Distributed Systems Assignment

Basic information

- Release Date: 17 February 2018.
- **Due: 27 March 2018, 4pm.**
- This assignment is worth 25% of the final mark and will be marked out of 100. This assignment asks you to implement and analyze different distributed rumour spreading schemes.

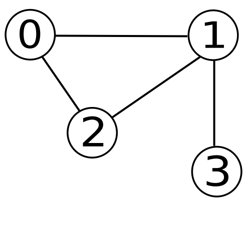
This assignment has two parts. Please read the entire description including submission instruction to see what you need, before you start work.

Part A: Simulation of rumour spreading

A *simulator* is a program that mimics the behaviour of a real system and lets us understand its properties. Unlike real distributed systems, the parameters in simulators can be changed easily to study system properties. Developing a simulator requires careful thought and understanding of what may happen in a real distributed system. In this coursework, you will develop a simple simulator for distributed systems and run rumor spreading algorithms.

This should be done using Java (Oracle Java 1.8), and should run on DICE systems. They will be tested on DICE. The following are the instructions to build your simulator:

- The underlying graph of the distributed system is specified using edge lists of its nodes in a text file. See Figure 1 for an example. Each line in the input text file shall have a comma separated list of its neighbors in the graph. The nodes are numbered starting from 0. *Make sure you follow this format as we will test your implementation on different inputs.*
- Each node or process in the graph should run on a separate thread and the inter-node communications are managed by a separate thread running an object called *network*.



```
0:1,2
1:0,2,3
2:0,1
3:1
```

Figure 1: Sample graph and its adjacency list.

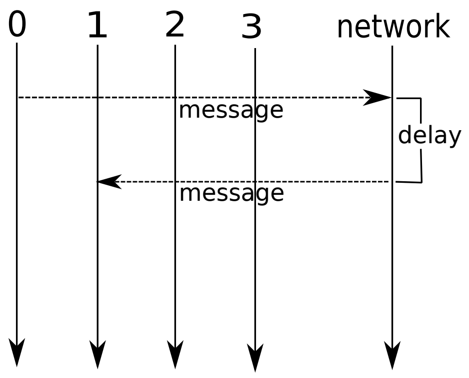


Figure 2: Sending message from node 0 to node 1.

- A node can send the rumour message to one of its neighbors via the network thread. Figure 2 shows an example sequence diagram for sending a message from node 0 to node 1.
- The network thread should add a delay between receiving a message from the source and delivering it to the destination to simulate transmission delay. Your simulator should independently choose a delay for each message in milliseconds uniformly at random between, say, 900 and 1100 milliseconds. You can add a similar delay for a node to attempt the next transmission. A message must not be delayed due to the delay introduced by a previous message. The suggested delay is to make the execution visible in Q2. You can consider using a smaller or larger delay depending on various parameters, so that the events are visible, yet the simulation runs in a reasonable time.

  Note that unlike the theoretical analysis that operates in synchronous rounds, in this simulation we are testing an un-synchronized setup. *Hint: In the network you may schedule a TimerTask for delivering a message when it receives one.*
- (Optional) Consider having an option to change the default delay range to allow faster simulation on large networks.

Question 1: Implement PUSH protocol (30/100 points)

Implement the rumour spreading PUSH protocol taught in the class. The input would be (on command line): a text file and a starting node (more specific input format is given in submission instructions section). The output shall be a text file containing the list of nodes ordered by the time when it received the rumour. Each line in the output text file should have a single node ID.

Question 2: Visualize PUSH protocol (10/100 points)

In this question you need to visualize the rumour spreading in the graph using *GraphStream* (http://graphstream-project.org/) library. A node should be colored green if it has the rumour and red otherwise. So, initially all the nodes except the starting node should be colored red, and the program stops when all nodes turn into green.

An example for using the library is given with this assignment.

Question 3: Probabilistic message drop model (20/100 points)

In this question, a message may be dropped at the network with probability p. Note that this model is different from the one in Part B. You need to run the same PUSH protocol as in previous questions with different values of p, starting from 0.05 to 1 with an increment of 0.05. You need to produce a graph where the $x-$axis denotes different values of p and the $y-$axis denotes the time (in milliseconds) to spread the rumour to all nodes in the corresponding setup. The graph needs to have a single point plotted for each (p, time) pair, this is called as scattered plot.

