Pattern Mining

Pattern mining consists of using/developing data mining algorithms to discover *interesting*, unexpected and useful *patterns* in databases.

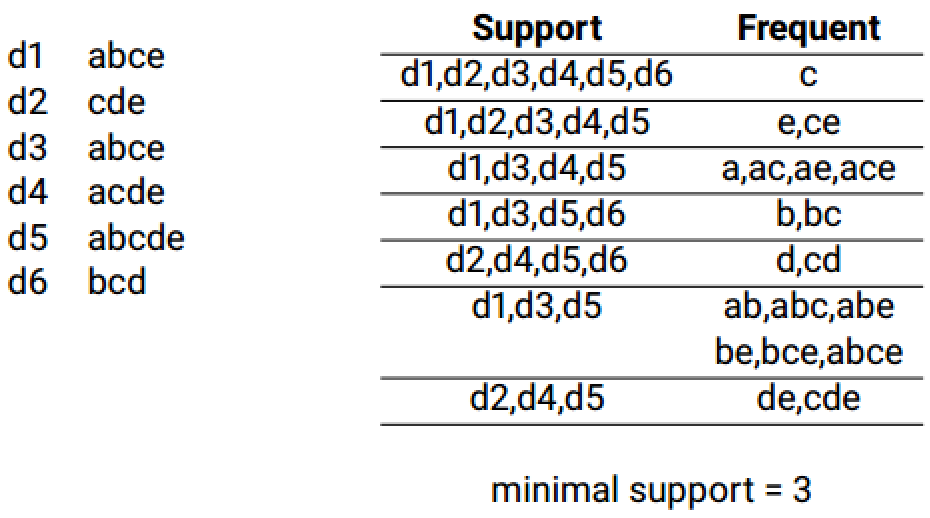
Pattern mining algorithms can be applied on various types of data such as transaction databases, sequence databases, streams, strings, spatial data, graphs, etc. These algorithms can be designed to discover various types of patterns: subgraphs, associations, indirect associations, trends, periodic patterns, sequential rules, lattices, sequential patterns, high-utility patterns, etc.

There are different definition for *interesting pattern*, from **frequent patterns** to **high confidence patterns**. We will focus on the first ones.

Frequent patterns

Suppose having *D* as a dataset of patterns (a series of transactions *t* for example), and *min_sup* is a constant (minimum support).

THe **support** *support(t)* is the number of patterns in *D* that are superpatterns of *t*. A pattern *t* is **frequent** if **support(t) >= min_sup**.



d1	abce
d2	cde
d3	abce
d4	acde
d5	abcde
d6	bcd

Support	Frequent
d1,d2,d3,d4,d5,d6	c
d1,d2,d3,d4,d5	e,ce
d1,d3,d4,d5	a,ac,ae,ace
d1,d3,d5,d6	b,bc
d2,d4,d5,d6	d,cd
d1,d3,d5	ab,abc,abe be,bce,abce
d2,d4,d5	de,cde

minimal support = 3

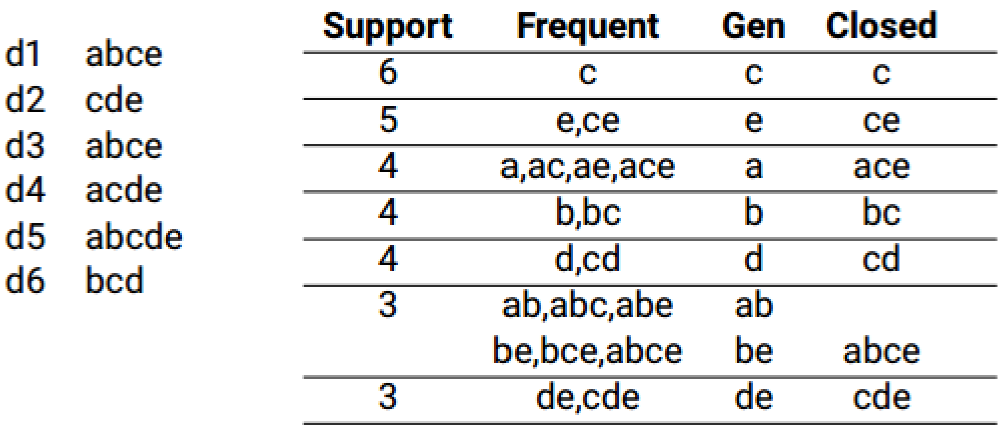
Let's suppose having the dataset of transaction on the left in the image. Finding the support of the patterns in this dataset means looking for every combination of letters and count the number of appearances of said combination. On the right in the cicture, only frequent patterns are reported, which means only those appearing three or more times.

