ALGORITMI PARALELI ȘI DISTRIBUIȚI
Homework #2 The miners and the sleepy wizards

Termen de predare: 25 Noiembrie 2018

Responsabili Temă: **Cristian Chilipirea, Radu Ciobanu**

Alexandru-Adrian Achiritoaei, Tiberiu Sosea

It is forbidden to have any I/O in the submitted solution. No reads, no writes, to standard output or files.

Objectives

Let there be **Miners** and **Wizards**. The goal of the miners is to explore all the tunnels in their cave system. The cave system has many rooms, each with its puzzle to solve. The miners can solve the puzzles and with the solutions, the wizards can break the magic barriers protecting the cave and gain access to the adjacent rooms.
The Wizards are not always available, as sometimes they sleep. When a wizard is awake, he takes puzzle solutions from the miners, and does wizardly things to unlock the paths to all adjacent rooms. If the puzzles are solved incorrectly, when the wizard tries to unlock the paths, a great explosion kills everyone, ends the program, and the student is given 0 points.
The solution should be implemented in **Java** using **threads**. You are only responsible for implementing the miners and the communication channel between them and the wizards.

The miners and their puzzles

Each puzzle starts with a random string found on a piece of paper inside the room. To solve the puzzle, a miner needs to... well... mine it [1]. To do this the miner starts calculating the hash of the hash of the hash of ... of the hash of the string. The number of times to repeat the hash operation is fixed and equal to N.
The miners start outside of the cave, and they know in which room to go by getting this information from the wizards. When this happens, the wizards also tell the miners what the string that needs to be hashed is.

The cave system

The cave system contains many rooms. There are multiple cave entrances and all of them can be accessed at the start. There is no guarantee that any room can be reached from any other room using the cave system. However, there is a guarantee that all rooms can be reached if all entrances are used.
Each room is protected by magic that needs to be taken down by the wizard. Once a room is unlocked, the adjacent, undiscovered rooms are revealed. One room can be connected with any number of rooms [2].

[1] A real life distributed system that uses lots and lots of hash computations is Bitcoin. In Bitcoin, the processes that calculate all the hashes are called miners.
[2] How can there be so many connected rooms? Magic.

The wizards

A wizard gets tired fast and when he gets tired he simply sleeps. It is not clear what triggers a sleep stage for a wizard, as the time between two sleep periods is not fixed and doesn't seem to be correlated to the number of rooms he unlocks.
When the wizard is awake, he takes a puzzle solution along with the name of the current room as well as the name of the previous room, and unlocks the current room. An incorrectly solved puzzle will trigger the explosion.
Once a barrier is down the wizard gives the list of adjacent rooms to the miners, so they can start work on the next puzzles.

The communication channel

The miners and the wizards communicate using a communication channel. For every puzzle solved, the miners give the wizards the following information in a single message: the room name, the name of the previous room (the parent room) and the solution of the puzzle.
For every puzzle solution, the wizards give the miners the list of rooms adjacent to the room in which the puzzle was solved. It the wizards get an incorrect solution or incorrect previous room, an explosion is triggered.
When there are no more rooms, the wizards put EXIT messages in the communication channel, one for each miner. They also trigger the end of the program.
Hint: The wizards give the miners the list of adjacent rooms in separate messages that arrive in the following order: *current room, adjacent room1, current room, adjacent room2, ... , END*. If two wizards give the miners this information at the same time, it might confuse them. Each of the adjacent room messages contains the string to be hashed by the miners in the new room (in the "data" field of the message).

Scalability

It should be obvious by now that, if we had more miners solving puzzles, the cave system would be discovered faster. Because of this, the number of miners should match the number of threads and the execution time should scale with the number of miners.

Running, testing and uploading

Programs would be run as follows:

```
java Homework <caveInfoPath> <numberOfHashes> <numberOfWizards> <numberOfMiners>
```

As an example, the program in the skeleton can be run as shown below:

```
java Homework test_cases/test01 100000 4 4
```

In order to help with the testing, we have provided you with one set of input files, located in the folder called *test_cases*. The file *test01_answer.txt* contains the correct answers for each room (which should be computed by the miners and verified by the wizards), *test01_data.txt* contains the values to be hashed (i.e., the random strings found on pieces of paper inside each room), while *test01_graph.txt* contains the layout of the cave system. All test files should follow the same naming format (e.g., if your first parameter when running the homework is "mytest", then you should have the files *mytest_answer.txt*, *mytest_data.txt* and *mytest_graph.txt*). The files are read only by the wizards, whereas the miners only receive data from the wizards.

How to use the skeleton files

In the homework archive, you can find the code for the main class (*Homework.java*), which starts the miners and the wizards and waits for them to finish, and stub files for the communication channel (*CommunicationChannel.java*) and the miners (*Miner.java*), which you must implement.
The code for the wizards and the messages exchanged on the communication channel can be found in two class files (*Wizard.class* and *Message.class*), with information regarding the public methods and fields from these classes available in the Javadoc located in the archive provided to you (all you need to do to access it is to go to the *doc* folder in the archive and open the *index.html* file in the web browser of your choice).
If you are working in Eclipse, you can perform the next steps to import the .java and .class files in a project:

- File → New → Java Project
- Give a project name, make sure you use Java 1.7, then click Next and then Finish
- Right click on the "src" folder in the project → Import → File System → select "src" folder from the skeleton folder → select .java files → press Finish
- Right click on your project → Properties → Java Build Path → Libraries → Add External Class Folder → select "src" folder from the skeleton folder
- Run the program normally

If you are working in IntelliJ IDEA, you can perform these steps to import the .java and .class files in a project:

- Import Project
- Go to the location of the skeleton and select the "src" folder
- Choose "Create project from existing sources", press Next, give the project a name, click Next again, make sure **only** the skeleton folder is selected and then press Next again until you finish
- Right click on the project → Open module settings → Libraries
- Press "+" → Java → select "src" folder from skeleton folder
- Run the program normally

If you are working in command line, you can perform the following steps to build and execute your program:

- Compile all the .java files with "javac *.java"
- Run the program normally (e.g., "java Homework test_cases/test01 100000 4 4")

Cave system generator and room solver

To help you with your testing, we have added a Python program that can generate a cave system (i.e., a graph) and the values found on pieces of paper in each room (i.e., the values to be hashed by the miners). It is located in the folder called *generator*, and can be run as follows (where we create a graph with 100 nodes and 150 edges, stored in files *test01_graph.txt* and *test01_data.txt*):

```
python generate_graph.py -f test01 -s 100 -e 150
```

In the homework archive, you can also find a tool for computing consecutive hashes on some input strings, which can be used to generate the input files with answers that the wizards use to verify the correctness of the solutions received from the miners. The wizard implementation in the skeleton uses those same hash functions to verify the correctness of the miners' computations.
With these two tools, you can create your own complete tests and verify your program even before uploading it to the checker.

Wizard and Message implementation

We have also provided the Java implementation for the Wizard and Message components, which can be found in the "java" folder of the skeleton.

Upload

You have to upload a .zip archive which contains .java files, a *Makefile* and a *Readme.txt*. The archive should not contain any other files or directories.
Your solution needs to compile and run on the cluster (this might mean using a different version of Java compared to what you use on your local system). Currently on the cluster java 1.7 is installed.

Any attempt to trick the checker system or falsify the results (including the speedup), would result in a penalty that would remove all points for the 3 homework assignments.

It is forbidden to print anything on screen, at stdout or stderr.

Points:
40 points - Correctness
60 points - Speedup
A solution that doesn't compile or that does not pass any tests will be awarded 0 points.