The CONGEST model

The CONGEST model is a synchronous message passing model where the size of each message is only allowed to contain bitstrings of length $O(\log n)$, where $n$ is the number of nodes (do not confuse message size with message complexity). Further, we assume that the nodes have IDs in $\{1,\ldots,n\}$. This means that each message may contain for example (the binary representation of) a constant number of integers $\leq n^c$ for some constant $c$ (in particular IDs). However, a node can not send the IDs of all its neighbors in a single message, as the degree of the network may be large.

Exercise 1: Leader Election

(a) 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. You may not assume that nodes initially know $D$.

(b) Analyze the message complexity of the algorithm. Show that your bound is tight.

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 acknowledgement (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 neighbour). 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.

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 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: $k$-Selection Problem in Graphs

(10 Points)

Given a graph $G$ with $n$ nodes that have pairwise distinct input values $\leq n^c$ for some constant $c$. In order to solve the $k$-selection problem in the distributed setting for some $k \leq n$, the $k^{th}$-smallest value in the graph needs to be announced by exactly one node.

Our goal is to describe a randomized distributed algorithm in the CONGEST model that always 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 $v$.

Hint: Use the algorithm from a).

c) Assume a tree $T$ of depth $D$, where each node $v$ a boolean $b_v$ as input. 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 $b_v = True$. Also, modify the algorithm from b) to choose uniformly at random a node among all nodes $v$ with $b_v = True$.

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

Hint: Use the previous algorithms.

Sample Solution

a) We can root the tree using the algorithm from exercise 1. We then 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 \( s_v \) from each child \( u \) in \( T \), it sets \( s_v := 1 + \sum u \text{ child of } v \ s_u \) and sends \( s_v \) to its parent.

b) The high level idea is that we either pick the current node, or we choose a child which has to pick a random node, and this choice will be random and biased depending on the size of each subtree rooted at each child. In more detail, each node computes the size of its subtree using the algorithm of a). Also, each node stores the size of the subtree of each child. Let \( u_1, \ldots, u_k \) be the children of \( v \). Recall that \( s_v := 1 + \sum_{i=1}^k s_{u_i} \). Starting from the root we execute the following recursive procedure. The current node \( v \) chooses a random number \( x \in \{1, \ldots, s_v\} \). We partition that interval further \( \{1, \ldots, s_v\} = I_1 \cup \cdots \cup I_k \cup \{s_v\} \), where the subintervals are \( I_j := \{1 + \sum_{i=1}^{j-1} s_{u_i}, \ldots, \sum_{i=1}^j s_{u_i}\} \) (formally \( s_{v_0} := 0 \)). If \( x = s_v \) then \( v \) chooses itself as random node (and can subsequently broadcast its identifier to everyone). If \( x \in I_j \) then \( v \) instructs the \( u_j \) to pick a random node from its subtree. Note that the the probability for each subtree is proportional to its size, which ultimately ensures that every node has the same probability to be picked.

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 \( s_v' = b_v + \sum u \text{ child of } v \ s_u \) (where we interpret True as 1 and false as 0). The algorithm of part b) uses the modified numbers \( s_v' \) instead of \( s_v \), for choosing the according child, whereas \( v \) has a chance to choose itself only if \( b_v = 1 \).

d) Let \( \alpha_v \) be the input value of each node \( v \). Initially we compute a tree \( T \) on \( G \) using exercise 1 and all nodes set a boolean variable \( \beta_v = 1 \) (True). Then the algorithm operates in phases, where in each phase more nodes become inactive by setting \( \beta_v = 0 \) in order to reduce the “search space” (number of active nodes). We repeat these phases until the \( k^{th} \) smallest element is announced and all nodes are ordered to terminate. In each phase we do the following steps (high level pseudo code):

1. Use the algorithm from c) to compute random node \( u \) among all active nodes (i.e., with \( \beta_v = 1 \))
2. Let \( x := \alpha_u \) and announce \( x \) with flooding.
3. Each active node \( v \) (i.e., with \( \beta_v = 1 \)) sets \( b_v = 1 \) if \( \alpha_v > x \) else \( b_v = 0 \).
4. Use part c) to compute \( s_v \) based on the bits \( \beta_v \) and \( s_v' \) based on the bits \( b_v \).
5. Now the root node knows how many of the remaining active nodes have a larger and smaller value than \( x \) (which is \( s_v' \) and \( s_v - s_v' \)) respectively.
6. From this the root determines \( \text{rank}(x) := s_v - s_v' \), i.e., the position of \( x \) in the sorted list of all values.
7. If \( \text{rank}(x) = k \) the root announces \( x \) and orders all nodes to terminate, else it broadcasts \( \text{rank}(x) \) to all nodes.
8. If \( \text{rank}(x) < k \) the root node updates the value \( k \) to \( k - \text{rank}(x) \) and floods the new value to all nodes. Furthermore each node \( v \) with \( \alpha_v < x \) becomes inactive by setting \( \beta_v = 0 \).
9. If \( \text{rank}(x) > k \) each node \( v \) with \( \alpha_v > x \) becomes inactive by setting \( \beta_v = 0 \).

**Runtime Analysis:** At the end of each phase, either all nodes with \( \alpha_v > x \) or those with \( \alpha_v < x \) become inactive (set \( \beta_v = 0 \)) or the algorithm terminates. We call a phase *good* if at least one third (rounded up) of the active nodes have a smaller value than \( x \) and at least one third (rounded up) have a larger value than \( x \), which means that at least 1/3 of the remaining nodes will become inactive. Let \( T \) be the number of good phases until there is at most one node left. This is true if \( T \) satisfies

\[
\begin{align*}
n \cdot (1/3)^T &< 2 \\
\iff n \cdot 3^{-T} &< 2 \\
\iff \log_3 n - T \log_3 3 &< \log_3 2 \\
\iff T &> \log_3 n - \log_3 2
\end{align*}
\]
This means that after at most $T \in \mathcal{O}(\log n)$ good phases there is at most one node left and then, at the latest, the final remaining node announces itself the solution. A phase is good with constant probability $c > 0$ (the probability of a good phase is actually a bit smaller than $1/3$ due to the rounding) so it takes $cT \in \mathcal{O}(\log n)$ rounds in expectation. Each phase takes $\mathcal{O}(D)$ rounds, so overall we are done after $\mathcal{O}(D \log n)$ rounds.