Advanced Programmin A.Y. 19/20: final project

1. Abstract

Design and develop a server that, based on a text- and message-oriented protocol, takes requests of computation consisting of one or more mathematical expressions and input values and replies with the results.

2. Specifications

2.1 Domain definitions

Let e be a *mathematical expression* composed of the binary operators $O = \{+, -, \times, \div, \mathrm{pow}\}$ and of zero or more named variables $V_e \in V$.

Example: with $e = \frac{x+1}{y-2^x}$, $V_e = \{x, y\}$.

Let $a : V \to \mathbb{R}^*$ be a *variable-values function* that associates a list of numerical values $a(v) \in \mathbb{R}^*$ with a variable v.

2.2 Protocol

Upon connection with a client C, the server S performs iteratively these operations:

1. waits for a *request* r
2. closes the connection or replies with a *response* s, depending on the content of r

2.2.1 Request format

A request is a line of text with the following format (literal text is shown between double quotes `""`, regexes between single quotes `''`):

```
Request = QuitRequest
        | StatRequest
```

```
        | ComputationRequest
```

The format of a *quit request* is:

```
QuitRequest = "BYE"
```

The format of a *stat request* is:

```
StatRequest = "STAT_REQS"
            | "STAT_AVG_TIME"
            | "STAT_MAX_TIME"
```

The format of a *computation request* is:

```
ComputationRequest = ComputationKind"_"ValuesKind";"VariableValuesFunction";"Expressions
```

```
ComputationKind = "MIN"
                | "MAX"
                | "AVG"
                | "COUNT"
```

```
ValuesKind = "GRID"
           | "LIST"
```

A variable-values function can be specified with the following format:

```
VariableValuesFunction = VariableValues
                       | VariableValuesFunction","VariableValues
```

```
VariableValues = VarName":"JavaNum":"JavaNum":"JavaNum
```

```
VarName = '[a-z][a-z0-9]*'
```

and `JavaNum` is a string that can be correctly parsed to a `double` using the Java `Double.parseDouble()` method. A list of expressions can be specified with the following format:

```
Expressions = Expression
            | Expressions";"Expression
```

```
Expression = VarName
           | Num
           | "("Expression""Op""Expression")"
```

```
Num = '[0-9]+(\.[0-9]+)?'
```

```
Op = "+"
   | "-"
   | "*"
   | "/"
   | "^"
```

2.2.1.1 Examples

Some examples of valid requests are (one per line):

```
BYE
STAT_MAX_TIME
MAX_GRID;x0:-1:0.1:1,x1:-10:1:20;((x0+(2.0^x1))/(1-x0));(x1*x0)
COUNT_LIST;x0:1:0.001:100;x1
```

Some examples of **not valid** requests are:

```
bye
MIN_GRID;x0:-1:0.1:1,x1:-10:1:20;((x0+(2.0^x1))/(1-x0));log(x1*x0)
COUNT_LIST;x0:1:0.001:100;
MAX_LIST;x0:0:0,1:2;(x0+1)
```

2.2.2 Response format

A response is a line of text with the following format:

```
Response = ErrorResponse
         | OkResponse
```

The format of an *error response* is:

```
ErrorResponse = ERR";"`[^;]*`
```

The format of an *ok response* is:

```
OkResponse = OK";"JavaNum";"JavaNum
```

where `[^;]*` does not include new line characters.

2.3 Request processing specifications

If the request r is a *quit request*, the server S must immediately close the connection with the client C.

Otherwise, S must reply with a response s. If s is an error response, the part of s following `ERR;` must be a human-comprehensible, succint textual description of the error. Otherwise, if s is an ok response, the first of two numbers following `OK;` must be the *response time*, i.e., the number of seconds S took to process r, with at least 3 digits after the decimal separator (millisecond precision).

2.3.1 Stat requests

If r is a stat request, S replies with an ok response where the second number is:

- the number of ok responses served by S (excluding r) to all clients since it started, if r is `STAT_REQS`;
- the average response time of all ok responses served by S (excluding r) to all clients since it started, if r is `STAT_AVG_TIME`;
- the maximum response time of all ok responses served by S (excluding r) to all clients since it started, if r is `STAT_MAX_TIME`.

2.3.2 Computation requests

If r is a computation request, S does the following steps:

1. parse a variable-values function a from the `VariableValuesFunction` part of r
2. build a list T of *value tuples* from a, each value tuple specifying one value for each v of the variables for which $a(v) \neq \emptyset$, depending on the `ValuesKind` part of r
3. parse a non-empty list $E = (e_1, \dots, e_n)$ of expressions from the `Expressions` part of r
4. compute a value o on T and E depending on the `ComputationKind` part of r

If any of the steps above fails, S replies with an error response. Otherwise S replies with an ok response s where the second number in s is o.

