Distributed Algorithms, MIT Course 6.852, Fall 2015
Lecture 4: Coloring

Stephan Holzer Nancy Lynch
MIT
Cambridge, MA, USA
holzer@mit.edu lynch@csail.mit.edu

09/22/2015

If you have comments or spot mistakes, please email Stephan: holzer@mit.edu.

So far the synchonous model we considered assumed unique IDs (UIDs). In this lecture we refine the model and assume that the n nodes of network G have unique IDs in a certain range that is a function of n, for simplicity we choose $\{1, \ldots, n\}$. We also assume that the message size is $O(\log n)$ and thus, e.g., a constant number of IDs can be exchanged in each round. This model is known as the CONGEST model, as it takes limited bandwidth into account, which might lead to congestion when one node in the network should forward several messages for other nodes.

In this lecture we consider the coloring problem, which is closely related to the MIS problem covered in lecture 3.

Definition 1 (Coloring)**.** *A coloring of a graph is an assignment of "colors" to nodes. A coloring is a legal coloring if no two adjacent nodes have the same color. A coloring is called c-coloring if it is a legal coloring that uses at most c colors, e.g., $\{1, \ldots, c\}$.*

We are interested in keeping the number of colors small, while reducing the time needed to compute a c-coloring.

1 A Simple Color Reduction Technique

Let Δ be the maximum degree in a network.

Theorem 1. *Given a legal c-coloring, Algorithm 1 produces a legal $(\Delta + 1)$-coloring of network G within $c - \Delta - 1$ rounds.*

Given a legal coloring with $c > \Delta + 1$ colors, we reduce the number of colors to $\Delta + 1$ by removing one color from the palette in each of $c - \Delta - 1$ rounds. To do so, we observe that the Δ neighbors of any node v can utilize at most Δ different colors from $\{1, \ldots, c\}$. We conclude that if v's color c_v is not in $\{1, \ldots, \Delta + 1\}$, then there exists at least one color $c'_v \in \{1, \ldots, \Delta + 1\}$ not chosen by any neighbor of v such that v could recolor itself to c'_v. We cannot ask all nodes with $c_v \notin \{1, \ldots, \Delta + 1\}$ to recolor themselves at the same time, as this might not lead to a legal coloring. Imagine e.g. a graph containing just two nodes that are connected and have color 10 and 12 and then both decide to recolor themselves to color 1. The following algorithm uses our previous observation to recolor nodes in a systematic way:

Proof. (of Theorem 1.) *Runtime:* The bound on the number of rounds follows immediately from the description of the algorithm.

Algorithm 1 Simple Color Reduction - executed by node v.
Converts a legal c-coloring into a legal $\Delta + 1$-coloring.

Assume:

- A legal coloring $C = \{c_1, \ldots, c_n\} \subset \{1, \ldots, c\}^n$ is given.
- Each node v knows its color c_v and the colors of adjacent nodes.
- Each node knows the maximal degree Δ in the graph.
- The graph is undirected.

Required:

- Legal coloring $C' = \{c'_1, \ldots, c'_n\} \subset \{1, \ldots, \Delta + 1\}^n$.
- Each node v should output its new color c'_v.

Pseudocode:

```
    //** Initialization
1:  $c'_v := c_v$; //** stores node $v$'s most recent color
2:  for each neighbor $u$ of $v$ do
3:      $c'_u := \infty$; //** stores most recent known color for neighbor $u$ of $v$
4:  end for
    //** Algorithm
5:  for $i = \Delta + 2, \ldots, c$ do
        //** Exchange current color with neighbors
6:      send $c'_v$ to all neighbors;
7:      for each neighbor $u$ of $v$ do
8:          $c'_u :=$ the color received from $u$;
9:      end for
        //** update $v$'s color if applicable
10:     if $c'_v = i$ then
11:         $c'_v := \min(\{1, \ldots, \Delta + 1\} \setminus \{c'_u | u \in \Gamma(v)\})$
12:     end if
13: end for
14: output $c'_v$;
```

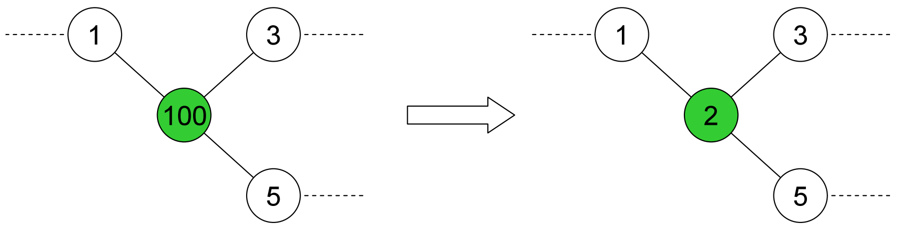


Figure 1: Vertex 100 receives the lowest possible color.

