Introduction to Algorithmic Data Analysis

Esther Galbrun

Autumn 2023

UNIVERSITY OF EASTERN FINLAND

Last updated on 2023-07-04 10:42

Part I

Frequent Itemset Mining

Problem

Frequent Itemset Mining

Discover items that often co-occur in a dataset

Classical setting: *Shopping basket data*

- Each product of the supermarket is an item
- Record customer transactions as sets of items
- Identify products that are often bought together
  Frequent itemset {butter, bread, ham, pickles}
- Extract rules that capture typical buying behaviour
  Association rules {bread, ham} $\Rightarrow$ {butter, pickles}
- Insights for marketing and shelf placement

Frequent Itemset Mining

Discover items that often co-occur in a dataset

Shopping basket data Customer transactions
Identify products often bought together

Text mining Bag of word model
Identify co-occurring terms and keywords

More complex data types (spatio-)temporal data, graph data

Other analysis tasks Building block for clustering, classification, outlier detection

Pizzeria example

A pizzeria offers to compose your pizza by freely choosing ingredients among *ham, jalapeno, mozzarella, olives and tuna*

To put together a menu, the pizzaiolo would like to know what are favorite combinations

Data

The database $\mathcal{T}$ is a collection of sets, called transactions, from a universe U of items

$$\mathcal{T} = \{T_1, T_2, \ldots, T_n\}, \text{ where } T_k \subseteq U, \forall k \in [1, n]$$

The total number of items is $m = |U|$

If we fix an order over U, each transaction can be represented as a binary vector of size m

Then, the database can be represented as a binary matrix with n rows and m columns

Each transaction has a unique identifier, its *tid*

Pizzeria example

The universe of items is the set of five ingredients

$$\{\text{ham}, \text{jalapeno}, \text{mozzarella}, \text{olives}, \text{tuna}\}$$

For short, $U = \{\mathtt{h}, \mathtt{j}, \mathtt{m}, \mathtt{o}, \mathtt{t}\}$

Each pizza constitutes a transaction, represented by the corresponding set of ingredients

For instance, a ham and mozzarella pizza is represented as $T = \{\mathtt{h}, \mathtt{m}\}$, also simply denoted $\mathtt{hm}$

Ordering the items alphabetically according to corresponding ingredient names, this pizza is represented by the binary vector $\langle 1, 0, 1, 0, 0 \rangle$, also simply written 10100

Pizzeria example

The database then records all pizzas sold

tids	pizzas	sets	matrix
1)	ham mozzarella olives	$\{h, m, o\}$	■□■■□
2)	mozzarella	$\{m\}$	□□■□□
3)	jalapeno mozzarella	$\{j, m\}$	□■■□□
4)	ham jalapeno mozzarella olives	$\{h, j, m, o\}$	■■■■□
5)	ham jalapeno mozzarella olives	$\{h, j, m, o\}$	■■■■□
6)	ham	$\{h\}$	■□□□□
7)	ham jalapeno mozzarella tuna	$\{h, j, m, t\}$	■■■□■
8)	mozzarella	$\{m\}$	□□■□□
9)	olives	$\{o\}$	□□□■□
10)	ham jalapeno mozzarella olives tuna	$\{h, j, m, o, t\}$	■■■■■
11)	ham mozzarella tuna	$\{h, m, t\}$	■□■□■
12)	ham mozzarella	$\{h, m\}$	■□■□□
⋮	⋮	⋮	⋮

Itemset and support

An itemset I is a set of items, i.e. $I \subseteq U$
A *k-itemset* is an itemset that contains exactly k items,
i.e. such that $|I| = k$

The support set of an itemset I in $\mathcal{T}$ is the set of transactions from $\mathcal{T}$ that contain I

$$\text{supp}_{\mathcal{T}}(I) = \{T \in \mathcal{T}, I \subseteq T\}$$

We call $|\text{supp}_{\mathcal{T}}(I)|$ the *absolute support* of I in $\mathcal{T}$
and $|\text{supp}_{\mathcal{T}}(I)| \,/\, |\mathcal{T}|$ its *fractional support*

We denote $\text{supp}\,\%_{\mathcal{T}}(I)$ the fractional support given as a percentage, i.e.

$$\text{supp}\,\%_{\mathcal{T}}(I) = 100 \cdot \frac{|\text{supp}_{\mathcal{T}}(I)|}{|\mathcal{T}|}$$

Itemset and support

An itemset I is a set of items, i.e. $I \subseteq U$

A *k-itemset* is an itemset that contains exactly k items,

i.e. such that $|I| = k$

The support set of an itemset I in $\mathcal{T}$ is the set of transactions from $\mathcal{T}$ that contain I

$$\text{supp}_{\mathcal{T}}(I) = \{T \in \mathcal{T}, I \subseteq T\}$$

We call $|\text{supp}_{\mathcal{T}}(I)|$ the *absolute support* of I in $\mathcal{T}$

and $|\text{supp}_{\mathcal{T}}(I)| \,/\, |\mathcal{T}|$ its *fractional support*

! There are variations in the use of support terminology

! The database is often left out from the notation,

as it is clear from the context

Pizzeria example

tid	set
1)	{h, m, o}
2)	{m}
3)	{j, m}
4)	{h, j, m, o}
5)	{h, j, m, o}
6)	{h}
7)	{h, j, m, t}
8)	{m}
9)	{o}
10)	{h, j, m, o, t}
11)	{h, m, t}
12)	{h, m}

In the database consisting only of transactions 1 to 12,
for itemset $I = \{\texttt{t}\}$

$$\begin{aligned} \text{supp}(I) &= \{7, 10, 11\} \\ |\text{supp}(I)| &= 3 \\ \text{supp}\,\%(I) &= 25 \end{aligned}$$

Pizzeria example

tid	set
1)	{h, m, o}
2)	{m}
3)	{j, m}
4)	{h, j, m, o}
5)	{h, j, m, o}
6)	{h}
7)	{h, j, m, t}
8)	{m}
9)	{o}
10)	{h, j, m, o, t}
11)	{h, m, t}
12)	{h, m}

In the database consisting only of transactions 1 to 12,
for itemset $I = \{\mathtt{h}, \mathtt{m}\}$

$$\mathrm{supp}(I) = \{1, 4, 5, 7, 10, 11, 12\}$$

$$|\mathrm{supp}(I)| = 7$$

$$\mathrm{supp}\,\%(I) = 58.33$$

Problem definition

Frequent Itemset Mining

Given a set of transactions $\mathcal{T} = \{T_1, T_2, \ldots, T_n\}$, where each transaction T_i is a subset of items from U, and a minimum support threshold σ, determine all itemsets I that occur as a subset of at least σ transactions in $\mathcal{T}$.

Pizzeria example

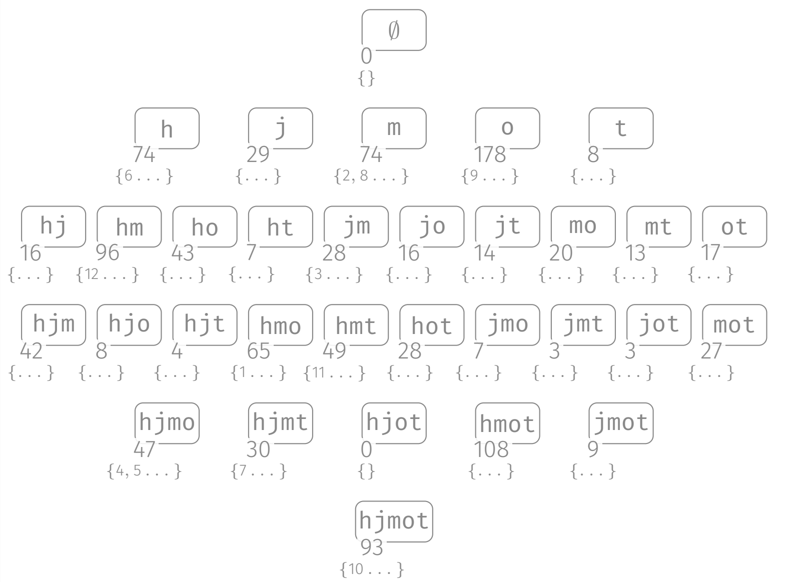
Enumerate all distinct pizzas

	count	tids
hmo	65	{1 . . . }
m	74	{2, 8 . . . }
jm	28	{3 . . . }
hjmo	47	{4, 5 . . . }
h	74	{6 . . . }
hjmt	30	{7 . . . }
o	178	{9 . . . }
hjmot	93	{10 . . . }
hmt	49	{11 . . . }
hm	96	{12 . . . }

