ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

FS 2021

Prof. R. Wattenhofer
Diana Ghinea

Principles of Distributed Computing
Sample Solution to Exercise 1

1 Vertex Coloring

If the nodes omit the "undecided" messages, then each node sends exactly two messages to each neighbor, one in the first round and one after assigning a color.

a) To express the worst-case message complexity in terms of n, we have to think of a topology that maximizes the number of neighbors of each node. Such a topology is a clique, where each neighbor has $n-1$ neighbors. Then, as each node sends 2 messages to each of its neighbors, the message complexity is $2n(n-1)$ in the worst case.

b) There are 4 messages sent over each edge: two in the first round, two after assigning a color. Hence, the total number of messages is $4m$.

2 TDMA

a) The resulting coloring is depicted in Figure 1. Note that the clique of size 3 needs at least 3 colors. Hence, our solution is optimal.

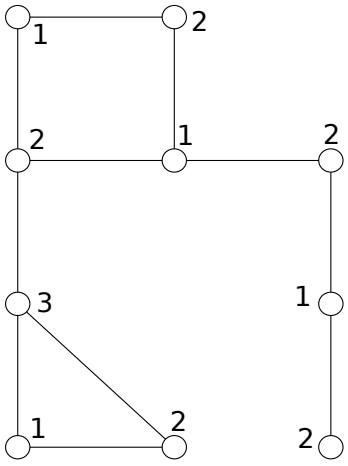

Figure 1: The slots for the wireless network

b) We will use additional edges to model the new interferences. Two neighbors of a node are not able to send messages at the same time if they have different colors. Therefore, for every node, we will add an edge between every two neighbors of that node.

The resulting coloring is depicted in Figure 2. There are multiple cliques of size 4 in the new graph; any of these needs at least 4 colors. Hence, our solution is optimal.

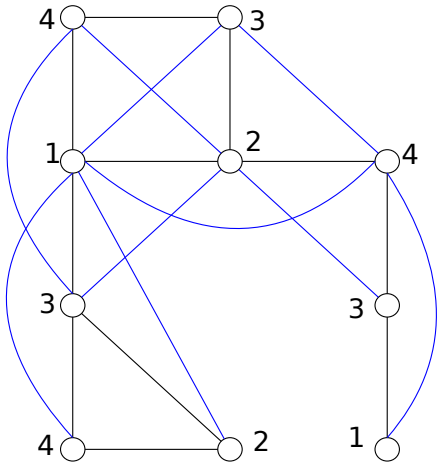

Figure 2: The slots for the wireless network with improved communication. The additional constraints are shown in blue.

c) Every pair of lectures that is selected by a student cannot take place at the same time. Thus, they cannot be colored with the same color. This leads us to the solution shown in Figure 3. The graph can be colored with 3 colors.

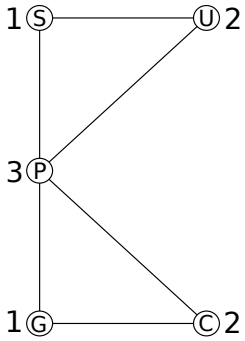

Figure 3: The resulting graph with P being shorthand for Principles of Distributed Computing, S for Statistical Learning Theory, U for Ubiquitous Computing, G for Graph Theory, and C for Cryptography. Note that 3 colors are sufficient.

3 Coloring Trees

a) 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 argument as for trees, it can be shown that no two neighboring nodes choose the same color. Note that we can omit the "shift" step, as all "children" (i.e., the right neighbor) always have the same color.

b) We build the algorithm step by step. The shape of the algorithm is similar to the solution of the previous task, consisting of a *"6-color"* phase and a *Reduction* phase.

What happens when the nodes do not know n?

Firstly, let's think of what happens in the algorithm from the previous task when n is unknown.

- the nodes cannot compute $\log^*(n)$, so they do not know when they can look for a color in $\{0, 1, 2\}$;
- the nodes only know their own color and the colors of their neighbors.

When can a node stop the *"6-color" phase*?

A natural question is the following: *Can a node stop the "6-color" phase when its own color and the colors of its neighbor are in $\mathcal{R}$?*

Well, *almost*. Somewhere in the ring, there might be a node with a color greater than 5 that will trigger other colors in the ring to change and might cause conflicts.

