AE2ACE coursework 2019-2020

This coursework is worth **25%** of the final mark.

Stock Trading & Algorithm Correctness. Deadline: 29 November 2019 16:00

Part I: Stock Trading (12 marks)

Warning: If program cannot be complied, it will result in an immediate **zero mark** for this part. You are strongly advised to check your code thoroughly before submission.

Basics on Stock Trading

An online computer system for stock trading needs to process orders of the form "buy 100 shares at ￥x each" or "sell 100 shares at ￥y each". A BUY order for ￥x can only be processed if there is an existing SELL order with price ￥y such that $y \leq x$. Likewise, a SELL order for ￥y can only be processed if there is an existing BUY order with price ￥x such that $y \leq x$. If a BUY or SELL order is entered but cannot be processed, it must wait for a future order that allows it to be processed.

Task Descriptions

The task is: given a set of existing SELL orders, and a set of existing BUY orders, you should determine whether a new order, either SELL or BUY, could be executed or not, or in some cases partially executed.

Functionalities of the Java Program

1. A method InsertBuyOrder() to insert a BUY order into the BUY priority queue.

For example, you have the following BUY orders.

{{￥102, 100}, {￥103, 200}, {￥98, 300}, {￥99, 400}}.

After you enter them into the priority queue, it looks like:

{{￥103, 200}, {￥102, 100}, {￥99, 400}, {￥98, 300}}.

If you have a new BUY order, e.g. {￥102, 400}, the priority queue will become:

{{￥103, 200}, {￥102, 100}, {￥102, 400}, {￥99, 400}, {￥98, 300}}.

You may give each order a transaction ID to track each order, e.g. For each order, it consists of {Price, Amount, ID}. The priority queue for BUY orders are max-oriented priority queue, i.e. whoever offers a higher price, his/her order will be executed first.

2. A method InsertSellOrder() to insert a SELL order into the SELL priority queue.

As opposite to the priority queue for BUY orders, the priority queue for SELL orders is min-oriented priority queue, i.e. whoever sells at a lower price, his/her order will be executed first.

For example, the priority queue for SELL orders is:

{{￥95, 200}, {￥98, 100}, {￥102, 100}, {￥103, 200}}.

If you have a new SELL order, e.g. {￥98, 200}, the priority queue will become:

{{￥95, 200}, {￥98, 100}, {￥98, 200}, {￥102, 100}, {￥103, 200}}.

3. A method ExecuteOrder() to execute orders, i.e. execute BUY orders and SELL orders that could be executed.

For example, give two priority queues as:

PQ_BUY = {{￥103, 200}, {￥102, 100}, {￥102, 400}, {￥99, 400}, {￥98, 300}}.

PQ_SELL= {{￥95, 200}, {￥98, 100}, {￥98, 200}, {￥102, 100}, {￥103, 200}}.

After calling ExecuteOrder(),

PQ_BUY = {{￥102, 100}, {￥99, 400}, {￥98, 300}}.

PQ_SELL= {{￥103, 200}}.

You should also record and display (if needed):

- The cut-off BUY price and SELL price,

E.g. ￥102 for both in the above example.

- The executed BUY orders:

E.g. {{￥103, 200}, {￥102, 100}, **{￥102, 300}**}.

- The executed SELL orders:

E.g. {{￥95, 200}, {￥98, 100}, {￥98, 200}, {￥102, 100}}

Illustration

1. You may implement the priority queues using heaps.
2. You may generate a set of random numbers for orders in the following way:

For example, the BUY price ￥x is randomly distributed in [80, 110]. The SELL price ￥y is randomly distributed in [100, 130]. You may keep two decimal places for the price. The amount for BUY/SELL order is a multiple of 100, e.g. 100, 200, 300, etc. Maximum amount is 1000. In another word, the amount is a random number drawn from {100, 200, ..., 1000}.

3. You should provide the test code as well.

Test case 1: For two priority queues given in PQ_BUY and PQ_SELL shown above, your program should work fine.

Test case 2: You should generate a set of BUY orders and SELL orders as stated in Illustration 2, for different number of orders, e.g. n = 10, 100, 1000, 10000, ...

You may self-study the relevant algorithms, and implement it in Java. You may refer to the implementation on the textbook, internet or other open resources, but with proper acknowledgement. Please include proper comments in your code.

Marking Criteria

1. In **Test Case 1**, your program runs successfully and returns correct results, without any mistakes in Section 3. (**6 marks**)
2. In **Test Case 2**, your program runs successfully and returns correct results, without any mistakes in Section 3. (**6 marks**)
3. Common mistakes for mark deduction:

- No comments or very few comments.
- Too many hardcoded magic numbers.
- The core algorithms and the test program should be separated, but they are not.

- The program is not a Java-style program, but a more c-style program.
- Input is hardcoded, not input by user, nor read from an input file.
- Many others that are not good programming practice.

Part II: Algorithm Correctness (13 marks)

See the detailed description in the next page.

Part II: Algorithm Correctness

Use the proof calculus to construct proofs for the following questions.

Question 1 (3 marks)

Let P be the program:

$$
\begin{array}{l}
if\ (x > y)\{ \\
\quad\quad z = x; \\
\}\ else\ \{ \\
\quad\quad z = y; \\
\}
\end{array}
$$

Show that $\vdash_{tot} \{\top\}\ P\ \{z = \max(x, y)\}$ is valid, where $\max(x, y)$ is the largest number of x and y. [3 marks]

Question 2 (10 marks)

Let *Fac1*(x) be the program:

$$
\begin{array}{l}
a = x; \\
y = 1; \\
while\ (a\ > 0)\{ \\
\quad\quad y = y * a; \\
\quad\quad a = a - 1; \\
\}
\end{array}
$$

Show that $\vdash_{tot} \{x \geq 0\}\ Fac1(x)\ \{y = x!\}$ is valid.

1. Write down a proper loop invariant which is useful for constructing the correctness proof. [2 marks]
2. Write down a proper variant which is useful for proving the termination of the program. [1 mark]
3. Provide the full proof using proof rules. [4 marks]
4. Justify the correct uses of the implied rule in three places of the proof in English. [3 marks]

Plagiarism

If you use code you found in a textbook or on the web, you must acknowledge it. I will run the plagiarism detector tools to check for similarities between submissions and web-based material.

You are reminded of the School's Policy on Plagiarism.

How to submit

Online submission (code and report) via Moodle. Please submit the report as a pdf file, the java files and any other files you need to make these java files compile. Please note that every next submission overwrites all the files in the previous one, so if you submit several times, make sure that your last submission includes all the necessary files.

For **late submission**, the standard late submission policy applies, i.e. 5% mark deduction for every 24 hours.