University of Freiburg
Dept. of Computer Science
Prof. Dr. F. Kuhn
Z. Parsaeian, G. Schmid

Distributed Graph Algorithms
Sample Solution Exercise Sheet 3

Exercise 1: MST with Unique Weights *(10 Points)*

Let $G = (V, E)$ be an edge-weighted graph where each edge $e \in E$ has a unique edge weight $w(e)$. Prove that this guarantees that G has only one *Minimum Spanning Tree*.

Sample Solution

Assume for contradiction that a weighted graph $G = (V, E)$ with pairwise distinct edge weights has two different minimum spanning trees T_1 and T_2. Since the trees differ, there exists an edge $e \in T_1 \setminus T_2$. Adding e to T_2 creates a unique cycle C (since T_2 is a tree). On that cycle, there must be some edge $f \in C \setminus T_1$.
Because edge weights are unique, either $w(e) < w(f)$ or $w(f) < w(e)$. But:

- T_1 is a minimum spanning tree, so replacing e by f cannot decrease its weight, hence $w(f) > w(e)$.
- T_2 is a minimum spanning tree, so replacing f by e cannot decrease its weight, hence $w(e) > w(f)$.

This yields a contradiction. Therefore, G must have a unique minimum spanning tree.

Exercise 2: Leader Election in CONGEST *(10 Points)*

Given a graph G, describe a deterministic algorithm in the CONGEST model that allows each node to learns the smallest ID in the graph in $O(D)$ rounds. This can be used to elect the smallest ID node as a leader. The protocol should terminate, i.e., each node at some point has to know that it is done. You may not assume that nodes initially know D. The protocol should also make sure that after the execution, each node should also know some parent node that points into the direction of the leader (i.e., the minimum ID node).

Hint: Start a BFS construction from each node, but make sure that only the BFS construction of the minimum ID node completes. In the BFS construction from the lecture, one can make sure that the root node is informed after the construction is finished by using a convergecast.

Sample Solution

Algorithm

Each node starts a BFS search using its own ID as the "source-ID". Messages contain:

$$(\text{source-ID}, \text{distance-from-source}).$$

We use the following rule:

- A node only forwards BFS messages whose *source-ID* is strictly smaller than any source-ID it has forwarded before.

Thus:

- All BFS waves with non-minimal source-ID die out quickly.
- Only the BFS originating from the globally smallest ID continues until it spans the entire graph.

Once the BFS tree of the smallest-ID node finishes expanding, the leaves initiate a convergecast sending an “OK” message upward.
It is not clear how nodes realise that they are leaves of the BFS tree. For this, whenever any node receives a new smallest source-ID, it sends back an ACK(nowledgdment) to the sender of that message (or to one arbitrary one if it received the same message multiple times). A node is then a leaf, if it does not receive any ACKs from other nodes (in the round after it propagtes a new source-ID). A non-leaf node then propagates an ”OK” only after it has received an ”OK” from all of its children (so from all of the nodes from which it received an ACK before).
When the root (the minimum-ID node) has received responses from all its children, it broadcasts a final `done` message down the BFS tree.
Every node stores the parent from which it received the first BFS message of the minimum-ID BFS tree.

Correctness

Every node eventually receives the BFS message from the globally smallest ID node, since that BFS wave is never suppressed. No other wave can ever complete the entire algorithm, as the minimal source ID propagates faster through the network than any ”OK” ever would.
The convergecast and final broadcast ensure that each node knows when the algorithm ends.
Thus, when the algorithm terminates, every node knows:

- the smallest ID in the network,
- a parent pointer toward it.

Running Time

Each BFS wave travels at most distance D. Each node forwards the BFS message with minimal source-ID in the next round, after it received such a message. So the total number of rounds before the smallest-ID wave spans the network is $O(D)$.
The convergecast and broadcast also each take $O(D)$ rounds.

Exercise 3: Distributed Aggregation in CONGEST *(20 Points)*

Assume you are given a graph G and a (pre-computed) BFS spanning tree T of G, rooted at some node r. Recall that the height of T is at most D, the diameter of G. As input, each node $v \in V$ obtains two $O(\log n)$-bit integers, (a_v, x_v). Let $A = \{a_v \mid v \in V\}$ be the set of all distinct a-values in the graph and let $k := |A|$. Let $a_1, \dots, a_k$ be the distinct a-values, i.e., assume that $A = \{a_1, \dots, a_k\}$. The goal is for all nodes to compute the set of pairs $\{(a_i, y_i) \mid i \in \{1, \dots, k\}\}$, where for all i,

$$y_i := \min_{\text{nodes } v \text{ for which } a_v = a_i} x_v.$$

Give a distributed algorithm that solves this task in $O(D + k)$ rounds in the CONGEST model (and prove that the algorithms terminates after $O(D + k)$ rounds).

Hint: There are different ways to coordinate the k minimum computations. You can for example first make sure that everybody knows A and then use pipelining to do the k global minimum computations.

Sample Solution

Algorithm

Step 1: Collect all distinct a-values. Perform a convergecast: Each node v keeps track of a set $P_v \subset A$ of a values that it has already propagated (so already sent up the tree) and a set $U_V \subset A$ of a values it has received, but not yet sent up the tree. In every round each node sends one of the values from U_v to its parent node (e.g. the one with the smallest index first). It then removes that value from U_v and adds it to P_V. Initially every node has an empty set of already propagated values $P_v = \emptyset$ and the set of unsent values is $U_v = \{a_v\}$ Every value a node v receives, is added to U_v, only if it is not yet in P.
This is sufficient so that the root r obtains the set A. This process takes at most $D + k$ rounds, since it is essentially equivalent to routing k tokens towards the root. Even if all tokens travel on exactly the same path, a token is only ever delayed by another token at most once and hence arrives with a delay of $k - 1$.
Then r broadcasts A to all nodes, by sending a_1 in round 1, a_2 in round 2, $\ldots$. These broadcasts perfectly pipeline and so every node has received all k values after $D + k$ rounds.
This takes $O(D + k)$ rounds upward and $O(D + k)$ downward.

Step 2: Pipeline k minimum computations. For each $a_i \in A$, we compute:

$$y_i = \min_{\{v:a_v=a_i\}} x_v.$$

To that end we simply act as if every node v had a value for each $a \in A$. To be more precise v acts as if it had k inputs $(a_1, x_1), (a_2, x_2), \ldots$, where $x_i = x_v$ for $a_i = a_v$ and otherwise $x_i = *$, where $*$ is some placeholder. We now perform k convergecasts, one for each a_i, but in a pipelined manner:
At each round:

- Every node sends the *next* minimum for the next a_i upward in the pipeline. Note that any "actual" value is smaller than the placeholder $*$
- Messages contain $(i, \text{current-min})$, which fits in $O(\log n)$ bits.

Because the BFS tree has depth D, after D rounds the first minimum reaches the root. Then, every round, the root receives the minimum for the next a_i. Thus, the pipeline delivers all k minima in:

$$O(D + k) \text{ rounds.}$$

The root then broadcasts the final set of pairs $\{(a_i, y_i)\}$ to all nodes in another $O(D)$ rounds.