CS 9868/4438 Internet Algorithmics

Solution for Assignment 2

1. *Design and implement in java a distributed synchronous algorithm for counting the number of processors in a network with a ring topology.*

- *Give an informal, high level description of the algorithm in English.*

  In the first round each processor sends a message with its own ID to its right neighbour and initializes a counter to 1. Then, whenever a processor receives a message with an ID different from its own, it increases its counter and forwards the message to its right neighbour in the next round. The algorithm ends when a processor receives a message with its own ID as then it has received messages from all other processors and, thus, the value of its counter indicates the total number of processors in the network.

- *Submit a java implementation of your algorithm.* ***Note****. Try statement was not included.*

```
public int countProcessors(String id) {
    Vector<String> v = neighbours();                    // Set of neighbours of this node.
    String rightNeighbour = (String) v.elementAt(1);    // First neighbour will be assumed
                                                        // as neighbour on the right.
    Message mssg = makeMessage(rightNeighbour, id);  // Send ID to right neighbour
    Message m;
    int count = 1;
    while (waitForNextRound()) {
        if (mssg != null) send(mssg);
        mssg = null;
        m = receive();
        if (m != null)
            if (equal(m.data(),id))                     // Processor got its own message back
                return count;
            else {
                mssg = makeMessage(rightNeighbour,m.data());
                ++ count;   // A message with a different ID from own was received and counted
            }
        }
    }
}
```

- *Compute the time complexity and communication complexity of your algorithm.*

  In the first round a processor sends a message with its own ID to its right neighbour. After n rounds the message gets back to the processor and it terminates. Therefore, the total number of rounds is n and so the time complexity is $O(n)$.
  Since in each round each processor sends a message, then in each round n messages are sent. The algorithm performs n rounds, to the total number of messages that are sent by all the processors is n^2, which is $O(n^2)$.

2. *Design and implement in java a synchronous distributed algorithm for solving the leader election problem on a line. Give an informal, high level description of the algorithm in English.*

   The leftmost processor sends in the first round a message with its own ID to its right neighbour. Whenever a processor receives a message from its left neighbour it compares the ID in the message with its own ID and selects the larger of the two. In the following round this processor sends a message with the larger ID to its right neighbour, unless the processor is the rightmost one, in which case the larger ID selected is the largest in the entire network.

Once the rightmost processor has determined the value of the largest ID in the network, it compares its own ID with the largest ID to determine whether it is the leader; then in the next round the processor sends a message to its left neighbour containing the largest ID and then it terminates. Whenever a processor receives a message from its right neighbour, it compares the ID contained in the message with its own: if the ID in the message is larger then the processor sets is status to “not leader”, otherwise it sets it to “leader”. The processor forwards the message to its left neighbour (if any) in the next round and then it terminates.

- *Submit a java implementation of your algorithm. **Note**. Try statement was not included.*

```
public String leaderElection(String id) {
    Vector<String> v = neighbours();      // Set of neighbours of this node.
    String leftNeighbour = (String) v.elementAt(0), rightNeighbour = (String) v.elementAt(1);
    boolean leftmostProcessor = false, rightmostProcessor = false;
    Message mssg = null, m;
    String status = "unknown", direction;
    // The leftmost processor sends its ID to the neighbour on the right in first round.
    if (equal(leftNeighbour,"0")) {
            leftmostProcessor = true;
            mssg = makeMessage(rightNeighbour, id);

    }
    if (equal(rightNeighbour,"0")) rightmostProcessor = true;
    while (waitForNextRound()) {
            if (mssg != null) {
                send(mssg);
                 if (equal(mssg.destination(),leftNeighbour)) return status; // End after sending message to left
            }
            mssg = null;
            m = receive();
            if (m != null)
                 if (equal(m.source(),leftNeighbour)) {                          // Message traveling to the right
                     if (rightmostProcessor) direction = leftNeighbour;
                     else direction = rightNeighbour;
                     if (larger(m.data(),id)) {  // Received a message with larder ID than own
                         mssg = makeMessage(direction,m.data());
                         status = "not leader";
                     }
                     else {  // Message received has smaller ID than own, send message with own ID to the right
                         mssg = makeMessage(direction,id);
                         if (rightmostProcessor) status = "leader";              // Rightmost processor has the largest ID
                     }
                }
                 else {                                                          // Message traveling to the left
                         if (equal(id,m.data())) status = "leader"; else status = "not leader";
                         if (leftmostProcessor) return status;
                         else mssg = makeMessage(leftNeighbour,m.data());
                }
        }
}
```

- *Prove that your algorithm terminates.*

In the first round the leftmost processor sends a message with its own ID to its right neighbour. In the following rounds, whenever a processor receives a message from its left neighbour, it will in the following round pass a message to its right neighbour containing the larger between its own ID and the ID in the message that it received.