The nodes could announce everyone when obtaining a color in $\mathcal{R}$: they could send a message to their neighbors, the neighbors would forward this message until this message reaches every node in the ring (in roughly $n/2$ steps). Intuitively, when everyone has announced that they found a color in $\mathcal{R}$, the nodes can start reducing the colors. However, an algorithm following this idea would require $O(n \log^* n)$ rounds, so it's much faster to simply color the ring with 3 colors in $n/2$ steps.

Then, let's think of the following idea: *when a node v gets a color in $\mathcal{R}$, it will stop the "6-color" phase*. What happens with v's left neighbor v_ℓ and right neighbor v_r?

- v_ℓ's color is in $\mathcal{R}$: it will stop the "6-color" phase as well;
- v_ℓ's color is not in $\mathcal{R}$: v_ℓ will obtain a color in $\mathcal{R}$ at a later round, and that color might be the same as v's color! To fix this issue, when v_ℓ sees that the color of its right neighbor (v) is in $\mathcal{R}$, it takes a special color r, with the meaning "The color of my right neighbor is in $\mathcal{R}$. I will now stop this phase" and stops the "6-color" phase;
- v_r's color is in $\mathcal{R}$: it will stop the "6-color" phase as well;
- v_r's color is not in $\mathcal{R}$: v_r's color is computed based on its current color and on v's color. Hence, v_r will never obtain a color in $\mathcal{R}$. To fix this issue, when v_r sees that the color of its left neighbor (v) is in $\mathcal{R}$, it takes a special color ℓ, with the meaning "The color of my left neighbor is in $\mathcal{R}$. I will now stop this phase", and stops the "6-color" phase.

To conclude: *when a node v gets a color in $\mathcal{R} \cup \{\ell, r\}$, it will stop the "6-color" phase.*

Algorithm 1 "$(6, \ell, r)$-Coloring" Phase - Version 1

```
1: send c_v to both neighbors
2: while c_v ∉ R ∪ {ℓ, r} do
3:    if c_ℓ ∈ R then
4:       c_v := ℓ;
5:    else if c_r ∈ R then
6:       c_v := r;
7:    else
8:       We will somehow use the "6-color" Algorithm here
9:    send c_v to both neighbors
```

How do we apply the ideas from the "6-color" Algorithm?

We will now discuss line 8 of the algorithm above: how can we use the "6-color" Algorithm? What happens with the nodes whose left neighbor is colored with ℓ or r? Intuitively, we will run the "6-color" Algorithm as if each *segment* between a node colored with ℓ and a node colored with r is a *tree*: the node whose left neighbor's color is ℓ will be the *root*.

For simplicity, each node will use the subroutine presented in Algorithm 2 for computing a new color (lines 5–8 of the "6-color" Algorithm).

Algorithm 2 `new_color`(c_ℓ, c_v)

```
1: interpret c_ℓ and c_v as bit-strings
2: let i be the index of the rightmost bit b where c_v and c_ℓ differ
3: return c_v = 2i + b
```

We now present the updated "$(6, \ell, r)$-Coloring" phase.

Algorithm 3 "$(6, \ell, r)$-Coloring" Phase - Version 2

send c_v to both neighbors
while $c_v \notin \mathcal{R} \cup \{\ell, r\}$ **do**
  if $c_\ell \in \mathcal{R}$ **then**
    $c_v := \ell$;
  else if $c_r \in \mathcal{R}$ **then**
    $c_v := r$;
  else if $c_\ell = \ell$ **then**
    $c_v =$ `new_color`(arbitrary color, c_v) ("root")
  else
    $c_v =$ `new_color`(c_ℓ, c_v)
  send c_v to both neighbors

The time complexity of this algorithm is $O(\log^* n)$. We need to show that two neighbors cannot obtain the same color. We will prove this in detail for ℓ.

Lemma. Two neighbors u and v cannot both be colored ℓ.

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

If u reaches color ℓ first, then v will reduce its color according to an arbitrary color from $\mathcal{R}$ in each following round and thus, it will never choose color ℓ.

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

A similar argument applies to the color r. The logic of Algorithm 6-color ensures that no two neighboring nodes get the same color $c \in \mathcal{R}$. Note that a node v to the right of a node colored ℓ acts like the root in a tree and can thus simply choose an arbitrary color $c \in \mathcal{R}$ in each round without causing any conflicts.

When a node exits this loop, its color will be in $R \cup \{\ell, r\}$. Can we start the reduction phase now? Not *yet*. We should wait until the colors of the neighbors are in $R \cup \{\ell, r\}$ as well.