Usually, there are too many frequent patterns. We can compute a smaller set, while keeping the same information. We can use **closed** frequent patterns:

> A frequent pattern *t* is **closed** if none of its superpatterns have the same support as it has. Frequent subpatterns and their supports can be generated from closed patterns.

The definition of closed frequent patterns is based on the **A priori property**:

> If *t'* is a subpattern of *t*, then *Support(t')>=Support(t)*

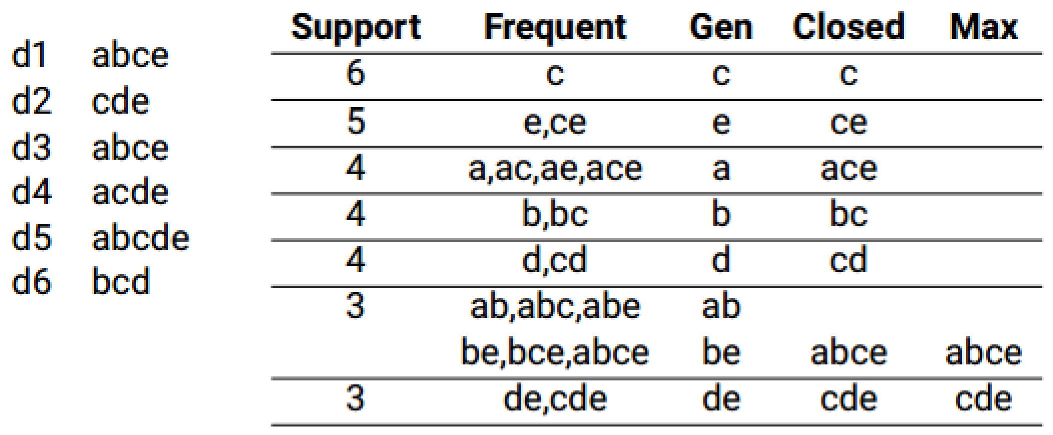


d1	abce
d2	cde
d3	abce
d4	acde
d5	abcde
d6	bcd

Support	Frequent	Gen	Closed
6	c	c	c
5	e,ce	e	ce
4	a,ac,ae,ace	a	ace
4	b,bc	b	bc
4	d,cd	d	cd
3	ab,abc,abe	ab	
	be,bce,abce	be	abce
3	de,cde	de	cde

It is possible to go one step further, and define **maximal frequent pattern**.

> A frequent pattern t is maximal if none of its proper superpatterns is frequent. Frequent subpatterns can be generated from maximal patterns, but not with their support.



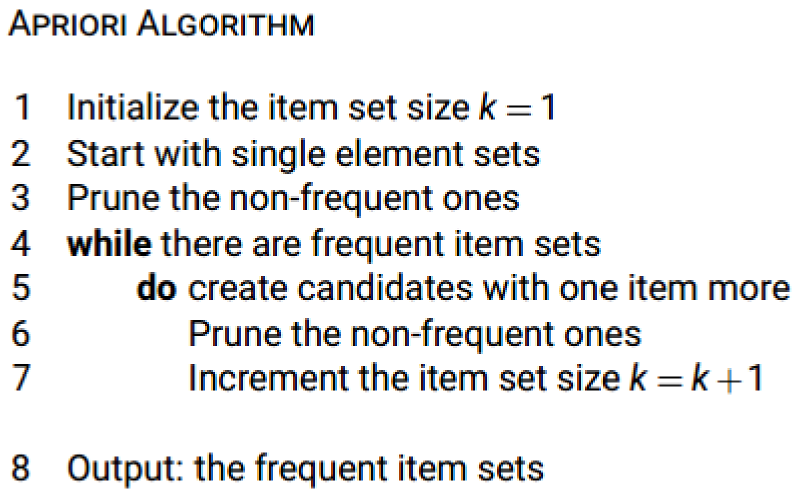
d1	abce
d2	cde
d3	abce
d4	acde
d5	abcde
d6	bcd

Support	Frequent	Gen	Closed	Max
6	c	c	c	
5	e,ce	e	ce	
4	a,ac,ae,ace	a	ace	
4	b,bc	b	bc	
4	d,cd	d	cd	
3	ab,abc,abe	ab		
	be,bce,abce	be	abce	abce
3	de,cde	de	cde	cde

Algorithms

The algorithms which solve the frequent itemset mining problem can be grouped into two categories, *breadth-first (levelwise)*, like **Apriori** algorithm, and *depth-first*, like **Eclat** or **FP-Growth** algorithms.