JFreeChart is a popular utility that can be used to get this plot, but you are free to use any other utility of your choice.

Part B: Theoretical analysis of rumour spreading

The rumour spreading PUSH protocol is one of the most fundamental distributed algorithms for information dissemination. When the underlying network G is a complete graph with n nodes, we have seen in class that the rumour spreading time

$$T(G) = \log n + \ln n + o(\log n),$$

where $T(G)$ is the number of synchronous rounds needed for all nodes to receive the rumour with high probability. Despite of the clear and tight bound above, assuming the underlying network G is fully connected, i.e., a complete graph, might be unrealistic for many applications, and deriving a bound of $T(G)$ for a more practical graph G is very desired.

In this problem we study a rumour spreading process in an evolving network. For formally, for two parameters $n \in \mathbb{N}$ and $p \in (0, 1]$, an evolving network can be viewed as a sequence of n-vertex graphs $G_0, G_1, \ldots$ such that, for any i, any pair of vertices in G_i is connected by an edge with probability p. The rumour spreading PUSH protocol in an evolving network is described as follows:

- Initially, one node in G_0 has the rumour;
- In each round $t \geq 0$, every informed node u sends the rumour to a node randomly chosen from the nodes that are connected to u by an edge in G_t.

Notice the following difference between rumor spreading in an evolving network and rumour spreading in a complete graph:

- In each round t every edge is present with the same probability, and this random event is independent with other edges' choices in t and choices of the edges in previous rounds;
- In each round t, every informed node u is only able to use the edges present in G_t to send a rumour. For instance, if an informed node u has only 5 neighbous in round t, then node u chooses a neighbour with probability 20% and sends the rumour to its picked neighbour.

Question 4 (16/100 points)

Let $p = 0.3$. Analyse the number of synchronous rounds needed for all nodes to receive the rumour with probability $1 - o(1)$.

Question 5 (24/100 points)

Generalising the analysis above, $p(n)$ is a general function of n, and you'll analyse the number of synchronous rounds needed for all nodes to receive the rumour with probability $1 - o(1)$. *[Remark: while a complete answer to this question is quite involved, we'll mark your solutions based on the best submitted solution of the course, i.e., the best submitted solution will get the full points, and the 2nd best solution will get the 2nd highest points, etc.]*

Submission Instructions

Your submission should be contained in a directory named as your *uun* (do include the *s*, e.g., *s*1234567). Your submission should compile and run in DICE environment. You may need to copy depdendancy jar files into the dependency directory for successful compilation.

Follow the following directory structure for your submission:

- readme.txt (max 1 page) - Any discussion, comment, interesting observation to any specific design decision you want us to know.
- Part A directory should contain the following items:
  - source - contains all the source files
  - build - compiled binary (jar file) (optional)
  - dependency - contain all jars required for compilation except standard java jar files.
  - log - Contains the log file generated for Question 1. And if we run your code for Question 1 then it should generate the output file here.
  - *run.sh* script to compile and run the program. With the following options.
    - *./run.sh compile* : compiles the source and builds binary into corresponding directory.
    - *./run.sh q1 <graph edge list file> <starting node id>* - Outputs a text file in the *log* directory.
    - *./run.sh q2 <graph edge list file> <starting node id>* - Shows the visualization while running.
    - *./run.sh q3 <graph edge list file> <starting node id>* - The program should print the value of p the system is currently using following the node ids (each in a new line) in the standard output as they get the rumor message for the first time. It should print for each p that is used.
- **Part B** directory should contain a pdf file containing your answer to Question 4 and 5.

Submit commant: Finally, make a .tar.gz file named as *<uun>.tar,gz* and submit using the *submit* command on DICE:

```
submit ds cw1 filename
```

Remarks

- To test your code and concepts, you can try on different types of graphs, such as tree, complete graph, star graph, etc.
- We do not expect the program to run for very large graphs. So, if your program runs well for ~ 500 nodes it should be fine.

University regulations

University regulations On good Scholarly Practice. Please remember the University requirement as regards all assessed work. Details about this can be found at:

http://web.inf.ed.ac.uk/infweb/admin/policies/academic-misconduct

Remember, if you use ideas from elsewhere (including other students), cite them. And try not use too much of these. The regulation says you can pick up “general ideas” but not “pivotal ideas”. But “general” and “pivotal” are subjective. Play safe and avoid getting into trouble.