Frequent Items Sets

Lecture 11

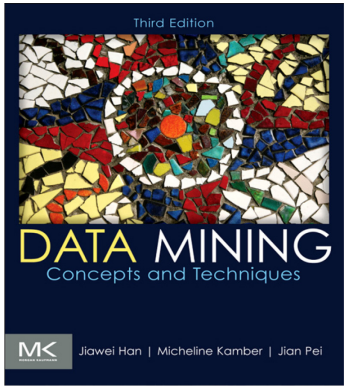
IDS-L11

Chair of Process and Data Science

RWTH AACHEN UNIVERSITY

Outline of Today's Lecture

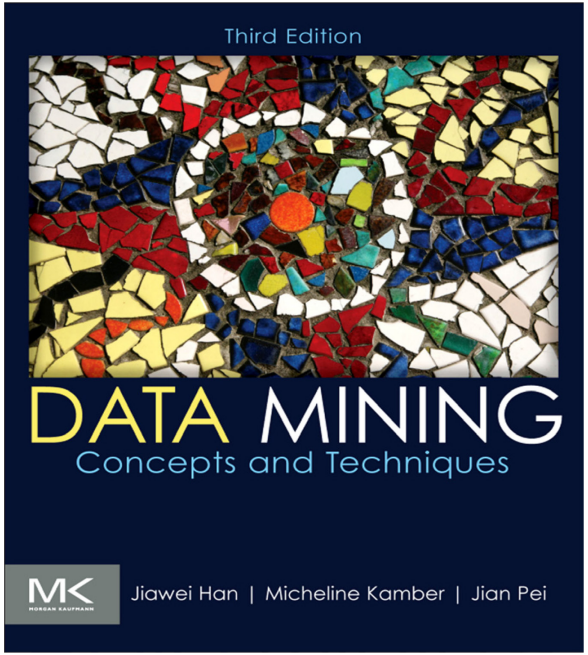
- Pattern mining
- Frequent itemsets
- Apriori algorithm
- FP-growth algorithm



Chapter 6 of Data Mining: Concepts and Techniques (3rd Edition) by Jiawei Han, Micheline Kamber, and Jian Pei. Morgan Kaufmann, June 2011.
&
Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach by Jiawei Han, Jian Pei, Yiwen Yin, Runying Mao. Data Min. Knowl. Discov. 8(1): 53-87 (2004)

Acknowledgement

The lecture mostly based on Chapter 6 of Data Mining: Concepts and Techniques (3rd Edition) by Jiawei Han, Micheline Kamber, and Jian Pei. Morgan Kaufmann, June 2011.



This book will be the main source of information for unsupervised learning (expect for process mining lectures).

As mentioned before, note the differences in terminology.

Pattern mining

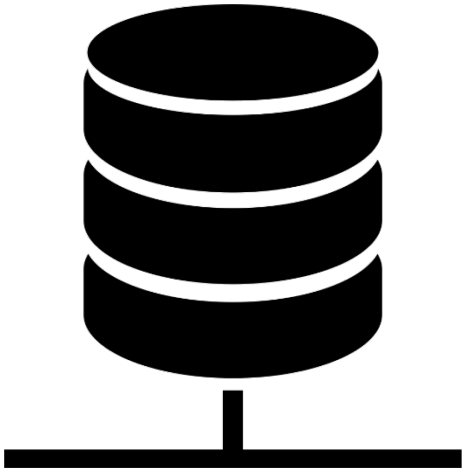
Pattern mining

- **Finding surprising repeating patterns in the input data.**
- **Types of patterns:**
  - **Frequent itemsets** 
  - **Association rules**
  - **Sequential patterns**
  - **Partial orders**
  - **Subgraphs**

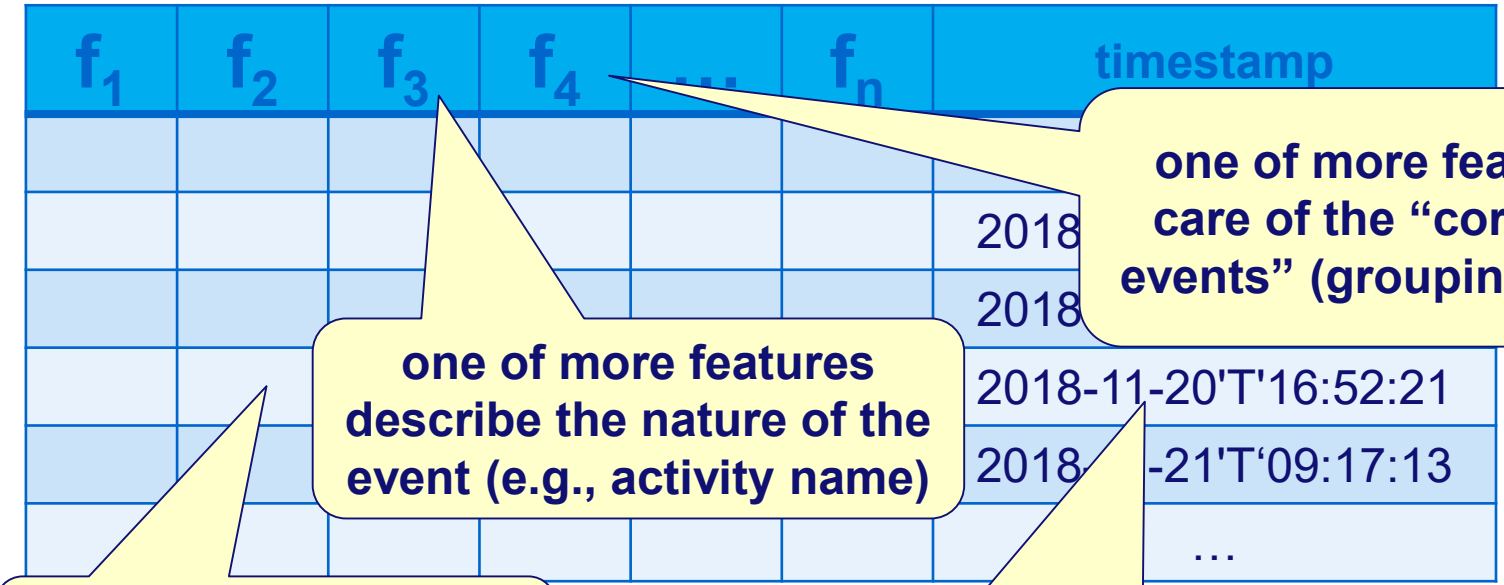
Pattern mining

We focus on two types of data:

1. **Event data (already preparing for process mining)**
2. **Itemset data**



Event data



one of more features take care of the "correlation of events" (grouping of events)

one of more features describe the nature of the event (e.g., activity name)

each row is an event (e.g., transaction)

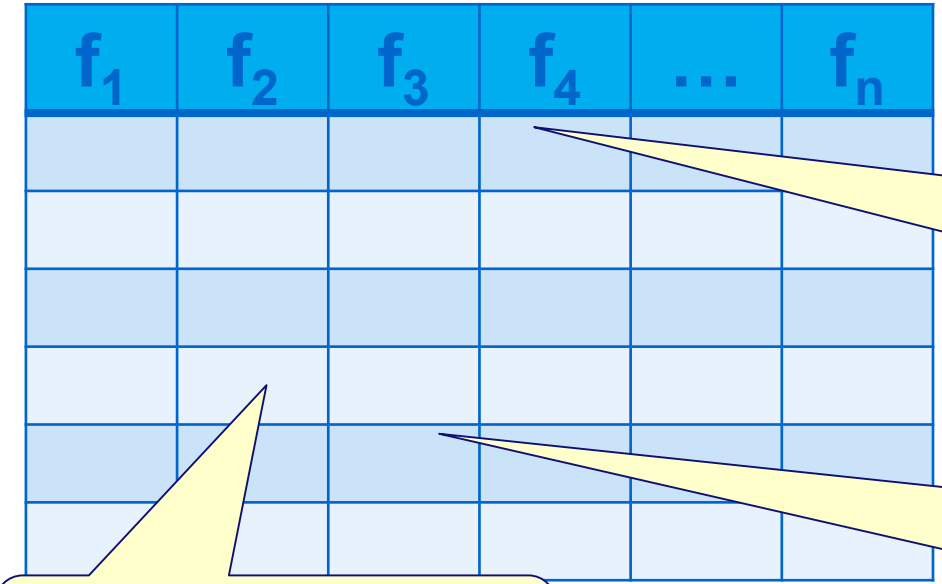
events are ordered based on their timestamps

Chair of Process and Data Science

Event data: Example

order	activity	resource	timestamp
99911	place order	John	2018-11-20'T'13:20:10
99912	place order	Sue	2018-11-20'T'15:33:55
99911	pay	John	2018-11-20'T'15:34:12
99911	collect	Mary	2018-11-20'T'16:52:21
99912	pay	John	2018-11-21'T'09:17:13
…			…

Itemset data

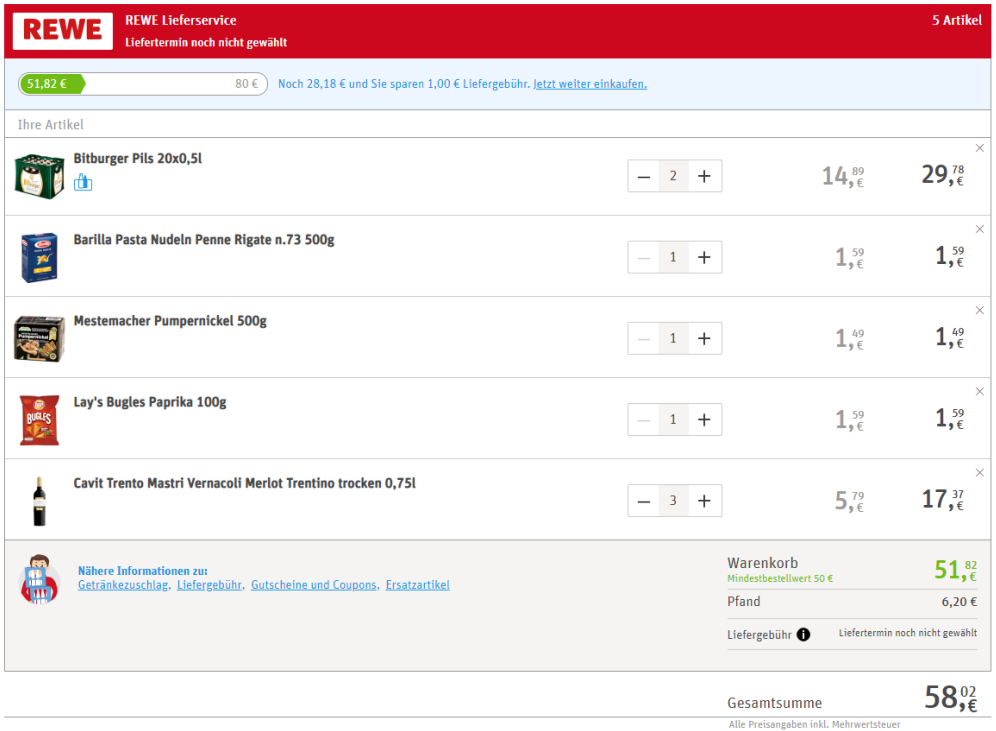


each feature refers to an item (e.g., a product, disease, course, or error code)

Each cell describes the presence of the item. This can be a Boolean (true/false or 0/1 or a natural number (count)

each row is a transaction

Itemset data: Example instance



Chair of Process and Data Science

Itemset data: Example

				...	
2	2	2	3		2
0	0	0	1		0
2	1	0	0		0
0	1	0	0		0
0	0	0	0		2
...	...	...	...		...

Itemset data: Example

				...	
true	true	true	true		true
false	false	false	true		false
true	true	false	false		false
false	true	false	false		false
false	false	false	false		true
...	...	...	...		...

Other itemset data examples

- **Rows: restaurant bills, columns: dishes.**
- **Rows: patients, columns: diseases.**
- **Rows: students graduating, columns: courses taken.**
- **Rows: repair bills, columns: components replaced.**
- **Rows: frequent travelers, columns: countries visited.**
- **Rows: Spotify users, columns: songs played.**
- **Rows: Netflix users, columns: movies watched.**
- **…**

Rows represent instances and columns represent items forming item sets.

Frequent item sets

- Frequent item sets: sets of items that that have a support *s* higher than some threshold.
- {Bitburg, Pumpernickel, Merlot} is frequent if there are many customer buying at least these three items.
- Frequent items sets are used to find association rules.

Examples of association rules (focus of next lecture)

- **{Bitburg, Pumpernickel} ⇒ {Merlot}**
  (people that buy Bitburg pilsner and Pumpernickel bread also tend to buy Merlot wine)
- **{Bitburg} ⇒ {Heineken, Palm}**
  (people that buy Bitburg pilsner tend to buy both Heineken and Palm pilsner)
- **{Carbonara, Margherita } ⇒ {Espresso, Tiramisu}**
  (people that buy Bitburg pilsner and Pumpernickel bread, also tend to buy Merlot wine)
- **{BPI, IDS} ⇒ {APM}**
  (students that take the BPI and IDS courses, also tend to take the APM course) We hope you do!
- **{part-245, part-345, part-456} ⇒ {part-372}**
  (when parts 245, 345, and 456 are replaced, then often also part 372 is replaced)

Frequent itemsets

Dataset is a multiset of transactions

- $\mathcal{I} = \{I_1, I_2, \ldots, I_m\}$ is the set of all items.
- $A \subseteq \mathcal{I}$ is an itemset.
- A transaction T is a non-empty itemset.
- A dataset D is a collection of transactions.
- Technically, $D \in \mathfrak{B}(\mathcal{P}(\mathcal{I}))$ such that $\emptyset \notin D$.

($\mathfrak{B}$ is the multiset and $\mathcal{P}$ is the powerset operator)

Chair of Process and Data Science

Example

$D \in \mathfrak{B}(\mathcal{P}(\mathcal{I}))$

- $\mathcal{I} = \{\text{milk, bread, butter,rice, cheese, ...}\}$ **is the set of all items.**
- $T = \{\text{milk, butter, cheese}\} \subseteq \mathcal{I}$ **is a transaction.**
- $D = [\{\text{milk}\}, \{\text{milk}\}, \{\text{bread}\}, \{\text{milk, bread}\}, \{\text{rice, bread}\}]$ **is a dataset with 5 transactions.**

Mapping example

$$D \in \mathfrak{B}(\mathcal{P}(\mathcal{I}))$$

 Bit	 Pum	 Bug	 Mer	...	 Pen
2	0	0	0		1
0	0	0	1		0
0	1	0	2		5
0	0	0	2		0

$$D = [\{\text{Bit,Pen}\}, \{\text{Mer}\}, \{\text{Pum,Mer,Pen}\}, \{\text{Mer}\}]$$

Chair of Process and Data Science

Generalization

$$D \in \mathfrak{B}(\mathfrak{B}(\mathcal{I}))$$

 Bit	 Pum	 Bug	 Mer	...	 Pen
2	0	0	0		1
0	0	0	1		0
0	1	0	2		5
0	0	0	2		0

$$D = [[\text{Bit}^2, \text{Pen}], [\text{Mer}], [\text{Pum}, \text{Mer}^2, \text{Pen}^5], [\text{Mer}^2]]$$

We will consider only itemsets hat are proper sets and not multisets. However, generalization is trivial.

Support

- The support of an itemset X given a dataset D is the fraction of instances in D that covers X.
- $support(X) = |[A \in D | X \subseteq A]|/|D|$ (relative)
- $support\_count(X) = |[A \in D | X \subseteq A]|$ (absolute, also called frequency or count)
- Minimum support threshold min_sup: lower bound for $support(X)$.
- Minimum support count threshold: lower bound for $support\_count(X)$.

Example support

$D \in \mathfrak{B}(\mathcal{P}(\mathcal{I}))$

$$D = [\{\text{Bit,Pen}\}, \{\text{Mer}\}, \{\text{Pum,Mer,Pen}\}, \{\text{Mer}\}]$$

$$support(\{\text{Mer}\}) = 3/4 \qquad support\_count(\{\text{Mer}\}) = 3$$

$$support(\{\text{Bit,Pen}\}) = 1/4 \qquad support\_count(\{\text{Bit,Pen}\}) = 1$$

$$support(\{\text{Mer,Pen}\}) = 1/4 \qquad support\_count(\{\text{Mer,Pen}\}) = 1$$

$$support(\{\text{Bit,Mer}\}) = 0/4 \qquad support\_count(\{\text{Bit,Mer}\}) = 0$$

Problem statement

Given data set $D \in \mathfrak{B}(\mathcal{P}(\mathcal{I}))$ and minimum support threshold min_sup, return all frequent non-empty itemsets:

$$\{X \subseteq \mathcal{I} | support(X) \geq min\_sup\}$$

Naïve approach

- Given $X \subseteq \mathcal{I}$ it is possible to check whether $support(X) \geq min\_sup$ by testing all instances.
- If there are m unique items, then there are $2^m - 1$ candidate itemsets that can all be tested individually.
- However, this can be very time consuming …

Number of unique items in a supermarket

Grocery stores carry 40,000 more items than they did in the 1990s

Published: June 17, 2017 7:57 a.m. ET

Here's how much grocery shopping has changed over the last three decades

Tesco cuts range by 30% to simplify shopping

By reducing number of products from 90,000, supermarket will be able to cut prices and improve availability on its shelves

Tesco trolleys. Photograph: Chris Radburn/PA

Tesco's new boss Dave Lewis is pulling up to a third of products off its shelves as it calls time on policy that left shoppers baffled by a choice of up to 90,000 products on their weekly shop.

