Network Algorithms: Exercise 10
Midterm questions

Dr. Stefan Schmid, Arne Ludwig, Srivatsan Ravi

1 Self-Stabilization (5 + 5 + 5 + 5 = 20 points)

Recall the voting stabilization problem from the lecture where the citizens of a little town call their friends to ask whether they vote for Democrats or Republicans. Assume that every citizen has an odd number of friends.

a) In a city with an even number n of citizens, every citizen will eventually vote for the same party. Prove this claim or give a counterexample for any $n > 1$.

b) After a certain time period, the "winning party" is always the same. Prove this claim or give a counterexample.

c) For which party will a citizen vote at day $t+2$ and at day $t+3$ if at day t, he votes for the Democrats and the majority of his out-going edges are good?

d) Give a topology with 6 nodes where each party initially has 3 votes and one party will eventually end up receiving all the votes.

2 Consensus

We consider the problem of achieving consensus in synchronous communication rounds. Look at the following algorithm:

- Broadcast own value to all other processes
- Receive values from all other processes
- Decide on the minimum value.

Now, answer the following questions:

(1) Suppose that there are no failures i.e. processes do not crash and messages always get delivered. Does the above algorithm achieve deterministic consensus? If not, present an execution to prove otherwise.

(2) Suppose that processes fail by crashing when sending messages to other processes. Does the above algorithm achieve deterministic consensus? If not, present an execution to prove otherwise.

(3) Suppose that at most f processes may crash. Give an algorithm similar to the above that achieves consensus. How many rounds does this algorithm of yours need?

3 Asynchronous Consensus

Recall the algorithm from Exercise 4 for solving 2-process consensus in an asynchronous system using queues and atomic registers when at most one process may fail by crashing.

(1) Assume that you are already given an initialized queue with a winner and a loser ball in it. Present an algorithm that is using this queue and atomic registers in order to solve consensus for 2 processes where up to one process may fail.

(2) Assume now that you do not have this initialized queue, but rather you have *two* initially empty queues. Take a look at the (incomplete) algorithm $propose_i(v)$ (Algorithm 1 in Exercise 4). Each process initially has a private input value v. Complete the algorithm with the code of the function call $func_i(Q_j, R_j, v)$ so that $propose_i(v)$ reaches consensus even if up to one process may fail.

(3) Is the resulting algorithm *wait-free*? i.e. every correct process returns an output after a finite number of its own steps.

4 MIS

Recall the Fast MIS from 1986 from the lecture.

(1) Argue why the algorithm does indeed produce a correct output.

(2) Let I be the output of Algorithm 17 in Step 2. Given that for all v, $Pr(v \in I) \geq \frac{1}{4d(v)}$, prove that if v if *good*, then $Pr(v \in N(I)) \geq \frac{1}{36}$, where $N(I)$ is the union of the neighbourhoods of every vertex in the set I (recall that a vertex is good if it has lots of neighbours of low degree; cf. MIS slides).

(3) Now, argue that the expected number of edges removed at a given phase is at least a constant fraction of the number of edges present.

(4) Now prove that the expected number of phases of the algorithm is $O(logn)$.

5 Self-stabilizing Spanning tree

(1) Prove an asyptotically optimal lower bound on the time-complexity of any self-stabilizing spanning tree algorithm.

(2) If the transformation from the lecture is applied to the Bellman-Ford BFS algorithm (cf. Chapter 3 in Lecture notes), is the time-complexity of the resulting self-stabilizing algorithm optimal? On average, how much more information per round must be transmitted compared to the original algorithm?

(3) Assume that the diameter D is known to the leader (i.e., the root of the constructed tree) Can you give an algorithm stabilizing up to constant factors in the same time, but sending not more information in each round than the unmodified Bellman-Ford algorithm? If yes, can your approach be generalized to other algorithms? If no, prove a corresponding lower bound.