Phase 1: Merging

(i) Create a lookup table (i.e. Hash Map) where we associate Strings and Bit Arrays (initialized as *false*)
(ii) Fix an ordering $C_1, C_2, ..., C_n$ of all columns (over all files).
(iii) Read an entry in a table. Assume it is in column C_i.
(iv) Mark bit i of the associated Bit Array as *true*.
(v) Repeat steps (iii) and (iv) until all entries have been processed.

Phase 2: Comparing

To decide weather column C_i is included in column C_j:

(i) Take a Bit Array you created in Phase 1.
(ii) If bit i is set *true* but bit j is *false*, this already means that $C_i \not\subseteq C_j$.
(iii) Repeat steps (i) and (ii) until all Bit Arrays have been processed.
(iv) If the algorithm has come to this point, it means that $C_i \subseteq C_j$.

Problems Fine for unary INGs, but creates massive (unnecessary, due to pruning) lookup tables for n-ary INGs.

Implementation

- Current version: Only unary ING implemented, only Phase 2 distributed.
- Uses actor pattern from demo, included full Reaper pattern and Master/Worker.
- Represent the Bit Arrays as BigIntegers and use Bit-operations.
- Represent the lookup table as HashMap and use Java 8's HashMap.merge() to update the BigIntegers.

Distribution

- Phase 1:
  - Can be distributed, but this need a lot of network traffic and still a lot computation power on the main node.
  - Idea: There are unique Workers which take care of entries of a fixed area of hash values of the entry.
  - Needs presorting at Input Reader level.

  Experimental runtime reduction: 40%, but Phase 1 doesn't need that much time at all
- Phase 2:
  - Can be distributed easily.
  - Idea: Worker gets a fixed number of BigInteger values and tells the Miner which INGs can be disqualified.