Assume $m = 50000$ products

$2^m - 1 =$

3,160,699,436,856,317,896,135,924,659,945,691,788,984,676,387,834,935,666,847,743,155,564,943,937,902,009,550,651,067,144,922,529,420,974,282,690,343,798,061,622,891,650,247,060,091,533,595,130,170,365,868,108,099,970,116,531,087,467,047,583,722,093,787,639,674,649,765,662,074,366,466,883,324,927,932,743,926,222,226,256,325,646,619,479,597,070,8

[illegible]

7,416,182,773,952,840,881,445,035,913,235,822,373,550,516,126,262,884,445,234,405,811,592,466,055,571,627,450,168,569,925,389,721,039,944,218,176,868,921,181,944,425,304,096,132,312,721,538,680,757,455,121,594,473,310,802,579,385,393,574,059,209,602,443,968,201,967,760,613,102,289,456,131,085,235,835,109,375

Good luck!

Of course most of them are not frequent, but still…

Assume $support(X) \geq min\_sup$ and $|X| = 100$

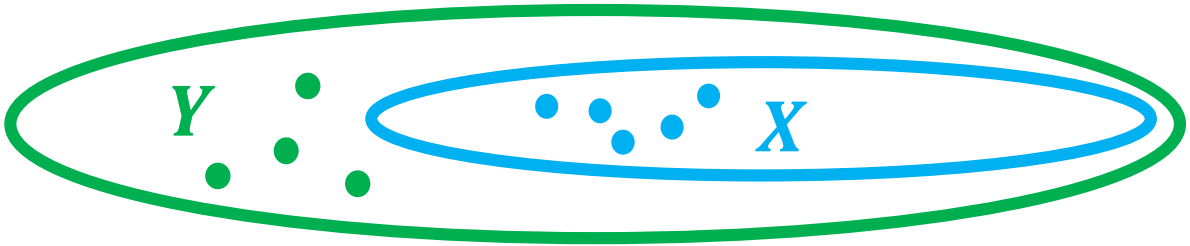
- Assume $X = \{a_1, a_2, \dots, a_{100}\}$ and $support(X) \geq min\_sup$.
- All subsets are also frequent!
- There are $\binom{100}{1} = 100$ frequent itemsets having 1 item.
- There are $\binom{100}{k}$ frequent itemsets having k items.
- There are $\binom{100}{1} + \binom{100}{2} + \cdots \binom{100}{99} = 2^{100} - 2 = 1.27 \times 10^{30}$ smaller frequent itemsets contained in X.

Conclusion

- **We should avoid exhaustively testing all candidate itemsets.**
- **We need to focus on the “interesting” ones.**

Closed itemsets

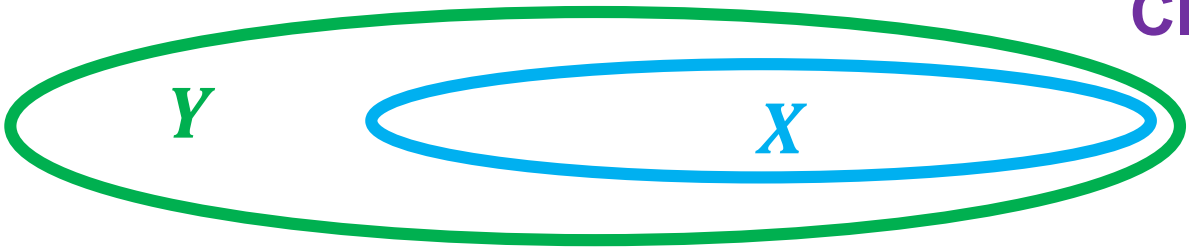
- An itemset X is closed in data set D if there is no proper superset $Y \supset X$ that has the same support.



If X is closed, then $support(X) > support(Y)$ for any $Y \supset X$.

Closed frequent itemsets

- An itemset X is closed in data set D if there is no proper superset $Y \supset X$ that has the same support.

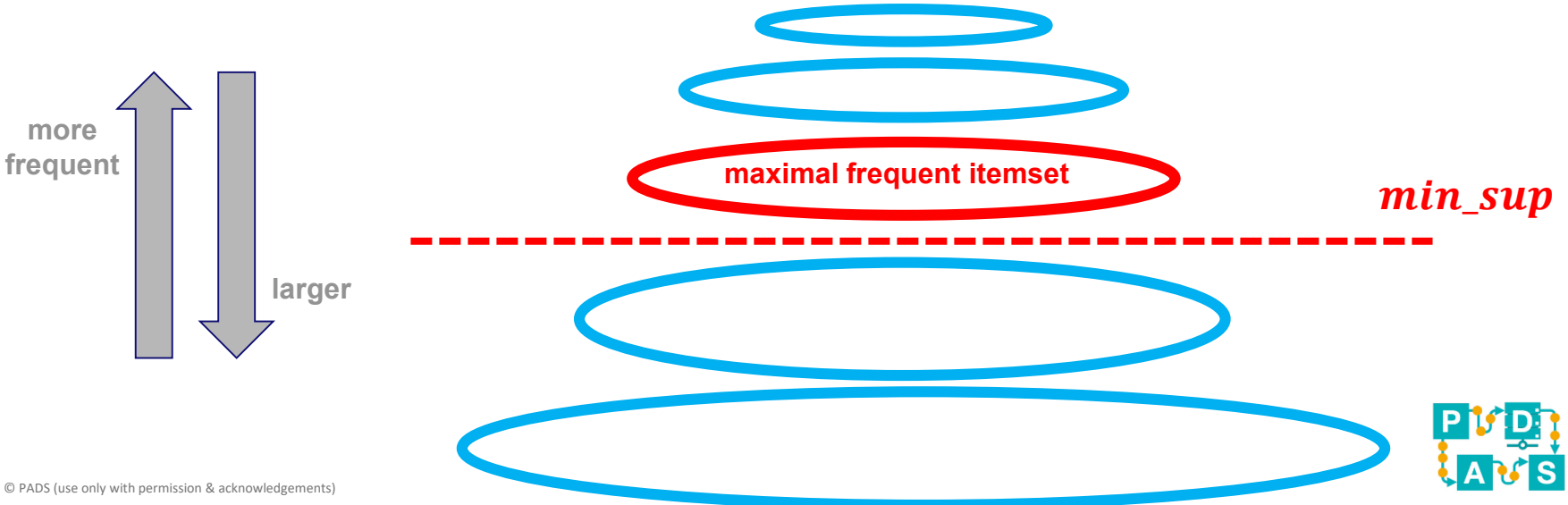


Closed frequent itemsets are closed and frequent.

If X is closed, then $support(X) > support(Y)$ for any $Y \supset X$.

Maximal frequent itemsets

- **An itemset X is a maximal frequent itemset in data set D if X is frequent, and there is no proper superset $Y \supset X$ that is also frequent.**



Chair of Process and Data Science

Example

- Assume $\mathcal{I} = \{a_1, a_2, \ldots, a_{100}\}, D = [\{a_1, a_2, \ldots, a_{50}\}^{10}, \{a_1, a_2, \ldots, a_{100}\}^{10}]$, and $min\_sup = \frac{5}{20}$.
- There are $2^{100} - 1 = 1.27 \times 10^{30}$ frequent itemsets.
- There are two closed frequent itemsets: $A = \{a_1, a_2, \ldots, a_{50}\}$ and $B = \{a_1, a_2, \ldots, a_{100}\}$. Note that $support(A) = \frac{20}{20}$ and $support(B) = \frac{10}{20}$.
- There is one maximal frequent itemset: $\{a_1, a_2, \ldots, a_{100}\}$.

Example

- Assume $\mathcal{I} = \{a_1, a_2, \dots, a_{100}\}, D = [\{a_1, a_2, \dots, a_{50}\}^{10}, \{a_1, a_2, \dots, a_{100}\}^{10}]$, and $min\_sup = \frac{15}{20}$.
- There are $2^{50} - 1 =$ frequent itemsets.
- There is only one closed frequent itemset: $\{a_1, a_2, \dots, a_{50}\}$.
- There is one maximal frequent itemset: $\{a_1, a_2, \dots, a_{50}\}$.

Example

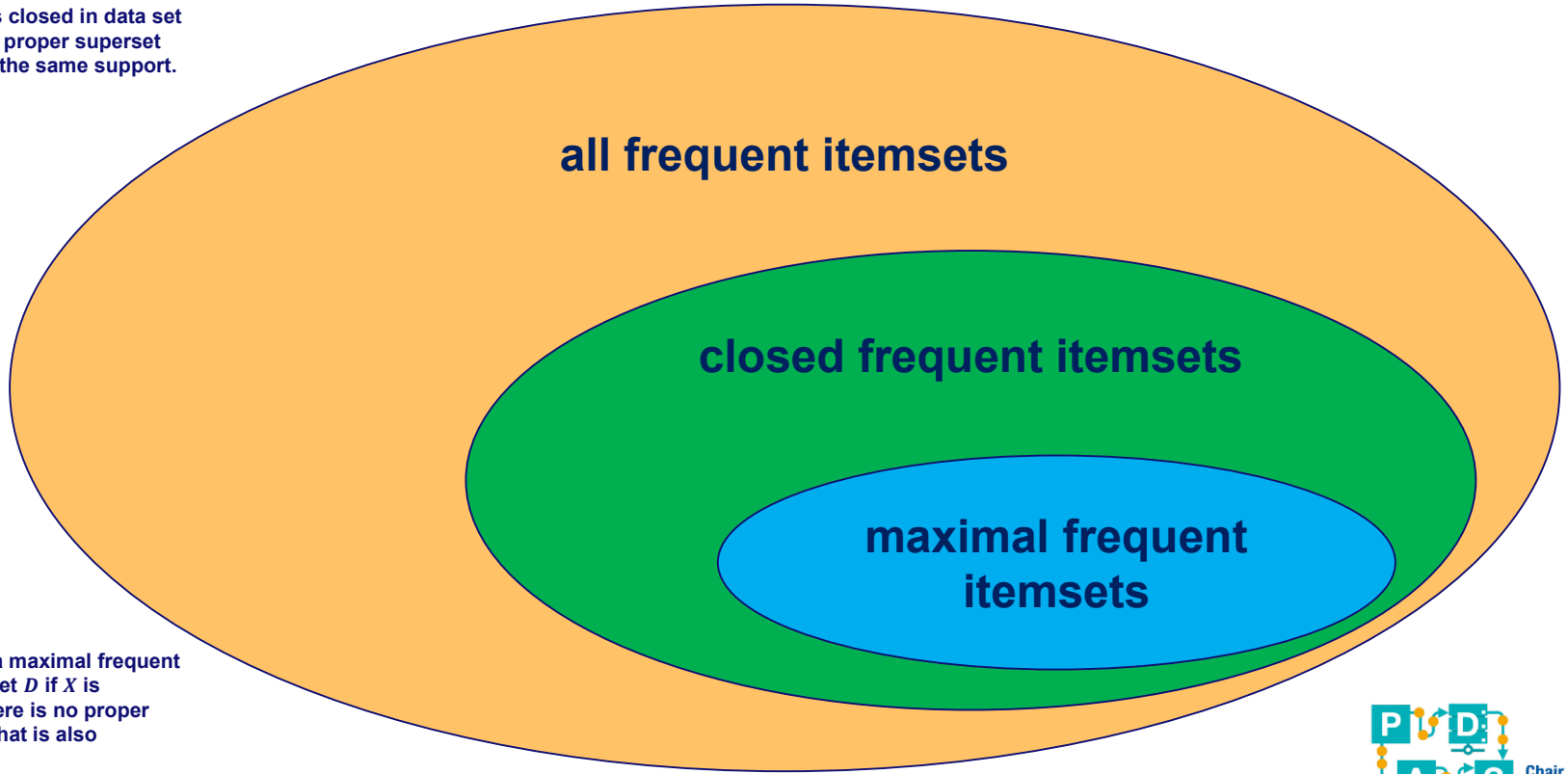
- Assume $\mathcal{I} = \{a_1, a_2, \dots, a_{100}\}, D = [\{a_1, a_2, \dots, a_{99}\}^{10}, \{a_2, a_2, \dots, a_{100}\}^{10}]$, and $min\_sup = \frac{15}{20}$.
- There are $2^{98} - 1$ frequent itemsets.
- There is one closed frequent itemset: $A = \{a_2, a_2, \dots, a_{99}\}$ with $support(A) = \frac{20}{20}$.
- There is one maximal frequent itemset: $\{a_2, a_2, \dots, a_{99}\}$.

Observation

- The frequencies of only the closed frequent itemsets provide complete information about the frequencies of all frequent itemsets.
- If $A \subset B$, B is a closed frequent itemset, and there is no closed frequent itemset B' such that $A \subseteq B' \subset B$, then $support(A) = support(B)$.
- Hence, it suffices to store only closed frequent itemsets
- (Maximal frequent itemsets provide less information.)

Relationships

An itemset X is closed in data set D if there is no proper superset $Y \supset X$ that has the same support.



An itemset X is a maximal frequent itemset in data set D if X is frequent, and there is no proper superset $Y \supset X$ that is also frequent.

Chair of Process and Data Science

Apriori algorithm

Apriori algorithm

- Introduced by Rakesh Agrawal and Ramakrishnan Srikant in “Fast Algorithms for Mining Association Rules in Large Databases. VLDB 1994: 487-499”.
- Apriori uses a “bottom up” approach, where frequent subsets are extended one item at a time (candidate generation), and superfluous checks are avoided by using information from smaller subsets.

Fast Algorithms for Mining Association Rules

Rakesh Agrawal Ramakrishnan Srikant*

IBM Almaden Research Center
650 Harry Road, San Jose, CA 95120

Abstract

We consider the problem of discovering association rules between items in a large database of sales transactions. We present two new algorithms for solving this problem that are fundamentally different from the known algorithms. Empirical evaluation shows that these algorithms outperform the known algorithms by factors ranging from three for small problems to more than an order of magnitude for large problems. We also show how the best features of the two proposed algorithms can be combined into a hybrid algorithm, called AprioriHybrid. Scale-up experiments show that AprioriHybrid scales linearly with the number of transactions. AprioriHybrid also has excellent scale-up properties with respect to the transaction size and the number of items in the database.

1 Introduction

Progress in bar-code technology has made it possible for retail organizations to collect and store massive amounts of sales data, referred to as the *basket* data. A record in such data typically consists of the transaction date and the items bought in the transaction. Successful organizations view such databases as important pieces of the marketing infrastructure. They are interested in instituting information-driven marketing processes, managed by database technology, that enable marketers to develop and implement customized marketing programs and strategies [6].

The problem of mining association rules over basket data was introduced in [4]. An example of such a rule might be that 98% of customers that purchase tires and auto accessories also get automotive services done. Finding all such rules is valuable for cross-marketing and attached mailing applications. Other applications include catalog design, add-on sales, store layout, and customer segmentation based on buying patterns. The databases involved in these applications are very large. It is imperative, therefore, to have fast algorithms for this task.

The following is a formal statement of the problem [4]: Let $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$ be a set of literals, called items. Let $\mathcal{D}$ be a set of transactions, where each transaction T is a set of items such that $T \subseteq \mathcal{I}$. Associated with each transaction is a unique identifier, called its *TID*. We say that a transaction T *contains* X, a set of some items in $\mathcal{I}$, if $X \subseteq T$. An *association rule* is an implication of the form $X \Rightarrow Y$, where $X \subset \mathcal{I}$, $Y \subset \mathcal{I}$, and $X \cap Y = \emptyset$. The rule $X \Rightarrow Y$ holds in the transaction set $\mathcal{D}$ with *confidence* c if c% of transactions in $\mathcal{D}$ that contain X also contain Y. The rule $X \Rightarrow Y$ has *support* s in the transaction set $\mathcal{D}$ if s% of transactions in $\mathcal{D}$ contain $X \cup Y$. Our rules are somewhat more general than in [4] in that we allow a consequent to have more than one item.

Given a set of transactions $\mathcal{D}$, the problem of mining association rules is to generate all association rules that have support and confidence greater than the user-specified minimum support (called *minsup*) and minimum confidence (called *minconf*) respectively. Our discussion is neutral with respect to the representation of $\mathcal{D}$. For example, $\mathcal{D}$ could be a data file, a relational table, or the result of a relational expression.

An algorithm for finding all association rules, henceforth referred to as the *AIS* algorithm, was presented in [4]. Another algorithm for this task, called the *SETM* algorithm, has been proposed in [13]. In this paper, we present two new algorithms, *Apriori* and *AprioriTid*, that differ fundamentally from these algorithms. We present experimental results showing

*Visiting from the Department of Computer Science, University of Wisconsin, Madison.

