Challenge 1 (“time series”)

Problem Statement

Part of our daily routine at Fyber consists in efficiently processing time series. One of the common ways to analyze them is to compute local information within a rolling time window of length `T`, such as:

- number of measurements in a window
- minimum of measurements in a window
- maximum of measurements in a window
- rolling sum.

When implemented well such an analyses could be run on inputs vastly exceeding an amount of RAM on a computer.

Your goal is to write a small Scala program performing analysis of price ratios in an efficient way. Your program should accept a path to a file containing time series on a local filesystem as a command-line argument and print out results of analyses to the standard output.

The length of rolling time window is 60 seconds.

Format of Input File

An input file is in plain text. Each line contains a timestamp of a measurement in seconds and the measurement of price ratio as floating point number (which is guaranteed to fit into Scala Double data type). They are separated by a space or a tab character. For example:

```
1355270609 1.80215
1355270621 1.80185
1355270646 1.80195
1355270702 1.80225
1355270702 1.80215
1355270829 1.80235
1355270854 1.80205
1355270868 1.80225
1355271000 1.80245
1355271023 1.80285
1355271024 1.80275
1355271026 1.80285
1355271027 1.80265
1355271056 1.80275
1355271428 1.80265
1355271466 1.80275
1355271471 1.80295
1355271507 1.80265
1355271562 1.80275
1355271588 1.80295
```

Format of Output

Your program should print results as a table. Each row represents analysis for one position of rolling window over time-series. Each row should have the following values:

- T — number of seconds since beginning of epoch at which rolling window ends.
- V — measurement of price ratio at time T.
- N — number of measurements in the window.
- RS — a rolling sum of measurements in the window.
- MinV — minimum price ratio in the window.
- MaxV — maximum price ratio the window.

The table should have a header aligned with values in the following rows. All floating point numbers should be rounded up to 5 decimal points. Given input file from above and window lenght T = 60 your program should print the following to standard output:

```
T          V       N RS      MinV    MaxV
-------------------------------------------
1355270609 1.80215 1 1.80215 1.80215 1.80215
1355270621 1.80185 2 3.604   1.80185 1.80215
1355270646 1.80195 3 5.40595 1.80185 1.80215
1355270702 1.80225 2 3.6042  1.80195 1.80225
1355270702 1.80215 3 5.40635 1.80195 1.80225
1355270829 1.80235 1 1.80235 1.80235 1.80235
1355270854 1.80205 2 3.6044  1.80205 1.80235
1355270868 1.80225 3 5.40665 1.80205 1.80235
1355271000 1.80245 1 1.80245 1.80245 1.80245
1355271023 1.80285 2 3.6053  1.80245 1.80285
1355271024 1.80275 3 5.40805 1.80245 1.80285
1355271026 1.80285 4 7.2109  1.80245 1.80285
1355271027 1.80265 5 9.01355  1.80245 1.80285
1355271056 1.80275 6 10.8163 1.80245 1.80285
1355271428 1.80265 1 1.80265 1.80265 1.80265
1355271466 1.80275 2 3.6054  1.80265 1.80275
1355271471 1.80295 3 5.40835 1.80265 1.80295
1355271507 1.80265 3 5.40835 1.80265 1.80295
1355271562 1.80275 2 3.6054  1.80265 1.80275
1355271588 1.80295 2 3.6057  1.80275 1.80295
```

Output should not include any other records like slf4j intialization messages or debug output.

Results

Please send us sbt to build your code and send us complete sbt project with layout of files which build out of the box with `sbt compile`. Alternatively, you can put your code on Github and send us a link.

Do not send over jar files or other binaries.

Challenge 2 (“tasks”)

Problem Statement

The challenge consists in writing a Scala program which simulates propagation of events in a network. Events are represented with strings. Network consists of nodes. Each node has input and output endpoints and it can perform one of the following operations on the event passing through it:

- echo: the output is the input string concatenated to itself;
- reverse: the output is the input string reversed;
- delay: the output is the previous input string; the initial output of a delay task is “tbb”;
- noop: the output is the input string.

Nodes are connected with links which route events from output endpoint of one node to input endpoint of another node. Inputs are processed in the order they were linked. If a task has multiple input links, and several events arrive at the same time they should be concatenated into one event before feeding the task. Concatenating consist of appending the strings, in the order in which links were created.

Program Interface

Your program should accept a path to a file containing descriptions of the network on a local filesystem as a command-line argument and print out results of sumulation to the standard output. Output should not include any other records like slf4j intialization messages or debug output. Input is plain text which has one statement per line. A statement starts with either “task”, “link” or “process”.

- A task statement looks like: `task <name> <operation>`. It defines a node in the network which is called `<name>` and which performs `<operation>` on events passing throug it. Names of possible operations are listed above. First task defined with a task statement becomes entrance point into the network.
- A link statement looks like `link <name1> <name2>`. It defines a link which connects output of node called `<name1>` to input of `<name2>`. Task must be defined before they can be linked and your application should enforce this rule.
- A process statement looks like: `process input1 input2 ...`. Process statement starts evaluation of events flow: `input1` is sent to entrance point of the network (first node defined with task statement), then `input2` is sent into it and so on. For each process statement in the input, an output line consisting in the outputs of the last task in the network should be printed. A process statement should leave the network “dry”: no events are left in the network once processing results are printed. The output should be limited to 16 times the number of input strings.

For example, for the following input file:

```
task reverse reverse
task delay delay
link reverse delay
process foo bar
```

your program should print the following to stdout:

```
tbb oof rab
```

Would you find any ambiguity, please solve them by making assumptions about the underlying network. Because your solution will be evaluated within the boundaries of these assumptions, please document them in the README file and send it over with your solution to us.

Results

Please send us your code together with assumptions you made about task, instruction on how to build and run your program (if build process does not use `sbt`) as an archive. Alternatively, you can put your code on Github and send us a link.

Do not send over jar files or other binaries.