Albert-Ludwigs-Universität
Institut für Informatik
Prof. Dr. F. Kuhn

May 2, 2013

Network Algorithms, Summer Term 2013
Problem Set 1 – Sample Solution

Exercise 1: Vertex Coloring

1. Note that an "undecided" message can be realized by sending nothing at all; thus we do not count such messages. Therefore, each node sends exactly two messages to each neighbor, one in the first round and one after assigning a color. Hence, the total number of messages is $4|E|$, as 4 messages are sent over each edge.

   The number of messages can be reduced to $3|E|$, if a "deciding" node does send its color only to undecided nodes in its neighborhood.

2. Yes, the algorithm still works, it could be reformulated in the following way (we assume that each node knows its degree):

Algorithm 1 Asynchronous "$\Delta + 1$"-Coloring

1: **send** node ID to all neighbors.
2: wait until all neighbor IDs have been received and all neighbors with a lower ID have chosen a color
3: choose smallest possible color
4: **send** chosen color to all neighbors

Exercise 2: Coloring Trees

1. The log-star algorithm for the ring is basically identical to the algorithm for trees. Nodes do not have a parent in the ring, therefore we simply define the left neighbor of any node to be its "parent". Given this definition, we can run the normal log-star algorithm. Using the same argumentation as for trees, it can be shown that no two neighboring nodes choose the same color. Note that is possible to leave out the "shift" step, as all "children" (i.e., the right neighbor) have always the same color.

2. We use two additional colors, ℓ and r, to solve the termination problem. Furthermore, we let each node send its color to both neighbors between each round of the log-star algorithm. This way, each node always knows the colors of both neighbors at the beginning of a round of the log-star algorithm. (Algorithm 5)

   The algorithm works as follows for a node v: As long as neither v nor one of its neighbors has a color in $\mathcal{R} \cup \{\ell, r\}$, it executes Algorithm 5. If v learns that the color of its left neighbor is in $\mathcal{R}$ (regardless of the color of the right neighbor), and vs color is not in $\mathcal{R}$, then v recolors itself with the color r and waits until both its neighbors have a color in $\mathcal{R} \cup \{\ell, r\}$ (v is at the right end of a sequence of nodes with colors in $\mathcal{R}$). If a node v learns that the color of its right neighbor is in $\mathcal{R}$, while the color of its left neighbor and its own color are both not in $\mathcal{R}$, then v recolors itself with the color ℓ and waits until both its neighbors have a color in $\mathcal{R} \cup \{\ell, r\}$ (v is a the left end of a sequence of nodes with colors in $\mathcal{R}$). Additionally, as a node v to the right of a node

Algorithm 2 Synchronous "3"-Coloring of the Ring (alg. for node v)

1: **send** c_v to both neighbors;
2: **receive** c_ℓ from left neighbor, c_r from right neighbor;
3: **while** $c_v \notin \mathcal{R} \cup \{\ell, r\}$ **do**
4: **if** $c_\ell \in \mathcal{R}$ **then**
5: $c_v := r$
6: **else if** $c_r \in \mathcal{R}$ **then**
7: $c_v := \ell$
8: **else if** $c_\ell = r$ **then**
9: do one step of Alg. 5 (Lines 6–9) with an arbitrary color $c \in \mathcal{R}$ as parent color
10: **else**
11: do one step of Alg. 5 (Lines 6–9)
12: **end if**
13: **send** c_v to both neighbors;
14: **receive** c_ℓ from left neighbor, c_r from right neighbor;
15: **end while**
16: **if** $c_v > 2$ **then**
17: **wait until** both neighbors' colors are in $\mathcal{R} \cup \{\ell, r\}$
18: **wait until** both neighbors have smaller colors (assuming $5 < \ell < r$)
19: choose new color $c_v \in \{0, 1, 2\}$ using **First Free**
20: **send** c_v to both neighbors
21: **end if**

colored r no longer receives new colors, we need the rule that v simple takes an arbitrary color $c \in \mathcal{R}$ as the new color of its parent and computes its new color based on c and its own color in each round (Line 9).

Inside the first while-loop, v eventually reaches a color in $\mathcal{R} \cup \{\ell, r\}$. After that, nodes with colors larger than 2 wait until their neighbors have also acquired a color in the range $\mathcal{R} \cup \{\ell, r\}$ in order to start the color reduction phase. In the color reduction phase, we essentially use Alg. 3 from the lecture notes ("Reduce"), except that instead of starting with unique ID's, we start with an initial coloring.

To prove correctness, we have to show that when reaching Line 13 of the algorithm no two neighboring nodes u and v have picked the same color in $\mathcal{R} \cup \{\ell, r\}$. We show this for color r in more detail:

Lemma 1. *Two neighbors u and v cannot both be colored r.*

Proof. W.l.o.g., let u be the parent (left neighbor) of v. A node only adopts color r if its color is not in $\mathcal{R}$, but its parent color is in $\mathcal{R}$. This condition cannot hold for two neighboring nodes at the same time. Hence u and v cannot reach color r in the same round.

If u reaches color r first, then v will reduce its color according to an arbitrary parent color from $\mathcal{R}$ in each following round until itself or its right neighbor reaches a color in $\mathcal{R}$. In the first case, v keeps its color in $\mathcal{R}$, in the second case, it chooses color ℓ and thus, it will never choose color r.

If v reaches color r first, u's color must be in $\mathcal{R}$ and thus u will not change its color again, in particular not to color r, proving that two neighboring nodes cannot be colored with color r. □

The same argumentation also applies to color ℓ.

Lemma 2. *Two neighbors u and v cannot both be colored with color $c \in \mathcal{R}$.*

Proof. W.l.o.g., let u be the parent (left neighbor) of v. The logic of Alg. 5 ensures that no two neighboring nodes get the same color $c \in \mathcal{R}$ in one step from outside $\mathcal{R} \cup \{r, \ell\}$. If both nodes

are inside that set, then the two *wait* conditions prevent simultaneous color changes and since all changes are done using *First Free*, a collision is not possible. Similarly, if one node is in $\mathcal{R}$, then it stops acting due to the *wait* conditions, and if the other one is not in that set, it assumes either r or ℓ in its next round. □

The running time of the algorithm is $\log^* n + O(1)$.