ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

FS 2024 Prof. R. Wattenhofer

Principles of Distributed Computing
Sample Solution to Exercise 3

1 Leader Election in an "Almost Anonymous" Ring

a) Yes, it is possible with a 1-round algorithm:

Algorithm 1 Leader Election (all but one of the nodes have the same ID)

```
1: send ID to each neighbor
2: if both received ID's differ from the own ID then
3:    output "I am the leader"
```

b) There exits an algorithm if and only if the nodes with ID 1 have unequal distances. This is only guaranteed if the number of nodes is odd.

- Suppose the number of nodes is even, and that the nodes with ID 1 are positioned exactly opposite to each other. Say we run a synchronous deterministic leader election algorithm, and on some round r some node v decides that it is the leader. The local view of v is always completely identical to the local view of the node v' positioned exactly opposite to v; hence, on round r, the node v' also decides that it is the leader.
- The unequal distances let us design the following uniform algorithm.

Algorithm 2 Leader Election (two nodes at unequal distances have ID 1, others have ID 0)

```
1: r_self := 0
2: r_other := 0
3: if own ID is 1 then
4:    send "relay" to the counterclockwise neighbor
5: for arbitrarily many rounds do
6:    upon receiving "relay" from the clockwise neighbor do
7:       if own ID is 1 then
8:          r_self := r, where r is the current round number
9:          send r to the counterclockwise neighbor
10:      else
11:         forward "relay" to the counterclockwise neighbor
12:   upon receiving a round number r from the clockwise neighbor do
13:      if own ID is 1 then
14:         r_other := r
15:      else
16:         forward r to the counterclockwise neighbor
17:   if r_self ≠ 0, r_other ≠ 0 and r_self < r_other then
18:      send "done" to the counterclockwise neighbor
19:      output "I am the leader" and terminate
20:   upon receiving "done" from the clockwise neighbor do
21:      forward "done" to the counterclockwise neighbor
22:      output "I am not the leader" and terminate
```

Consider the unique shortest path whose first and final endpoints v and v' are the two nodes with ID 1. The r_{self} value of v' will be lower than the r_{self} value of v. Hence, v' will decide it is the leader after receiving the r_{self} value of v and storing it as its r_{other} value. The node v' will then start a chain reaction of "done" messages which will cause every node to decide.

2 Distributed Computation of the AND

a) Assume for contradiction that some algorithm A solves the problem. Examine what happens on a ring of $n \geq 3$ nodes, where all inputs are 1. In any round, the states of all nodes are identical (by induction, as in the proof of Lemma 3.4 in the lecture notes). Observe that these states do not depend on the size n of the ring, as the algorithm is uniform and the local topology is independent of n. Thus, there must be some constant round number r that does not depend on n such that all nodes terminate and output 1 in round r (as we assumed the algorithm to be correct).

Now run A on a ring of size $2r+2$, where exactly one node v has 0 as its input bit. Consider the node v' opposite to v; all nodes at a distance of at most r to v' have the input 1. Hence (again analogously to Lemma 3.4), until round $r+1$ begins, the node v node will have the same state it would have in a ring where all nodes have the input 1. This will make v' terminate and output 1 in round r, contradicting the assumption that A is correct and should output 0 at all nodes.

b) The nodes propagate AND accumulations in both directions, attaching to the accumulations hop counters which indicate how many inputs have been accumulated. An accumulation with the hop counter n is the AND of all input bits. Eventually, some node v receives such an accumulation. The node v cannot silently output the accumulation and halt; that would break the message relaying system which causes every node to decide. So, before halting, the node v starts a chain reaction which causes every node to decide and halt.

Algorithm 3 AND in the Ring: asynchronous, non-uniform (n is the number of nodes)

```
1: send (b_own, 1) to both neighbors, where b_own is the own input bit
2: upon receiving a message (b, h) from a neighbor do
3:    if h = n then
4:       send (out, b) to both neighbors
5:       output b and terminate
6:    else
7:       forward (b ∧ b_own, h + 1) to the other neighbor
8: upon receiving a message (out, b) from a neighbor do
9:    forward (out, b) to the other neighbor
10:   output b and terminate
```

c) The following synchronous algorithm efficiently computes the AND. Consider a node with the value 0 to be "on fire." Each round, a node catches fire if an adjacent node is on fire. By induction, after $\lfloor \frac{n}{2} \rfloor$ rounds, a node is on fire if and only if some node was initially on fire.

Algorithm 4 AND in the Ring: synchronous, non-uniform (n is the number of nodes)

```
1: val := b_own, where b_own is the own input bit
2: for ⌊n/2⌋ rounds do
3:    if val = 0 then
4:       send 0 to both neighbors (do this only once)
5:    upon receiving 0 from a neighbor do
6:       val := 0
7: output val and terminate
```

It is not possible to reduce the round complexity below $\lfloor \frac{n}{2} \rfloor$. A node v must indeed receive information from every other node before computing the AND, which takes at least a number of steps equal to the diameter $D = \lfloor \frac{n}{2} \rfloor$ of an n node ring.