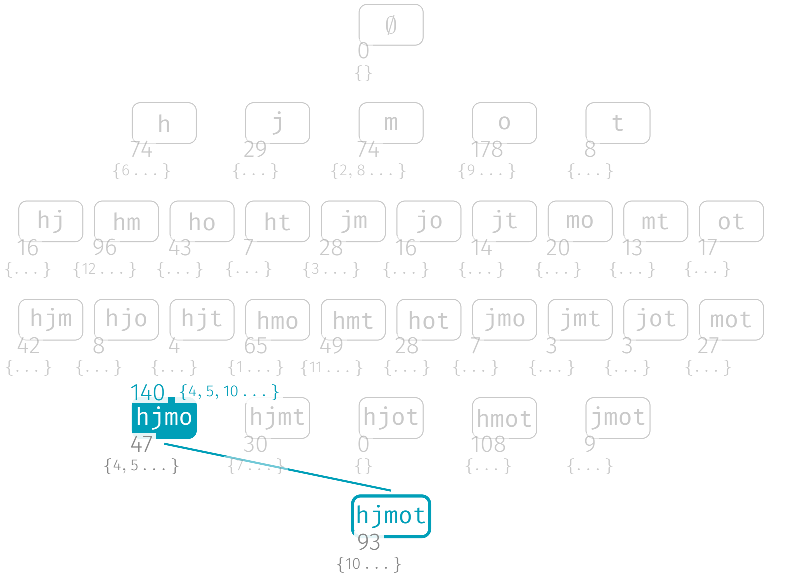
	count	tids
hj	16	{. . . }
hjm	42	{. . . }
hjo	8	{. . . }
hjot	0	{}
hjt	4	{. . . }
hmot	108	{. . . }
ho	43	{. . . }
hot	28	{. . . }
ht	7	{. . . }
j	29	{. . . }

	count	tids
jmo	7	{. . . }
jmot	9	{. . . }
jmt	3	{. . . }
jo	16	{. . . }
jot	3	{. . . }
jt	14	{. . . }
mo	20	{. . . }
mot	27	{. . . }
mt	13	{. . . }
ot	17	{. . . }
t	8	{. . . }

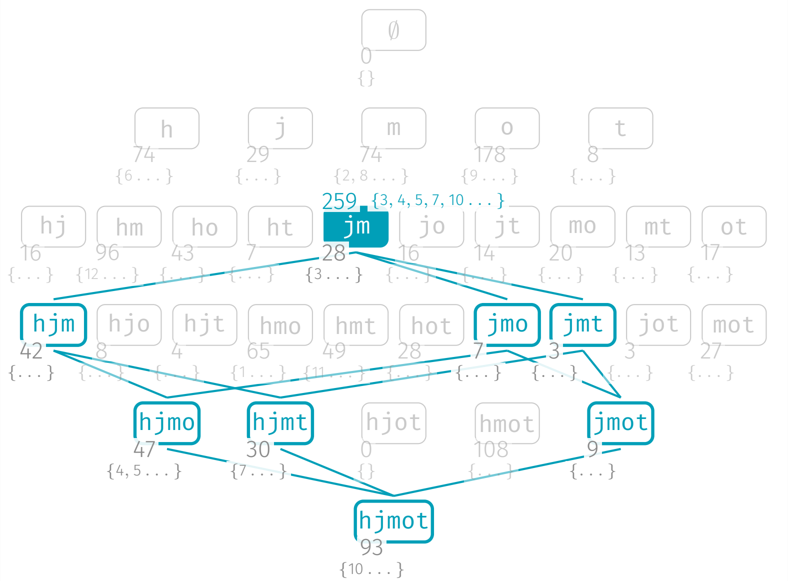
Pizzeria example: Enumerating all distinct pizzas



Pizzeria example: Aggregating supports



Pizzeria example: Aggregating supports



Support properties

Monotonicity of support

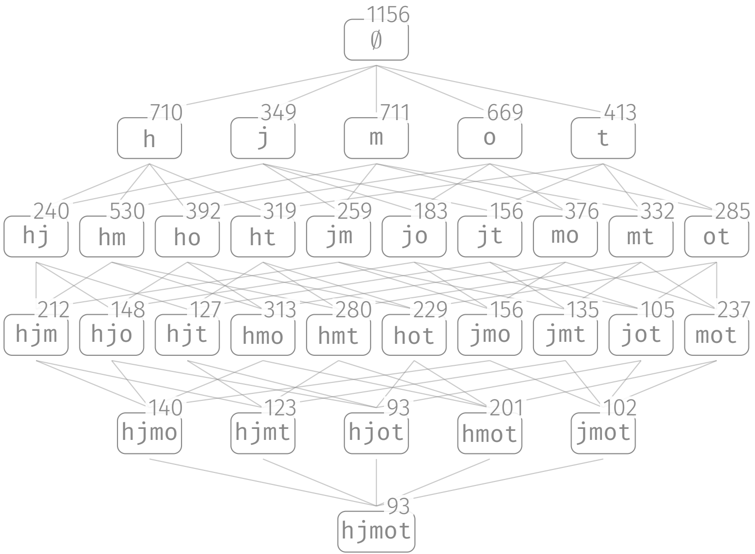
The support of every subset J of I is at least equal to that of the support of itemset I

$$\forall J \subseteq I, \quad \text{supp}(I) \subseteq \text{supp}(J)$$
$$\text{and hence} \quad |\text{supp}(I)| \leq |\text{supp}(J)|$$

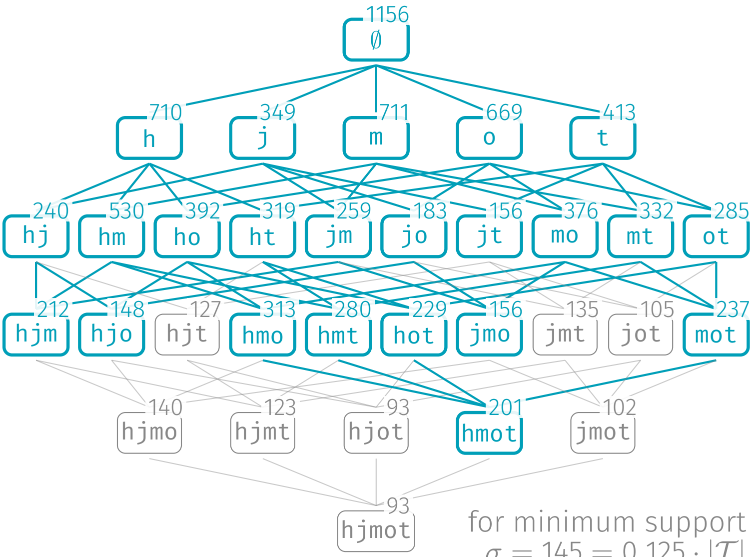
Downward closure property

Every subset of a frequent itemset is also frequent

Pizzeria example: Lattice of ingredient combinations

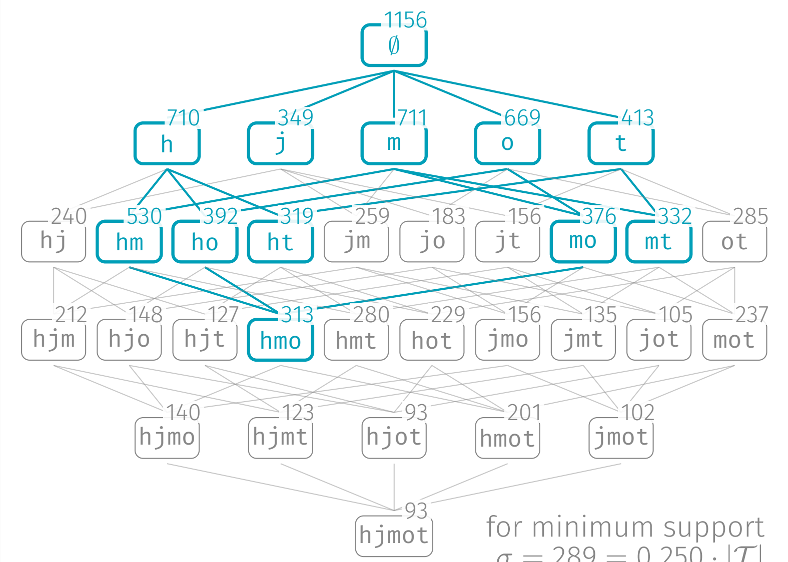


Pizzeria example: Frequent ingredient combinations



for minimum support
$\sigma = 145 = 0.125 \cdot |\mathcal{T}|$

Pizzeria example: Frequent ingredient combinations



for minimum support
$\sigma = 289 = 0.250 \cdot |\mathcal{T}|$

The empty set

Considered as an itemset, the empty set has a fractional support equal to one, since it is a subset of every transaction in the database

However, the empty set is generally not listed among frequent itemsets, because it does not provide any interesting information

Closed and maximal frequent itemsets

An itemset I is closed if none of its supersets have exactly the same support count

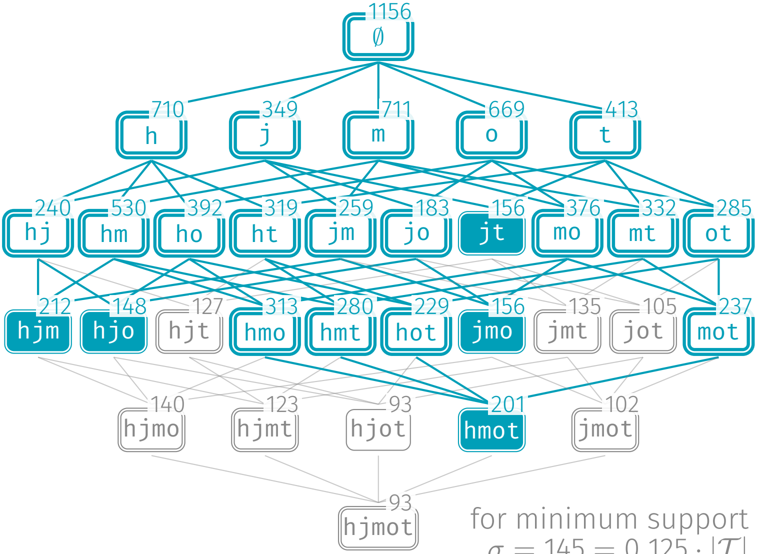
A frequent itemset I is maximal at a given minimum support level σ, if it is frequent and none of its superset is frequent

