Albert-Ludwigs-Universität, Inst. für Informatik
Prof. Dr. Fabian Kuhn
Philipp Bamberger

Distributed Systems, Summer Term 2020
Exercise Sheet 2

In the following exercises we consider the CONGEST model. This is a **synchronous** message passing model with the additional property that the **size** of each message is bounded. If we assume that the nodes have IDs in $\{1, \dots, n\}$ and communicate by exchanging bitstrings, then each message is only allowed to contain $O(\log n)$ bits. This means that each message may contain for example (the binary representation of) a constant number of integers $\leq n^c$ for some constant c. However, it is not possible that a node sends another node the IDs of all its neighbors in a single message, as the degree of the network may not be bounded.

Remark: Do not confuse the message size and the message complexity.

1. k-Selection Problem in Graphs

Given a graph G with n nodes that have pairwise distinct input values $\leq n^c$ for some constant c, the k-selection problem for a $k \leq n$ is the problem of finding the k^{th}-smallest value in the graph.
Our goal is to describe a randomized distributed algorithm in the CONGEST model that solves the k-selection problem with an expected runtime of $O(D \cdot \log n)$.

a) Assume a tree T of depth D. Describe an algorithm that computes in $O(D)$ rounds for every node v a value s_v which equals the size (number of nodes) of the subtree with root v.

b) Assume a tree T of depth D and root r in which each node is able to flip coins. Describe a method to choose a node from the tree uniformly at random (i.e., each node has the same probability to be chosen) in time $O(D)$.

Hint: Use the algorithm from a).

c) Assume a tree T of depth D, where each node v has in input a boolean b_v. Modify the algorithm of a) such that for every node v, the value s_v is equal to the number of nodes in the subtree rooted at v that have $b = \text{True}$. Also, modify the algorithm from b) to choose uniformly at random a node among all nodes that have $b = \text{True}$.

d) Describe a randomized algorithm that solves the k-selection problem with an expected runtime of $O(D \cdot \log n)$.

Hint: Use the algorithms from c).

Sample Solution

a) We recursively compute the size of each subtree as follows. At first, each leaf node v sets $s_v = 1$ and sends s_v to each parent. Then, once a non-leaf node v received a message s_{c_i} from each child c_i, it sets $s_v = 1 + \sum c_i$ and sends s_v to its parent.

b) A way to choose a node uniformly at random in a tree is the following. Each node starts by computing the size of its subtree using the algorithm of point a). Also, each node stores the size of the subtree of each child. Then, the high level idea is that we either choose the current node, or we choose a child where to recurse, and this choice will be biased depending on the size of each

subtree rooted at each child. Hence, we start from the root and we execute the following recursive procedure. The root v chooses at random a number in $[0, c]$, where c is the number of children of v, in the following way: number 0 will be chosen with probability $1/s_v$, while number i will be chosen with probability s_{c_i}/s_v. If the chosen number is 0, then the chosen node is the current node. Otherwise, if the choice is $i > 0$, we recurse with the same procedure on the i-th child of v.

c) We modify the algorithm of point a) as follows: a leaf sets s_v to 1 if b is True, while it sets $s_v = 0$ otherwise. Then, when a node received a message from each child, it sets $s_v = 1 + \sum c_i$ if b is True, and $s_v = \sum c_i$ otherwise. We modify the algorithm of point b) as follows: instead of computing the size of each subtree, we compute the number of nodes in each subtree that have $b =$ True. Then, we modify the random choice as follows: if b is False, the probability of choosing the value 0 is set to 0 instead of $1/s_v$.

d) At the beginning, all nodes set $b =$ True. The algorithm operates in phases. With the algorithm from b), the value x of some random node is chosen. With flooding/echo every node learns this value. Each node can then set a new boolean indicating if its own value is larger or smaller than x, and we can use the modified version of the algorithm of point a) presented in point c) to let the root know how many nodes have a smaller value and how many nodes have a larger value. That is, the root knows the rank of x (its position in the sorted list of all values). If $\text{rank}(x) = k$, then x is the k^{th}-smallest value. If $\text{rank}(x) < k$, this information is broadcasted and each node with a value $\leq x$ sets $b =$False. Also, in this case, k is updated to $k - \text{rank}(x)$ (that is, if we originally wanted to find the k-th value among all values, we now want to find the $(k - \text{rank}(x))$-th value among the values larger than x). If $\text{rank}(x) > k$, each node with a value $\geq x$ sets $b =$False. Next, the value y of some node having $b =$ True is chosen randomly and the process is repeated.

Runtime Analysis: Let u be the random node that is chosen in some phase. At the end of the phase, either all nodes with smaller value or all nodes with larger value set $b =$ False (or the algorithm terminates). We call a phase *good* if at least one third of the nodes with $b =$ True have a smaller value than u and at least 1/3 have a larger value than u, which means that at least 1/3 of the remaining nodes will set $b =$ False. After $\log_{3/2} n$ good phases, no node has $b =$ True and the algorithm terminates. The probability that a round is good is 1/3. It follows that after $3 \log_{3/2} n$ phases, the expected number of good phases is at least $\log_{3/2} n$. So the algorithm needs in expectation at most $3 \log_{3/2} n$ phases, each taking $O(D)$ rounds.

2. Leader Election

Given a graph G, describe a deterministic algorithm in the CONGEST model such that every node learns the smallest ID in the graph and terminates after $O(D)$ rounds. Analyse the message complexity of the algorithm.

Sample Solution

Consider the following algorithm for building a spanning tree in the case where a leader is *already provided*. The leader starts by sending a *parent request* (P) message to each neighbor. When a node receives a P message for the first time it sends back an *acknowledgement* (ACK) message. All the other times that a P request is received, 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. When a node becomes a leaf, it sends a *termination* (T) message to its parent. After a node received, from all neighbors, either a T or a NACK message, it sends a T message to its parent. When the leader receives a T from all neighbors, the tree is built and the root can broadcast a *stop* (S) message to all children and stops. When a node receives an S message from the parent, it sends S to all children and stops.
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 identifier 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 identifiers). 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 a node ignores a message of some execution, such execution will never terminate. In particular, consider two executions, started from nodes with identifiers x and y, such that $x < y$. In order for the execution started by y to stop, it is necessary that y starts sending S messages, and this is only done when all neighbors of y sent a T message to y. A node sends a T message only when all children already sent a T. Since a tree must span all nodes, then the tree that y is trying to build must include node x. Since x will ignore messages regarding the execution of y, then x will never send a T message to its parent, and thus the parent of the parent of x will never receive a T message as well, and so on. Hence, node y will never receive a T message from all children, and thus y will never send S messages. On the other hand, consider the execution started by the node with smallest identifier x. All nodes will participate in such execution, since no node will ignore such messages. Hence, at some point node x will receive a T message from all neighbors, and will start to broadcast S messages.

Also, 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, in each direction, is sent, and hence, since the original (spanning tree construction) algorithm works in the CONGEST model, the leader election one works in the CONGEST model as well. Also, we can bound the running time as follows: nodes terminate when they receive an S message regarding the execution of the smallest identifier. Since such execution takes $O(D)$, also this 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. One can check that such execution takes linear time, and at each time all nodes of the clique send a message to each other.