Problem Set 1, Part b

Due: Thursday, February 21, 2008

Reading:

Sections 4.2-4.4, 5.1, 6.1-6.3 of *Distributed Algorithms*

Reading for next week: Sections 6.4-6.7, 7.1-7.3 of *Distributed Algorithms*
Aguilera, Toueg paper, listed in Handout 3
(Optional) Keidar, Rajsbaum paper

Problems:

0. Email Calvin to join the course mailing list and to select a problem set to grade, if you have not already done so (details on course website).

1. (a) Exercise 4.3 (Recall that the number of messages is at most $diam|E|$, which is $O(n^3)$.)
   (b) Show a weighted digraph (with nonnegative edge weights) with 5 nodes (including a distinguished source node) such that some node changes its *parent* component 4 times in an execution of *BellmanFord*, or else prove that no such graph exists.
   (c) Show a weighted digraph (with nonnegative edge weights) with 5 nodes (including a distinguished source node) such that *two* nodes change their *parent* components 4 times each in an execution of *BellmanFord*, or else prove that no such graph exists.

2. Consider a variation of the Shortest Paths problem described in Section 4.3 of the textbook where:

   A. Halting is not required: we require only that *eventually* a shortest paths tree over the processes exists and does not change,
   B. Each process's state contains a Boolean constant, *source*, which is set to *true* for i_0 and *false* for every process $i \neq i_0$, and
   C. (modified:) Except for *source* and edge *weight* constants, the initial state of each process is arbitrary: each non-*source*, non-*weight* state variable v of a process, (including any *rounds* or *status* variables), is initially set to an arbitrary value from $\mathbf{type}(v)$.

   Note that these conditions mean that different processes might think they are in different rounds, and processes can't tell if they are just starting an algorithm execution or are in the middle of an execution.

   (a) Explain informally why the $BellmanFord$ algorithm described on p. 62 of the textbook does not solve this new version of the Shortest Paths problem.
   (b) Describe informally a modified version of $BellmanFord$ that solves the Shortest Paths problem under these conditions.
   (c) Give pseudocode in the style in the book for your new algorithm.
   (d) Think about how you would prove your algorithm correct. (Don't turn anything in for this part.)

3. Recall that *SynchGHS* proceeds in phases, each of which consists of a fixed number of rounds upon which processes agree ahead of time. We argued that this fixed number can be $O(n)$, where n is the number of nodes in the network graph. Determine constants a and b such that every phase can be $an + b$ rounds (i.e., no phase requires more than $an + b$ rounds), and argue that $an + b$ rounds are sufficient for any phase. Try to find the smallest possible value for a and b, particularly a. (Do not try to improve *SynchGHS*; just analyze it as described on page 67 of the book.)

4. Consider the following variant of the coordinated attack problem from Section 5.1 (the one that is supposed to tolerate message losses). Assume that the network is a complete graph of $n = 4$ nodes. The termination and validity requirements are the same as those in Section 5.1. However, the agreement requirement is weakened to say: "If any process decides 1, then at least three processes decide 1." Is this problem solvable or unsolvable? Prove.

5. Section 6.3.3 contains a simple algorithm (TurpinCoan) for Byzantine agreement on an arbitrary value domain V. This algorithm uses a Byzantine agreement algorithm for bits as a "subroutine". At the cost of two extra rounds, this algorithm manages to substantially reduce the bit complexity, over the standard Exponential Information Gathering Byzantine Agreement algorithm for V.

   (a) Read the description of this algorithm, and its correctness proof.

   (b) Do Exercise 6.22, which asks you to generalize the algorithm slightly.

6. Modify the *FloodSet* algorithm of Section 6.2.1 by adding a local *early decision* test condition, in order to obtain the following additional *early decision* time bound property:
   If the execution has only $f' \leq f$ failures, then all nonfaulty processes decide (but don't halt) by the end of round $f' + 2$.
   Prove that your algorithm works, that is, that it solves the stopping agreement problem for stopping failures, and that it has the additional time bound property.
   Note: Recall that we require the *uniformity* property, which says that all processes that decide (even if they later fail), decide on the same value.