Condensed representations

Knowledge of the maximal frequent itemsets allows to reconstruct the set of frequent itemsets, but not their supports

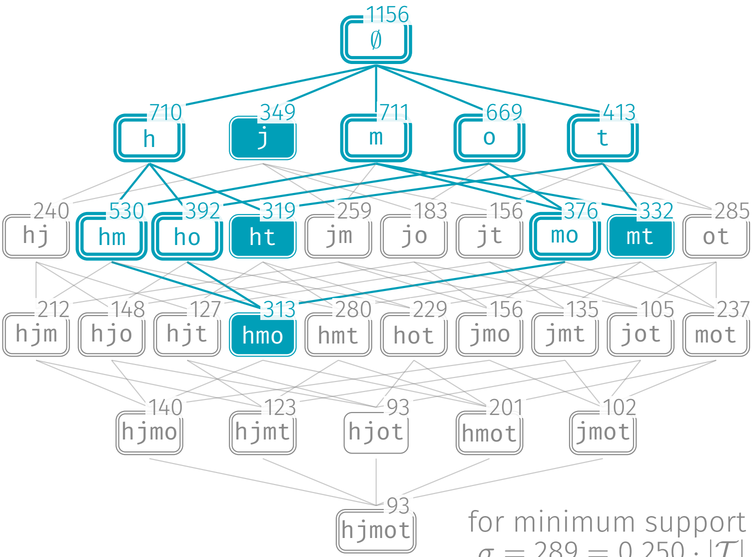
Knowledge of the closed frequent itemsets allows to also recompute the supports

Pizzeria example: Closed and maximal frequent itemsets



for minimum support $\sigma = 145 = 0.125 \cdot |\mathcal{T}|$

Pizzeria example: Closed and maximal frequent itemsets



for minimum support
$\sigma = 289 = 0.250 \cdot |\mathcal{T}|$

Algorithms

Algorithms for mining frequent itemsets

Support counting is expensive

Explore the space of itemsets by increasing lengths,
i.e. *level-wise enumeration*

Avoid generating itemsets twice by using a canonical order

Exploit the *downward closure property* to prune itemsets

Level-wise enumeration: *Apriori* algorithm

$k \leftarrow 1$
$\mathcal{F}_k \leftarrow$ {all frequent singleton itemsets}
while $\mathcal{F}_k \neq \emptyset$ **do**
    Generate $\mathcal{C}_{k+1}$ by extending itemsets from $\mathcal{F}_k$
    Prune itemsets that violate downward closure
    $\mathcal{F}_{k+1} \leftarrow \{\mathcal{S} \in \mathcal{C}_{k+1}, \text{supp}_{\mathcal{D}}(\mathcal{S}) \geq \theta\}$
    $k \leftarrow k + 1$
return $\bigcup_i \mathcal{F}_i$

Candidate generation

$k \leftarrow 1$
$\mathcal{F}_k \leftarrow$ {all frequent singleton itemsets}
while $\mathcal{F}_k \neq \emptyset$ **do**
  Generate $\mathcal{C}_{k+1}$ by extending itemsets from $\mathcal{F}_k$
  Prune itemsets that violate downward closure
  $\mathcal{F}_{k+1} \leftarrow \{\mathcal{S} \in \mathcal{C}_{k+1}, \text{supp}_{\mathcal{D}}(\mathcal{S}) \geq \theta\}$
  $k \leftarrow k + 1$
return $\bigcup_i \mathcal{F}_i$

Candidate pruning

$k \leftarrow 1$

$\mathcal{F}_k \leftarrow$ {all frequent singleton itemsets}

while $\mathcal{F}_k \neq \emptyset$ **do**

Generate $\mathcal{C}_{k+1}$ by extending itemsets from $\mathcal{F}_k$

Prune itemsets that violate downward closure

$\mathcal{F}_{k+1} \leftarrow \{\mathcal{S} \in \mathcal{C}_{k+1}, \text{supp}_{\mathcal{D}}(\mathcal{S}) \geq \theta\}$

$k \leftarrow k + 1$

return $\bigcup_i \mathcal{F}_i$

Support counting

$k \leftarrow 1$

$\mathcal{F}_k \leftarrow$ {all frequent singleton itemsets}

while $\mathcal{F}_k \neq \emptyset$ **do**

  Generate $\mathcal{C}_{k+1}$ by extending itemsets from $\mathcal{F}_k$

  Prune itemsets that violate downward closure

  $\mathcal{F}_{k+1} \leftarrow \{\mathcal{S} \in \mathcal{C}_{k+1}, \mathrm{supp}_{\mathcal{D}}(\mathcal{S}) \geq \theta\}$

  $k \leftarrow k + 1$

return $\bigcup_i \mathcal{F}_i$

Pizzeria example: *Apriori* algorithm

We return to our pizzeria example, where each of the 1156 transactions in the database represents a pizza sold

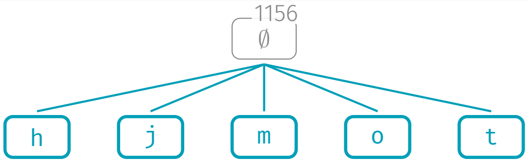
We look how the *Apriori* algorithm can be applied to mine frequent itemsets from this database

In this example, we set the minimum support threshold to $\sigma = 289$ (i.e. 25%)

tid	set
1)	{h, m, o}
2)	{m}
3)	{j, m}
4)	{h, j, m, o}
5)	{h, j, m, o}
6)	{h}
7)	{h, j, m, t}
8)	{m}
9)	{o}
10)	{h, j, m, o, t}
11)	{h, m, t}
12)	{h, m}
⋮	⋮

Pizzeria example: *Apriori* algorithm

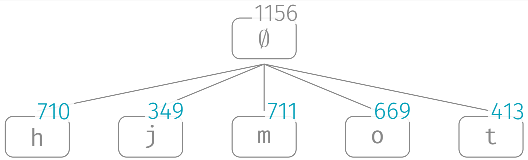
Enumerate singleton itemsets



for $\sigma = 289$

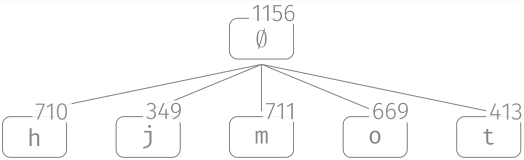
Pizzeria example: *Apriori* algorithm

Count supports



for $\sigma = 289$

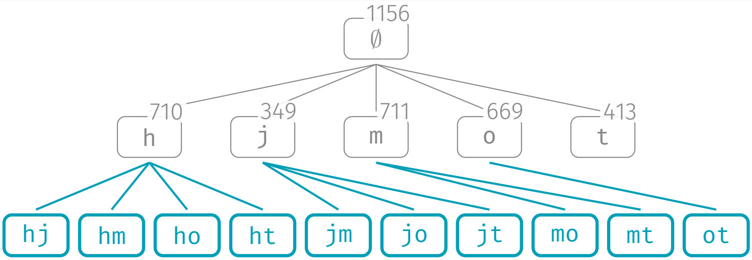
Frequent singleton itemsets



for $\sigma = 289$

Pizzeria example: *Apriori* algorithm

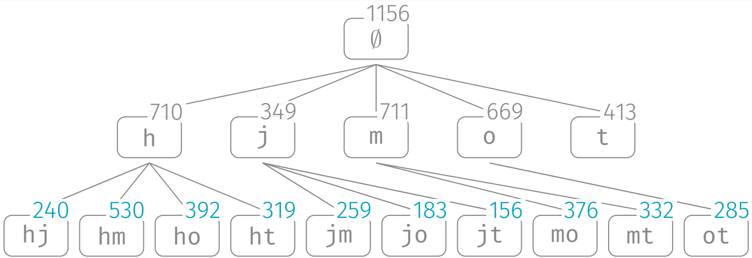
Generate candidates itemsets of length 2



