# Réponses aux exercices

**Réponse 1.** Soit $G$ un graphe connexe, et $T$ un arbre couvrant de $G$. Enraciner $T$ en un sommet $r$ quelconque de $G$, et colorier chaque sommet $u$ de $G$ par sa distance à $r$ dans $T$ modulo 2. Chaque sommet a une couleur différente de son parent dans $T$, et $r$ à au moins un enfant, celui-ci étant de couleur différente de $r$.

**Réponse 2.** Considérons l'algorithme distribué probabiliste suivant. Chaque sommet $u$ n'ayant pas encore décidé une couleur choisit la couleur $c(u) = 0$ ou 1 aléatoirement uniformément. Si, à la suite de ce choix, $u$ a un voisin de couleur différente de la sienne, alors $u$ décide la couleur choisie.

Cet algorithme réalise une 2-coloration faible. En effet, avec probabilité 1, tout sommet $u$ finit par choisir une couleur différente de la couleur choisie (ou décidée précédemment) par un voisin $v$. De plus, lorsqu'un sommet $u$ décide une couleur, nécessairement un de ses voisins, $v$, a choisi une couleur $c(v)$ différente, et $v$ décide $c(v) \neq c(u)$. Le sommet $u$ satisfait donc la 2-coloration faible.

A chaque phase, la probabilité que $u$ décide une couleur est au moins $1/2$, car, pour tout voisin $v$ de $u$, $\Pr[c(u) = c(v)] = 1/2$. Soit $X_u$ la variable aléatoire égale au temps auquel $u$ décide sa couleur. On a $\Pr[X_u \geq t] \leq 1/2^t$, et donc $\Pr[X_u \geq 2\log_2 n] \leq 1/n^2$. Soit $X = \max_u X_u$ le temps auquel tous les sommets ont une couleur. Par sous linéarité des probabilités, on obtient :

$$\Pr[X \geq 2\log_2 n] \leq \sum_{u \in V(G)} \Pr[X_u \geq 2\log_2 n] \leq n/n^2 = 1/n.$$

**Réponse 3.** There are $\binom{c'}{\lfloor c'/2 \rfloor}$ subsets of $\{1, \ldots, c'\}$ with cardinality $\lfloor c'/2 \rfloor$. Label these subsets from 1 to $\binom{c'}{\lfloor c'/2 \rfloor}$ in lexicographic order. Since $\binom{c'}{\lfloor c'/2 \rfloor} \geq c$, one can associate a subset $S_i$ to each color $i \in \{1, \ldots, c\}$. Let $v$ be a node colored $i \in \{1, \ldots, c\}$, and let $w$ be a neighbor of $v$ colored $j \in \{1, \ldots, c\}$ with $i \neq j$. Since $S_i \neq S_j$, there exists $x \in S_i \setminus S_j$. Node $v$ recolors itself with color $x$. If every node does so, one gets a weak $c'$-coloring since $x \in \{1, \ldots, c'\}$ and the neighhbor $w$ of $v$ picks color $y \in S_j$ necessarily different from the color $x \notin S_j$.

**Réponse 4.** Nodes are initially weakly colored with colors in $\{1, \ldots, c\}$, and we recolor them with two colors $a$ and $b$ so that to get a weak 2-coloring. We proceed in $c$ rounds $r = 1, \ldots, c$. At round $r$, we recolor the nodes initially colored $r$.

Procedure RECOLOR (at node $v$) :

```
for r = 1 to c do
    if v is initially colored r then
        if v has all its neighbors initially colored ≥ r then v recolors itself a (Rule 1)
        else (i.e., v has a neighbor which was recolored) (Rule 2)
            if all recolored neighbors of v have color b
                then v recolors itself a
                else v recolors itself b.
```

By construction, if node $v$ recolors itself according to Rule 2, then $v$ has a neighbor colored differently from itself. So, let $v$ be a node that recolors itself $a$ according to Rule 1. Assume $v$ was initially colored $r$. Since $v$ has all its neighbors colored with colors $\geq r$, and since the initial colors form a weak coloring, $v$ necessarily possesses a neighbor $w$ with initial color $r' > r$. At round $r'$, $w$ will apply Rule 2, and will recolor itself $b$. Hence, $v$ and $w$ will have different colors.