2.3.2.1 Step 1: parsing of `VariableValuesFunction` to a

First, a list I of tuples $(v, x_{\text{lower}}, x_{\text{step}}, x_{\text{upper}})$ is obtained by parsing each `VariableValues`. If, for any tuple, $x_{\text{step}} \leq 0$, the step fails.

Second, $a : V \to \mathcal{P}(\mathbb{R})$ is built as follow:
$a(v) =$
$$\begin{cases} \emptyset & \text{if no tuple for } v \text{ exists in } I \\ (x_{\text{lower}} + kx_{\text{step}} : x_{\text{lower}} + kx_{\text{step}} \leq x_{\text{upper}})_{k \in \mathbb{N}} & \text{otherwise} \end{cases}$$

Example: `x0:-1:0.1:1,x1:-10:1:20` is parsed such that $a($`x0`$) = (-1, -0.9, \dots, 0.9, 1)$, $a($`x1`$) = (-10, -9, \dots, 19, 20)$, and $a(v) = \emptyset$ for any other v.

2.3.2.2 Step 2: building of value tuples T from a

If `ValuesKind` is `GRID`, than T is the cartesian product of all the non empty lists in the image of a.

Otherwise, if `ValuesKind` is `LIST`, if the non empty lists in the image of a do not have the same lenght, the step fails. Otherwise, T is the element-wise merging of those lists.

For example, for an a parsed from `x:1:1:3,y:2:2:6`:

- $T = ((1, 2), (2, 2), (3, 2), \dots, (1, 6), (2, 6), (3, 6))$ if `ValuesKind` is `GRID`;
- $T = ((1, 2), (2, 4), (3, 6))$ if `ValuesKind` is `LIST`.

where `x` and `y` are omitted in T elements for brevity.

2.3.2.3 Step 3: parsing of `Expressions` to E

For each `Expression` token in `Expressions`, an expression e is built and added to E by parsing the `Expression` token based on the corresponding context-free grammar. If any of the expression parsing fails, the step fails.

A sample code for performing this step is provided in the form of a few Java classes. The student may freely get inspiration from or reuse this code.

2.3.2.4 Step 4: computation of o from T and E

Let $V_t \in V$ be the set of variables for which a tuple t defines the values and let $e(t) \in \mathbb{R}$ be the value of the expression e for the variables values given by t such that $V_t \supseteq V_e$.

Then:

- if `ComputationKind` is `MIN`, $o = \min_{e \in E, t \in T} e(t)$, or the step fails if $\exists e \in E : V_t \not\supseteq V_e$;
- if `ComputationKind` is `MIN`, $o = \max_{e \in E, t \in T} e(t)$, or the step fails if $\exists e \in E : V_t \not\supseteq V_e$;
- if `ComputationKind` is `AVG`, $o = \frac{1}{|T|} \sum_{t \in T} e_1(t)$, or the step fails if $V_t \not\supseteq V_{e_1}$;
- if `ComputationKind` is `COUNT`, $o = |T|$.

2.4 Examples of request-response pairs

Some examples of request-response pairs (one request or response, interleaved, per line, starting with a request):

```
MAX_GRID;x0:-1:0.1:1,x1:-10:1:20;((x0+(2.0^x1))/(21.1-x0));(x1*x0)
OK;0.040;52168.009950
COUNT_LIST;x0:1:0.001:100;x1
OK;0.070;99000.000000
MIN_GRID;x0:-1:0.1:1,x1:-10:1:20;((x0+(2.0^x1))/(1-x0));log(x1*x0)
ERR;(ComputationException) Unvalued variable log
STAT_MAX_TIME
OK;0.000;0.070000
```

3. Non-protocol specifications

The server must:

- log on the standard output or standard error significant runtime events as:
  - new connection from client
  - disconnection from client
  - errors
- listen on port p specified as command-line argument
- handle multiple clients at the same time
- never terminate, regardless of clients behavior
- at any time, do at most n computation for processing computation requests at the same time, with n being equal to the number of available processors on the machine where the server is running.

Moreover, the server must:

- be a Java application delivered as a `.jar` named after the student last name and first name in upper camel case notation (e.g., `MedvetEric.jar`);
- be executable with the following syntax `java -jar MedvetEric.jar` p (e.g., `java -jar MedvetEric.jar 10000` for $p = 10000$)

4. Delivery of the project

The student must deliver the project to the teacher **within the deadline** by email, with **a single `.zip` attachment** containing:

- the `.jar` file, in the root of the `.zip`
- at most one (i.e., optional) pdf with a brief description of key design choices
- all the source files for the project, properly organized

No tests are required; no documentation is required.