for $\sigma = 289$

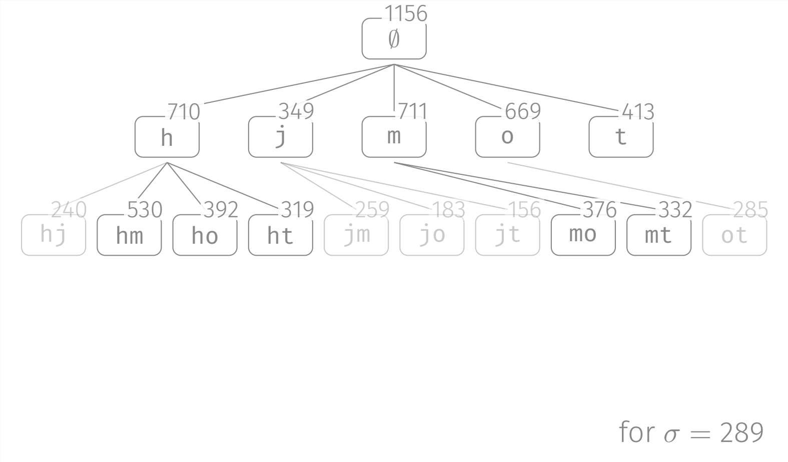
Pizzeria example: *Apriori* algorithm

Count supports



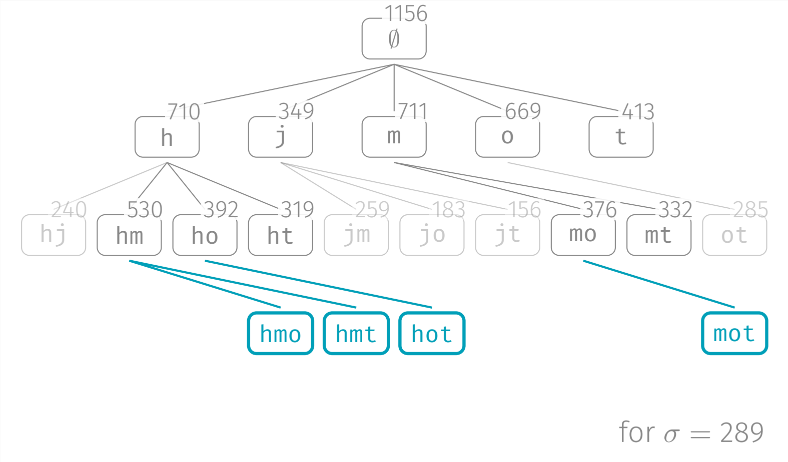
for $\sigma = 289$

Frequent itemsets of length up to 2



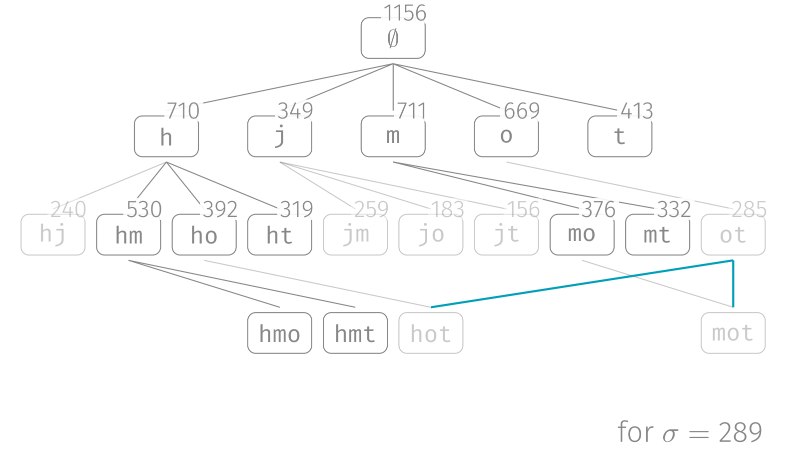
Pizzeria example: *Apriori* algorithm

Generate candidates itemsets of length 3

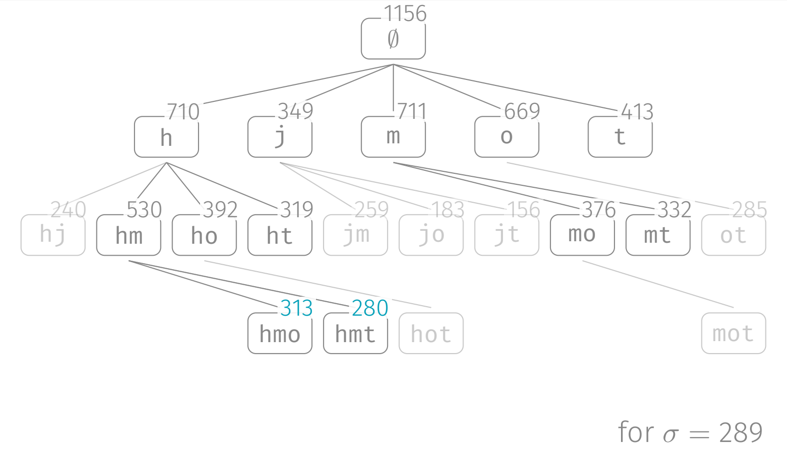


Pizzeria example: *Apriori* algorithm

Prune candidates

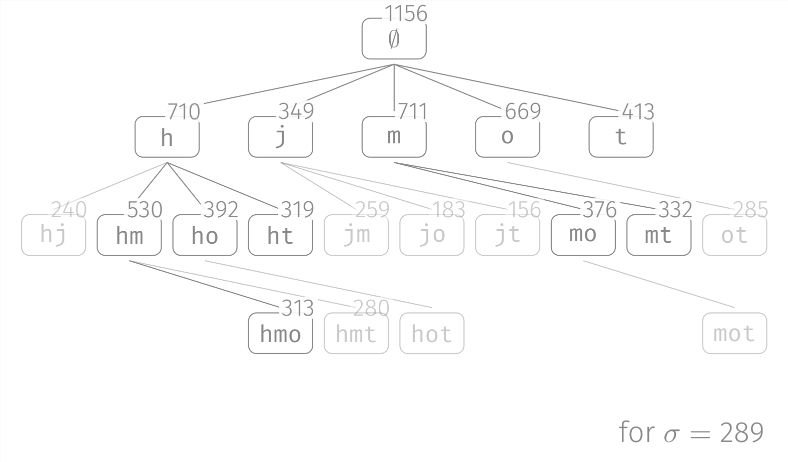


Count supports



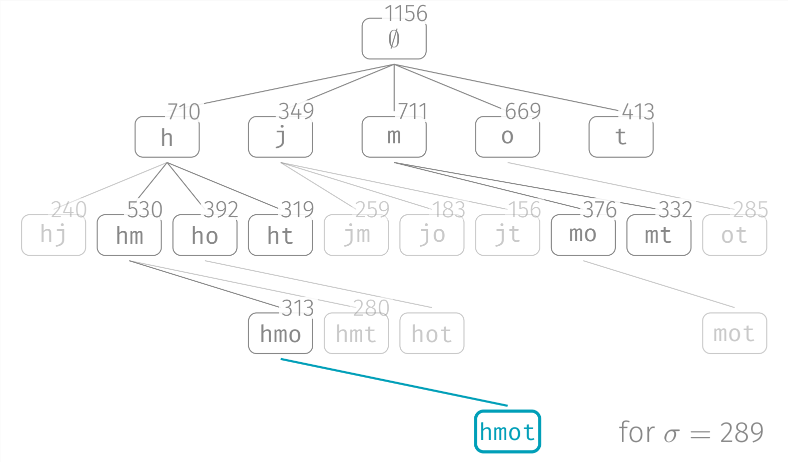
Pizzeria example: *Apriori* algorithm

Frequent itemsets of length up to 3

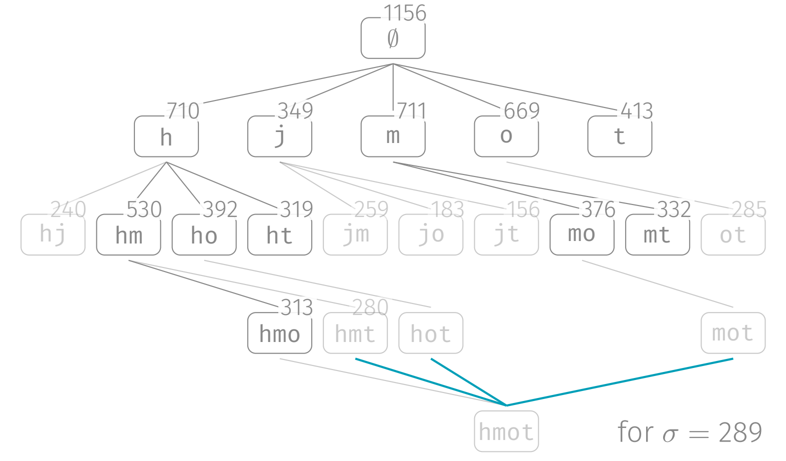


Pizzeria example: *Apriori* algorithm

Generate candidates itemsets of length 4

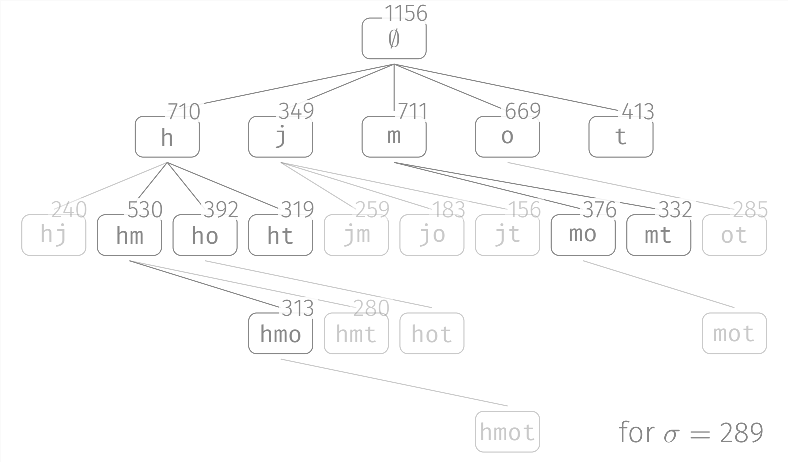


Prune candidates



Pizzeria example: *Apriori* algorithm

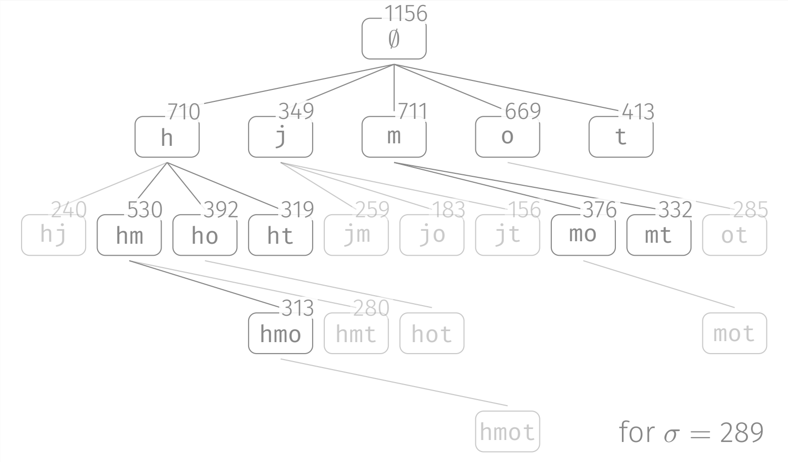
Frequent itemsets of length up to 4



for $\sigma = 289$

Pizzeria example: Enumeration tree

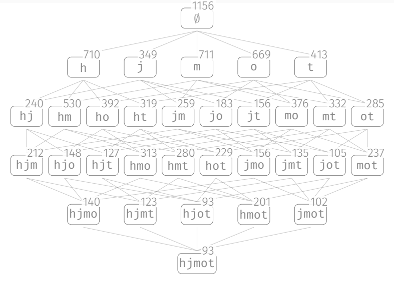
Items ordered alphabetically, prefix growth

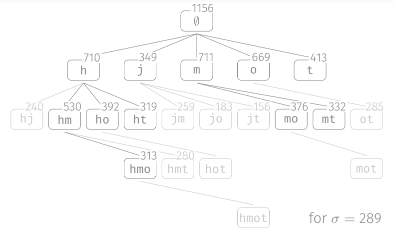


Pizzeria example: Enumeration tree

Note the difference

itemset lattice vs. enumeration tree

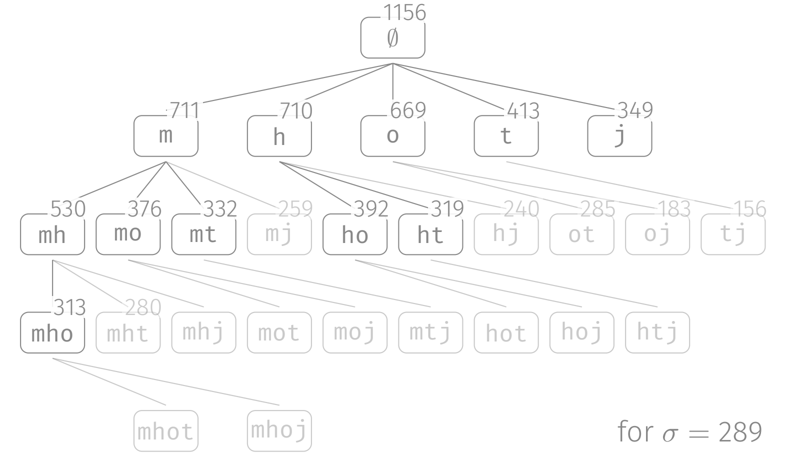




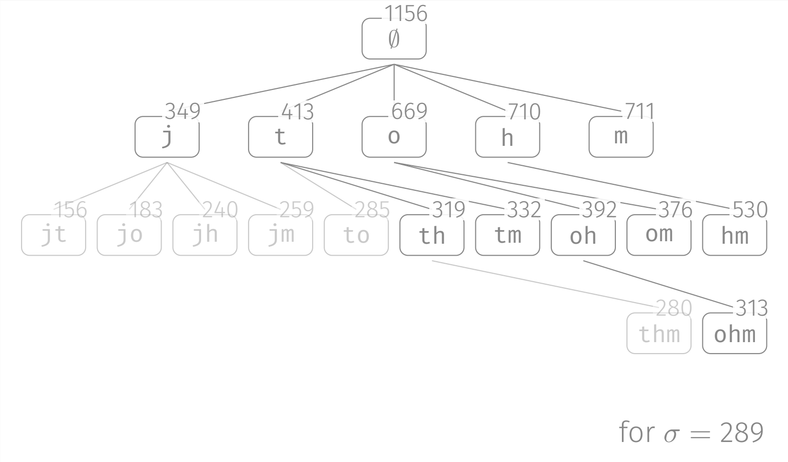
Shows the space of itemsets
Edges represent subset relationships

Shows the enumeration structure
Edges indicate from which subset each itemset was generated

Items ordered by decreasing frequency, prefix growth



Items ordered by increasing frequency, prefix growth



Support counting is expensive

According to the *monotonicity of support*

$$\forall J \subseteq I, \quad \mathrm{supp}(I) \subseteq \mathrm{supp}(J)$$

Make support counting more efficient

- Prune irrelevant transactions
- Reuse support counting from previous steps

Recursively project the database down the enumeration tree

Vertical apriori algorithm

$k \leftarrow 1$

$\mathcal{F}_k \leftarrow$ {all frequent singleton itemsets}

Generate *tid* list for each frequent singleton itemsets

while $\mathcal{F}_k \neq \emptyset$ **do**

Generate $\mathcal{C}_{k+1}$ by joining pairs of itemsets from $\mathcal{F}_k$

Prune itemsets that violate downward closure