Proceedings of the 20th VLDB Conference
Santiago, Chile, 1994

Basic idea: Antimonotinicity

- Let $D \in \mathfrak{B}\left(\mathcal{P}(\mathcal{I})\right)$ be a dataset and min_sup a chosen threshold.
- For any $A \subseteq \mathcal{I}$ and $B \subseteq \mathcal{I}$.
  - If $A \subseteq B$, then $support(A) \geq support(B)$.
  - If $A \subseteq B$ and $support(B) \geq min\_sup$, then $support(A) \geq min\_sup$.
  - If $A \subseteq B$ and $support(A) < min\_sup$, then $support(B) < min\_sup$.
- Antimonotinicity will be used to prune search space.

Basic idea: Leveling

- Let $D \in \mathfrak{B}\left(\mathcal{P}(\mathcal{I})\right)$ be a dataset and min_sup a chosen threshold.
- $L_k = \{A \subseteq \mathcal{I} | support(A) \geq min\_sup \wedge |A| = k\}$.
- L_k are all frequent itemsets of length k.
- For any $A \in L_k$ there exist $A', A'' \in L_{k-1}$ such that $A = A' \cup A''$ $(k \geq 2)$.
- For any $A \in L_k$ and $A', A'' \subseteq \mathcal{I}$ such that $A = A' \cup A''$ and $|A'| = |A''| = k - 1$: $A', A'' \in L_{k-1}$ $(k \geq 2)$.

Basic idea: Leveling

- Assume that the items are ordered $(a_1, a_2, \ldots)$.
- Let $A = \{a_1, a_2, \ldots, a_{k-1}, a_k\} \in L_k$.
  - Then $A' = \{a_1, a_2, \ldots a_{k-2}, a_{k-1}\} \in L_{k-1}$.
  - And $A'' = \{a_1, a_2, \ldots a_{k-2}, a_k\} \in L_{k-1}$.
- Hence, any $A \in L_k$ can be obtained by joining $A' \in L_{k-1}$ and $A'' \in L_{k-1}$ such that the first $k-2$ elements are identical:
  - $A' \cap A'' = \{a_1, a_2, \ldots a_{k-2}\} = A'''$
  - $A' = A''' \cup \{a_{k-1}\}$, $A'' = A''' \cup \{a_k\}$, $A = A''' \cup \{a_{k-1}, a_k\}$.

Basic idea: Candidate set C_k

- Any $A = \{a_1, a_2, \dots, a_{k-1}, a_k\} \in L_k$ can be obtained by joining $A' = \{a_1, a_2, \dots a_{k-2}, a_{k-1}\} \in L_{k-1}$ and $A'' = \{a_1, a_2, \dots a_{k-2}, a_k\} \in L_{k-1}$.
- This can be done efficient without creating duplicates.
- Let C_k be the set of all candidates created from L_{k-1} in this manner.
- Prune the set C_k by removing itemsets that have infrequent subsets.

Basic idea: Pruning

- Let $A \in C_k$ be a candidate. A can be removed if any of it subsets is infrequent.
- For any $a \in A$:
  - check whether $A \backslash \{a\} \in L_k$
  - If not, remove A from C_k.
- Note that the creation of C_k and subsequent pruning is done without looking at the dataset!

Basic idea: Test candidates

- Consider all transactions in $t \in D \in \mathfrak{B}(\mathcal{P}(\mathcal{I}))$.
- For each candidate $c \in C_k$: increment the corresponding counter if $c \subseteq t$.
- Afterwards, we know the frequencies of the candidates and we can compute L_k from C_k.

$$L_k = \{c \in C_k | support(c) \geq min\_sup\}$$

Then move to level L_{k+1}

Algorithm

Algorithm: Apriori. Find frequent itemsets using an iterative level-wise approach based on candidate generation.

Input:

- D, a database of transactions;
- *min_sup*, the minimum support count threshold.

Output: L, frequent itemsets in D.

Method:

```
(1)   L1 = find_frequent_1-itemsets(D);
(2)   for (k = 2; Lk−1 ≠ ϕ; k++) {
(3)     Ck = apriori_gen(Lk−1);
(4)     for each transaction t ∈ D { // scan D for counts
(5)       Ct = subset(Ck, t); // get the subsets of t that are candidates
(6)       for each candidate c ∈ Ct
(7)         c.count++;
(8)     }
(9)     Lk = {c ∈ Ck|c.count ≥ min_sup}
(10)  }
(11)  return L = ∪k Lk;

procedure apriori_gen(Lk−1:frequent (k − 1)-itemsets)
(1)   for each itemset l1 ∈ Lk−1
(2)     for each itemset l2 ∈ Lk−1
(3)       if (l1[1] = l2[1]) ∧ (l1[2] = l2[2])
            ∧... ∧ (l1[k − 2] = l2[k − 2]) ∧ (l1[k − 1] < l2[k − 1]) then {
(4)         c = l1 ⋈ l2; // join step: generate candidates
(5)         if has_infrequent_subset(c, Lk−1) then
(6)           delete c; // prune step: remove unfruitful candidate
(7)         else add c to Ck;
(8)       }
(9)   return Ck;

procedure has_infrequent_subset(c: candidate k-itemset;
        Lk−1: frequent (k − 1)-itemsets); // use prior knowledge
(1)   for each (k − 1)-subset s of c
(2)     if s ∉ Lk−1 then
(3)       return TRUE;
(4)   return FALSE;
```

creating candidates C_k based on L_{k-1}

testing candidates to create L_k based on C_k

create candidates from L_{k-1} (avoiding duplicates)

prune set of candidates by checking whether subsets were frequent before

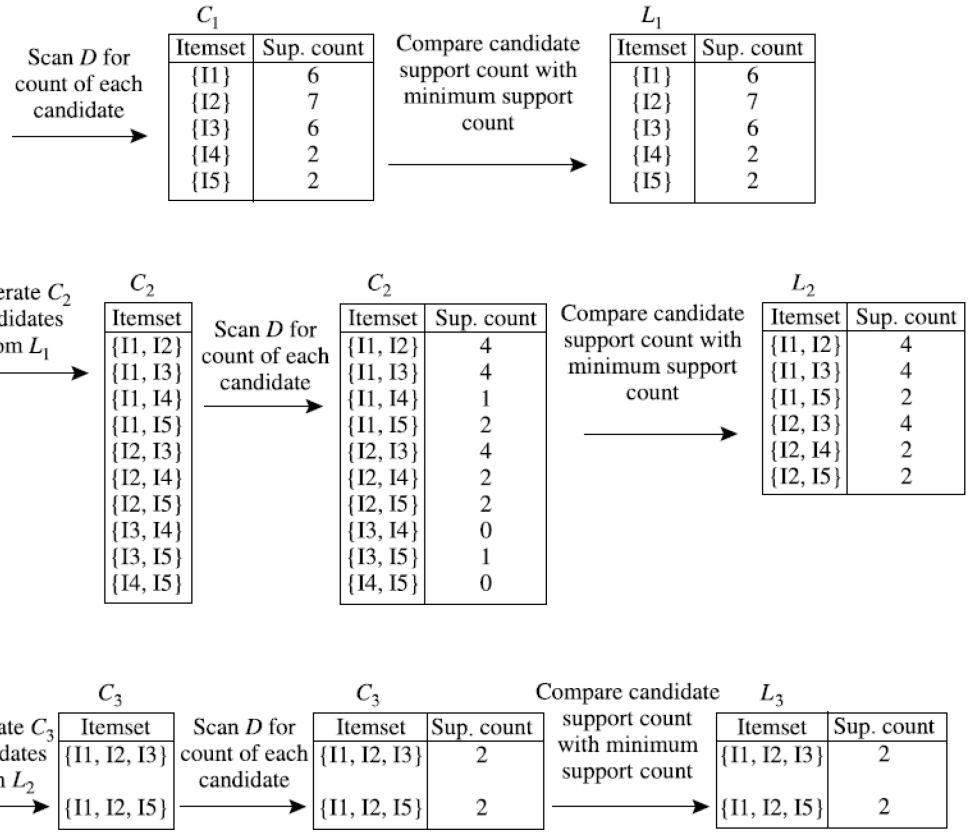
Chair of Process and Data Science

Example in book

Transactional Data for an *AllElectronics* Branch

TID	*List of item_IDs*
T100	I1, I2, I5
T200	I2, I4
T300	I2, I3
T400	I1, I2, I4
T500	I1, I3
T600	I2, I3
T700	I1, I3
T800	I1, I2, I3, I5
T900	I1, I2, I3

$min\_sup = \frac{2}{9}$



Optimizations

- Further optimizations:
  - Distribute the data (can be done in various ways).
  - Gradually remove transactions not containing any frequent itemset of length k.
  - Sampling.
- However, it may remain challenging to generate the candidate sets (may be huge).
- Each candidate needs to be tested against the whole dataset.
- FP-growth is an approach that tries to overcome these limitations.

FP-growth algorithm

Frequent Path Growth (FP-growth)

- **Introduced by Jiawei Han, Jian Pei, Yiwen Yin in Mining Frequent Patterns without Candidate Generation. SIGMOD Conference 2000: 1-12.**
- **Based on constructing the Frequent Path Tree (FP-tree).**
- **Avoid generating many candidates.**
- **Depth-first rather than breadth-first.**

Mining Frequent Patterns without Candidate Generation *

Jiawei Han, Jian Pei, and Yiwen Yin
School of Computing Science
Simon Fraser University
{han, peijian, yiweny}@cs.sfu.ca

Abstract

Mining frequent patterns in transaction databases, time-series databases, and many other kinds of databases has been studied popularly in data mining research. Most of the previous studies adopt an Apriori-like candidate set generation-and-test approach. However, candidate set generation is still costly, especially when there exist prolific patterns and/or long patterns.

In this study, we propose a novel frequent pattern tree (FP-tree) structure, which is an extended prefix-tree structure for storing compressed, crucial information about frequent patterns, and develop an efficient FP-tree-based mining method, FP-growth, for mining *the complete set of frequent patterns* by pattern fragment growth. Efficiency of mining is achieved with three techniques: (1) a large database is compressed into a highly condensed, much smaller data structure, which avoids costly, repeated database scans, (2) our FP-tree-based mining adopts a pattern fragment growth method to avoid the costly generation of a large number of candidate sets, and (3) a partitioning-based, divide-and-conquer method is used to decompose the mining task into a set of smaller tasks for mining confined patterns in conditional databases, which dramatically reduces the search space. Our performance study shows that the FP-growth method is efficient and scalable for mining both long and short frequent patterns, and is about an order of magnitude faster than the Apriori algorithm and also faster than some recently reported new frequent pattern mining methods.

* The work was supported in part by the Natural Sciences and Engineering Research Council of Canada (grant NSERC-A3723), the Networks of Centres of Excellence of Canada (grant NCE/IRIS-3), and the Hewlett-Packard Lab., U.S.A.

1 Introduction

Frequent pattern mining plays an essential role in mining associations [3, 12], correlations [6], causality [19], sequential patterns [4], episodes [14], multi-dimensional patterns [13, 11], max-patterns [5], partial periodicity [9], emerging patterns [7], and many other important data mining tasks.

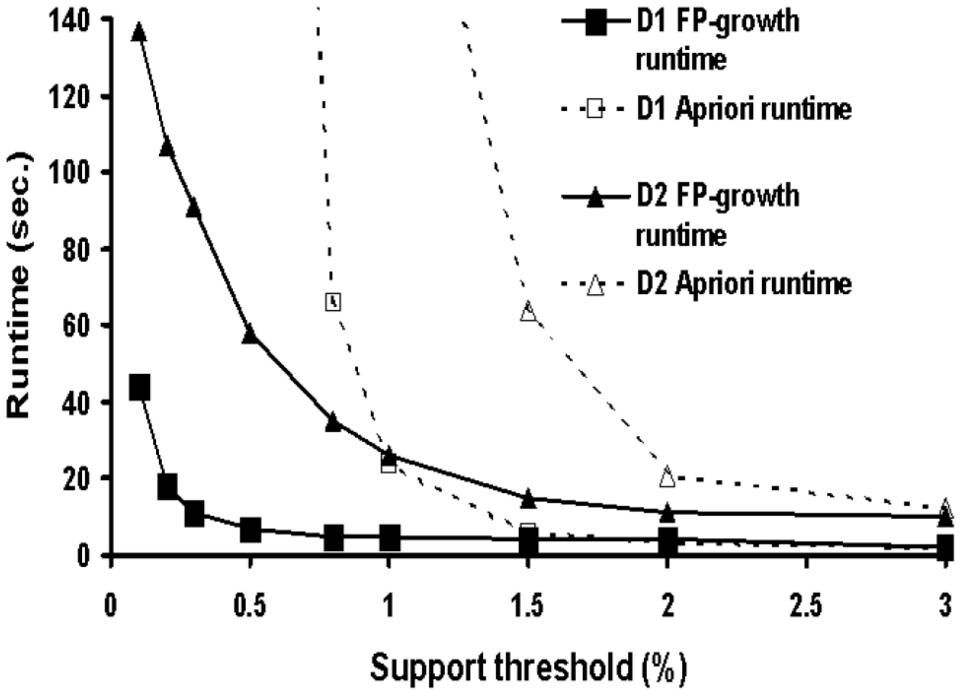
Most of the previous studies, such as [3, 12, 18, 16, 13, 17, 20, 15, 8], adopt an Apriori-like approach, which is based on an *anti-monotone* Apriori *heuristic* [3]: *if any length k pattern is not frequent in the database, its length $(k+1)$ super-pattern can never be frequent.* The essential idea is to iteratively generate the set of candidate patterns of length $(k+1)$ from the set of frequent patterns of length k (for $k \geq 1$), and check their corresponding occurrence frequencies in the database.

The Apriori heuristic achieves good performance gain by (possibly significantly) reducing the size of candidate sets. However, in situations with prolific frequent patterns, long patterns, or quite low minimum support thresholds, an Apriori-like algorithm may still suffer from the following two nontrivial costs:

- It is costly to handle a huge number of candidate sets. For example, if there are 10^4 frequent 1-itemsets, the Apriori algorithm will need to generate more than 10^7 length-2 candidates and accumulate and test their occurrence frequencies. Moreover, to discover a frequent pattern of size 100, such as $\{a_1, \ldots, a_{100}\}$, it must generate more than $2^{100} \approx 10^{30}$ candidates in total. This is the inherent cost of candidate generation, no matter what implementation technique is applied.
- It is tedious to repeatedly scan the database and check a large set of candidates by pattern matching, which is especially true for mining long patterns.

Also see Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach by Jiawei Han, Jian Pei, Yiwen Yin, Runying Mao. Data Min. Knowl. Discov. 8(1): 53-87 (2004)

Motivation



Taken from Jiawei Han, Jian Pei, Yiwen Yin, Mining Frequent Patterns without Candidate Generation. SIGMOD Conference 2000: 1-12.

D1 and D2 are two datasets.

Step 1: Build the FP-tree

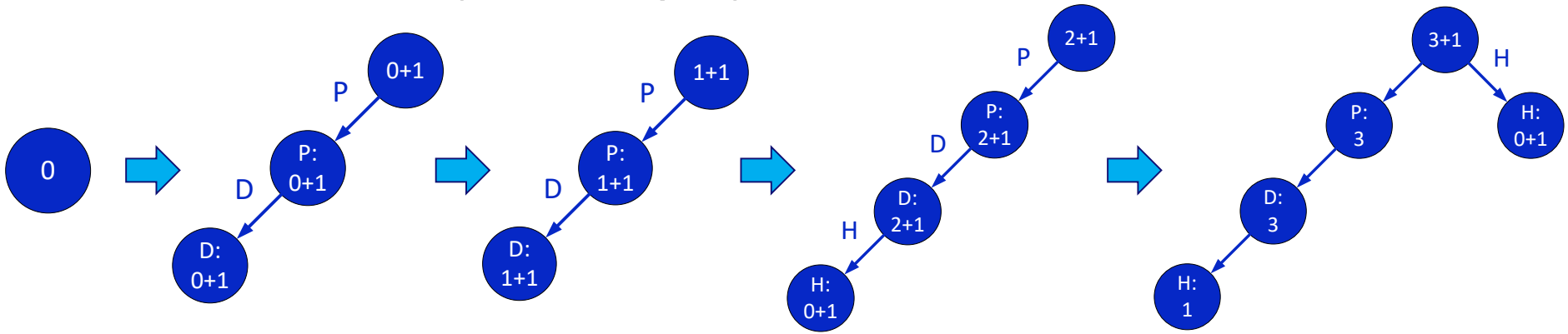
- Determine the frequency of each item. Requires one pass through the data set $D \in \mathfrak{B}(\mathcal{P}(\mathcal{I}))$.
- Sort $\mathcal{I} = \{a_1, a_2, \ldots, a_n\}$ based on frequencies. a_1 is most frequent and a_n is least frequent but still frequent (ignore other non-frequent items).
- The instances can be lexicographically ordered.
- This can be used to build a so-called prefix tree (second pass trough the data set D).
- The resulting FP-tree contains all information needed (no need to traverse the dataset again).

