Computer Science Department, UWO

CS9668
Assignment 2

Linxiao Wang
250888611

October 26, 2018

Problem 1

Description of the algorithm

For this question, we can use a similar approach as the second LeaderElection algorithm introduced in class. In the following description, we refer to the ID in the received message id_m, and the ID of the current processor id_p.

1 Every processor doesn't know whether it is the Leader of the network or not, so it starts to send a message to it's right neighbour with its own ID and a distance d = 1.

2 After receiving a message from its left neighbour, if id_m is smaller than id_p, the processor will discard the message. If id_m is larger than id_p, the processor will know that it is not the Leader. If the distance in the message is 1, the processor will send the message back to its left neighbour without a distance, else the processor should forward the message to its right neighbour with the distance minus 1.

3 After receiving a message from its right neighbour, if id_m does not equal to id_p, then the processor should forward the message to its left neighbour, otherwise, the processor should send a message with its own ID with a distance $d = 2d$ to its right neighbour.

4 If a processor receives a message from its left neighbour and id_m is equal to id_p, then the processor knows that it is the Leader. The leader should read the distance in the

message(dist), and the number of the processors should be $\#p = d - dist + 1$. The leader should send an "END" message with $\#p$ to its right neighbour and return $\#p$.

5 If a processor receives an "END" message, it knows that it is not the leader and it can read $\#p$ from the message. It should forward the message to its right neighbour and return $\#p$.

Implementation

See file CountProcessorsRing.java.
At the end of the algorithm, every processor will return the number of the processors in the network.

Complexity computation

We can separate the whole process into phases, in each phase, a message is sent from a processor and come back to the processor from its right neighbour. So starts from phase 0, in phase i , the initial distance in the message is 2^i which means there will be $2^i * 2 = 2^{i+1}$ rounds in that phase. So the total number of rounds is $\sum_0^k 2^{i+1} = 2^1 + 2^2 + \cdots + 2^{k+1} = 2^{k+2} - 2 = 2^2 * 2^k - 2$. When $n >= 2^k$ the program will terminate, to make it simple we assume $n = 2^k$.
Time complexity is $2^2 * 2^k - 2 + n = 4n - 2 + n = 5n - 2$ **which is** $O(n)$**.**
In each phase, the number of active processors will be at most $\frac{n}{2^{i-1}+1}$. So the number of messages sent each phase is at most $2^{i+1} * \frac{n}{2^{i-1}+1}$. And in the end, the largest id will be send around all processors and a END message will be sent around all the processors, that is another 2n messages.
Communication complexity is $2n+\sum_0^k 2^{i+1}*\frac{n}{2^{i-1}+1} < 2n+\sum_0^k 2^i*\frac{n}{2^{i-1}+1} = 2n+4nk = 2n+4n\log n$ **which is** $O(n \log n)$**.**

Problem 2

Description of the algorithm

In the following description, we refer to the processor who doesn't have a left neighbour the leftmost node, the processor who doesn't have a right neighbour the rightmost node, all the processors in between the middle nodes. Also, we refer to the ID in the received message id_m, and the ID of the current processor id_p.

1 At the beginning of the algorithm, every processor determines whether it is the rightmost node or leftmost node or a middle node.

2 Then the leftmost node sends its ID to its right neighbour.

3 For every middle node, after receiving a message from its left neighbour, the processor send the larger one between id_m and id_p to its right neighbour.

4 When the rightmost node receives a message, it compares the id_m with id_p, if id_m is larger, then the processor knows it is not the leader, else it is the leader. Then the processor send the larger one between id_m and id_p to its left neighbour, and return the status.

5 For every middle node, after receiving a message from its right neighbour, the processor compare id_m with id_p, if id_m is larger, then the processor knows that it is not the leader, else it is the leader. Then the processor forward the message to its left neighbour and return the status.

6 When the leftmost node receives a message, it compares the id_m with id_p, if id_m is larger, then the processor knows it is not the leader, else it is the leader, then return the status.

Implementation

See the LeaderElectionLine.java file.

Proof of termination

- Because its a line network, so there will be one and only one leftmost processor. Start from the leftmost processor, every processor will send a message to its right neighbour, so the right most processor will receive a message, determine its status and terminate according to step 4 above.
- The rightmost processor will send a message to its left neighbour, and the message will be forwarded to all the left neighbours by all the middle processors.
- According to step 5 and 6, all the middle processors and the leftmost processor will receive a message from their right neighbours, so they all will terminate.

