Challenge for the Position of Scala Developer at Fyber

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 file system as a command-line argument and print out results of analyses to the standard output.

The length of the rolling time window is 60 seconds.

Please, solve the problem using only the standard library and any test framework you like.

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 length 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
```

The output should not include any other records like slf4j initialization messages or debug output.

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.