Step 1: Build the FP-tree

- **Note that:**
  - **A first pass is needed to identify frequent singleton items.**
  - **These items are ordered by frequency (if items have the same frequency, pick an arbitrary, but fixed order): $a_1, a_2, \ldots, a_n$.**
  - **Infrequent items are ignored completely (do not appear in tree and are ignored in data set).**

FP-tree

Let's assume a threshold of 10 (P,D,H are frequent).

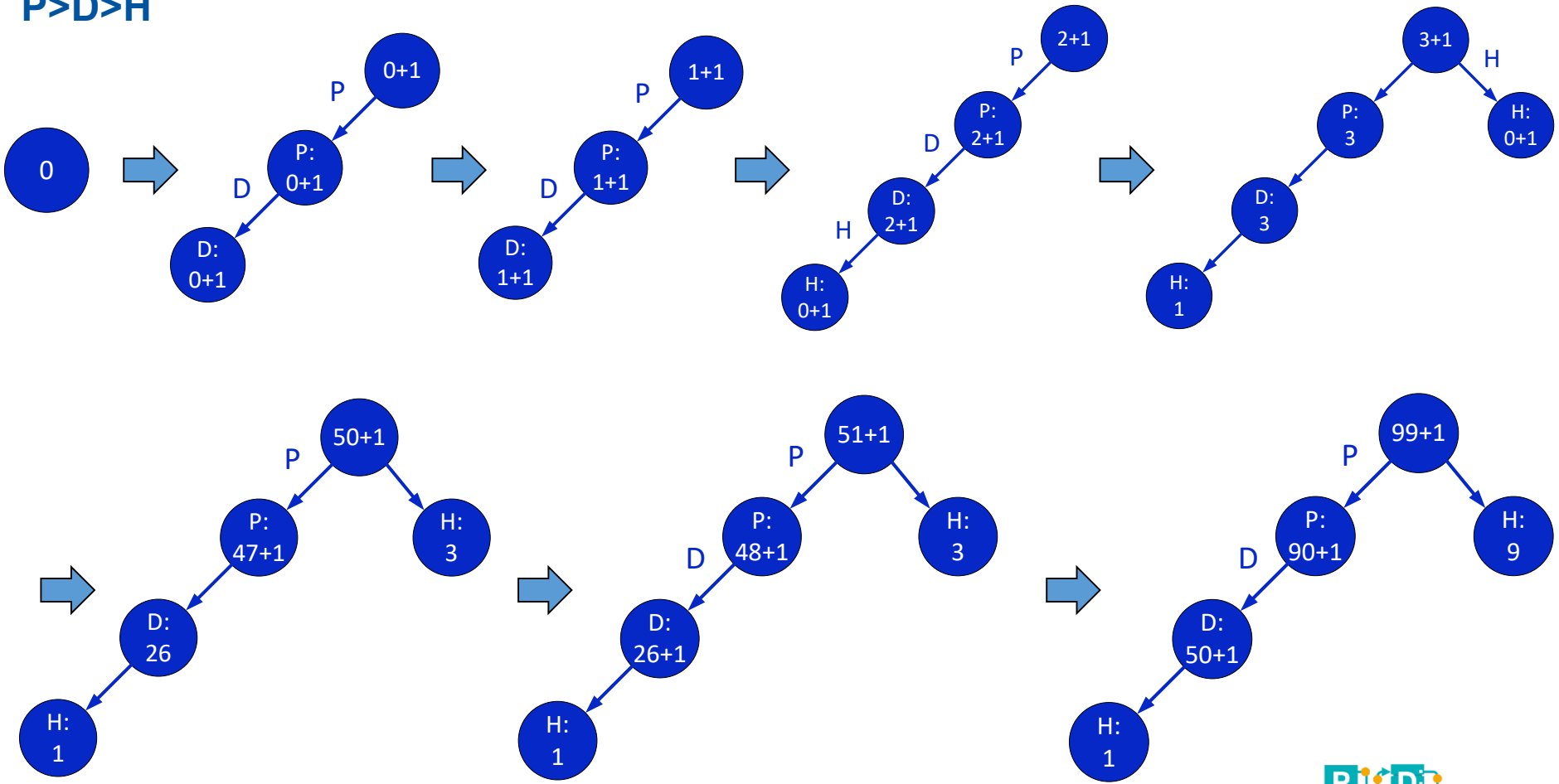


Simplified explanation:

- **Count overall frequency and remove infrequent items.**
- **Sort all projected instances in order of global frequency (here P>D>H).**
- **Build a tree adding the projected instances one-by-one (as shown above).**
- **Prune the tree (note that frequency decreases monotonically).**

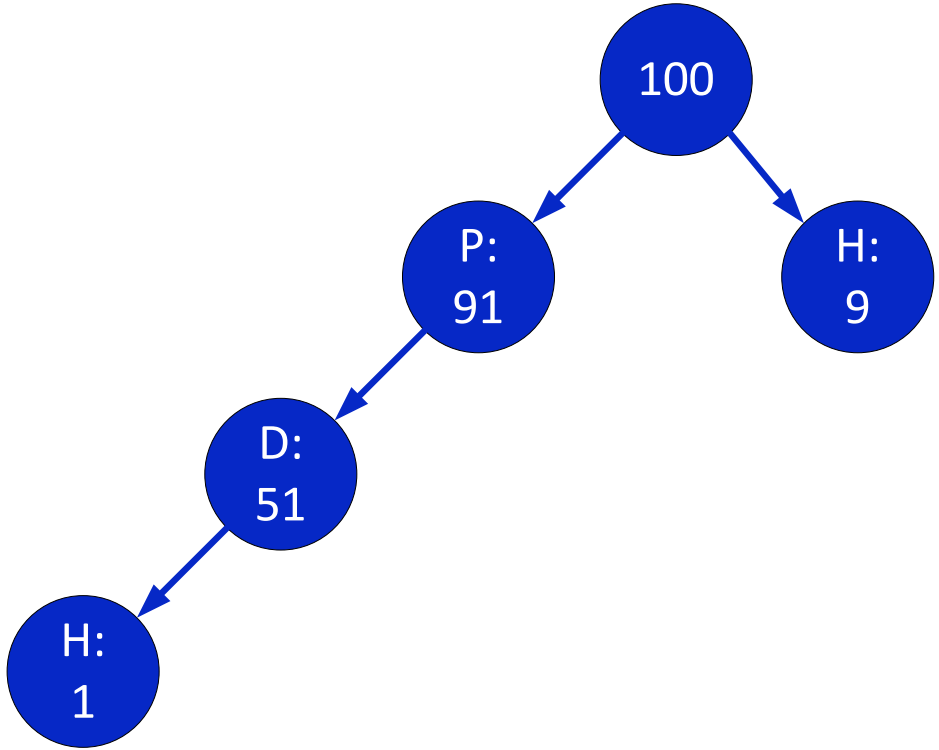
Chair of Process and Data Science

P>D>H

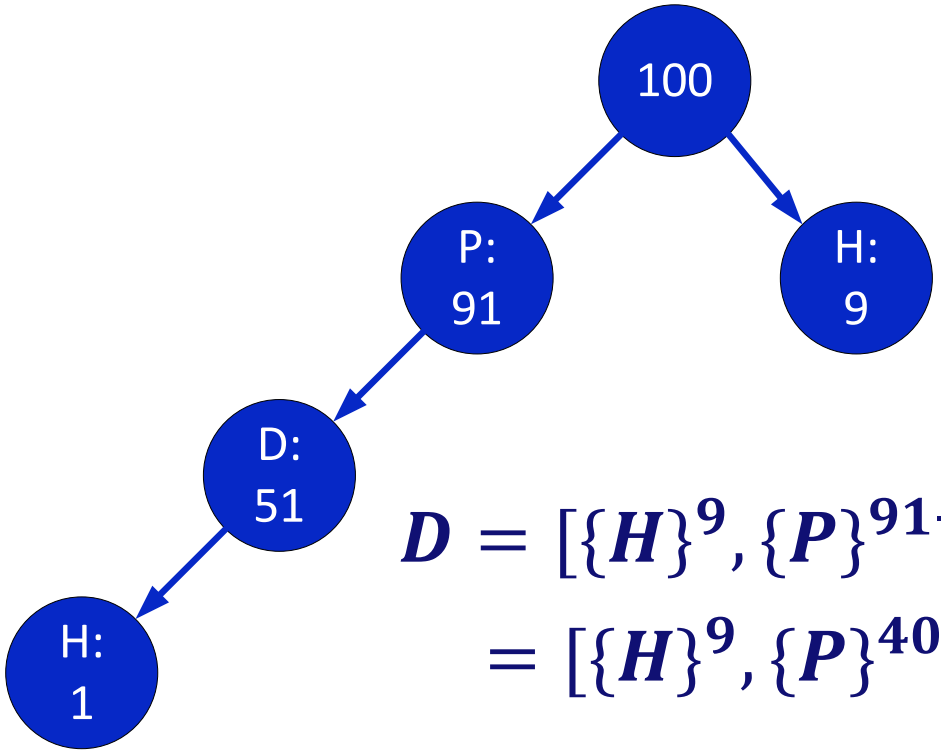


Chair of Process and Data Science

Resulting FP-tree

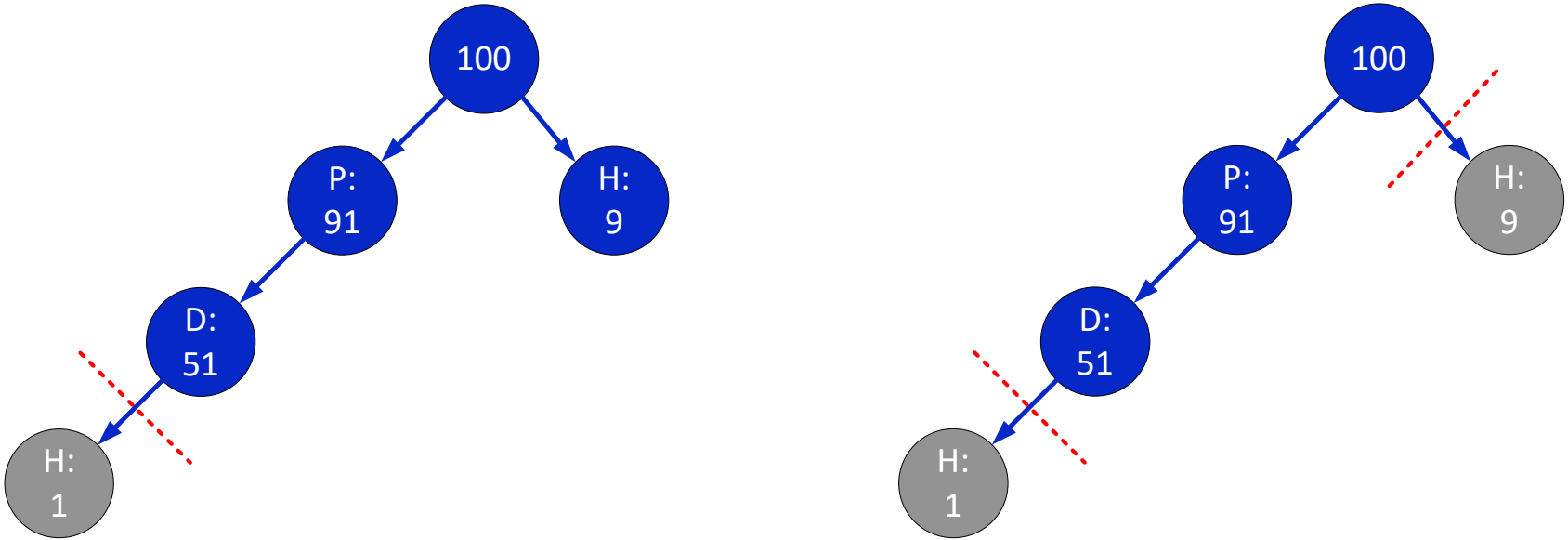


FP-tree encodes the whole data set



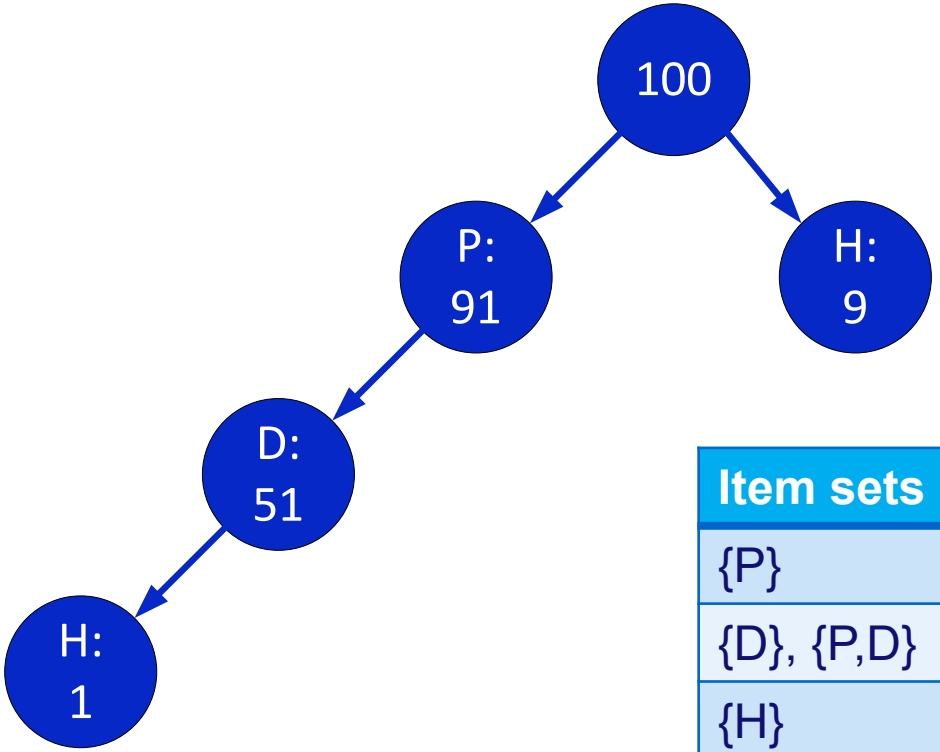
$$D = [\{H\}^{9}, \{P\}^{91-51}, \{P, D\}^{51-1}, \{P, D, H\}^{1-0}]$$
$$= [\{H\}^{9}, \{P\}^{40}, \{P, D\}^{50}, \{P, D, H\}^{1}]$$

Cannot cut FP-tree naïvely



H happens 10 times and this is above the threshold (otherwise not in tree)

Frequent item sets



Item sets	support
{P}	91
{D}, {P,D}	51
{H}	10

Step 2: Mine the FP-tree

- For each frequent item, create a conditional FP-tree (starting with the least frequent one).
- The conditional FP-tree tree considers all instances ending with this item.
- Apply this recursively.

Through the recursion we are also considering postfixes that contain multiple elements. The conditional FP-tree is as if the dataset was projected on instances that only contain the prefix (while removing items that have become infrequent). The ordering ensures that postfixes are considered only once.