Algorithm 4 "$(6, \ell, r)$-Coloring" Phase - Final Version

send c_v to both neighbors
while $c_v \notin \mathcal{R} \cup \{\ell, r\}$ **do**
  if $c_\ell \in \mathcal{R}$ **then**
    $c_v := \ell$;
  else if $c_r \in \mathcal{R}$ **then**
    $c_v := r$;
  else if $c_\ell = \ell$ **then**
    $c_v =$ `new_color`(arbitrary color, c_v) ("root")
  else
    $c_v =$ `new_color`(c_ℓ, c_v)
  send c_v to both neighbors
wait until both neighbors' colors are in $\mathcal{R} \cup \{\ell, r\}$

At this point, the nodes can reduce the colors from $\mathcal{R} \cup \{\ell, r\}$ to $\{0, 1, 2\}$.

From $6 + 2$ colors to 3 colors

Algorithm 5 presents the intuitive shape the *Reduction phase*. Note that since every node has two neighbors, every node can obtain an admissible color in $\{0, 1, 2\}$. There is still a question left: *When is it* `my_turn()` *to pick a color in* $\{0, 1, 2\}$*?*

It is possible that a node and its neighbors start the reduction phase at different rounds. If we simply iterate through the colors (e.g. nodes with color ℓ go first, then nodes with color

Algorithm 5 Reduction Phase

```
1: while c_v ∉ {0,1,2} do
2:   if my_turn() then
3:     choose smallest available color c_v ∈ {0,1,2}
4:     send c_v to both neighbors
```

r, afterwards nodes with color 5 etc.), a node and its neighbor, although they end up with different colors in the first phase, they might choose a color in $\{0, 1, 2\}$ at the same time. They might even choose the same color! To ensure this does not happen, we will use the `round_number`: since the nodes operate synchronously, they have the same `round_number` at any time, even if they are in different phases of the algorithm. We will define the subroutine `my_turn()` as follows.

Algorithm 6 `my_turn`(`round_number`, c_v)

```
1: x := (round_number mod 5) + 3;
2: if (c_v = x) then
3:   return true
4: else if x = 6 and c_v = l then
5:   return true
6: else if x = 7 and c_v = r then
7:   return true
8: else
9:   return false
```

Putting it all together

Algorithm 7, presented below, is the final solution.

To summarize, 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 ("6-color").

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 "6-color". If v learns that the color of its left neighbor is in $\mathcal{R}$ (regardless of the color of the right neighbor), and v's color is not in $\mathcal{R}$, then v recolors itself with the color ℓ and waits until both its neighbors have a color in $\mathcal{R} \cup \{\ell, r\}$. 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 r and waits until both its neighbors have a color in $\mathcal{R} \cup \{\ell, r\}$. Additionally, as a node v to the right of a node colored ℓ no longer receives new colors, we need the rule that v simply 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.

Inside the first while-loop, v eventually reaches a color in $\mathcal{R} \cup \{\ell, r\}$. After that, node v waits until its neighbors have also acquired a color in this range in order to start the color reduction phase. In each round of the color reduction phase, one color of $\{3, 4, 5, \ell, r\}$ is replaced by a legal color in $\{0, 1, 2\}$. As all nodes know the absolute number of rounds that have passed so far, this can be done without interference.

Algorithm 7 Synchronous "3"-Coloring on Ring

send c_v to both neighbors
while $c_v \notin \mathcal{R} \cup \{\ell, r\}$ **do**
  if $c_\ell \in \mathcal{R}$ **then**
    $c_v := \ell$;
  else if $c_r \in \mathcal{R}$ **then**
    $c_v := r$;
  else if $c_\ell = \ell$ **then**
    $c_v =$ `new_color`(arbitrary color, c_v) ("root")
  else
    $c_v =$ `new_color`(c_ℓ, c_v)
  send c_v to both neighbors
wait until both neighbors' colors are in $\mathcal{R} \cup \{\ell, r\}$
while $c_v \notin \{0, 1, 2\}$ **do**
  if `my_turn`(`round_number`, c_v) **then**
    choose smallest available color $c_v \in \{0, 1, 2\}$
    send c_v to both neighbors

`new_color`(c_ℓ, c_v)

interpret c_ℓ and c_v as bit-strings
let i be the index of the rightmost bit b where c_v and c_ℓ differ
return $c_v = 2i + b$

`my_turn`(`round_number`, c_v)

$x :=$ (`round_number` mod 5) + 3;
return $(c_v = x)$ or $(x = 6$ and $c_v = l)$ or $(x = 7$ and $c_v = r)$