**Réponse 5.** Each round ensables to reduce the number of colors from $c$ to $c'$ where $\binom{c'}{\lfloor c'/2 \rfloor} = \frac{4^{c'}}{\sqrt{\pi c'}}(1 + o(1)) \geq c$. We have $c' = \log_4 c + O(\log\log c)$, and thus $c' \leq \log_2 c$ for $c$ large enough. The number of rounds to reach a weak 4-coloring is thus $O(\log^* k)$. Since reducing from 4 to 2 colors requires a constant number of rounds, we get that weak 2-coloring a graph is possible in constant number of rounds whenever there is a way to weakly color the nodes with $k = O(1)$ colors. This is for instance the case of the regular graphs with maximum degree 3. (See Moni Naor, Larry J. Stockmeyer : What Can be Computed Locally ? SIAM J. Comput. 24(6) : 1259-1277 (1995)).

# EPIT2017: Lower Bounds in Distributed Computing Solutions

## 1 Covering and valence in consensus algorithms

*Consider an obstruction-free binary consensus algorithm using atomic read-write registers.*

*Let $P$ be bivalent from $C\beta$, where $\beta$ is a block write by some $R \subseteq P$ . Let $\gamma$ be a schedule of some $z \notin P$ such that $z$ decides in $C\gamma$. Show that $z$ must write to a register not covered by $R$ in $C$.*

Suppose not, i.e., for some $\gamma \in z^*$, $z$ decides a value $v \in \{0, 1\}$ in $C\gamma$ writing only to registers covered by $R$ in $C$. Thus, in no process in $P$ can distinguish $C\gamma\beta\alpha$ from $C\beta\alpha$ where $\alpha$ is a $P$-only schedule. As $C\beta$ is bivalent, we can choose $C\beta\alpha$ and, thus, $C\gamma\beta\alpha$, to decide $1 - v$—a contradiction.

## 2 Space complexity of mutual exclusion

1. *Show that any 2-process read-write mutual-exclusion alogirithm requires at least 2 registers.*

   Let $p_1$ run its TS (trying section) until it is about to perform its first write: such a schedule must exist, as otherwise $p_2$ may enter its CS without noticing $p_1$. Let $\alpha_1$ be the corresponding schedule. Since $\alpha_1$ is indistinhguishable to $p_2$ from an empty schedule, there exists a schedule $\alpha \in p_2^*$ such that $p_2$ is in its CS at the end of $\alpha_1\alpha$.

   If $p_2$ only writes to the register covered by $p_1$ in $\alpha_1\alpha$, then we can wake up $p_1$ and let it overwrite all the traces of $p_2$ and enter its CS—a contradiction.

   Thus, $p_2$ must write to a distinct register in $\alpha_1\alpha$.

2. *What about 3 processes? Can you show that 3 registers are necessary?*

3. *Finally, prove the general statement: $n$-process algorithm requires $n$ registers.*

   We prove the general case, without wasting time on the 3-process scenario.

   By induction, we are going to prove the following claim:

   > Let $C$ be any configuration in which every process is in its remainder section (RS). For all $k = 1, \ldots, n$, there exists a schedule $\alpha$ by $P_k = \{p_1, \ldots, p_k\}$, such that:
   >
   > - every process in $P_k$ is about to write to a distinct register in $C\alpha$, and
   > - there exists a schedule $\alpha'$ by $P_k$ such that (1) every process is in the remainder section in $C\alpha'$ and (2) $C\alpha$ and $C\alpha'$ differ only in the local states of processes in $P_k$.

The base case $k = 1$ is immediate: simply run $p_1$ from $C$ until it is about to perform its first write. No other process can distinguish the resulting configuration from $C$.

Now suppose that the claim holds for some $k = 1, \ldots, n-1$. Let $C_0$ be the configuration after $\alpha$. Let $D_0 = C_0\beta_0\gamma_0$ be the extension of $C_0$ by $P_k$, where $\beta_0$ is the block write by $P_k$ (on a set $k$ distinct registers $\mathcal{B}_0$), such that every process in $P_k$ is in its RS in $D_0$. Since the algorithm deadlock-free, such an extension exists.

Now we can reuse the induction hypothesis and get a configuration $C_1 = D_0\alpha_0$ in which $P_k$ cover a (possibly different from $\mathcal{B}_0$) set of $k$ registers $\mathcal{B}_1$, etc.