Another example

facdgimp
abcflmo
bfhjo
bcksp
afcelpmn

threshold = 3

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

f:4
c:4
a:3
b:3
m:3
p:3

facdgimp
abcflmo
bfhjo
bcksp
afcelpmn

Filter and reorder based on frequency

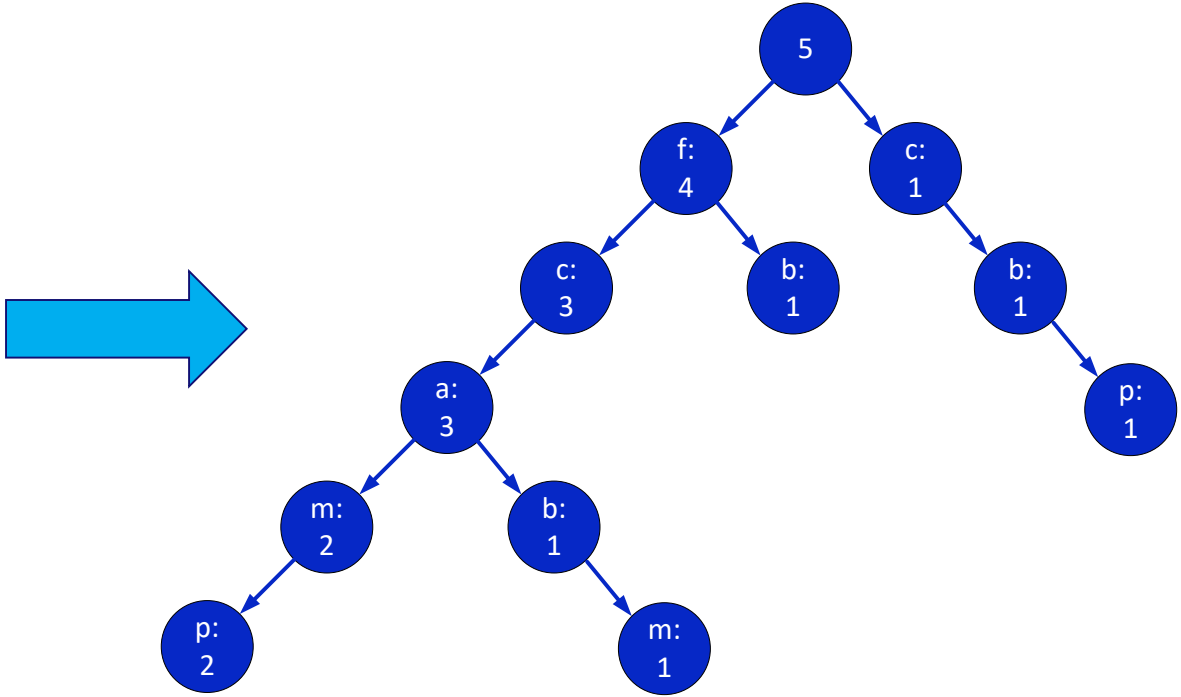
facdgimp
abcflmo
bfhjo
bcksp
afcelpmn

f:4
c:4
a:3
b:3
m:3
p:3

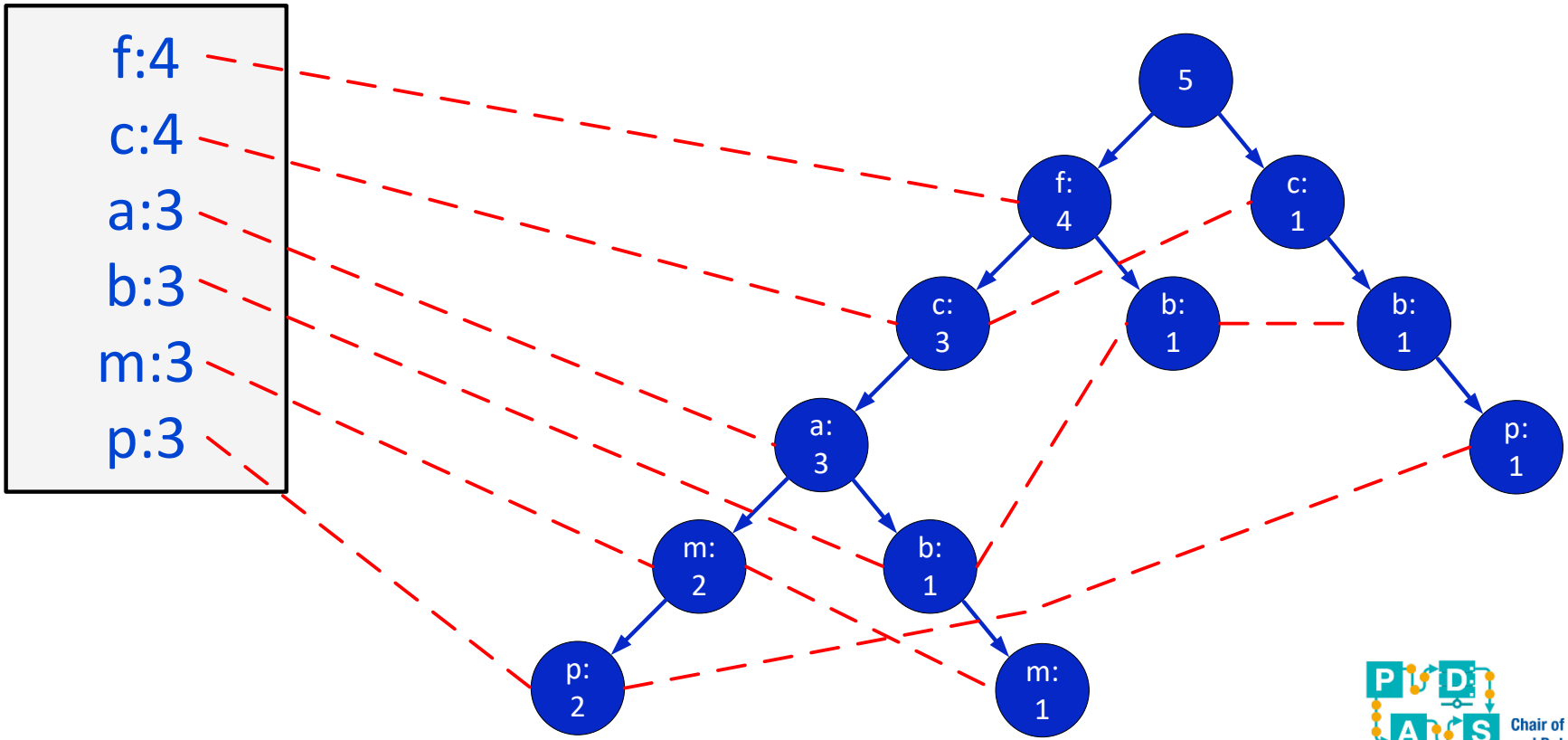
fcamp
fcabm
fb
cbp
fcamp

Build FP-tree

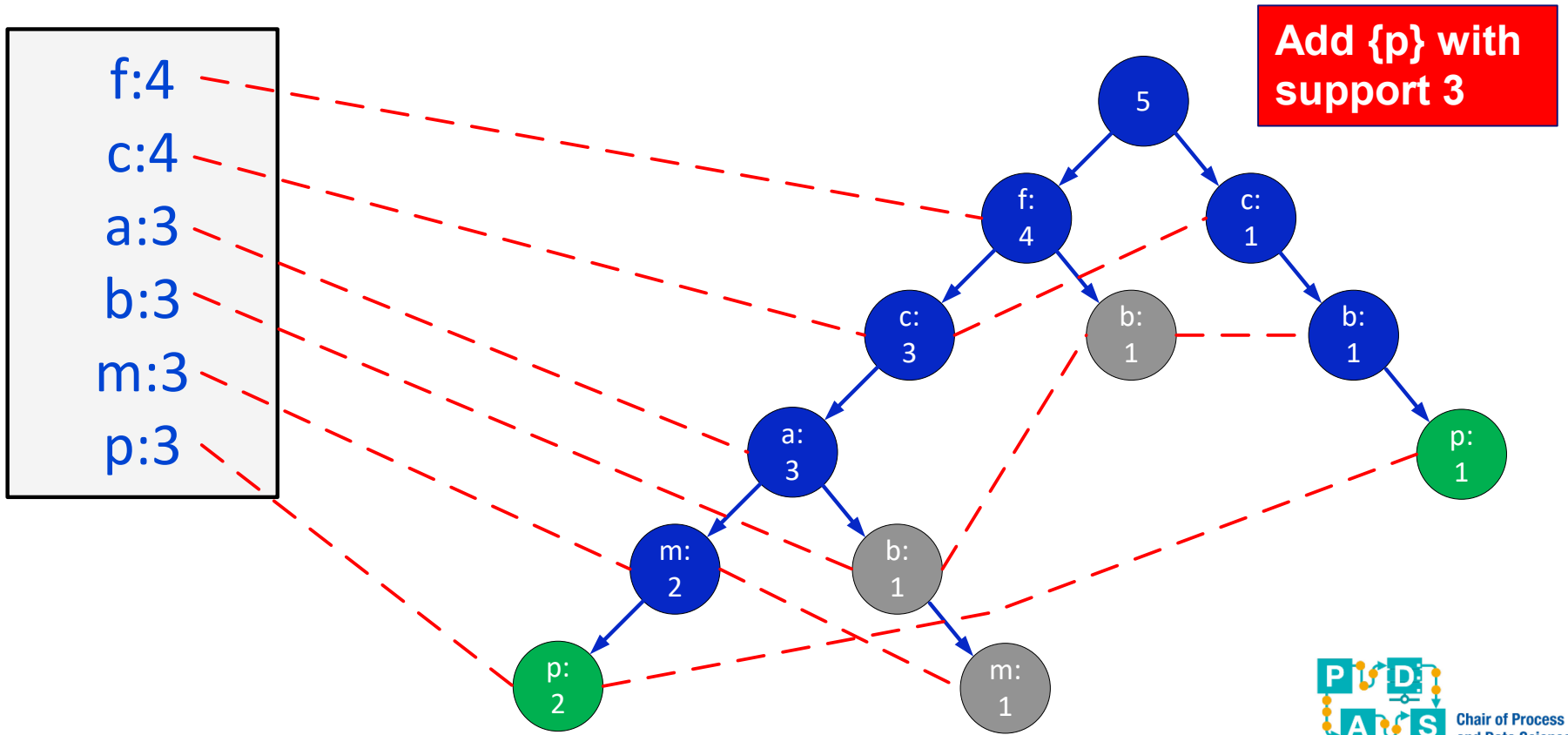
fcamp
fcabm
fb
cbp
fcamp



Node links

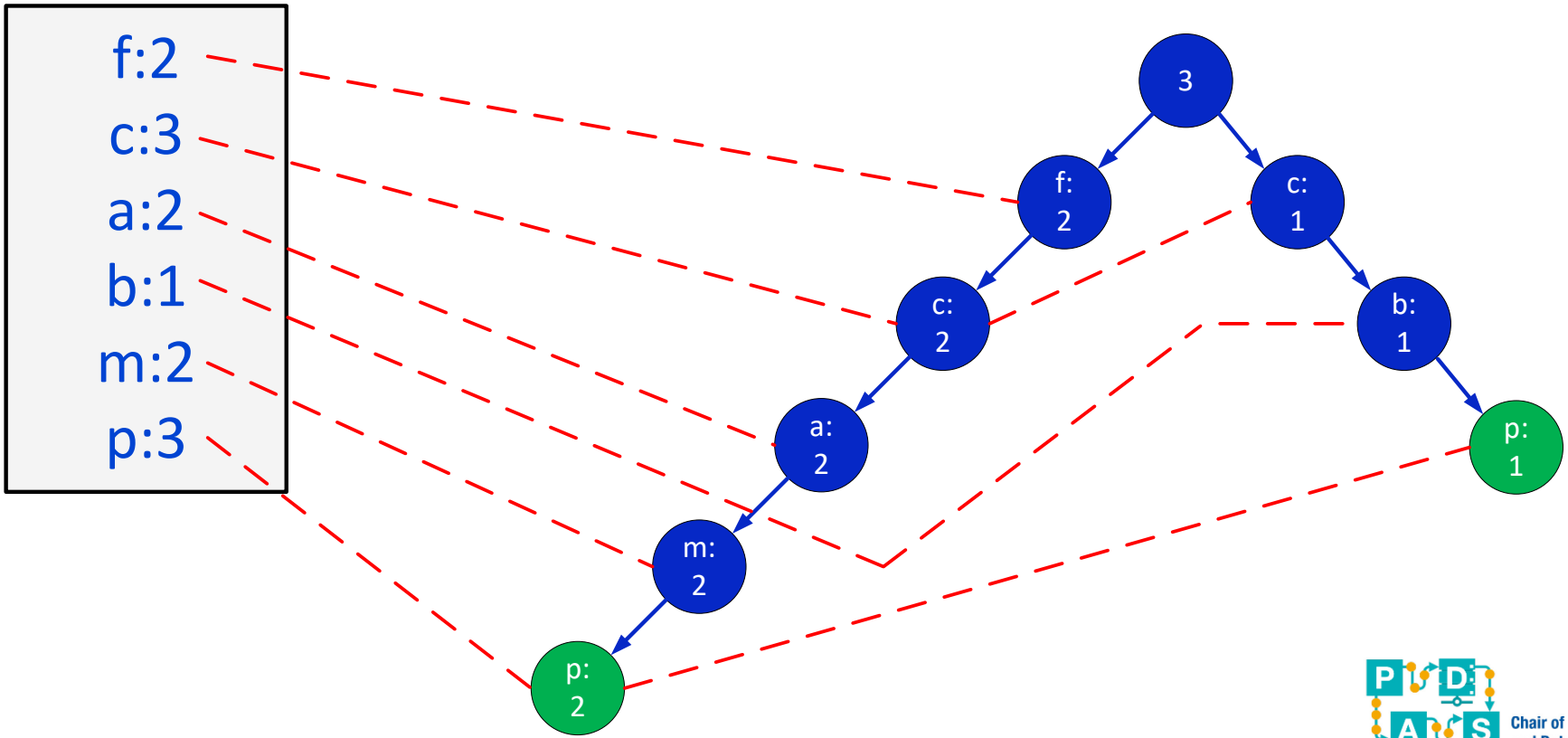


Consider postfix p



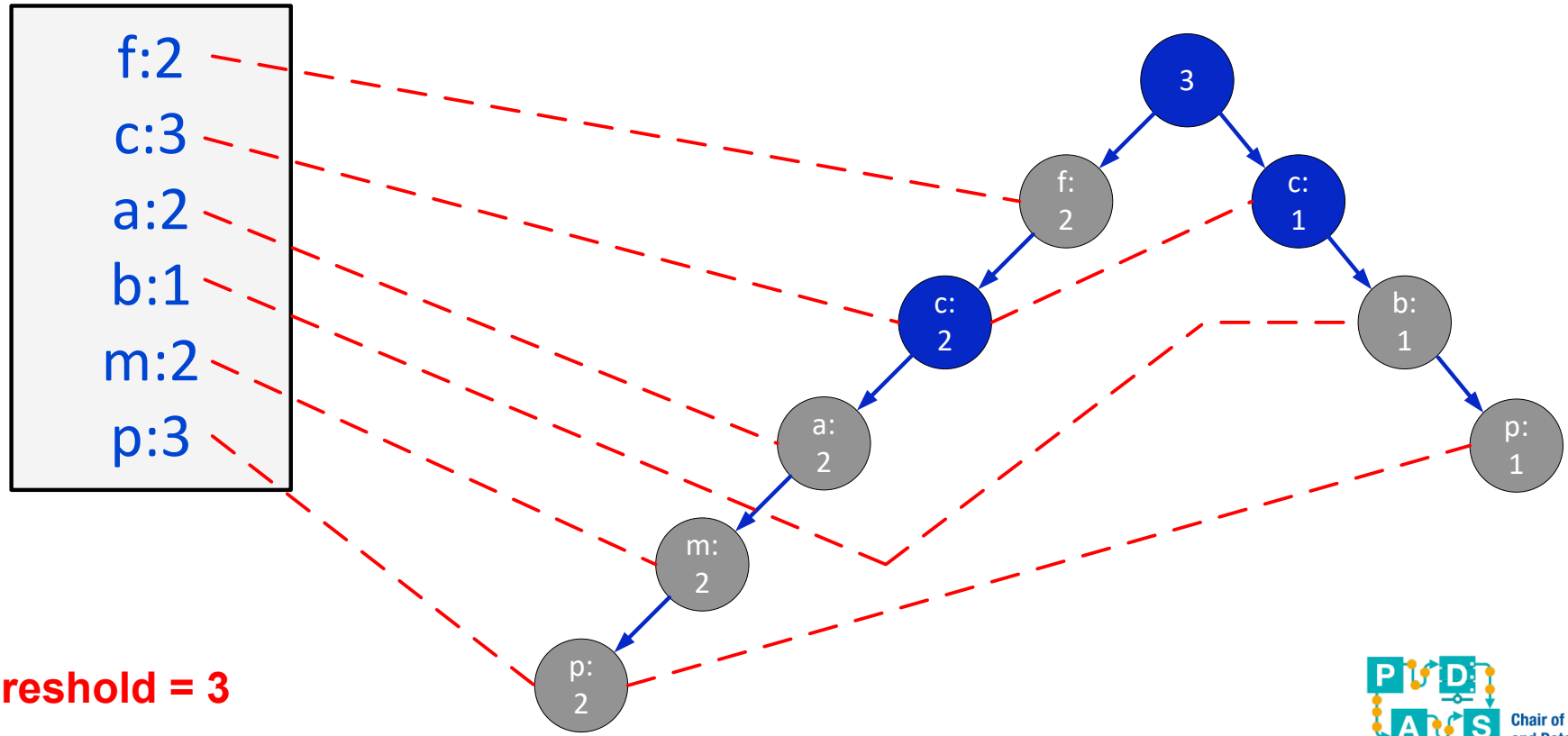
Chair of Process and Data Science

Conditional pattern-base for postfix p



Chair of Process and Data Science

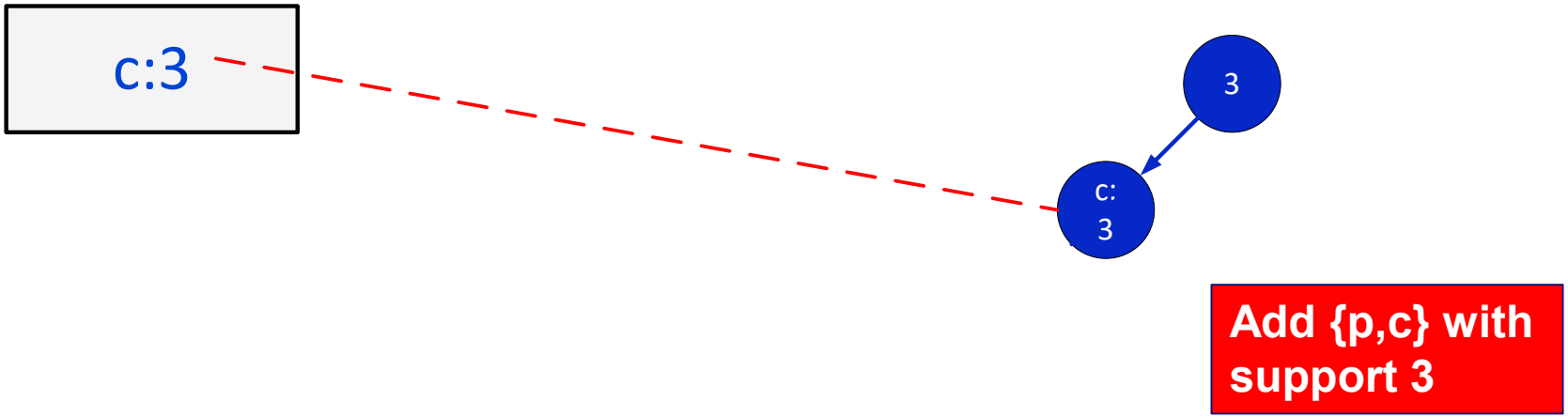
Towards the conditional FP-tree for postfix p