Generate *tid* list for each candidate by intersecting

tid lists of associated pair of k-itemsets

$\mathcal{F}_{k+1} \leftarrow \{\mathcal{S} \in \mathcal{C}_{k+1}, \text{supp}_{\mathcal{D}}(\mathcal{S}) \geq \theta\}$

$k \leftarrow k + 1$

return $\bigcup_i \mathcal{F}_i$

Vertical apriori algorithm

Vertical database representation

$k \leftarrow 1$

$\mathcal{F}_k \leftarrow$ {all frequent singleton itemsets}

Generate *tid* list for each frequent singleton itemsets

while $\mathcal{F}_k \neq \emptyset$ **do**

  Generate $\mathcal{C}_{k+1}$ by joining pairs of itemsets from $\mathcal{F}_k$

  Prune itemsets that violate downward closure

  Generate *tid* list for each candidate by intersecting tid lists of associated pair of k-itemsets

  $\mathcal{F}_{k+1} \leftarrow \{\mathcal{S} \in \mathcal{C}_{k+1}, \text{supp}_{\mathcal{D}}(\mathcal{S}) \geq \theta\}$

  $k \leftarrow k + 1$

return $\bigcup_i \mathcal{F}_i$

Space–time trade-off

Tid lists

- Allow to compute supports faster
- Require memory space for storage

Use dedicated data structures that support efficient counting

FP-growth algorithm

The FP-tree is a compact representation of the database

- Extract conditional projected database for a given suffix
- Update counts efficiently

FP-growth is a recursive suffix-based pattern growth algorithm

Pizzeria example: Construction of the FP-tree

We return to our pizzeria example, where each of the 1156 transactions in the database represents a pizza sold

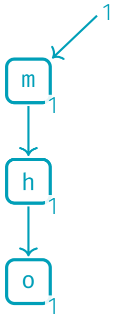
We look how the *FP-growth* algorithm can be applied to mine frequent itemsets from this database

In this example, we set the minimum support threshold to $\sigma = 289$ (i.e. 25%)

The first step is to construct the *FP-tree* representing the database

tid	set
1)	$\{h, m, o\}$
2)	$\{m\}$
3)	$\{j, m\}$
4)	$\{h, j, m, o\}$
5)	$\{h, j, m, o\}$
6)	$\{h\}$
7)	$\{h, j, m, t\}$
8)	$\{m\}$
9)	$\{o\}$
10)	$\{h, j, m, o, t\}$
11)	$\{h, m, t\}$
12)	$\{h, m\}$
⋮	⋮

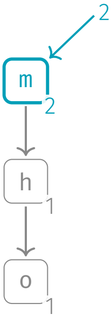
Inserting transactions (items sorted by decreasing frequency)



Step# 1
Transaction mho

Pizzeria example: Construction of the FP-tree

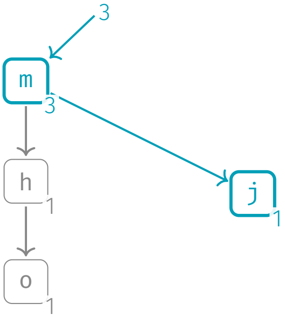
Inserting transactions (items sorted by decreasing frequency)