Apriori

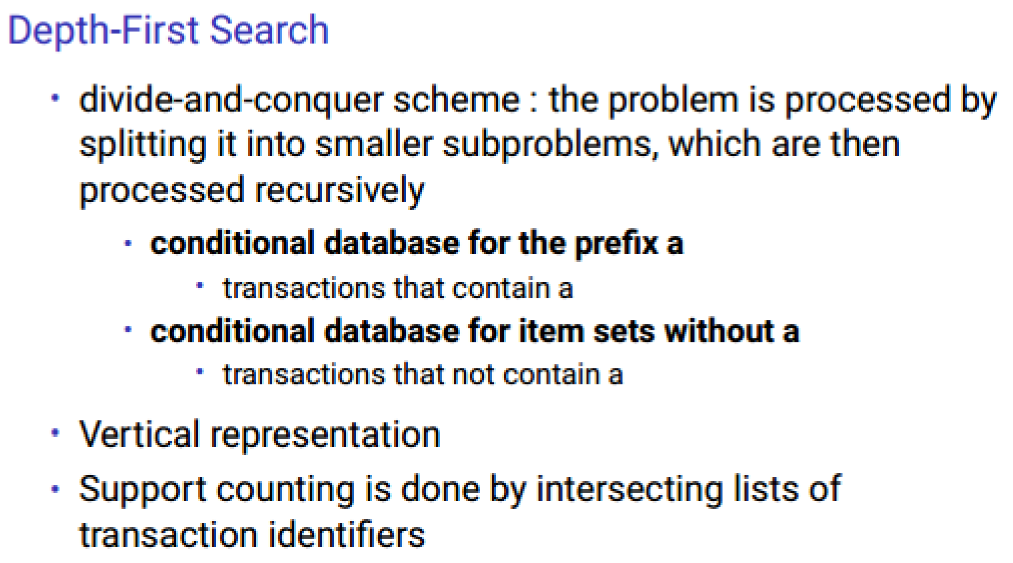


```
APRIORI ALGORITHM

1  Initialize the item set size $k = 1$
2  Start with single element sets
3  Prune the non-frequent ones
4  while there are frequent item sets
5        do create candidates with one item more
6           Prune the non-frequent ones
7           Increment the item set size $k = k + 1$

8  Output: the frequent item sets
```

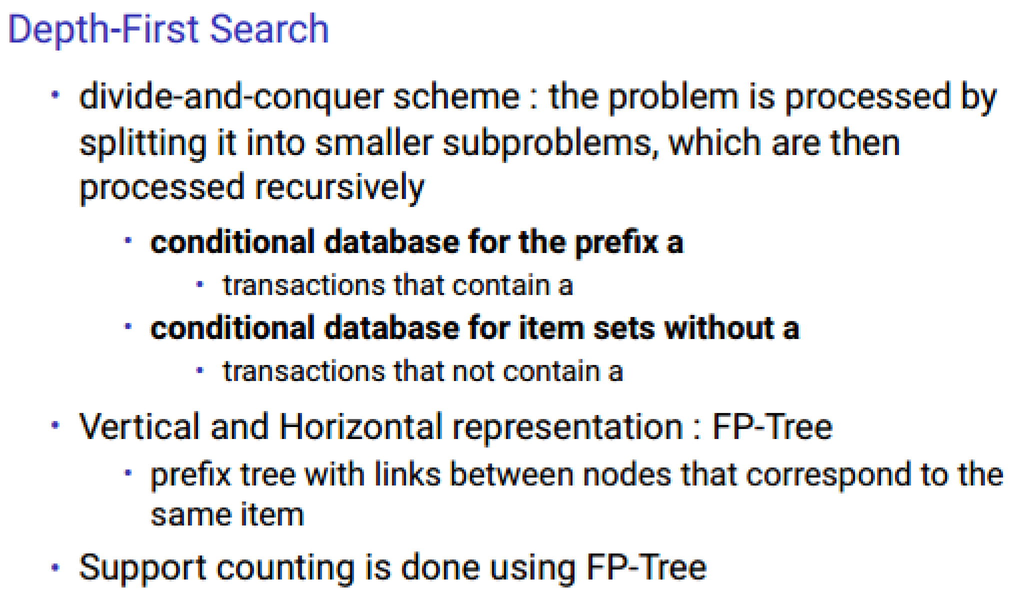
Eclat



Depth-First Search

- divide-and-conquer scheme : the problem is processed by splitting it into smaller subproblems, which are then processed recursively
  - **conditional database for the prefix a**
    - transactions that contain a
  - **conditional database for item sets without a**
    - transactions that not contain a
- Vertical representation
- Support counting is done by intersecting lists of transaction identifiers

FP-Growth



Depth-First Search

- divide-and-conquer scheme : the problem is processed by splitting it into smaller subproblems, which are then processed recursively
  - **conditional database for the prefix a**
    - transactions that contain a
  - **conditional database for item sets without a**
    - transactions that not contain a
- Vertical and Horizontal representation : FP-Tree
  - prefix tree with links between nodes that correspond to the same item
- Support counting is done using FP-Tree