University of Freiburg
Dept. of Computer Science
Prof. Dr. F. Kuhn
S. Faour, A. Mályusz

Theory of Distributed Systems
Sample Solution Exercise Sheet 2

Due: Wednesday, 8th of May 2023, 12:00

The CONGEST model

The CONGEST model is a **synchronous** message passing model where the **size** of each message is bounded, i.e., the bitstring representing a message consists of at most $O(\log n)$ bits, where n is the number of nodes (do not confuse message size with message complexity). Further, we assume that each node is initially equipped with a unique ID in $\{1, \dots, n^3\}$.

Note that sending such an ID requires at most $\lfloor \log_2 n^3 \rfloor + 1 = O(\log n)$ bits. However, a node can not send the IDs of all its neighbors in a single message, as the degree of the node could be large.

Exercise 1: Leader Election *(10 Points)*

a) Given a graph G, describe a deterministic algorithm in the CONGEST model such that every node learns the smallest ID in the graph (hence, the node with the smallest ID is our leader) and *terminates* after $O(D)$ rounds. You may *not* assume that nodes initially know D. Note that after the execution each node should know the ID of the leader as well as some parent node that points into the direction of the leader.

   Hint: Try to provide a spanning tree algorithm assuming that the ID of the leader is already known to each node and in the next step try to use this algorithm to solve the actual task.

b) Analyze the message complexity of the algorithm in terms of n. Show that your bound is tight i.e., if $B(n)$ is your message complexity, give a family of graphs where your algorithm indeed has a message complexity of $\Omega(B(n))$.

Sample Solution

a) Consider the following algorithm for building a spanning tree in the case where a leader is *already provided*. We assume that G has more than one node, otherwise this is trivial. The leader starts by sending a *parent request* (P) message to each neighbor. When a node receives a P message from some node u for the first time it sends back an *acknowledgement* (ACK) message and forwards P to all its neighbors except u. For all P requests that are received later, a node sends back an *negative aknowledgement* (NACK) message. If two requests are received at the same round, an arbitrary one is chosen to be the first. A node considers itself to be a leaf when it accepted a parent request, and it received a NACK from all its other neighbors, or there are no others (parent is the only neighbor). When a node becomes a leaf, it sends a *termination* (T) message to its parent. After a node received either a T or a NACK message from all neighbors it sends a T message to its parent. When the leader receives a T from all neighbors, the root broadcasts a *stop* (S) message to all children and stops. When a node receives S from the parent, it sends S to all children and stops. The final tree consists of all edges over which ACK was sent. This spanning tree construction terminates within $O(D)$ rounds, since it takes at most D rounds until all nodes know their parent and can send T, an additional D rounds are required until the leader receives

T from all its neighbors and finally D rounds are enough to send S. Further, the messages ACK, NACK, T and S can be encoded as $O(1)$-bit strings.

In order to perform leader election, we can do the following. Each node pretends to be the leader and starts running the above algorithm (hence we have many different executions of the same algorithm that are done in parallel). Also, each message is marked with the ID of the node that started the execution of the algorithm. At the same round, nodes may receive messages that belong to different executions (messages marked with different ID's). Each node keeps track of the smallest identifier ever seen, and *ignores* messages that do not belong to the execution regarding the smallest identifier ever seen. When an S message is received, a node knows that the leader is the node with the smallest identifier ever seen (and can send S to the neighbors and stop).

The idea behind the algorithm is the following: if some node ignores the messages of an execution initiated by some root r, then this execution will never terminate. In particular, the corresponding root r will never receive the T message from all its children. To see that, consider two executions started from nodes with identifiers x and y such that $x < y$. Since the system is synchronized, there must be some node v where a message of the execution of x arrives *not later than* that of y. Then this node v will henceforth ignore the execution with ID y. In particular it will never send the T message for that execution. But than the T message of its branch in the tree of rooted at y can never reach y, thus y will neither terminate nor conclude that it is the leader. Consequently, the only execution whose root will terminate and announce itself the leader is that of the node with minimum ID.

b) Notice that, while there may be multiple executions performed at the same time, each node at each round will participate in a single execution, hence at most one message for each edge is sent in each direction. Since the original (spanning tree construction) algorithm works in the CONGEST model, the leader election one works in the CONGEST model as well. The running time of the whole algorithm corresponds to the spanning tree algorithm of the node with minimum identifier, because its messages are never delayed by any other node and after this execution is done, every node terminates. Since that execution takes $O(D)$, also the whole algorithm takes $O(D)$.

   Regarding the message complexity, an upper bound can be given by multiplying the running time with the number of edges m, that is $O(Dm)$, that can be $O(n^3)$ in the worst case. There actually is a family of input instances where $\Omega(n^3)$ messages are sent, hence the bound is tight. Such input instances are composed of a path of size $n/2$ connected to a clique of size $n/2$, where the identifiers are assigned in an increasing order while starting from one endpoint of the path and going towards the clique. Such an execution takes $O(n)$ rounds, and each round all pairs of nodes of the clique send a message to each other.