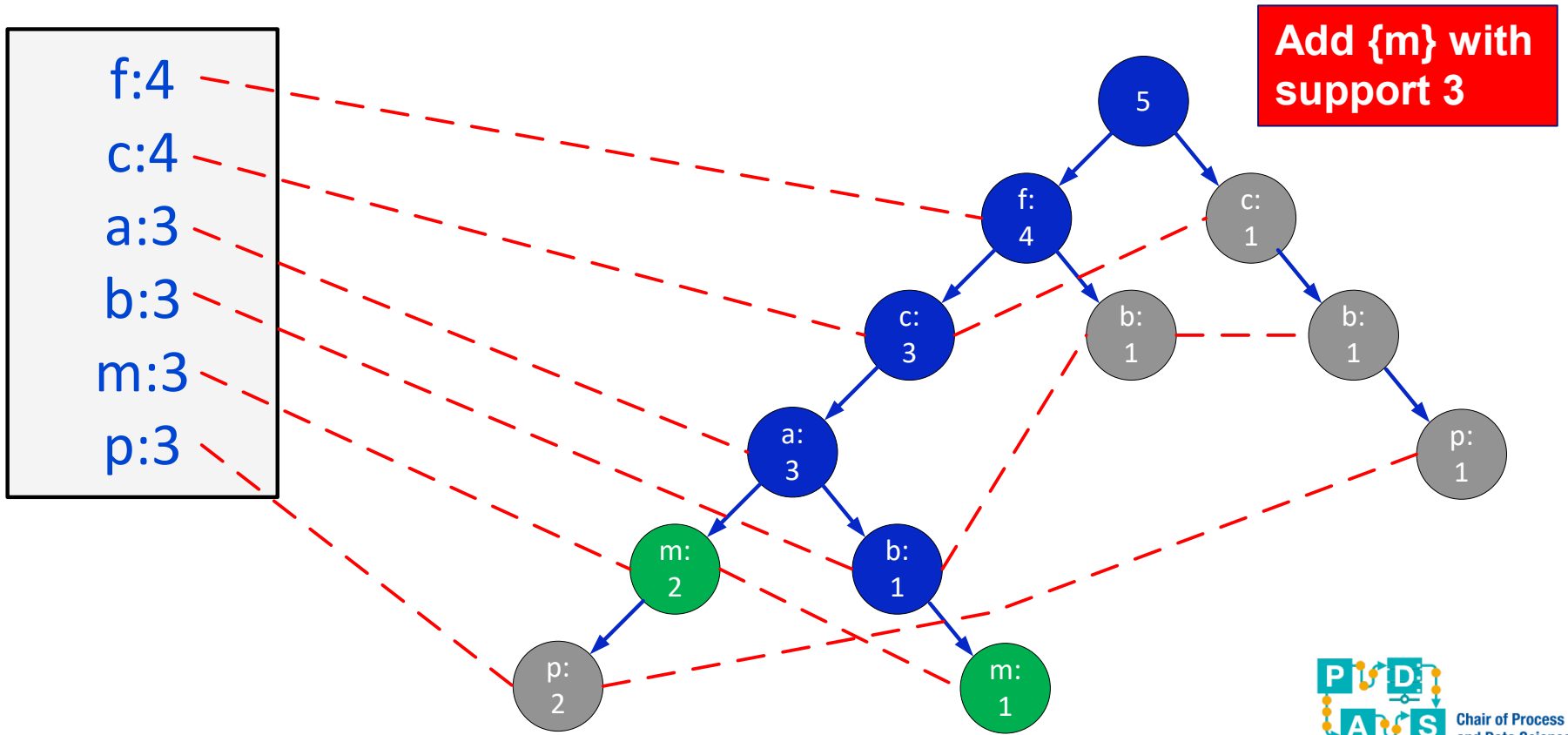
threshold = 3

Chair of Process and Data Science

Conditional FP-tree for postfix p

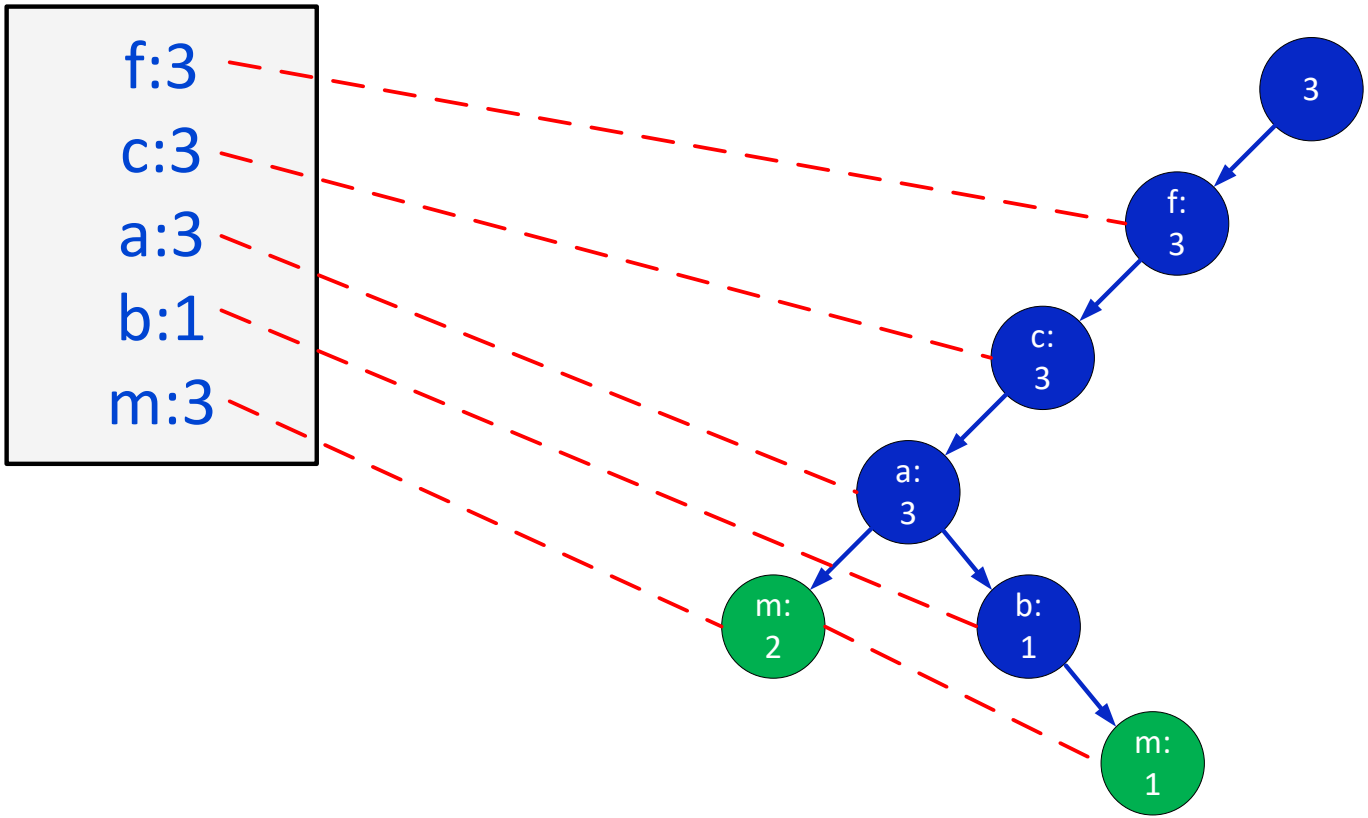


Consider postfix m



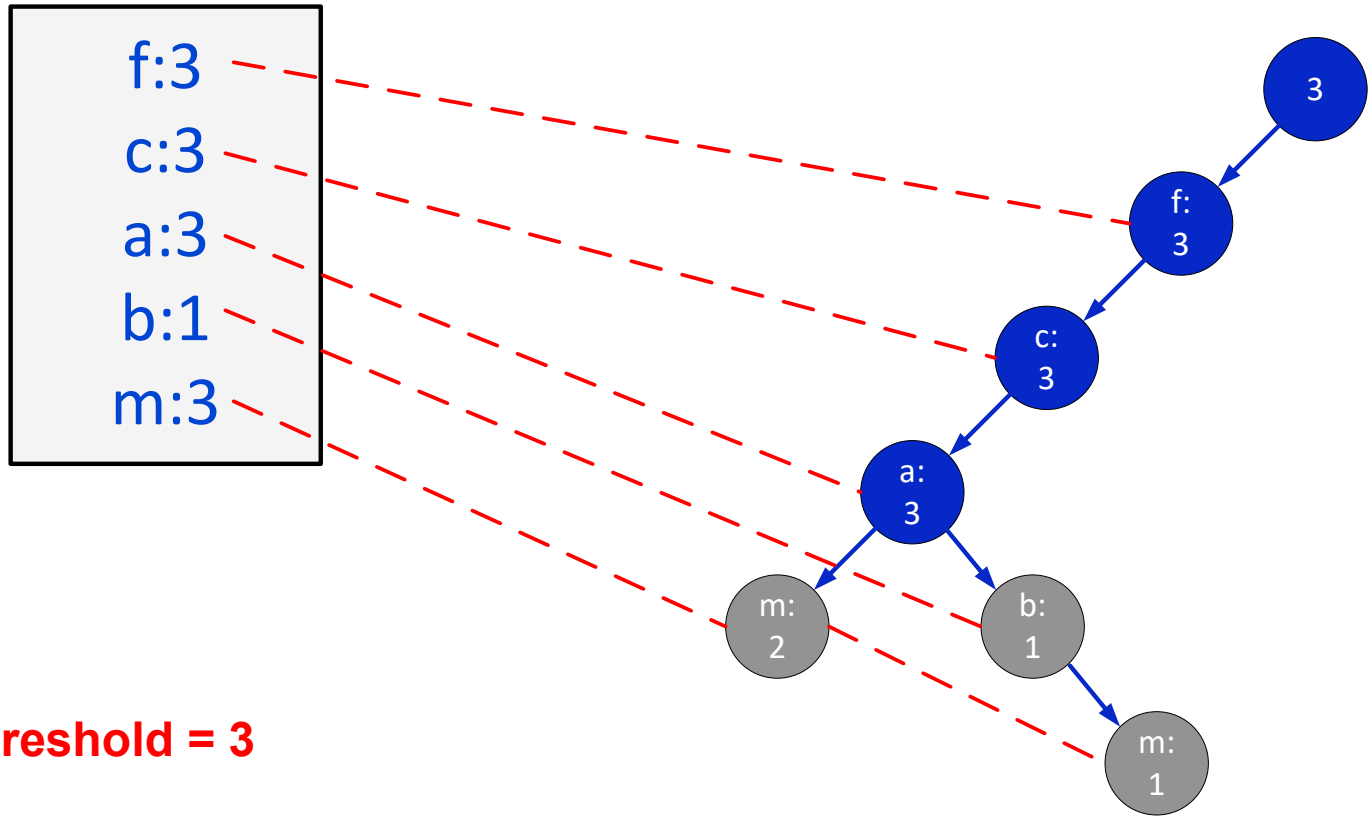
Chair of Process and Data Science

Conditional pattern-base for postfix m



Chair of Process and Data Science

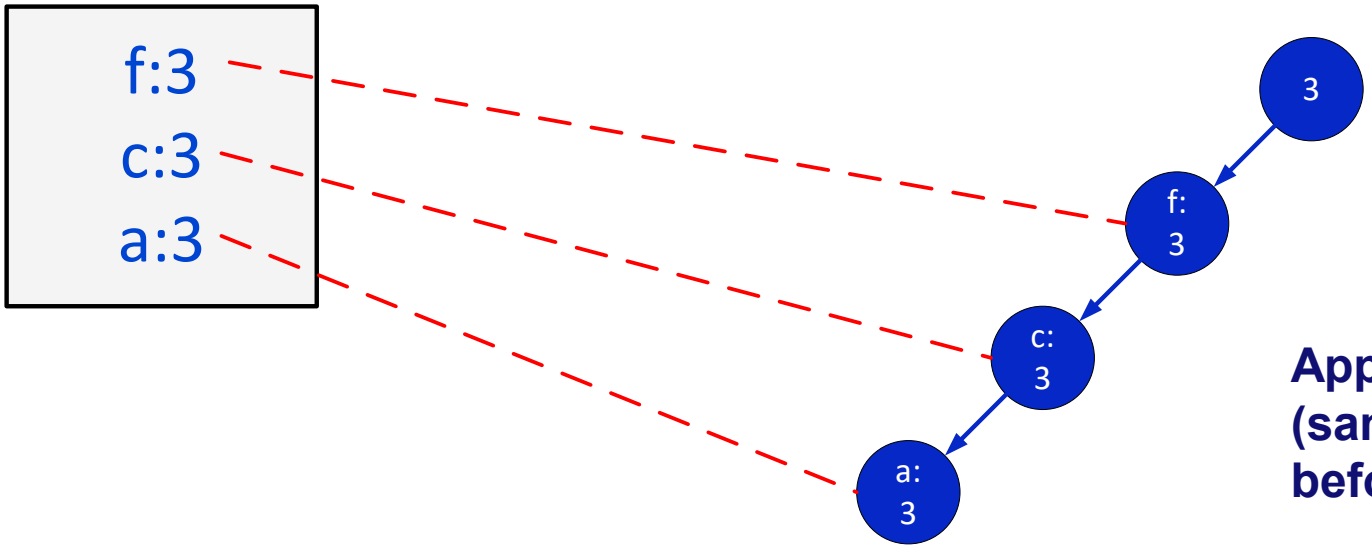
Towards a conditional FP-tree for postfix m



threshold = 3

Chair of Process and Data Science

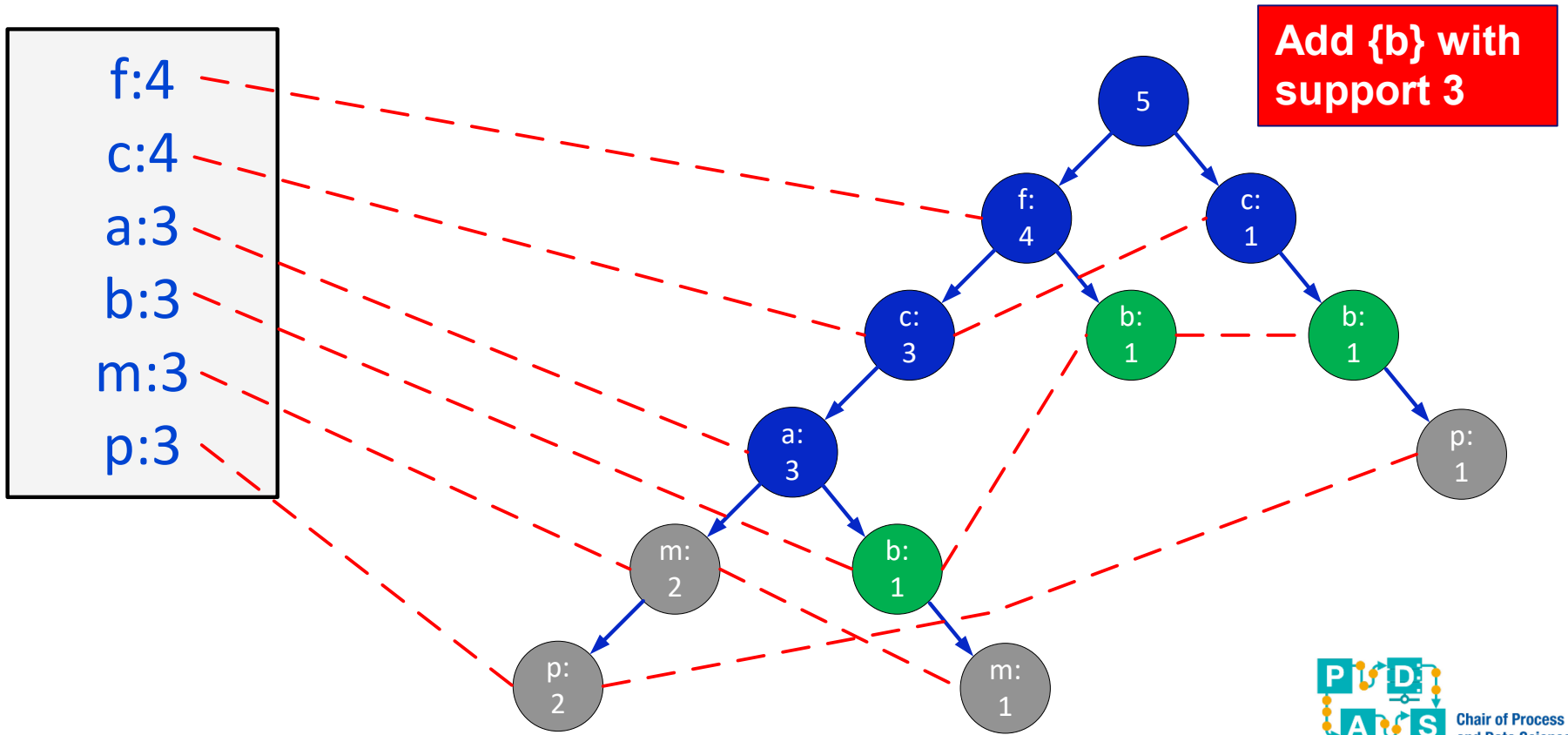
Conditional FP-tree for postfix m



Apply recursion (same setting as before).