Step# 2
Transaction m

Pizzeria example: Construction of the FP-tree

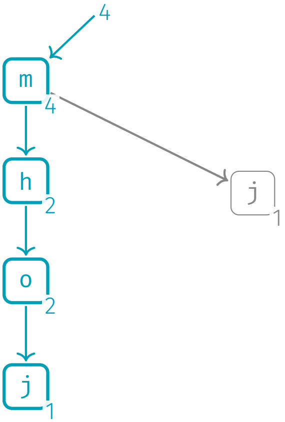
Inserting transactions (items sorted by decreasing frequency)



Step# 3
Transaction mj

Pizzeria example: Construction of the FP-tree

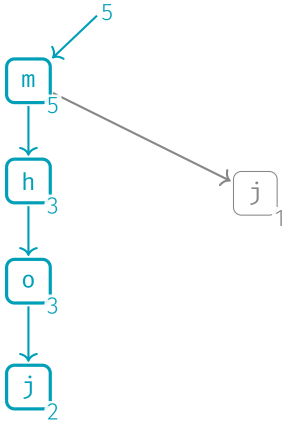
Inserting transactions (items sorted by decreasing frequency)



Step# 4
Transaction `mhoj`

Pizzeria example: Construction of the FP-tree

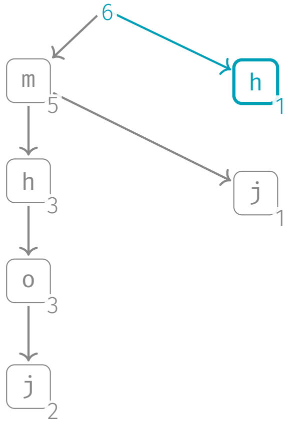
Inserting transactions (items sorted by decreasing frequency)



Step# 5
Transaction mhoj

Pizzeria example: Construction of the FP-tree

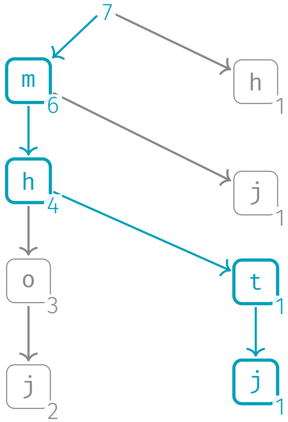
Inserting transactions (items sorted by decreasing frequency)



Step# 6
Transaction h

Pizzeria example: Construction of the FP-tree

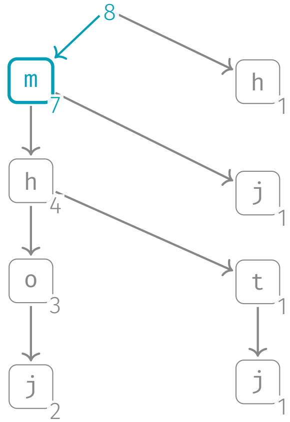
Inserting transactions (items sorted by decreasing frequency)



Step# 7
Transaction `mhtj`

Pizzeria example: Construction of the FP-tree

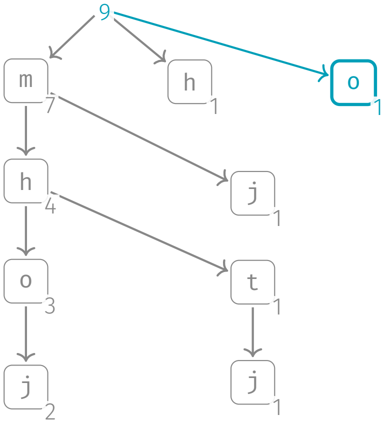
Inserting transactions (items sorted by decreasing frequency)



Step# 8
Transaction m

Pizzeria example: Construction of the FP-tree

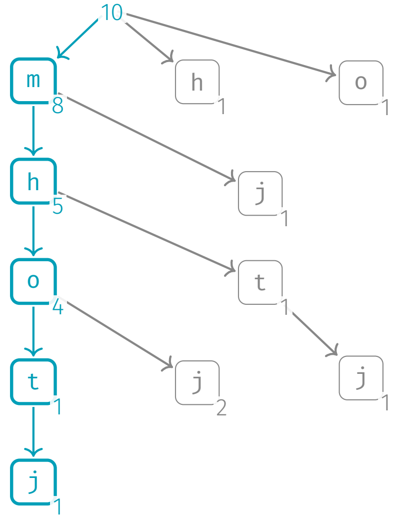
Inserting transactions (items sorted by decreasing frequency)



Step# 9
Transaction o

Pizzeria example: Construction of the FP-tree

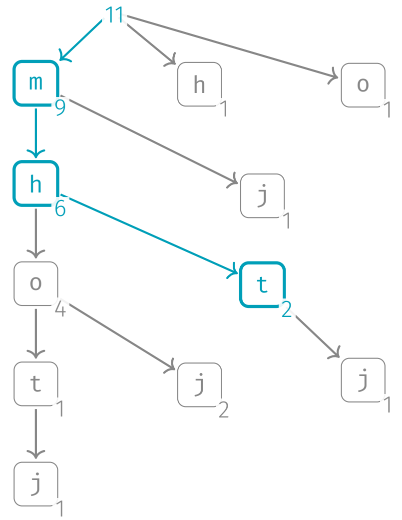
Inserting transactions (items sorted by decreasing frequency)



Step# 10
Transaction `mhotj`

Pizzeria example: Construction of the FP-tree

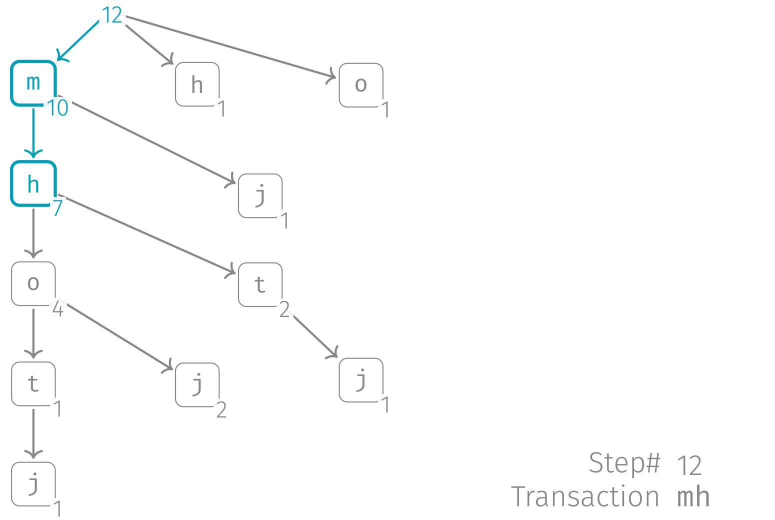
Inserting transactions (items sorted by decreasing frequency)



Step# 11
Transaction `mht`

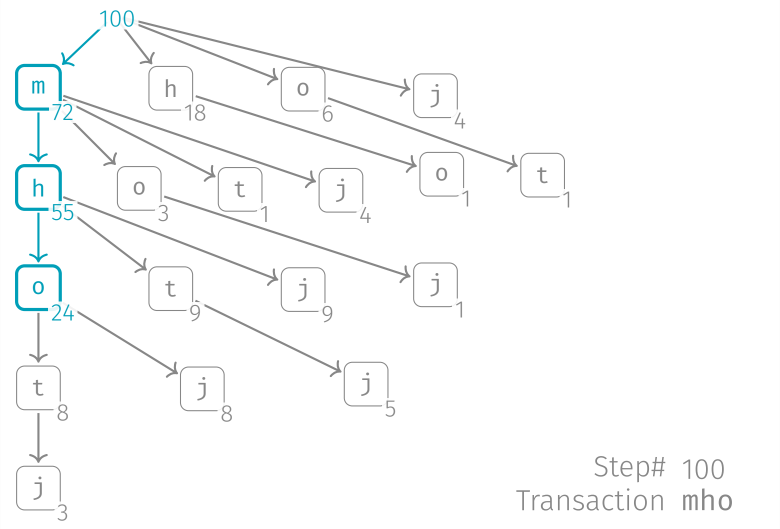
Pizzeria example: Construction of the FP-tree

Inserting transactions (items sorted by decreasing frequency)



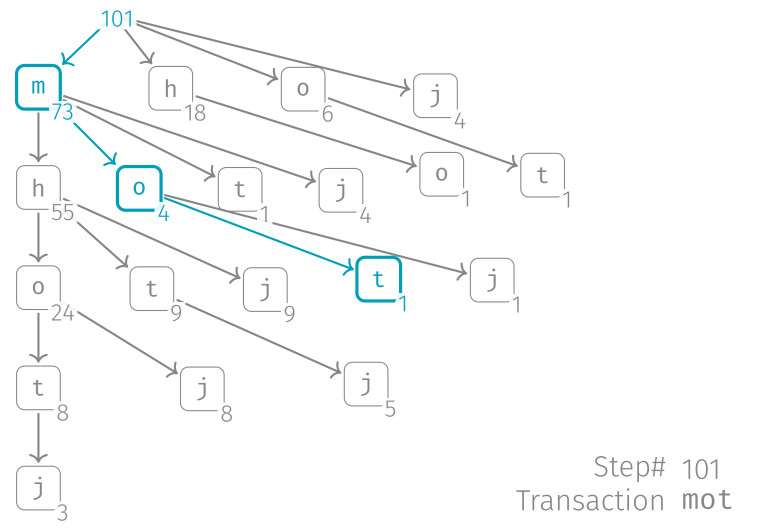
Pizzeria example: Construction of the FP-tree