Since messages are always passed top the right, after n-1 rounds a message reaches the rightmost processor. After the rightmost processor receives the message, in the following rounds a message with the largest ID will travel to the left. This message is always forwarded, so every processor will receive it. After a processor sends a message to its left neighbour, it terminates. The leftmost processor terminates as soon as it receives a message from its right neighbour. Therefore, all processors will terminate.

- *Prove that your algorithm produces the correct output.*

The first message sent by the algorithm travels to the right and it contains the ID the leftmost processor. As the message travels to the right, each processor p compares its own ID with the one stored in the message and it selects the larger of the two, which is then forwarded to the right neighbour. Notice that then the ID selected by p is the largest ID among all the processors in the network between the leftmost processor and p. Thus, when the message reaches the rightmost processor, the ID in it must be the largest in the network.
Once the rightmost processor knows the largest ID it sends a message containing this ID which will travel to the left. This message is always forwarded so every processor will receive it. Hence, each processor will compare its ID with the largest one. If the ID's are the same the processor correctly changes its status to "leader" and otherwise it changes its status to "not leader". Since all ID's are different, only one processor will have its status set to "leader" and all others will have it as "not leader".

- *Compute the time complexity and communication complexity of your algorithm.*

The algorithm needs n-1 rounds for the message initially sent by the leftmost processor to reach the rightmost one. Then another n-1 rounds are needed for the message containing the largest ID to be sent to all the processors. Therefore, the total number of rounds is 2(n-1), which is O(n).
In each round the algorithm sends only one message, so the total number of messages sent is 2(n-1), which is O(n).

3. *Design and implement in java an algorithm for solving the leader election problem on a mesh*.

- *Give an informal, high level description of the algorithm in English.*

Each processor in the first column of the mesh sends in the first round a message containing its own ID to its right neighbour. When a processor receives a message from its left neighbour, it compares its own ID with the ID in the message and keeps the larger one, which is forwarded to the right neighbour (if any) in the next round. When the messages arrive to the processors in the last column of the mesh, they will know the largest ID in their row.
In the second part of the algorithm the processor at the top of the rightmost column sends a message with the largest ID in its row to the neighbour below it. When a processor in the last column receives a message from the processor above it, it compares the largest ID in its row with the ID in the message and keeps the larger of the two; this larger ID is put in a message and forwarded to the processor below it (if any). When the processor at the bottom of the rightmost column receives a message from the neighbour above it, it will know the largest ID in the entire network.
This processor then compares its own ID with the largest one to determine if it is the leader, changing its status accordingly. A message with the largest ID is then sent to the neighbour on the left and the one on top. When a processor receives a message from its neighbour on the right or below, it compares its own ID with the ID in the message to determine whether it is the leader and changes its status accordingly. Messages received from the right are forwarded to the left and messages received from the bottom are forwarded to the top. A processor terminates the execution of the algorithm after forwarding messages to the left and, if needed, to the top.