Proof of correctness

- Start from the leftmost processor, every processor will send a message contains the largest ID so far. So when the rightmost processor receive a message, it will contain the largest ID from all the processors except the rightmost one.
- The rightmost one can compare the largest one from all the processors from its left with its own, so it will know whether it is the leader or not.
- Then the rightmost processor will send the largest ID among all the processors to its left neighbour, and the largest ID will be forwarded to all the processors. All the other processors will compare that largest ID with their own and know whether they are the leader or not.

Complexity computation

Here we assume n is the number of the processors in the network.
Start from the leftmost processor, one message will be send per round, until the right most processor get an message. This will take $n-1$ round and $n-1$ messages are sent.
Then a message will be sent from the rightmost processor to the leftmost processor, it will take $n-1$ rounds and $n-1$ messages are sent.

Time complexity: $2(n-1) = 2n-1$ which is $O(n)$.
Communication complexity: $2(n-1) = 2n-1$ which is $O(n)$.

Problem 3

Description of the algorithm

We refer to the ID in the received message id_m, the ID of the current processor id_p and the largest id a processor has seen so far id_l.

1 The first step of the algorithm is to find the largest id in each row. So all the processors in the left cloumn will start to send its id to the right. After getting a message from the left, a processor will choose the larger id between id_m and id_p and send it to the right.

2 When all the processors in the right cloumn gets the largest id from its row, the top-right processor will start to send id_l to the bottom processor. After getting a message from the top, a processor will send the larger one between id_m and id_l to the bottom.

3 When the right-bottom processor get a message from the top, it will know the largest id in the whole grid. It can determine it's status and send the message to top and left and return.

4 Each processor in the right column will get the largest id and determine its status and send the id to the top and left and return,

5 Other processors will determine their status when getting the largest id, send it to the left and return.

6 When the processors in the left column get the largest id, they will determine their status and return.

Implementation

See the LeaderElectionMesh.java file.

Proof of termination

There are 3 types of processors in this algorithm, the right column, the left column and all the other processors. We gonna prove by case how these three types of processors will terminate.

1 There's only one bottom-right processor, so after it receives the largest id, it will start to send the id to the other processors.

2 For all the right column processors, after sending the message to their top neighbour if any, they will set the roundLeft to 1. Since all the right column processors except the top one will have a top neighbour, they will have roundLeft 1. If a right column procssor doesn't have a top neighbour(the top processor), or the roundLeft is 1(all the other ones), it will send the message to it's left neighbour, and set the roundLeft to 0. When the roundLeft is

0, a processor will return the status and stop. So all the right column processors will have a 0 roundLeft eventually.

3 For the left column processor, after receiving any message, they will decide their status and set the roundLeft to 0.

4 For all the other processor, after receiving a message from their right neighbours, they will prepare a message to send to the left, after send a message to their left neighbours, they will set the roundLeft to 0.

5 When the roundLeft reachs 0, a processor will terminate, from the above steps, we can see that all the processors will have a 0 roundLeft, so they all will terminate.

Proof of correctness

Evey processor has a unique id, so only one of them is going to be elected as Leader.

1 Start with the left column processors, all the processor will send the largest id so far to its right. So the right column processors will get the largest ids of their row.

2 Then start with the top-right processor, each right column processor will send the largest id so far to its bottom neighbour. So that the bottom-right processor will get the largest id of the grid.

3 A message contains the largest id will be sent from the bottom-right processor to the right column processors, so all the right column processors will see the largest id, after comparing the largest is and their own ids, the right column processors will know their correct status.

4 Then the right row processors will forward the largest to the left and the messages will be forwarded to all the processors in each row, so all the processors will know whether or not they are the leader. Here all the processors will know the correct status.

Complexity computation

Time complexity: $O(W + L)$
Getting the largest id in each row will take L rounds.
Then it will take W rounds for the bottom-right processor to get the largest id in the grid.
It will take W rounds for all the right row processors the get the largest id.
Then it will take L rounds to pass the largest id to the left row.
So the total number of rounds is $2W + 2L$ which is $O(W + L)$**.**
Communication complexity: $O(WL)$
To get the largest id of each row, it will need $(L - 1) \times W$ messages.
Then it will need $W - 1$ messages to get the largest id to the bottom-right.
It will follow the same path backward to send the largest id to all the processors, so there are another $(L - 1) \times W + (W - 1)$ messages.
So the total messages that are sent is $2((L - 1) \times W + (W - 1)) = 2WL - 2$ which is in the order of $O(WL)$**.**