Building a Direct Follower Matrix

The following exercise will guide you through a fundamental data-extraction problem in Process Mining; the extraction of all *direct follower relations* that are contained in a log.

First, here are some (very) simplistic definitions used in this task, which hopefully do not assume any further knowledge in the field, but will come in handy:

- An **Event:** One data point in the world of Process Mining. It represents the digital footprint of one unit of work, also called activity (i.e. Receive user application, send invoice, process order, ship parcel etc…), that was done and logged in a given process. It contains at least one time-stamp, the name of the activity which produced the event, and a trace identifier, often referred to as a case ID. Extending these basic data are attributes which can be either numeric or nominal and contain any additional information needed.
- A **Trace**: Consists of multiple events which share the same case ID and thus belong to one instance of a process, i.e. an insurance claim which is processed in multiple steps at an insurance company. The events are ordered - most often by their timestamp - to form a sequence of events.
- A **Log:** Is comprised of a number of *traces* and their events. The form the log is stored in varies widely between use-cases. They can reside in (distributed) databases of various forms and sizes as well as files of differing type. Most common are relational databases and CSV files.

Get the code and submit

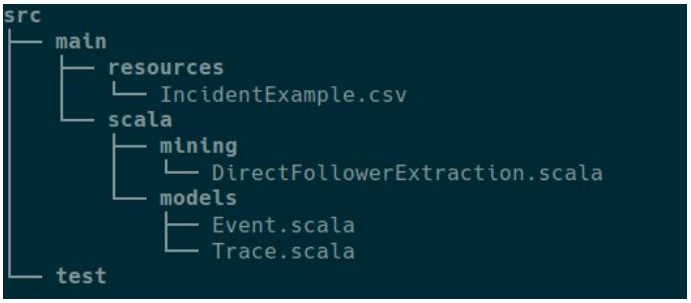
You can check out the baseline code for your exercise from https://github.com/LanaLabs/follower-matrix.git. You will not be allowed to push to this repository. You still can use local commits, if you like to. Please provide your final results as a zip file.

Provided

Given is an sbt project on GitHub which contains case classes to represent the above named concepts (located in the **models** package) and one entry point to the application in the **mining** package, which you will use to run your code using the **sbt run** command.

Besides that, feel free to use any structure for your own code. Note,

The log is contained in the IncidentExample.csv file in the resources.



Tasks

I. CSV Import

Here we assume that the log is given in the form of a comma separated value file, namely *IncidentExample.csv.*
One line corresponds to one event and the columns refer to the data's dimensions, such as the timestamp, the case ID and the attributes. The headers contain the names of the columns. The example log has columns for the case ID, the activity name, a complete timestamp for when the activity was finished and one attribute called "classification", which is nominal.
Here is a small excerpt from the log:

```
1 Case ID,Activity,Start,Complete,Classification
2 trace_0,Incident logging,2016/01/04 12:09:44.000,2016/01/04 12:09:44.000,
3 trace_0,Incident classification,2016/01/04 12:10:44.000,2016/01/04 12:17:44.000,Citrix
4 trace_0,Initial diagnosis,2016/01/04 12:34:44.000,2016/01/04 12:39:44.000,
5 trace_0,Functional escalation,2016/01/04 12:41:44.000,2016/01/04 12:48:44.000,
```

For the sake of this exercise, please consider this structure to be **fixed**, meaning that column 1 (or 0 for us regular people) always will be the Case_ID, column 2 the Activity etc… You are not asked to build a fully configurable parser for this and it is ok (and encouraged) to hard code the elements of the CSV.
Please read all the events from the file into a collection over **Events**, either drop the **Complete** time and the **Classification** or adapt **Event.scala** accordingly.

II. Building the Traces

The next step is to take the Events and cluster them according to their **trace Ids** to form **Traces.** Again, build a collection of **Traces** where each trace contains its events *sorted by their start time.*

III. Building the Direct Follower Matrix

Once you have the traces, the last step is to build a data structure of your choosing, representing the direct follower matrix of activities found in the traces. Two events are only direct followers, when they proceed each other without another event in between in the same trace. The matrix should contain the counts of activity **M** followed by activity **N** and allow for quick access to this information. Lastly, quickly display the collected direct follower relations to the command line. Please don't bother about providing a nice visualization of this information, but we won't stop you if you want to practise your ASCII art.

Here is a simple visualization of such a matrix, where e.g. the follower relation from **a** to **b** has 12 occurrences.

	a	**b**	**c**
a	0	12	3
b	1	5	13
c	2	3	0

IV. Filtering the results based on dates

Implement functionality to filter the follower matrix computation based on a date range. The level of filtering here is based on **traces**, meaning that only those traces which start **and** end in the given time range are allowed. An allowed range should be chosen and the results should be efficiently recomputed.