Inserting transactions (items sorted by decreasing frequency)



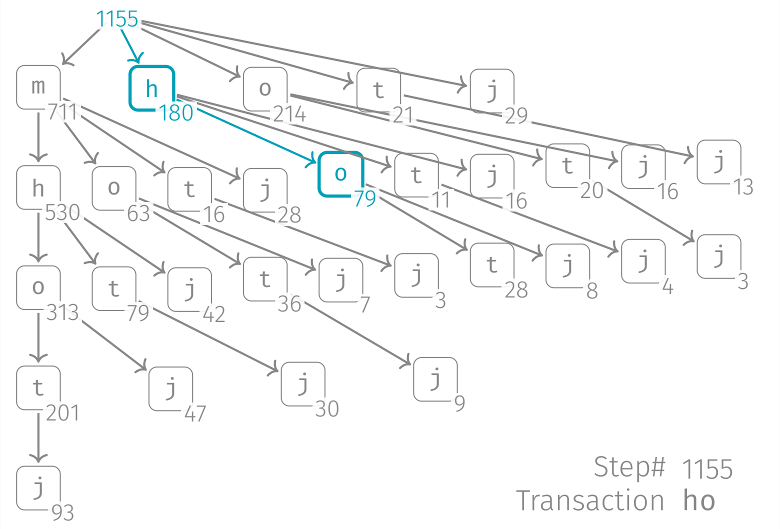
Pizzeria example: Construction of the FP-tree

Inserting transactions (items sorted by decreasing frequency)

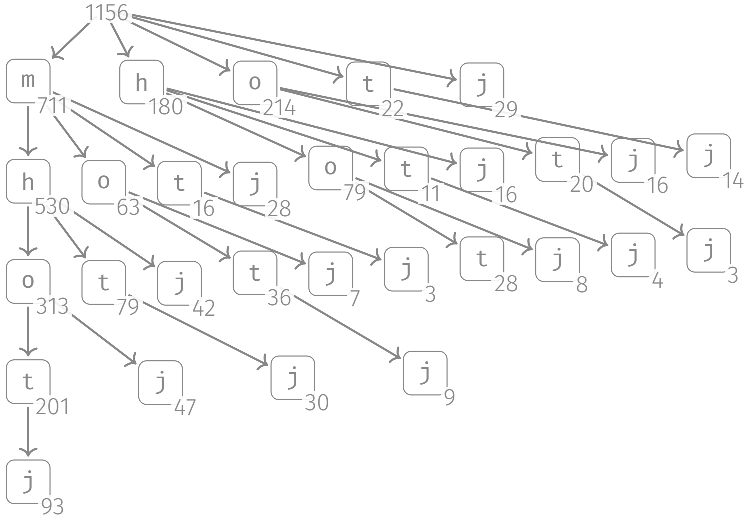


Pizzeria example: Construction of the FP-tree

Inserting transactions (items sorted by decreasing frequency)

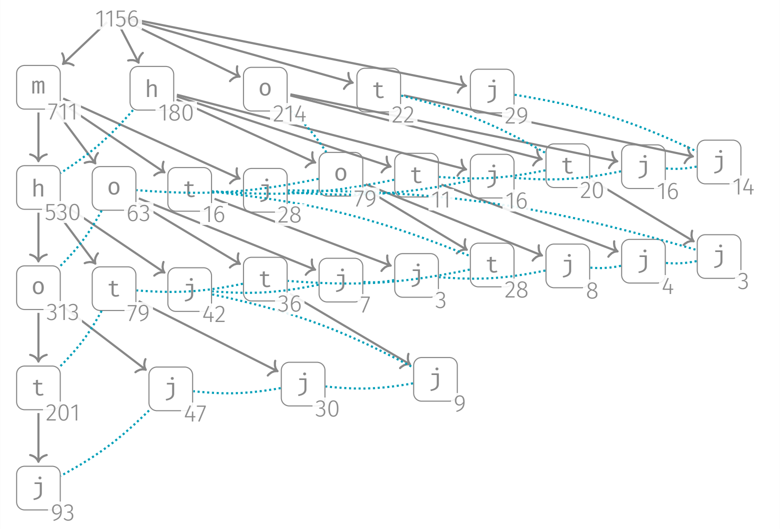


Having inserted all transactions, we obtain the full *FP-tree*



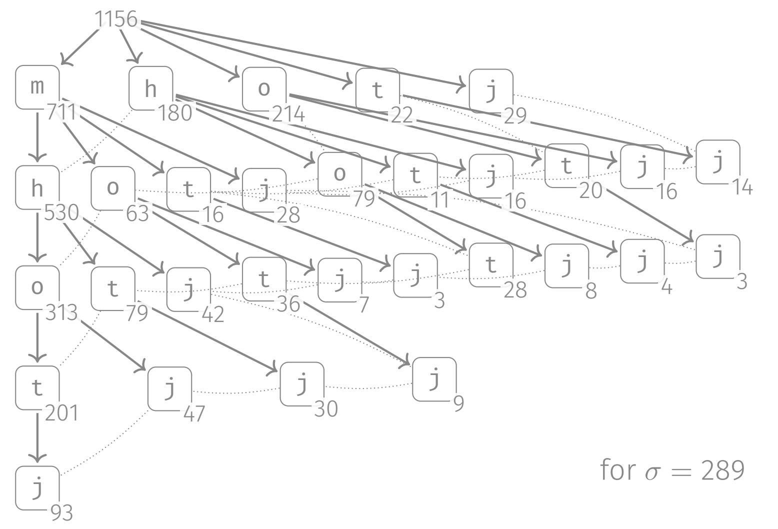
Pizzeria example: Construction of the FP-tree

We add pointers linking nodes representing the same item



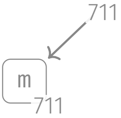
Pizzeria example: Suffix-based pattern growth with the FP-tree

Recursive pattern growth



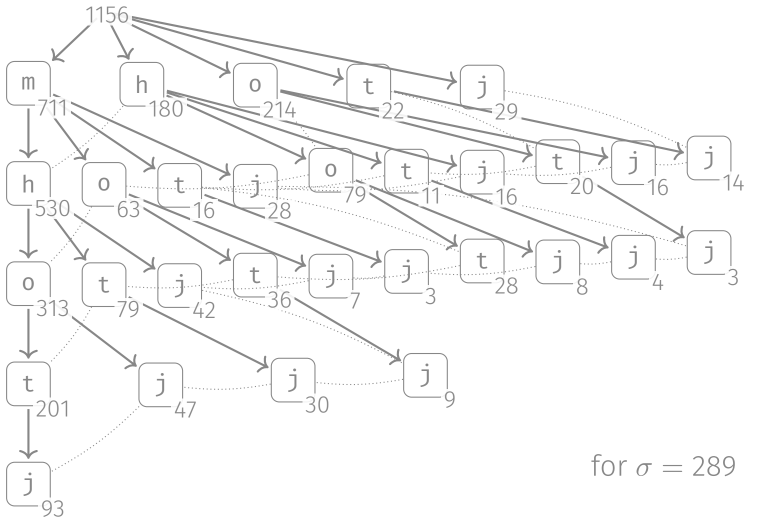
Pizzeria example: Suffix-based pattern growth with the FP-tree

Recursive pattern growth



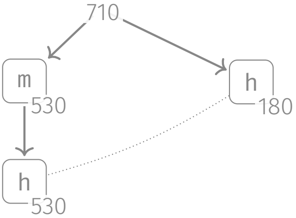
for $\sigma = 289$
Suffix m

Recursive pattern growth



Pizzeria example: Suffix-based pattern growth with the FP-tree

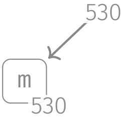
Recursive pattern growth



for $\sigma = 289$
Suffix **h**

Pizzeria example: Suffix-based pattern growth with the FP-tree

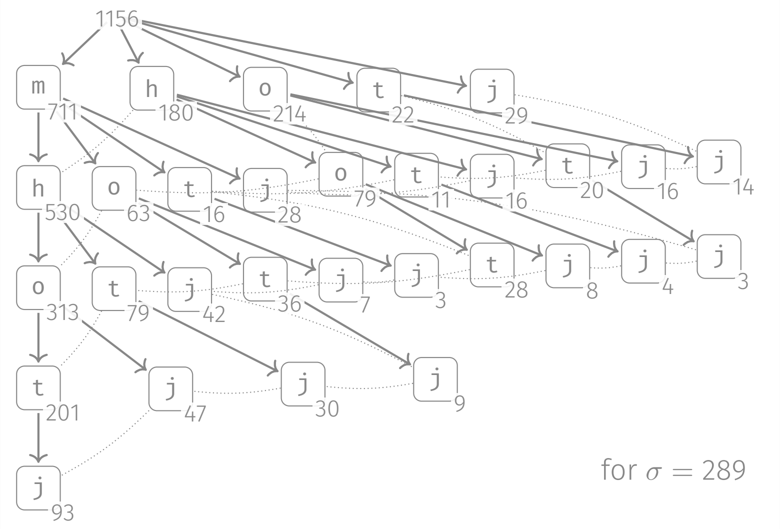
Recursive pattern growth



for $\sigma = 289$
Suffix **mh**

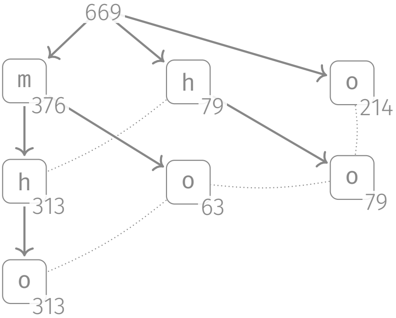
Pizzeria example: Suffix-based pattern growth with the FP-tree

