Exercise 9: Recovering from Christmas

Task 1: Self-Stabilization Survey

In this exercise you will check out some algorithms from the lecture and the exercises, and ponder how to apply our knowledge about self-stabilization on it. Specifically, we would like to know whether a problem or the presented algorithm:

triv: already is self-stabilizing (includes trivial modifications),

tran: well-suited for the transformation from the lecture,

mod: self-stabilizing with straightforward modifications,

unfit: solving a problem unsuitable for a self-stabilizing algorithm,

slow: there is a different transformation that targets “slow” algorithms and makes them self-stabilizing,

hunch+/-: you believe it can/can’t be made self-stabilizing, but it’s not that simple to prove/disprove, or

uhh: you don’t have the foggiest idea.

a) Roughly describe a transformation that takes a non-stabilizing algorithm that runs in R rounds, and outputs an algorithm that stabilizes in $\mathcal{O}(R + D)$. This transformation should impose an overhead of $\mathcal{O}(\log n)$ bits on each message, such that it can meaningfully handle both CONGEST and LOCAL algorithms.

b) Make a table, marking each algorithm according to the above list:

(1) 2-coloring by message passing

(2) Pointer jumping

(3) Cole-Vishkin

(4) Linear color-reduction

(5) Synchronizers (α, β, γ)

(6) BFS construction (Dijkstra; Bellman-Ford)

(7) Clustering (Exercise 2 Algorithm 1)

(8) Consensus (randomized asynchronous; deterministic synchronous from Exercise 3 Task 2)

(9) Randomized MIS

(10) Safe Broadcast (Exercise 4 Algorithm 1)

(11) Low-arboriticy MDS (Exercise 5 Algorithm 1)

(12) Forest decomposition (Exercise 5 Algorithm 2)

(13) MST on general graphs (distributed Kruskal; GHS; GKP)

(14) MST on clique (Exercise 6 Task 1)

(15) Communication complexity (equality; disjoint set)

(16) Pipelined Bellman-Ford

(17) APSP

(18) Dijkstra’s Token Ring

Some of these are very straight-forward, some are rather difficult, and sometimes it depends on how you combine the problem/algorithm with self-stabilization.

c) Check out the lectures and exercises on your own. What problems and algorithms are missing from the table? (There are at least two.) What is their classification on the above list?

d) As a new column of the table, provide a bit of explanation where you think it's helpful or needed.

e)* Discuss unclear cases in the exercise session!

Task 2: Nice and Naughty

A class has students S. Some $N \subseteq S$ students are *naughty* and the others $S \setminus N$ are *nice*. If nice students get good grades for their exercise sheets, they stay nice; naughty students stay naughty while receiving good grades because they get away with it. If nice students receive bad grades, they turn naughty out of frustration. However, a naughty student $s \in N$ eventually sees reason and turns nice when getting bad grades for $k_s \in \mathbb{N} \cup \{\infty\}$ consecutive weeks.

Since we have a perfect TA, he can correctly and deterministically decide whether a student was nice or naughty from the solutions that have (or have not) been handed in. Unfortunately, this takes two weeks, and he needs to determine grades already within one week. He thus must decide on the grades for week W based on niceness/naughtiness for weeks $1, \ldots, W-1$ only. This also means that the TA has the luxury of starting from a known initial state, so he reliably remembers his name and who the lecturer is. Also note that the TA is not omniscient, so the TA does not know k_s in advance.

Example for some specific student $s \in S \setminus N$: In week 1, student s hands something in. A few days later, still during week 1, the TA has to put a grade on it and show it to student s, despite not having had enough time to truly appreciate it. Thus the TA doesn't know whether s was nice or naughty in week 1. Let's assume the TA decides to hand out a bad grade in week 1. This turns our previously-nice student s naughty!
In week 2, student s hands more things in, and again the TA doesn't have the time to really appreciate what he is given, and has to put a grade on it. However, he now knows that the week 1 submission was really great, and that the student was nice last week, in week 1. And so on.

We hope that playing around with this model, tweaking the knobs and observing the effects to the model's self-stabilization properties, build some intuition about the topic as a whole.

a) Devise a grading scheme that is self-stabilizing, where we consider an execution suffix correct iff all nice students remain nice and all naughty students remain naughty. What is the stabilization time?

b) We say that $s \in S$ has a *good heart* iff $k_s \neq \infty$. Improve the grading scheme such that it is still self-stabilizing if we keep the requirement that all nice students remain nice, and add the additional requirement that students with good hearts eventually become (and remain) nice, while the remaining students receive bad grades. What is the stabilization time?

c) Show that there is an algorithm $\mathcal{A}_k$ (i.e., the algorithm knows k) that can stabilize faster than that in b) if $k_s = k \neq \infty$ for all naughty students $s \in N$. (It should still stabilize if this is not the case, just be faster if it holds true!)

d) Argue that, in some sense, your solution from b) is doing well, with respect to stabilization time.

e)* What if the TA has to be fully self-stabilizing, i.e., has to gracefully handle starting in arbitrary states?

Task 3*: Have a Happy New Vote

a) Program a simulator for Democrats vs. Republicans!

b) Make the simulator determine the number of steps until stabilization!

c) Come up with examples maximizing the time until stabilization as a function of the number of nodes n. How does the maximum possible stabilization time behave asymptotically?

d) Show off your results in the exercise session!