Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

FS 2025 Prof. R. Wattenhofer

Principles of Distributed Computing
Exercise 9

Recall that an algorithm is *self-stabilizing* if, starting from any configuration, it eventually reaches a *legitimate configuration*. In addition, once the system reaches a legitimate configuration, all further configurations are legitimate.

1 Self-Stabilizing $(\Delta + 1)$-Coloring

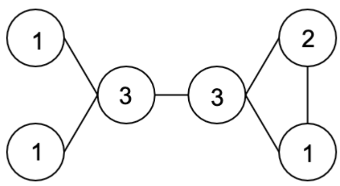
As a warm-up, we will take a look at the algorithm below, meant to compute a proper $(\Delta+1)$-coloring. We assume that Δ is publicly known and cannot be corrupted.

Algorithm 1 $(\Delta + 1)$-Coloring?

Node u holds a variable c_u, and a variable c_v for each neighbor v.
When receiving c'_v from neighbor v, u sets $c_v := c'_v$.
At all times, node u checks whether both of the conditions below hold:
    c_u is an integer between 1 and $\Delta + 1$.
    $\forall v \in N(u) : c_u \neq c_v$.
  If any of these conditions fails, node u updates c_u:
    $c_u :=$ lowest integer between 1 and $\Delta + 1$ such that $\forall v \in N(u) : c_u \neq c_v$.
In every round, node u sends c_u to all neighbors.

a) Define a legitimate configuration for the $(\Delta + 1)$-coloring problem.

b) We run the Algorithm 1 on the graph below. The nodes' labels represent the initial colors c_u, and each node already knows its neighbors' colors. In addition, no transient faults occur. Do we reach a legitimate configuration?



c) Is Algorithm 1 indeed a self-stabilizing $(\Delta + 1)$-coloring algorithm? If yes, prove that this is indeed the case. If not, briefly explain why and describe how to fix the issues.

2 Self-stabilizing Spanning Tree

We now move our focus to spanning trees. In this exercise, we are searching for efficient, deterministic, self-stabilizing spanning tree algorithms. We assume a leader node root. Each node except for the root needs to hold a parent node p_u such that the edges (p_u, u) define a tree.

a) Show that any deterministic self-stabilizing spanning tree algorithm requires $\Omega(D)$ rounds, where D denotes the diameter of the graph we are running the algorithm on.

b) We now want to apply the transformation from the lecture to the Bellman-Ford BFS algorithm (see Algorithm 2). We assume that the root's identity is hardcoded (and cannot be corrupted). In addition, we assume that the nodes know the diameter of the graph D (which is also hardcoded, and cannot be corrupted).

Is the time complexity of the resulting self-stabilizing algorithm optimal? How much more information per round must be transmitted compared to the original algorithm?

Algorithm 2 Bellman-Ford BFS

Each node u stores an integer d_u which corresponds to the distance from u to the root, and a node p_u representing its parent in the spanning tree. Initially $d_{\text{root}} := 0$ and $d_u := \infty$ for every non-root node u. Every node sets $p_u := \perp$.
The root starts the algorithm by sending "1" to all neighbors.
if a node u receives a message "y" with $y < d_u$ from a neighbor v **then**
    node u sets $d_u := y$ and $p_u := v$
    node u sends "$y + 1$" to all neighbors (except v)
end if

3 Crash Failures

Transient faults are not the only issues that can occur in a distributed system. In this exercise, we will focus on coloring algorithms that can tolerate *permanent crash failures.*

Our network is synchronous and shaped as a path of n nodes. The nodes know n and can distinguish between their left and their right neighbor (if any). Out of the n nodes, $t < n$ nodes may crash permanently at any point.

Our goal is to obtain a proper coloring in this setting, i.e., to give different colors to any two adjacent non-crashing nodes. Note that we do not require output from any of the crashing nodes.

a) Describe a deterministic algorithm obtaining a proper 2-coloring in this setting.

b) What if the network is asynchronous, i.e., the only guarantee is that messages get delivered eventually? Is there a deterministic algorithm obtaining a proper 2-coloring in this setting? If your answer is yes, then describe such an algorithm. Otherwise, explain why no such algorithm exists.