1 Overall marking scheme

The coursework for consists of two assignments contributing altogether 30% of the final mark. The contribution of the individual assignments is as follows:

Assignment 1	15%
Assignment 2	15%
TOTAL	30%

2 Objectives

This assignment requires you to design and implement in Java a distributed algorithm for leader election in a *ring of rings* network and then to experimentally validate its correctness and evaluate its performance.

3 Description of coursework

The network considered in this assignment is a *ring of rings*, defined as follows (see Figure 1 for an illustration while reading the definition). There is a *main ring* G of n processors $V = \{u_1, u_2, \ldots, u_n\}$. There is a non-empty subset $V' \subseteq V$ of *interface processors*, such that every $v \in V'$ is associated with a distinct ring subnetwork R_v. We call the remaining processors *non-interface processors*; these are the non-interface processors of G and the processors of the ring subnetworks. Initially, every non-interface processor u has a *unique id* stored in a variable $myID_u$, while every interface processor v has $myID_v := -1$ to denote that its id is to be determined through its subnetwork. The subnetworks are not connected to the main ring via communication links, instead every R_v is implicitly connected to its interface v via shared variables. In particular, we make the assumption that whenever an elected processor w of R_v sets the value of a variable $leaderID_w$, then $myID_v := leaderID_w$, that is, as soon as a processor w is elected in R_v, the interface v knows the id of the elected processor through the shared variable $leaderID_w$. It follows, that every interface processor can eventually obtain an elected id from its subnetwork.

In our setting, the processors do not know the number of processors in the main ring or in any of the subnetworks in advance, but they do know the general structure of the network and all but the interface processors are equipped with *unique* ids. The ids are not necessarily consecutive and for simplicity you can assume that they are chosen from $\{1, 2, \ldots, \alpha N\}$, where $\alpha \geq 1$ is a small constant and N is the total number of non-interface processors (e.g., for $\alpha = 3$, the N non-interface processors will be every time assigned unique ids from $\{1, 2, \ldots, 3N - 1, 3N\}$). Processors execute in synchronous rounds, as in every example we have discussed so far in class.

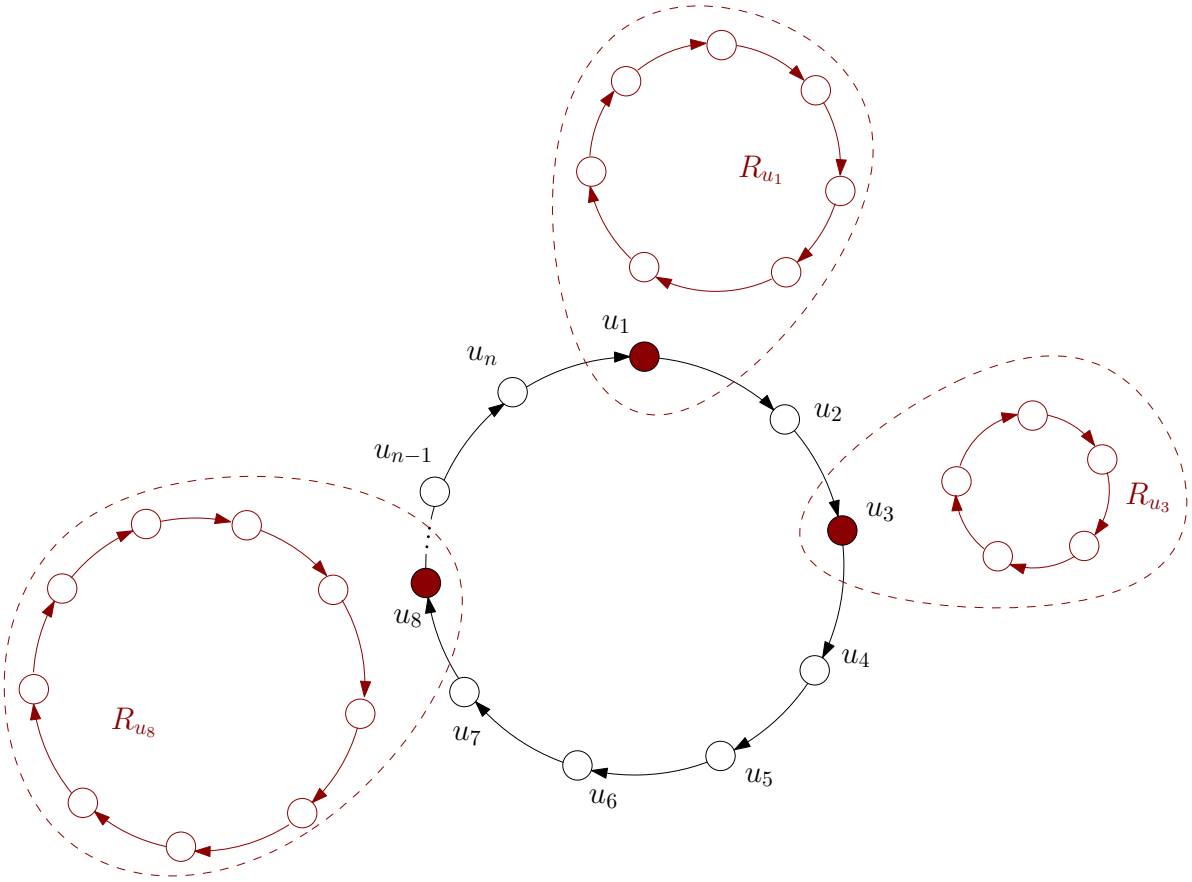

Figure 1: A ring of rings network. The red nodes represent the interface processors whose ring subnetwork is included within the dashed area. The white nodes represent the N non-interface processors and each of those has initially a unique id. The goal of a leader-election algorithm in a ring of rings is to eventually elect the maximum id over all the initial ids of the non-interface nodes.

