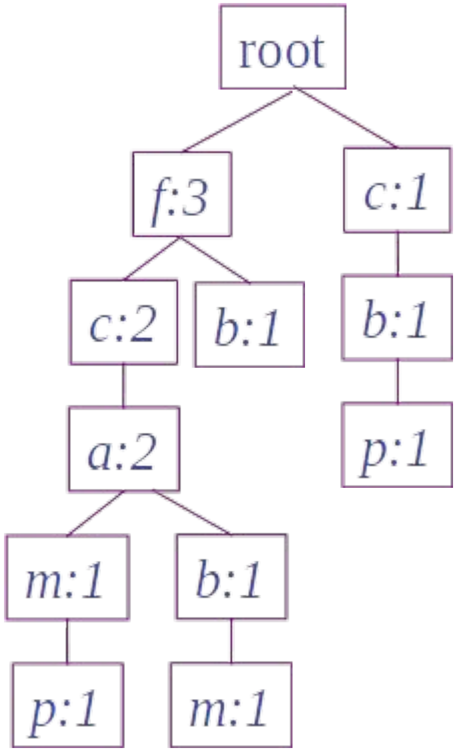


FP-Tree Construction

(ordered) frequent items

{f, c, a, m, p}
{f, c, a, b, m}
{f, b}
{c, b, p}
{f, c, a, m, p}

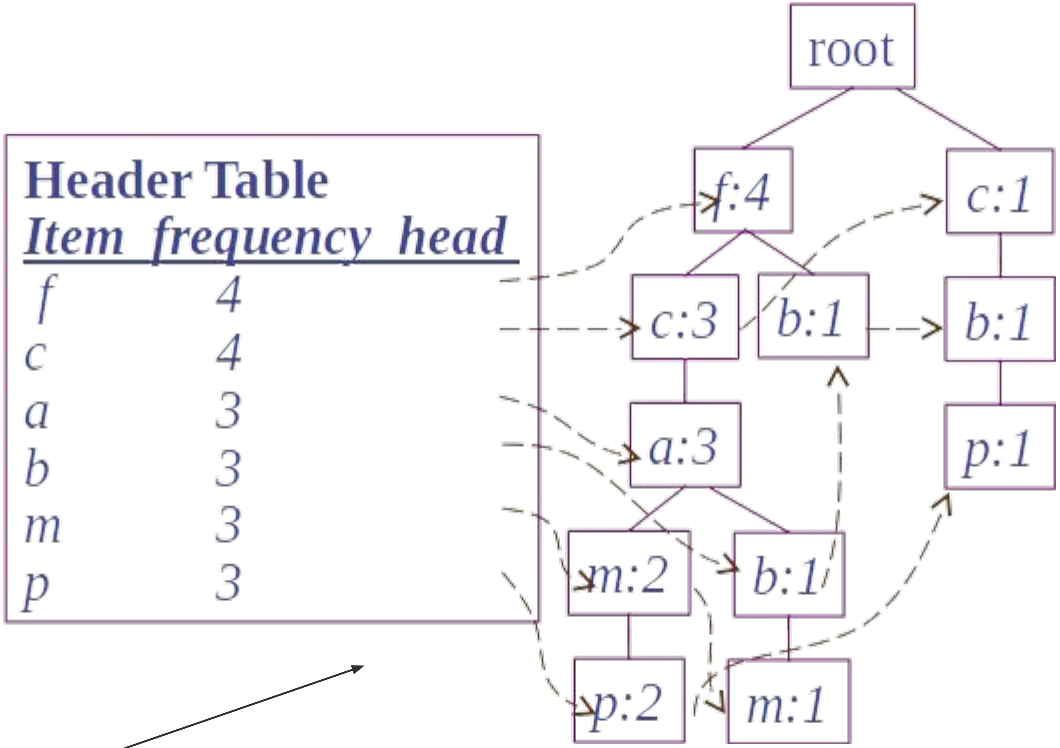
Item	frequency
f	4
c	4
a	3
b	3
m	3
p	3



FP-Tree Construction

<u>(ordered) frequent items</u>

{f, c, a, m, p}
{f, c, a, b, m}
{f, b}
{c, b, p}
{f, c, a, m, p}



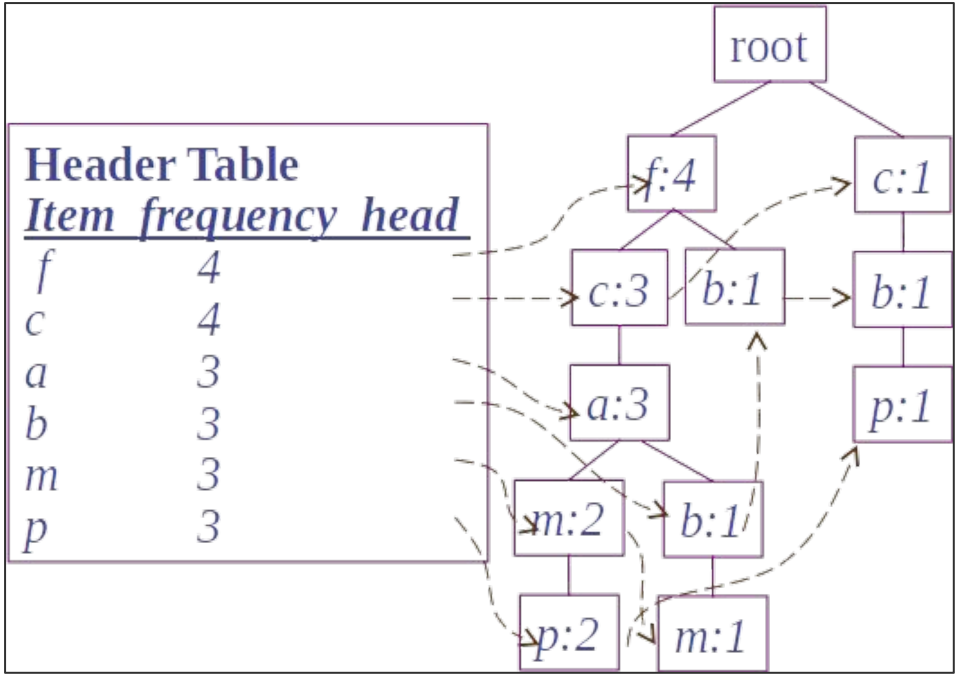
FP Tree

FP-Growth Algorithm (No candidate itemset generation)

FP-Growth allows frequent itemset discovery without candidate itemset generation. Two step approach:

1. Build a compact data structure called the FP-tree
   a. Built using 2 passes over the database
2. Extracts frequent itemsets directly from the FP-tree

Obtain frequent itemsets from FP-Tree



Goal: Find frequent itemsets whose last element is p

Conditional Pattern Base for p

- {f, c, a, m} : 2
- {c, b} : 1

Item	count
f	2
c	3
a	2
m	2
b	1

p-FP Tree

```
   root
   /
 c
```

Obtain frequent itemsets from FP-Tree

Frequent itemsets ending with p are :

Rules obtained from (p-FP Tree

```
   root
   /
  c
```

) + “p” = (null, c) + p = {p, cp}

Similarly frequent itemsets ending at item m, b, a, c and f can be obtained.

Why is Frequent Pattern Growth Fast?

- No candidate generation, no candidate test
- Use compact data structure
- Eliminate repeated database scan

References

- Agrawal, R., & Srikant, R. (1994). Fast algorithms for mining association rules. In Proceedings of 20th international conference on VLDB (pp. 487–499).

- Han, J., Pei, J., Yin, Y., & Mao, R. (2004). Mining frequent patterns without candidate generation: A frequent-pattern tree approach. Data Mining and Knowledge Discovery, 8(1), 53–87.

- http://www2.cs.uregina.ca/~dbd/cs831/notes/itemsets/itemset_apriori.html (pseudo code)

- https://chih-ling-hsu.github.io/2017/03/25/apriori (images)