Recursive pattern growth



Pizzeria example: Suffix-based pattern growth with the FP-tree

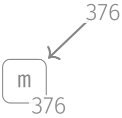
Recursive pattern growth



for $\sigma = 289$
Suffix o

Pizzeria example: Suffix-based pattern growth with the FP-tree

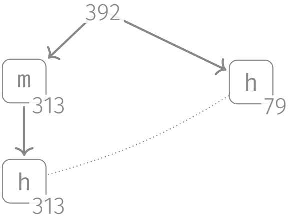
Recursive pattern growth



for $\sigma = 289$
Suffix `mo`

Pizzeria example: Suffix-based pattern growth with the FP-tree

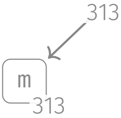
Recursive pattern growth



for $\sigma = 289$
Suffix **ho**

Pizzeria example: Suffix-based pattern growth with the FP-tree

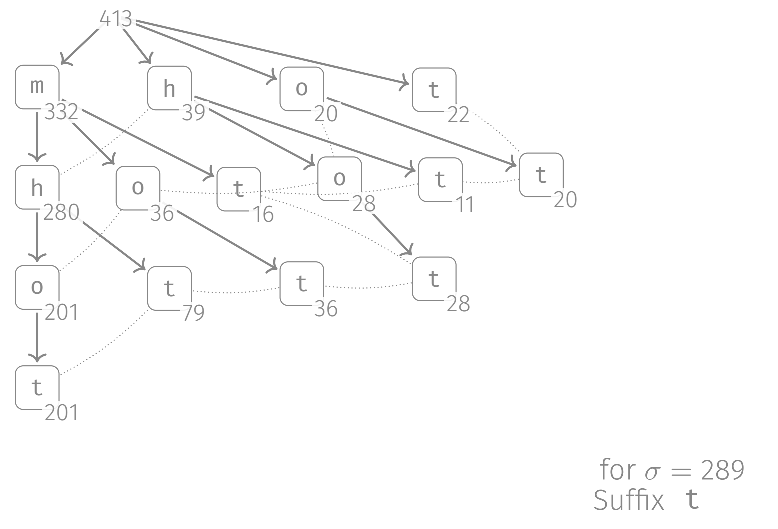
Recursive pattern growth



for $\sigma = 289$
Suffix **mho**

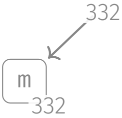
Pizzeria example: Suffix-based pattern growth with the FP-tree

Recursive pattern growth



Pizzeria example: Suffix-based pattern growth with the FP-tree

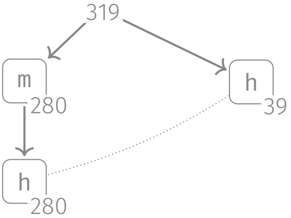
Recursive pattern growth



for $\sigma = 289$
Suffix `mt`

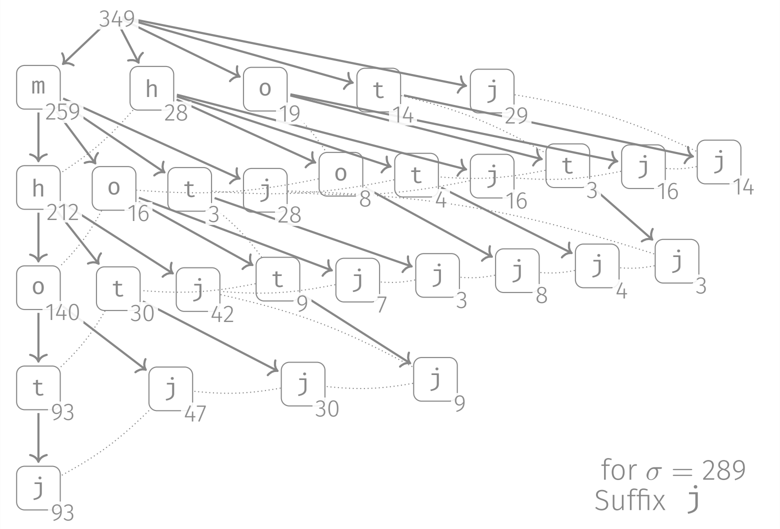
Pizzeria example: Suffix-based pattern growth with the FP-tree

Recursive pattern growth

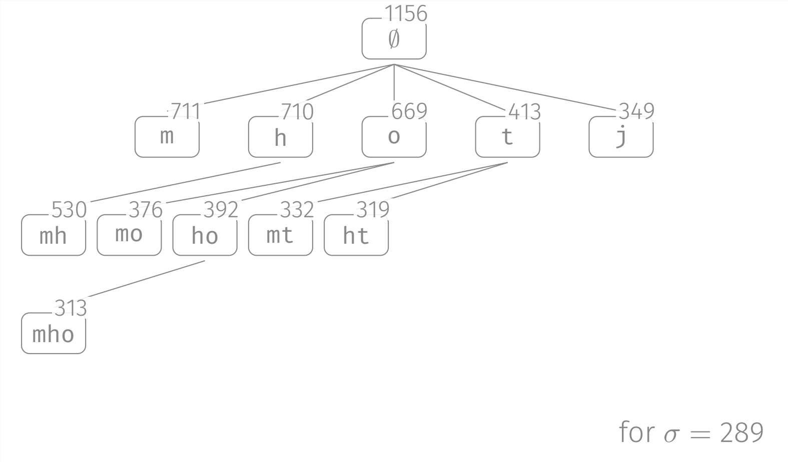


for $\sigma = 289$
Suffix **ht**

Recursive pattern growth



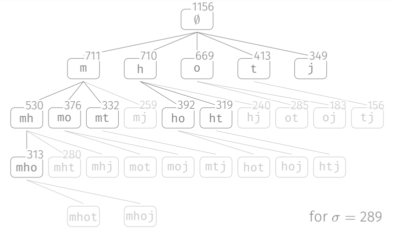
Items ordered by decreasing frequency, suffix growth

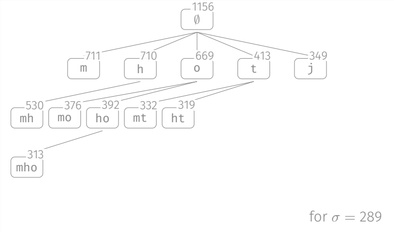


Pizzeria example: Enumeration tree

Note the difference

Apriori algorithm enumeration tree vs. *FP-growth* algorithm enumeration tree





Prefix-based

Traversed breadth-first

Suffix-based

Traversed depth-first

Association rules

Frequent itemsets can be used to generate *association rules*

Classical setting: *Shopping basket data*

- Identify products that are often bought together
  Frequent itemset $\{\text{butter}, \text{bread}, \text{ham}, \text{pickles}\}$
- Extract rules that capture typical buying behaviour
  Association rules $\{\text{bread}, \text{ham}\} \Rightarrow \{\text{butter}, \text{pickles}\}$
- Insights for marketing and shelf placement

Association rules

Frequent itemsets can be used to generate *association rules*

Consider two itemsets X and Y such that

$$X \subset U, \quad \emptyset \neq Y \subseteq U, \quad X \cap Y = \emptyset$$

The confidence of the association rule $X \Rightarrow Y$ is
the *conditional probability* that a transaction contains $X \cup Y$
given that it contains X

$$\mathrm{conf}(X \Rightarrow Y) = \frac{|\mathrm{supp}(X \cup Y)|}{|\mathrm{supp}(X)|}$$

X and Y are called the antecedent and consequent of the rule, respectively

Association rules

$X \Rightarrow Y$ is an association rule at minimum support σ and minimum confidence γ if

$$\mathbf{supp}(X \cup Y) \geq \sigma \quad \text{and} \quad \mathbf{conf}(X \Rightarrow Y) \geq \gamma$$

Mining association rules

1. Mine all the frequent itemsets for minimum support σ
2. Split the frequent itemsets into association rules of minimum confidence γ

Monotonicity of confidence

Let X_a, X_b and I be itemsets such that $X_a \subset X_b \subset I$, then

$$\text{conf}(X_b \Rightarrow I \setminus X_b) \geq \text{conf}(X_a \Rightarrow I \setminus X_a)$$