So we get an infinite chain of configurations:

$$C \overset{\alpha}{\rightsquigarrow} C_0 \overset{\beta_0\gamma_0}{\rightsquigarrow} D_0 \overset{\alpha_0}{\rightsquigarrow} C_1 \overset{\beta_1\gamma_1}{\rightsquigarrow} D_1 \overset{\alpha_1}{\rightsquigarrow} C_2 \overset{\beta_2\gamma_2}{\rightsquigarrow} \ldots\ldots$$

where each $C_i$ satisfies the claim for $P_k$.

Since there are only finitely many registers there must exist $C_i$ and $C_j$ $(i < j)$, such that the same set $\mathcal{B}$ of $k$ registers is covered by $P_k$ in $C_i$ and $C_j$.

Now we extend $C_i$ with steps of $p_{k+1}$ until it is about to write to a register not in $\mathcal{B}$. Such an extension $C_i' = C_i\psi$ exists, as otherwise $p_{k+1}$ can enters its CS and then all the traces of its presence in the CS will be overwritten by the subsequent block write $\beta_i$.

Notice that, since all steps of $p_{k+1}$ are overwritten by the block write in $C_i\psi\beta_i$, the resulting configuration $C_j' = C_j'\psi\beta_i\gamma_i\alpha_i \ldots \beta_{j-1}\gamma_{j-1}\alpha_{j-1}$ is indistintinguishable from $C_j$ for any process except $p_{k+1}$. Thus, $C_j'$ satisfies the claim for $P_{k+1} = \{p_1, \ldots, p_{k+1}\}$: (1) every process in $P_{k+1}$ covers a distinct register in $C_j'$ and (2) only processes in $P_k$ can distinguish $C_j'$ from some configuration in which every process is in its RS.

Typos and mistakes are possible in this draft. If you find any, please let me know: `petr.kuznetsov@telecom-paristech.fr`.

Prof. Patt-Shamir

# Solutions

**Problem 1.** We use the maximum-finding protocol shown in class, but each node $v$, instead of sending its input value $x_v$ (using $O(\log M)$ bits), sends $y_v \stackrel{\text{def}}{=} \lceil \log_{1+\varepsilon} x_v \rceil$. Let $r$ be the maximal value that reaches the root. The root outputs $(1+\varepsilon)^r$ as the maximum. Let $X = \max\{x_v\}$, i.e., the true maximum. The approximation is due only to rounding, so we have that

$$\begin{aligned} (1+\varepsilon)^r &= (1+\varepsilon)^{\lceil \log_{1+\varepsilon} X \rceil} \geq (1+\varepsilon)^{\log_{1+\varepsilon} X} = X \ , \text{ and} \\ (1+\varepsilon)^r &= (1+\varepsilon)^{\lceil \log_{1+\varepsilon} X \rceil} < (1+\varepsilon)^{\log_{1+\varepsilon} X + 1} = X \cdot (1+\varepsilon) \ . \end{aligned}$$

Regarding message length, for $0 < \varepsilon \leq 1$ we have that $\log_{1+\varepsilon} X = \frac{\log X}{\log 1+\varepsilon} = \Theta(\frac{\log X}{\varepsilon})$ , and therefore the number of bits requited to encode $y_v$ is $O(\log \frac{\log X}{\varepsilon}) = O(\log\log M + \log \varepsilon^{-1})$. (Note that this is actually better than the expression in the question.)

**Problem 2.** (Sketch.) The idea is to consider two rings of sizes $n > n' > 1$, and prove, by induction on time, that all nodes in both rings maintain the same state at step $t$, for $t \geq 0$. The base of the induction is by assumption of that all nodes have the same initial state, and the inductive step follows from the assumption that all nodes run the same code. This means that if a node in one ring outputs at time $t$ "size is $m$" then all nodes, in both rings, will make the same output at time $t$, which is wrong for at least one of the rings.

**Problem 3.** (Sketch.) First prove, by induction on time, that at time $t$, all nodes at distance at most $t$ have $d_v \leq t$. Next prove, by induction on distance, that at any time $t$, a node $v$ at distance $r \geq t$ from the source have $d_v \geq t$.

**Problem 4.** Consider the following labeled 5-node line graph: $(1) - (2) - (4) - (3) - (5)$. The edge $(3) - (4)$ is removed but it is not killed by any node. (The edge $(2) - (4)$ is killed by $(1)$.)