- *Submit a java implementation of your algorithm*. ***Note****. Try statement was not included.*

```
public String electLeader(String id) {
   Message mssg = null, m = null, mssgUp = null; // Messages might be sent up and to the left in the same round
   Vector<String> v = neighbours();                  // Set of neighbours of this node.
   String topNeighbour = (String) v.elementAt(0), rightNeighbour = (String) v.elementAt(1), bottomNeighbour =
                         (String) v.elementAt(2), leftNeighbour = (String) v.elementAt(3);
   String destination, status = "unknown, largestId = id;              // Largest ID seen so far is own processor's ID

   // The leftmost processor in each row sends its ID to the neighbour on the right
   if (equal(leftNeighbour,"0")) mssg = makeMessage(rightNeighbour, id); // First processor in a row of the mesh
   while (waitForNextRound()) {
         if (mssgUp != null) send(mssgUp);
         if (mssg != null) {
            send(mssg);
            if (equal(mssg.destination(),leftNeighbour)) return status; // Terminate after sending largest ID left
          }
          mssgUp = null; mssg = null;
          m = receive();
          if (m != null) {
              if (equal(m.source(),leftNeighbour)) {                                  // Message moving to the right
                   if (!equal(rightNeighbour,"0")) destination = rightNeighbour;        // Right neighbour exists
                   else if (equal(topNeighbour,"0")) destination = bottomNeighbour; // Uppermost, rightmost proc.
                   else destination = "none";                    // Processors in last column of mesh wait for messages
                   if (larger(m.data(),id)) {                    // ID in message is larger than own ID, save larger ID
                        largestId = m.data();
                        status = "not leader";
                    }
                    else if (equal(rightNeighbour,"0") && equal (bottomNeighbour,"0") &&
                             equal(topNeighbour,"0")) status = "leader";        // Mesh is a line
                    if (!equal(destination,"none"))
                        mssg = makeMessage(destination,largestId);
             }
             else if (equal(m.source(),topNeighbour)) {                          // Message moving down
                       if (larger(m.data(),largestId)) {                         // Compare own ID with message's ID
                            largestId = m.data();
                            status = "not leader";
                       }
                       else if (equal(bottomNeighbour,"0"))  status = "leader"; // Bottom right processor has largest ID
                       if (!equal(bottomNeighbour,"0")) destination = bottomNeighbour;
                       else {                                                    // Send message up and to the left
                           destination = leftNeighbour;
                            mssgUp = makeMessage(topNeighbour,largestId);
                       }
                       mssg = makeMessage(destination,largestId);
              }
              else {                                                             // Message moving left or up
                   if (equal(rightNeighbour,"0") && !equal(topNeighbour,"0"))  // Send message up
                        mssgUp = makeMessage(topNeighbour,m.data());
                   else mssgUp = null;
                   if (equal(m.data(),id)) status = "leader"; else status = "not leader";
                   if (equal(leftNeighbour,"0")) return status;                  // Processor in first column of mesh
                   else mssg = makeMessage(leftNeighbour,m.data());
              }
         }
    }}
```

- *Prove that your algorithm terminates. The number of processors in each row of the mesh is L and the number of processors in each column is W.*

  Observe that the algorithm never discards any messages. Messages move from the processors in the leftmost column to the right. After L-1 rounds the messages arrive to the rightmost column. A message then travels from the top of that column to the bottom and it arrives at the bottom-right processor after W-1 rounds. Messages then travel to the left and up. After a processor forwards a message to the left, it terminates. Hence, after the largest ID has been determined by the bottom-right processor in the mesh, L+W-2 additional rounds are needed for all the processors to terminate the execution of the algorithm.

- *Prove that your algorithm produces the correct output.*

  Note that during the first part of the algorithm messages travel along the rows of the mesh from the left to the right. Each processor compares its own ID with the ID in the message that it receives, and the larger ID is forwarded to the right in the following round. This part of the algorithm is the same as the algorithm for the previous question, so using the same argument as above we know that after L-1 rounds each processor in the rightmost column knows the largest ID of the processors in its row.
  In the second part of the algorithm the top-right processor sends a message with the largest ID in its row to the neighbour below it. Each processor in the last column compares the largest ID in its row with the ID in the message that it receives. This is again the same algorithm as the leader election on a line, so by the same argument in the last question we know that after W-1 rounds the bottom-right processor determines the largest ID in the entire mesh.
  In the last part of the algorithm messages with the largest ID in the network are forwarded to all the processors in the mesh. Each processor compares its own ID with the largest ID to determine whether they are the leader or not and changes its status accordingly. Hence, when the algorithm terminates only one processor will have its status set to “leader” and the other ones will have it set to “not leader”.

- *Compute the time complexity and communication complexity of your algorithm.*

  The algorithm needs L-1 rounds for the messages to travel from the processors in the first column to the processors in the last column. Then, W-1 rounds are needed for a message to travel from the top processor of the rightmost column to the bottom processor. Finally, L+W-2 additional rounds are needed for the messages containing the largest ID in the network to be delivered to all the processors in the system. Therefore, the total number of rounds is 2(L+W-2), which is O(L + W).
  In the first L-1 rounds of the algorithm one processor in each row sends one message; hence, W(L-1) messages are sent. In the next W-1 rounds only one message is sent per round for a total of W-1 messages.
  Finally, in the last part of the algorithm one message is sent to each one of the processors in the mesh (except the bottom-right processor), for a total of WL-1 messages.
  Therefore, the total number of messages sent by the algorithm is W(L-1)+W-1+WL-1, which is O(WL) or O(n), as n = WL is the number of processors.