Exercise 2: k-smallest ID problem *(10 Points)*

Given a graph G with n nodes that have ID's as described in the CONGEST model definition. In order to solve the k-smallest ID problem in the distributed setting for some $k \leq n$, the k^{th}-smallest ID in the graph needs to be announced by exactly one node.
Our goal is to describe a distributed algorithm in the CONGEST model that always solves the k-smallest ID problem with a runtime of $O(D \cdot \log n)$ rounds.
Note that our goal is to construct a deterministic algorithm, however, if you come up with an randomized algorithm that solves the problem in the same number of rounds (in expectation) you will also get full points.

a) Give an $O(D)$ round algorithm that computes a spanning tree T on a graph G, such that the root of T knows the minimum and the maximum ID of the nodes in G.

   Hint: You can adapt the algorithm from Task 1.

b) Assume the setting of a) where the root is given an additional value x. Give an $O(D)$ round algorithm that counts the number of nodes with $ID < x$, with $ID > x$ and $ID = x$. It is sufficient if the root node of the tree knows these values in the end.

c) Assume the setting from b) where each node v additionally has a boolean $b_v \in \{0,1\}$ as input. In the following we call a node active if $b_v = 1$, else we call it inactive. Modify the algorithm of b) such that the root knows the number of **active** nodes that have $ID < x$, $ID > x$ and $ID = x$.

d) Describe an algorithm that solves the k-smallest ID problem in time $O(D \cdot \log n)$.

Hint: In the beginning we have an ID space of $\{minID, ..., maxID\}$. Try to gradually decrease the size of the ID space in a way that the k-th smallest ID always remains within the remaining ID space. Nodes with an ID that is not in the space can be made inactive for the subsequent steps.

Sample Solution

a) We can compute the spanning tree with root node using the algorithm from exercise 1. For sure here the root node already knows the minimum ID. To get the maximum ID we can use a converge cast s.t. leaf nodes forward their ID and non-leaf nodes forward the maximum ID of their children and of the own ID. This costs an additional $O(D)$ time.

b) First, the root node floods the value x to all nodes in the spanning tree. Each node will then initialize 3 counters, let's call them k_v, l_v, m_v, where k_v is the number of nodes in the subtree that have an ID smaller than x (l_v the number of nodes that have an ID equal to x and m_v the number of nodes that have an ID larger than x). Each node sets k_v, l_v or m_v according to the own ID i.e. either k_v, l_v or m_v is set to 1, the other two variables are set to 0. Then the leave nodes will send these three values to their parent. They then update their values accordingly, adding their own values together with these of the children ... until the root node knows all values. Note that all 3 variables contains a number of size at most n, hence we can encode these as a $O(\log n)$ bit size message. Surely, this procedure takes $O(D)$ rounds.

c) Active nodes set their counters k_v, l_v or m_v as in b) while inactive nodes set all of them to 0. Besides that, no further changes are required.

d) First we compute a spanning tree T as in a) and set all nodes active $b_v := 1$. We know that at this point the minimum ID is ≥ 1 and the maximum ID is $\leq n^3$. Then the algorithm operates in phases, where in each phase more nodes become inactive by setting $b_v = 0$ in order to reduce the ID space. We repeat these phases until the k^{th} smallest ID is announced and all nodes are ordered to terminate. In each phase we do the following steps (high level pseudo code):

1. Let the current ID space be $\{y, ..., z\}$ where y and z are known by the root.
2. Let $x := \lfloor \frac{y+z}{2} \rfloor$ and compute the algorithm from c) with this x as input.
3. Now the root node knows how many of the remaining active nodes have a larger and smaller ID than x.
4. If $k-1$ active nodes have smaller IDs then x and there is a node with $ID = x$, then we have found the k-th smallest ID!
5. Otherwise:
   - If $R \geq k$ nodes have an ID $\leq x$, we set $z := x$ and set all nodes inactive with ID $> x$.
   - If $R < k$ nodes have an ID $\leq x$, then $y := x$ and set $k := k - R$ and set all nodes inactive with ID $< x$.
6. Repeat the process with new ID space (and new k).

Correctness: We have to show that whenever we do not find the k-th smallest ID in step 4, we guarantee with step 5, that this ID is part of the new ID space. Clearly, if the condition of the first case holds, the k-th smallest ID is smaller then x and hence we can ignore all nodes with ID's larger then x. Thus we can replace the upper bound of our ID space by x. The second case is slightly more involved. Here we have that the the k-th smallest ID is larger then x. So we can ignore nodes with ID's smaller then x (these are actually R nodes). However, the k-th smallest element is now the $(k - R)$-th smallest element of the remaining active nodes.

Runtime Analysis: In each phase the ID space decreases by a factor of 2. Since the ID space is initially of size n^3, after $\log_2(n^3)$ phases the ID space is of size at most 1 (and hence the problem is trivial). Since a phase runs in time $O(D)$, we can guarantee termination after $O(D \log n)$ rounds.