**Problem 5.** By Markov's Inequality,

$$\Pr[\text{at least } |E|/4 \text{ edges removed}] \geq \frac{1}{3} \ .$$

Call a round is *successful* if it eliminates at least a 1/4 of the edges. Then by the above, each round is independently successful with probability at least 1/3, and hence the expected number of successful rounds in any $T$ rounds is at least $T/3$. By Chernoff's bound,

$$\Pr[\text{at most } T/6 \text{ successful rounds ot of } T \text{ rounds}] \leq e^{-T/12} \ .$$

Now, since in each successful round the number of edges is reduced by a factor of at least 3/4, we may conclude that in $T$ rounds, with probability at least $1 - e^{-\Omega(T)}$, the number of surviving edges is at most $|E|(3/4)^{T/6}$. Plugging in $T \in \Omega(\log n)$ concludes the proof.

**Problem 6.** Say that a path is *nearly shortest* if its length is at most one more than the distance between its endpoints. First we prove that if a path is nearly shortest, then it can be broken into at most 2 paths such that each of them is a (truly) shortest path between its endpoints. It follows that any instance with $k$ packets with routes which are nearly shortest can be replaced by two packets whose routes are shortest paths. This immediately implies that no packet suffers more than $2k-1$ delays. Moreover, since a packet is never delayed by its sibling "subpacket", the number of delays is at most $2k-2$.

As a simple example, consider $k$ packets, where packet $i$ has the $i$th highest priority. The route of packet $i$ is (1) – (2) – $\cdots$ (i+1) – (i+1) – (i+2) (the $(i+1)$th edge is a self loop). Clearly the routes are nearly shortest, and it can be verified that packet $i$ is delayed $2(i-1)$ times. The average delay is therefore

$$\frac{1}{k}\sum_{i=1}^{k} 2(i-1) \;=\; \frac{2}{k}\cdot\frac{k(k-1)}{2} \;=\; k-1\ .$$

Spring School on Theoretical Informatics

Porquerolles, May 14-19, 2017

Correction sketches of the exercises related to Michel Raynal's lectures

# Exercise 1: on the kFK universal construction

Trivial.
It is easy to find a counter-example in which same pattern occurs in which STATE.SC(ls) returns always false.

# Exercise 2: on operations on memory locations

Objects that support read and one of decrement and multiply have consensus number 1 and cannot be combined to solve wait-free consensus for 2 or more processes.

However, a system that supports all three instructions can solve wait-free binary consensus for any number of processes. The protocol uses a single memory location initialized to 1. Processes with input 0 perform decrement(), while processes with input 1 perform multiply(n). The second operation by each process is read (). If the value returned is positive, then the process decides 1. If it is negative, then the process decides 0.

## 2 Solution to stateless flooding

Let us introduce notation that we use for the proofs. A device is *visited* if it receives the message at least once. An edge is *used* if the message was sent over it at least once. It is *unused* otherwise. A visited device is a *border* if it has an adjacent unused link. A visited device that is not a border is *internal*.

**Lemma 1** *In SF, every border device holds a message in SQ to be sent over every unused link and it never holds a message to be sent over a used link.*

**Proof:** We prove the lemma by induction. The source sends messages over the links to its neighbors. Therefore, right before the transmission, the source is a border device with every link unused and a message to transmit over this link. Therefore, the conditions of the lemma hold. Assume the conditions hold at some step of a computation. Let us consider the next step: a transmission of the message from device $a$ to device $b$. Device $b$ may be visited or not visited. If $b$ is not visited, then all its links, except for link to $a$, are unused. When $b$ receives a message from $a$, it becomes a border device and it holds a message to every neighbor except $a$. This satisfies the conditions of the lemma. If $b$ is already visited, then, by assumption, it has a message to be sent to $a$ in its $SQ$. This message is a mate of the message received by $b$ from $a$. By the algorithm, these two messages are discarded. That is, once the message is transmitted to a visited device and uses the channel, there are no messages to be sent over this used channel. Again, the conditions of the lemma hold. □

**Theorem 1** *SF guarantees termination and delivery from the source to all target devices connected to the source.*

**Proof:** Once the source device has a message to send, it sends to all its neighbors. That is, it becomes a border device. According to Lemma 1, every border device has a message to transmit over unused channels. Since we consider fair computations of routing algorithms, this message is eventually going to be transmitted. If the receiver device is not visited, it becomes visited and sends messages to all its neighbors. Eventually, all devices connected to the source will be visited, and all channels used. That is, SF delivers the message to all devices connected to the source. Note that according to Lemma 1, once the channel is used, there are no messages to be sent across it. That is, SF terminates. □