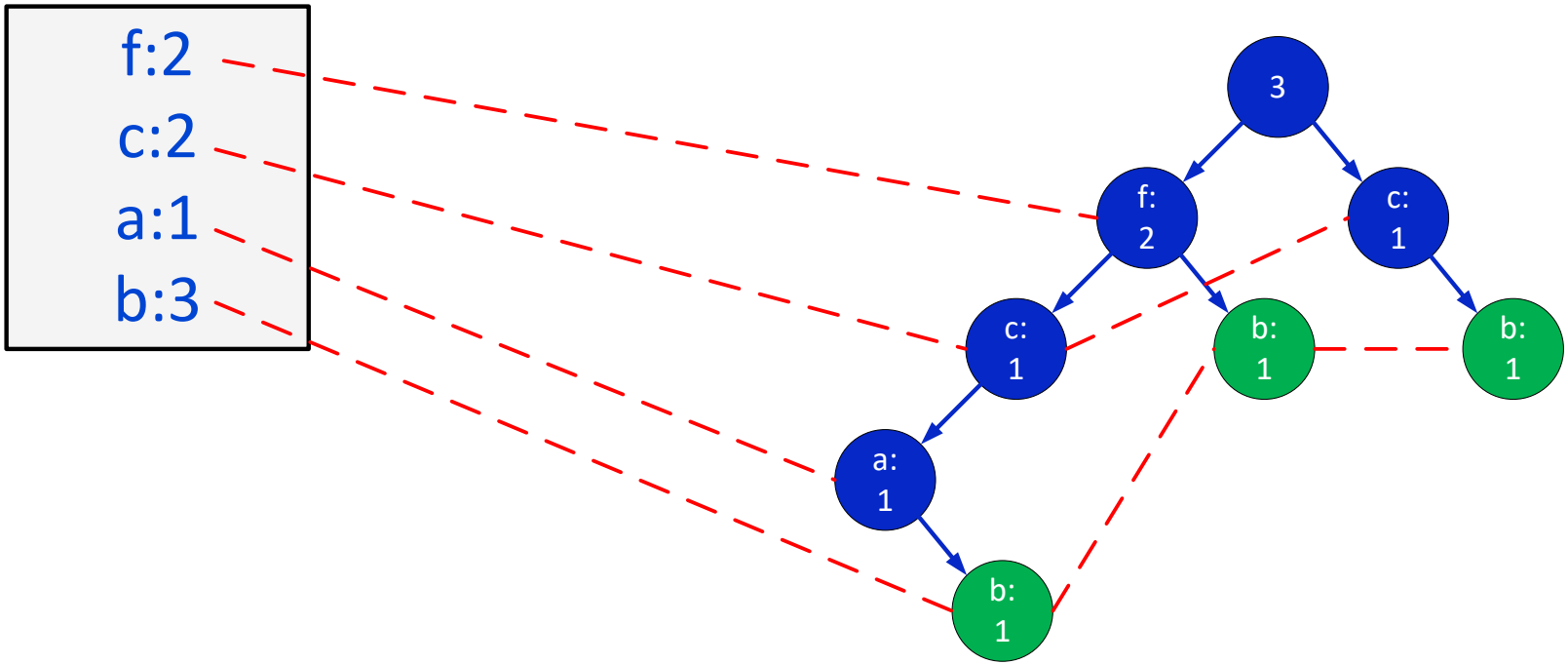
In the recussion we add: {m,a} with support 3, {m,c} with support 3, {m,f} with support 3, {m,c,f} with support 3, {m,a, f} with support 3, {m,a,c} with support 3, and {m,a,c,f} with support 3.

Consider postfix b



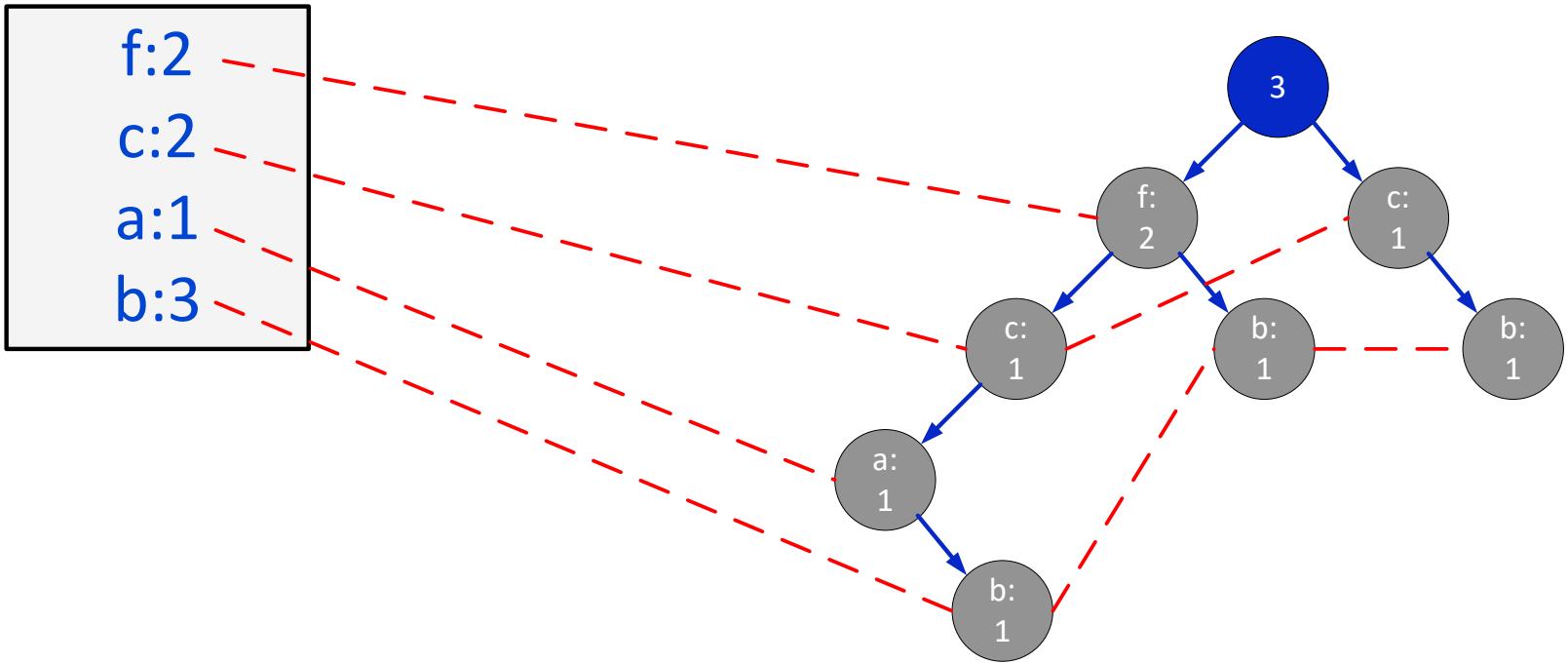
Chair of Process and Data Science

Conditional pattern-base for postfix b



Chair of Process and Data Science

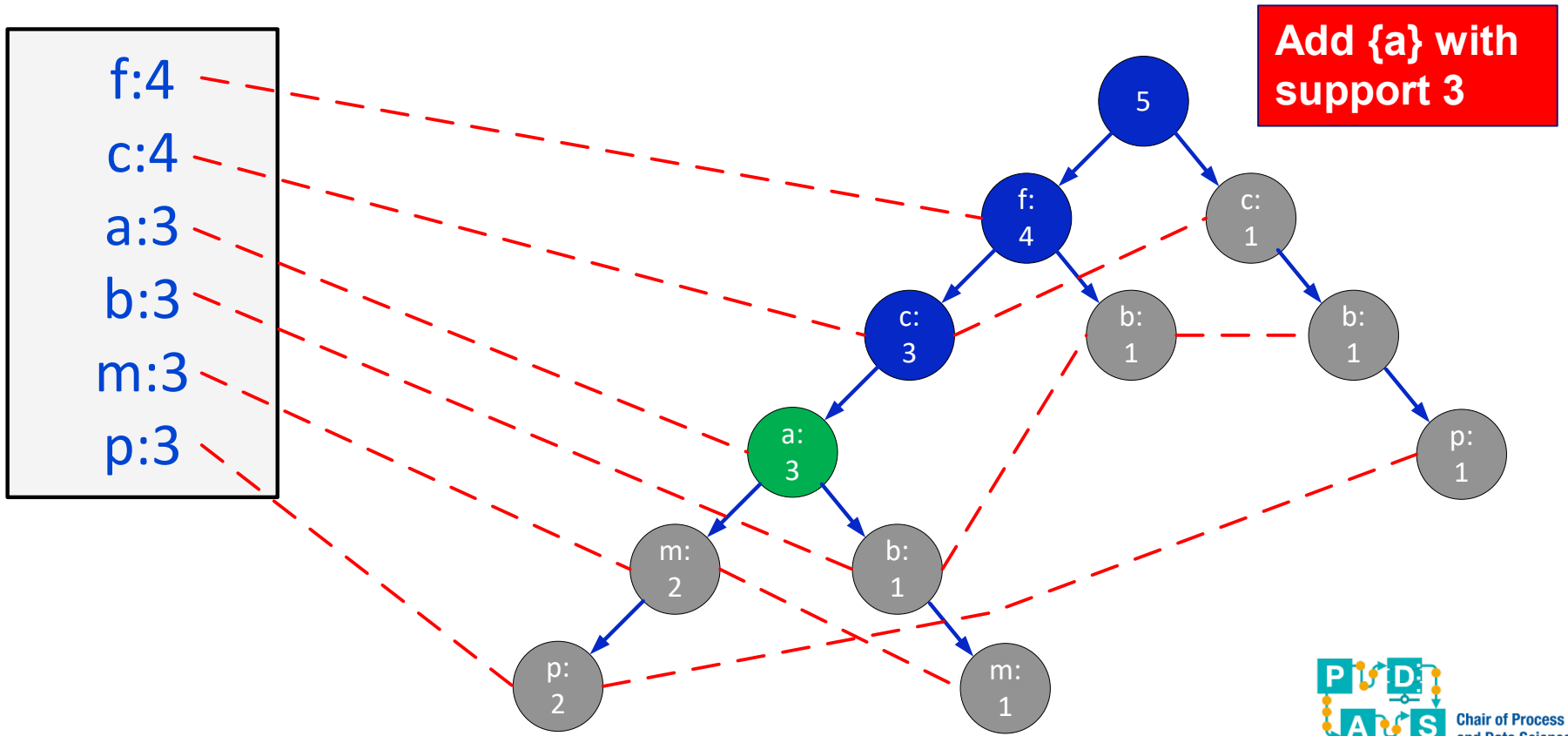
Towards a conditional FP-tree for postfix b



Conditional FP-tree for postfix b

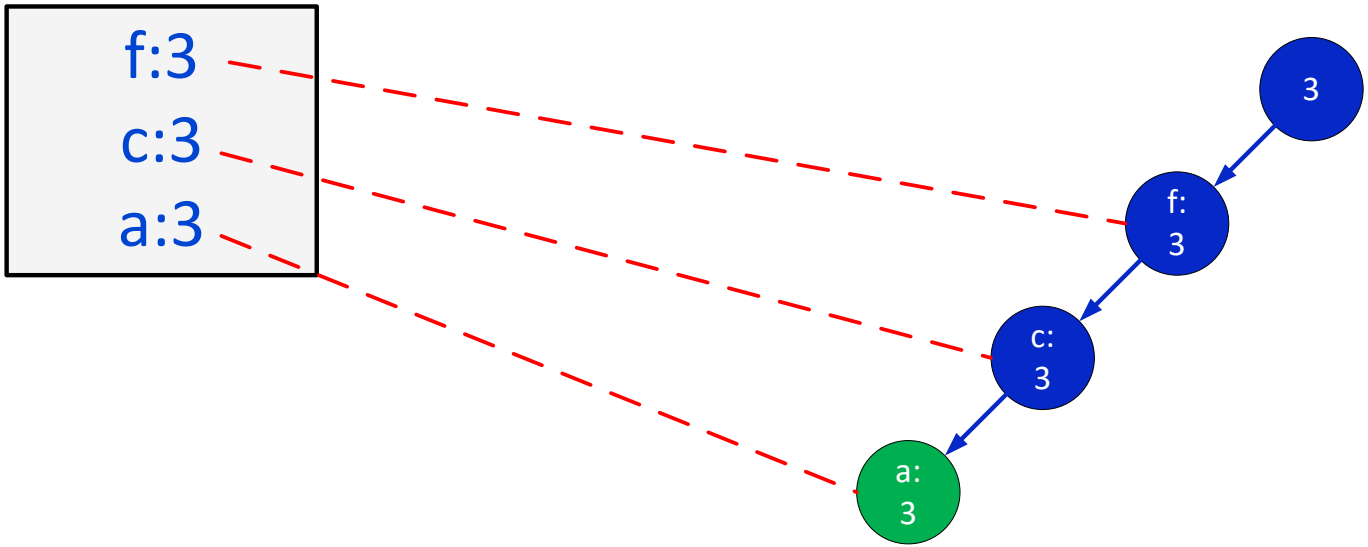
Conditional FP-tree is empty, i.e., no additional frequent itemsets next to {b}.

