Dept. of Computer Science and Engineering,

CS69011: Computing Lab 1

Assignment 5

Maximum Marks: 100

Leader Election is a common problem in distributed computing. For example, in a distributed database if the process coordinating a distributed transaction dies, then the remaining participating processes may elect a new transaction coordinator. There are many problems which require a leader to be elected to coordinate some activity among a set of processes.

In this exercise, we shall consider leader election among a set of agents arranged in the form of a given tree, where the vertices represent the agents (to be modeled by processes) and the edges represent bi-directional pipes through which the two agents who are incident on an edge may communicate. The format of the tree has been provided later. You have to read the tree structure and create the network of processes by repeated forking.

Each agent has an identifier (an integer), which is also specified in the tree structure (you should not confuse this ID with the process ID of the process representing that agent). The leader election algorithm is as follows:

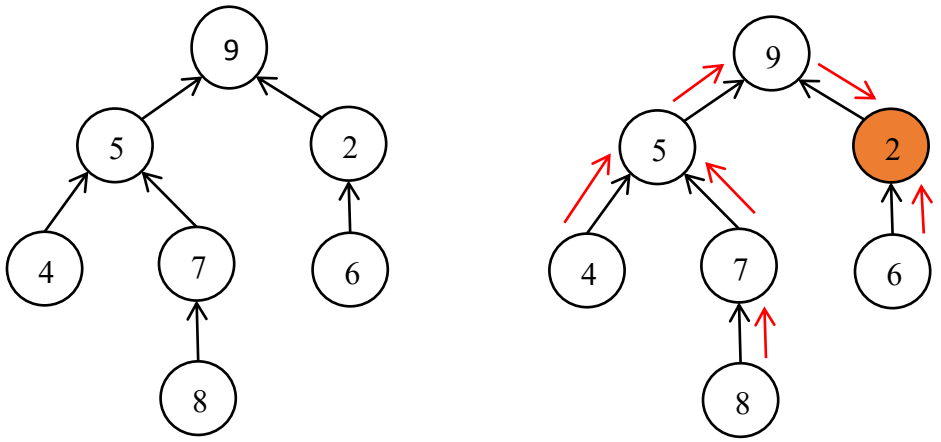
```
var ws_p    : Boolean init false;
    wr_p    : integer  init 0;
    rec_p[q] : Boolean   for each q ∈ Neigh_p init false;
    v_p : P init p
    state : (sleep, leader, lost) init sleep;

begin
    If p is the initiator then
    begin ws_p = true  for each q ∈ Neigh_p send <wakeup> to q  end

    while wr_p < #Neigh_p do
    begin     receive ⟨ wakeup ⟩ ; wr_p = wr_p + 1 end ;
              if not ws_p then
                 begin
                    ws_p = true  for each q ∈ Neigh_p send <wakeup> to q
                 end
    end

        while #{q : ¬ rec_p [q]} > 1 do
           begin
                    receive <tok, r> from q; rec_p [q] = true;
                    v_p= min (v_p, r);
            end
            send <tok, v_p> to q_0 with ¬ rec_p [q_0]
            receive <tok, r> from q_0
            v_p= min (v_p, r);
            if (v_p = p) then state = leader else state = lost;
            for each q ∈ Neigh_p, q ≠ q_0 send <tok, v_p> to q;
end
```

The algorithm elects the agent having the minimum identifier as the leader. At the end, each agent is aware of the leader, including the leader itself of course.



The figure on the left shows the process tree (with edges from children to parent). Once the leader is identified, the agents may redefine their parents so as to point towards the leader (shown on the right). Note that the process hierarchy remains the same, the red links are maintained by the agents to remember the direction to the leader.

The structure of the process tree is provided as an input, read by the root process. The input consists of a list of edges of the form ⟨ *node, parent* ⟩. For the figure shown above, the tree may be specified as:

```
7,5
4,5
6,2
8,7
5,9
2,9
```

Bidirectional pipes must be created for each of these process pairs. Once the process network is constructed, we will assume that the nodes do not know the structure, nor the IDs of the agents other than its own.

Task-1:
Develop a mechanism for creating the process network for a given input and the set of pipes. Then run the leader election algorithm. At the end of this algorithm, each process, J, must write the identity of the leader, K, using the following statement:

```
I am agent J and my leader is K
```

Task-2:
Now the leader wants to find out the sum of the IDs of the agents. Use a variant of the previous algorithm to perform this task. At the end, the leader must print the sum of the IDs of the agents.

Task-3:
Finally, the leader asks each process to terminate. Each process announces its intention to terminate just before termination. Each parent in the process hierarchy must wait for its children to terminate before announcing its own intention for termination.