ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

FS 2015

Prof. R. Wattenhofer
Barbara Keller

Principles of Distributed Computing
Sample Solution to Exercise 13

1 Self-stabilizing Spanning Tree

a) If the whole memory state of all nodes is deleted, nodes need at least R time to build a new tree, where R denotes the radius of the graph. This can be seen from the fact that a circle of length $D \leq R/2$ (D being the diameter) cannot be detected faster than that. Therefore, we have a lower bound of $\Omega(D)$, which trivially must be optimal.

b) The Bellman-Ford algorithm terminates in $R(r) \in \Theta(D)$ rounds, where $R(r)$ denotes the radius of the graph at the root r. This implies that its self-stabilizing variant needs exactly the same number of rounds to stabilize, matching the bound from **a)**. Since the transformation of an algorithm running in k rounds results in an algorithm simulating k instances of the original algorithm in parallel, on average we need to transmit $R(r) \in \Theta(D)$ times more information in each round.

c) As it takes $\Theta(D)$ time anyway to stabilize, we can also keep a previously computed solution for that time and only then start the algorithm again. Since the root knows D, it may start a new computation every $O(D)$ rounds, after the previous one should have finished if no errors occured. Along with building the tree, nodes are informed in how many rounds the result will be "switched" to the outcome of the current computation. Thus, at any time only a single instance of the Bellman-Ford algorithm is running, but the stabilization time is still in $O(D)$.

This technique can be applied whenever we have a "fail-safe" way to start a new computation after at least k (k being the running time of the original algorithm again) and at most l rounds, for some parameter $l \geq k$. Here fail-safe means that even if the system is corrupted, after at most l rounds a new instance of the original algorithm can be started in a globally consistent way. In particular, this must be independent of the memory states of the nodes! If we do so, the system will stabilize in $l + k$ rounds, which in case of $l \in O(k)$ means a stabilization time of $O(k)$.

This can always be done if we have a problem requiring $\Omega(D)$ time anyway and D is known: We simply fix one node as a leader, which initiates new runs every $\Theta(D)$ time. If—unlike in the Bellman-Ford algorithm—different nodes start to send messages on their own, this can be coordinated by starting the computation by means of a flooding, telling nodes at distance d from the leader to start the execution in $D - d$ rounds.