Consider postfix a

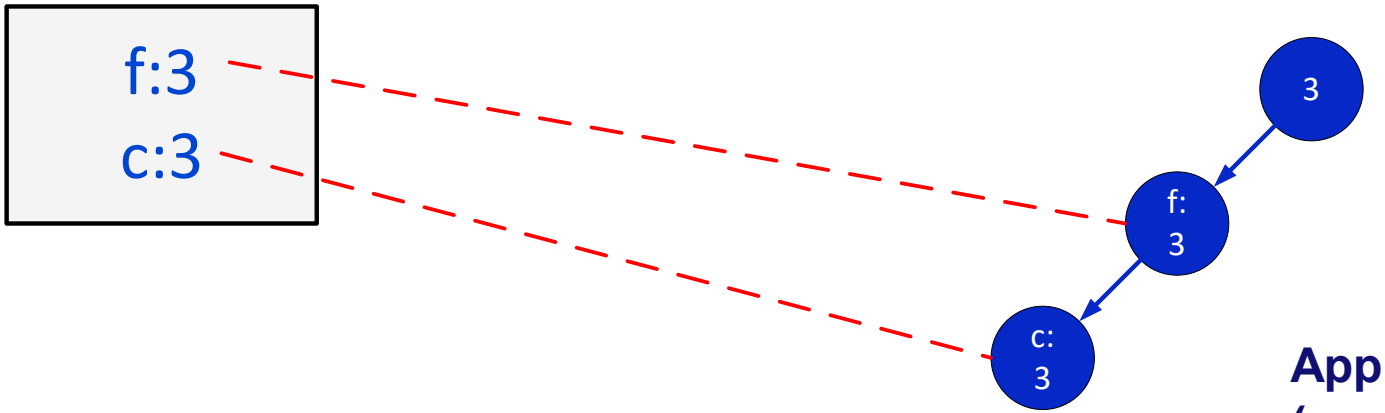


Chair of Process and Data Science

Conditional pattern-base for postfix a



Conditional FP-tree for postfix a

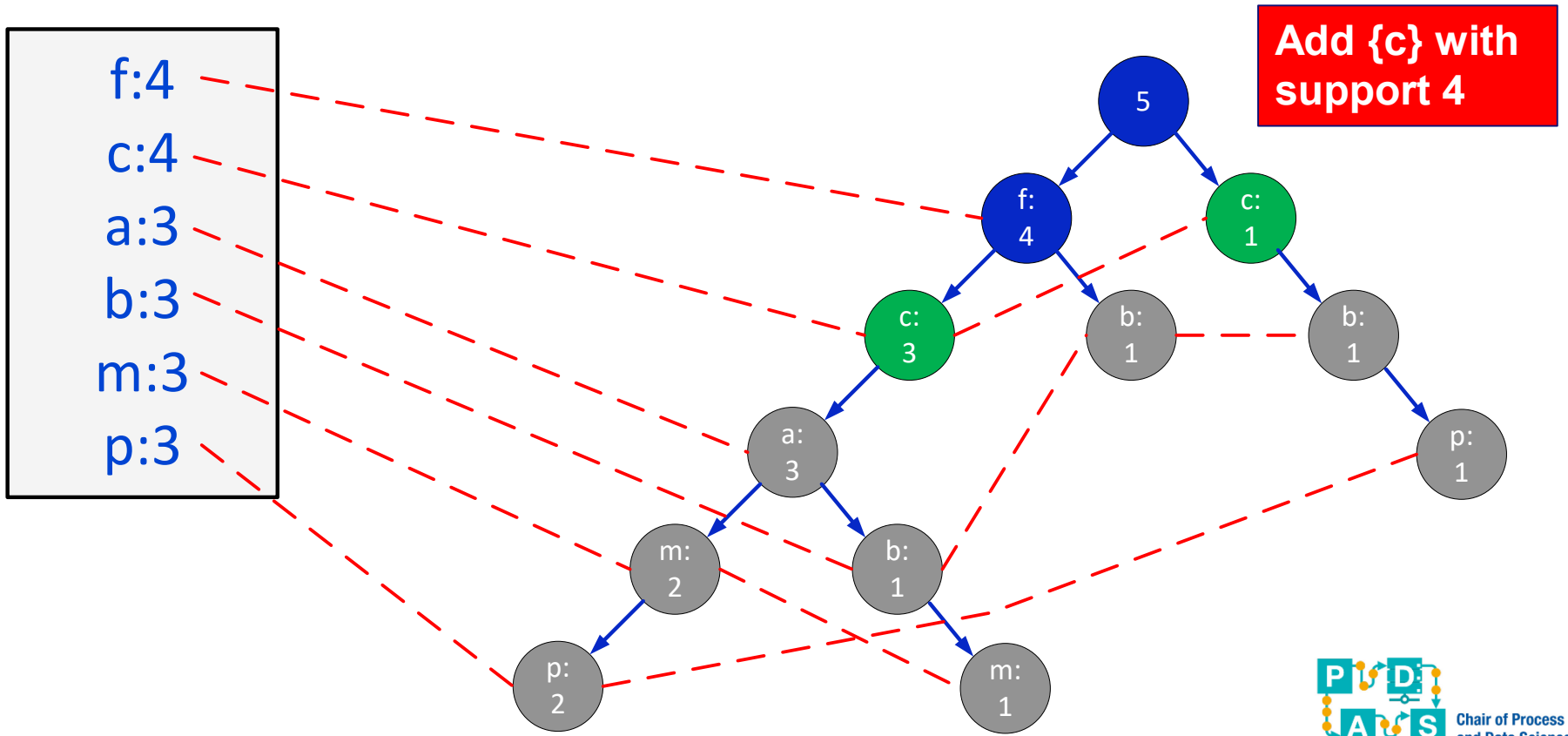


Apply recursion (same setting as before).

In the recussion we add: {a,c} with support 3, {a,f} with support 3, and {a,c,f} with support 3.

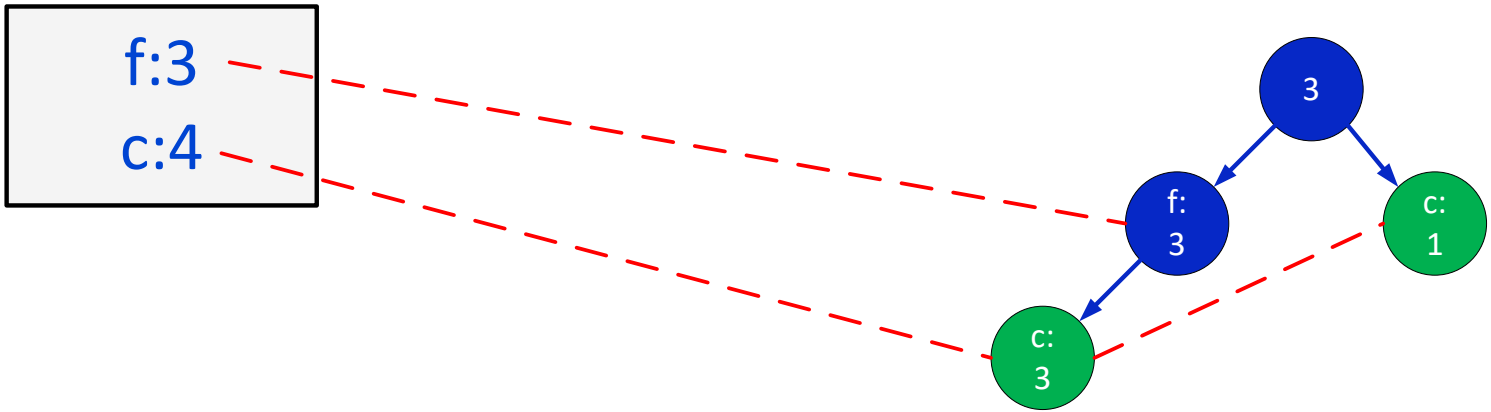
Actually, this is a special case that can be handled more efficiently.

Consider postfix c

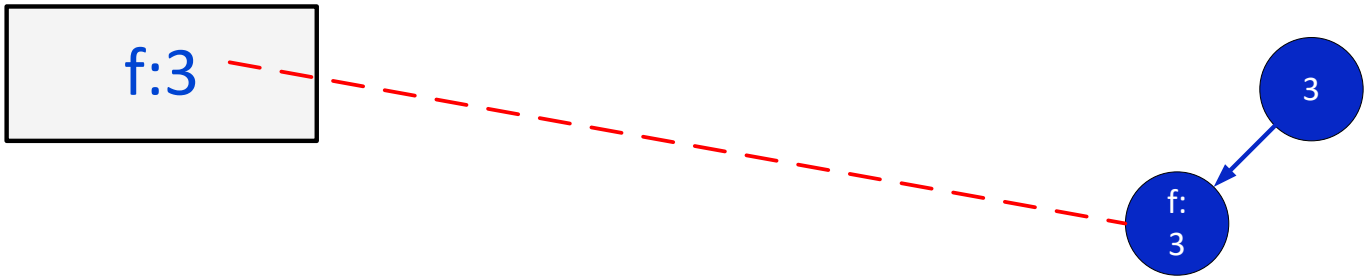


Chair of Process and Data Science

Conditional pattern-base for postfix c



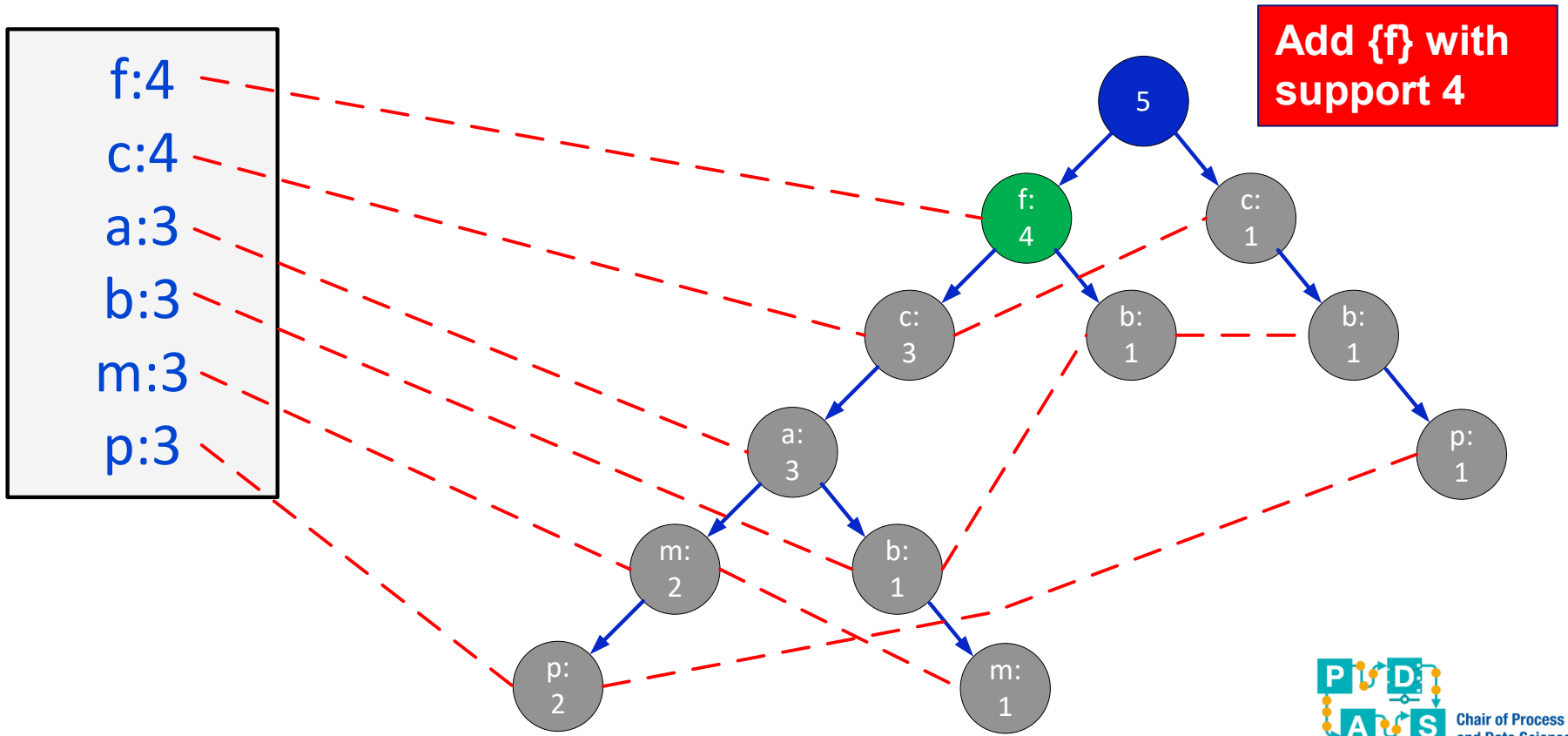
Conditional FP-tree for postfix c



Apply recursion (same setting as before).

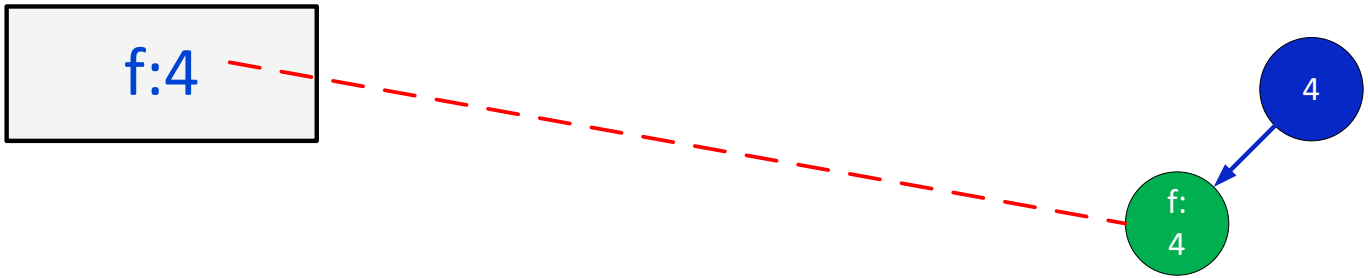
In the recussion we add: {c,f} with support 3.

Consider postfix f



Chair of Process and Data Science

Conditional pattern-base for postfix f

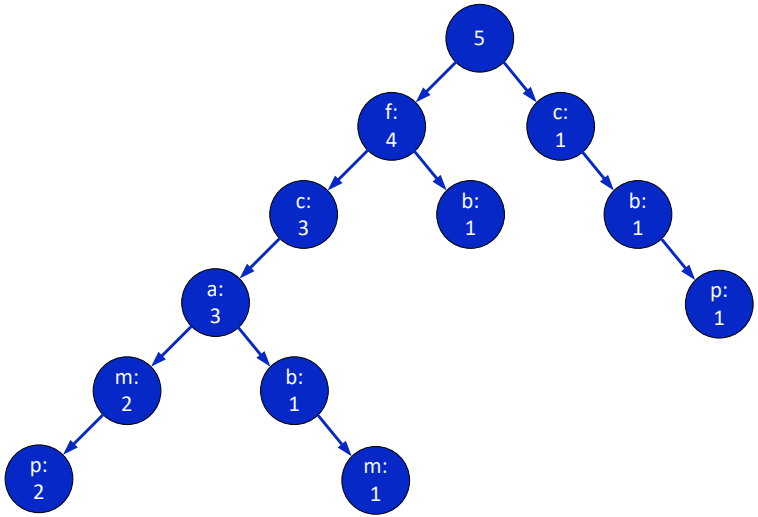


Conditional FP-tree for postfix f

4

Conditional FP-tree is empty, i.e., no additional frequent itemsets next to {f}.

All frequent item sets generated



Item sets	support
{f},{c}	4
{a}, {b}, {m}, {p}, {p,c}, {m,a,c,f}, {m,c,f}, {m,a,f}, {m,a,c}, {m,a}, {m,c}, {m,f}, {a,c,f}, {a,c}, {a,f}, {c,f}	3

FP growth: Summary

- **Idea: Frequent pattern growth based on FP-tree**
- **Method**
  - **For each frequent item, construct its conditional pattern-base, and then its conditional FP-tree**
  - **Repeat the process on each newly created conditional FP-tree until the resulting FP-tree is empty.***

*** An optimization is possible if the FP-tree contains only one path. For a single path one can generate all the combinations of its sub-paths, each of which is a frequent pattern.**

FP growth: Summary

- **Advantages:**
  - **Only two passes through the data are needed.**
  - **Avoiding testing many hopeless candidates.**
  - **Very fast when FP-tree fits in main memory.**
- **Approach has problems when FP-tree is too large to fit into memory.**
- **More important (and not specific for FP-growth): There may be many "patterns". How to determine which ones are surprising / interesting?**

Conclusion

Short summary of lecture

- Pattern mining is a form of unsupervised learning.
- Frequent itemsets are the basis for finding patterns (ideas can be transferred to other patterns).
- Two algorithms were discussed:
  - Apriori algorithm
  - FP-growth algorithm

Next: Association rules

- **{Bitburg, Pumpernickel} ⇒ {Merlot}**
  (people that buy Bitburg pilsner and Pumpernickel bread also tend to buy Merlot wine)
- **{Bitburg} ⇒ {Heineken, Palm}**
  (people that buy Bitburg pilsner tend to buy both Heineken and Palm pilsner)
- **{Carbonara, Margherita } ⇒ {Espresso, Tiramisu}**
  (people that bug Bitburg pils and Pumpernickel bread, also tend to buy Merlot wine)
- **{BPI, IDS} ⇒ {APM}**
  (students that take the BPI and IDS courses also tend to take the APM course)
- **{part-245, part-345, part-456} ⇒ {part-372}**
  (when parts 245, 345, and 456 are replaced, then often also part 372 is replaced)

Key challenge: What rules are interesting?

#	Lecture	date	day
Lecture 1	Introduction	10/10/2018	Wednesday
Lecture 2	Crash Course in Python	11/10/2018	Thursday
Instruction 1	*Python*	*12/10/2018*	*Friday*
Lecture 3	Basic data visualisation/exploration	17/10/2018	Wednesday
Lecture 4	Decision trees	18/10/2018	Thursday
Instruction 2	*Decision trees and data visualization/exploration*	*19/10/2018*	*Friday*
Lecture 5	Regression	24/10/2018	Wednesday
Lecture 6	Support vector machines	25/10/2018	Thursday
Instruction 3	*Regression and support vector machines*	*26/10/2018*	*Friday*
Lecture 7	Neural networks (1/2)	31/10/2018	Wednesday
Instruction 4	*Neural networks and supervised learning*	*02/11/2018*	*Friday*
Lecture 8	Neural networks (2/2)	07/11/2018	Wednesday
Lecture 9	Evaluation of supervised learning problems	08/11/2018	Thursday
Instruction 5	*Neural networks and supervised learning*	*09/11/2018*	*Friday*
Lecture 10	Clustering	14/11/2018	Wednesday
Lecture 11	Frequent items sets	15/11/2018	Thursday
Lecture 12	Association rules	21/11/2018	Wednesday
Lecture 13	Sequence mining	22/11/2018	Thursday
Instruction 6	*Clustering, frequent items sets, association rules*	*23/11/2018*	*Friday*
Lecture 14	Process mining (unsupervised)	28/11/2018	Wednesday
Lecture 15	Process mining (supervised)	29/11/2018	Thursday
Instruction 7	*Process mining and sequence mining*	*30/11/2018*	*Friday*
backup		31/01/2019	Thursday
extra	*Question hour*	*01/02/2019*	*Friday*