3.1 Implementing the Asynchronous-Start and Terminating LCR Algorithm—30% of the assignment mark

As a first step, you are required to implement a *variant* of the LCR algorithm for leader election in a **ring** network (**not** in a ring of rings yet). The pseudocode of the standard LCR can be found in the lecture notes and is also given here for convenience (Algorithm 1).

Algorithm 1 LCR (synchronous-start and non-terminating version)

Ring network
Code for processor u_i, $i \in \{1, 2, \ldots, n\}$:

Initially:
u_i knows its own unique id stored in $myID_i$
$sendID_i := myID_i$
$status_i :=$ "*unknown*"

```
1: if round = 1 then
2:     send ⟨sendID_i⟩ to clockwise neighbour
3: else// round > 1
4:     upon receiving ⟨inID⟩ from counterclockwise neighbour
5:     if inID > myID_i then
6:         sendID_i := inID
7:         send ⟨sendID_i⟩ to clockwise neighbour
8:     else if inID = myID_i then
9:         status_i := "leader"
10:    else if inID < myID_i then
11:        do nothing
12:    end if
13: end if
```

You are required to implement an *asynchronous-start and terminating version* of the LCR algorithm in which additionally to the standard LCR:

1. Every processor u_i has an associated *start round* $r_i \geq 1$, such that the processor wakes up and starts executing the algorithm at the beginning of round r_i. Before round r_i, processor u_i is asleep and any message transmitted to it is lost.

2. All processors eventually terminate and know the elected id.

3.2 Implementing a Leader-Election Algorithm for Rings of Rings—40% of the assignment mark

Next, you are required to come up with and implement an algorithm for leader election on a **ring of rings** network. Your algorithm should make use of the asynchronous-start and

terminating variant of the LCR algorithm that you developed in the previous step. It should eventually elect **the maximum id over all ids assigned initially**, let all processors **of the main ring** know of the elected id, and have all processors **of the main ring** terminate.

Hint. The ring subnetworks will be executing the terminating LCR, while the main ring will be executing its asynchronous-start variant. You are also allowed to assume that all rings are bidirectional with the processors still having a sense of clockwise (counter-clockwise, respectively) orientation. You could make use of this in order to avoid some message retransmissions, but doing so is not a requirement.

3.3 Experimental Evaluation & Report—30% of the assignment mark

After implementing the leader-election algorithm for a ring of rings, the next step is to conduct an experimental evaluation of its correctness and performance.

Correctness and Performance. Execute your algorithm in networks of increasing size (up to reasonable execution times, depending on your equipment) and varying structure (parameters to consider: number and positioning of interface nodes, size of subnetworks) and starting from various id assignments for each given setting. For instance, you could execute your algorithm on both specifically constructed id assignments (e.g., ids ascending clockwise or counterclockwise) and random id assignments. In each execution, your simulator should check that eventually the maximum id over all initial ids is elected. Of course, this will not be a replacement of a formal proof that your algorithm is correct as you won't be able to test it on all possible input configurations, but at least it will be a first indication that it might do as intended. In each execution, your simulator should also record the ***number of rounds*** and the ***total number of messages*** transmitted until termination.

After gathering the simulation data, plot them as follows. For each chosen combination of network structure and type of id assignment (e.g., fixed number, size, and positioning of subnetworks and random ids) the x-axis will represent an increasing parameter of size (e.g., the size n of the main ring or the number N of non-interface nodes) and the y-axis will represent the complexity measure (number of rounds or number of messages). Your plots can also compare against standard complexity functions, like n, $n \log n$, or n^2, so that it will become as clear as possible what are the asymptotic complexities of the algorithm. Generate **at least 4 plots** (each studying a different combination of parameters) that will help you draw meaningful conclusions about the time and communication complexities of your algorithm. You can use gnuplot, JavaPlot or any other plotting software that you are familiar with.

The final *crucial* step is to prepare a concise report (at most 5 pages including plots) clearly describing your algorithms, the main functionality of your simulator, the set of experiments conducted, and the findings of your experimental evaluation. In particular, in

the latter part you should try to draw conclusions about (i) the algorithm's correctness and (ii) its performance w.r.t. time and messages (e.g., what was the worst/best/average performance of the algorithm as a function of n?).

4 Deadline and Submission Instructions

- The deadline for submitting this assignment is **Thursday, 3rd March 2022, 17:00 UK time (UTC)**.
- Submit
  (a) The Java source code for all your programs,
  (b) A README file (plain text) describing how to compile/run your code to reproduce the various results required by the assignment, and
  (c) A concise self-contained report (at most 5 pages including everything) describing your algorithms, experiments, and conclusions in PDF format.

  Compress all of the above files into **a single ZIP** file (the electronic submission system won't accept any other file formats) and **specify the filename** as *Surname-Name-ID*.zip. It is extremely important that you include in the archive all the files described above and not just the source code!

Good luck to all!