Correctness: As argued earlier, if $c_v \notin \{1, \dots, \Delta + 1\}$, there exists a color in $\{1, \dots, \Delta + 1\}$ that v could pick while keeping a legal coloring. For each node v that has an original color $c_v \notin \{1, \dots, \Delta+1\}$, the algorithm picks such a color. By iterating through all original colors $\Delta + 2, \dots, c$ one after each other we avoid that adjacent nodes recolor to the same color: As C is a valid coloring, only non-adjacent nodes v can choose a new color at the same time, as nodes with same original color c_v cannot be neighbors in a legal coloring C. Thus the resulting coloring C' is legal, too. □

2 Coloring Oriented Trees Using 3 Colors

Definition 2 (Oriented Tree)**.** *A tree $T = (V_T, E_T)$ is* rooted *if it has a distinguished node r serving as the root. In a rooted tree, a node v is a* parent *of node u if $(u, v) \in E_T$ and $d_T(r, v) = d_T(r, u) - 1$, where d_T is the distance-function*[1]*. A rooted tree is* oriented *when each node (except the root) knows its parent.*

Note that there is a difference between edges being directed and oriented. Although the tree is oriented, communication through the edges is still undirected and therefore possible in both directions. Undirected communication also implies that nodes of an oriented tree not only know their parents, but also their children.

Before we start, observe that a (rooted) tree always has a 2-coloring: color all nodes v of even distance $d_T(v, r)$ to the root r with color 1 and of odd distance $d_T(v, r)$ to the root r with color 2. Such a 2-coloring can be computed by computing the distances to the root: The root broadcasts a message with a time-stemp t to the network - the time difference between t and the time that the message arrived at a node v represents the distance $d_T(v, r)$. In the p-set 2 you are asked to analyze whether this is the optimal runtime in case the tree is a line.

We use the following definition of $\log^* n$ to bound the runtime of the algorithm presented in this section. You may recall this notion from your basic algorithms class.

Definition 3 ($\log^* n$)**.** *The function $\log^* n$ is the iterated logarithm of n, defined as follows:*

$$\begin{aligned} \log^{(1)} n &:= \log n \\ \log^{(i+1)} n &:= \log(\log^{(i)} n) \\ \log^* n &:= \min\{i \in \mathbb{N} \mid \log^{(i)} n \le 2\} \end{aligned}$$

This is an extremely slow growing function: $\log^* 3 = 1$, $\log^* 10 = 2$, $\log^* 1,000 = 3$ and $\log^* 2^{1,000} = 4$.

In Lecture 3 we discussed a randomized algorithm that computes an MIS on an undirected graph of arbitrary topology and n nodes in time $O(\log n)$ with probability $1 - 1/n$. In this lecture, we consider deterministic versions. In Problem-Set 1 you were asked to revisit Luby's algorithm from

[1]In case this algorithms is applied on a tree that is a subgraph of another algorithm, d_T refers to the distance when using edges of T only.

lecture 3 (continued in lecture 4) to compute an MIS and analyze its performance on a ring with some probability. In this section we show that a coloring can be computed in time $O(\log^* n)$ on an oriented tree in a deterministic way. This is interesting, as MIS and colorings are closely related, as you will show in p-set 2. In Section 3 we present an $\frac{1}{2}\log^* n - 1$ lower bound for computing a 3-coloring on a ring. It can be shown that the upper bound for oriented trees can be transferred to the ring, such that this bound is tight.

We obtain the fast 3-coloring algorithm in two steps. First, we compute a 16-coloring of a tree in $O(\log^* n)$ rounds, see Theorem 2. Second, we show how to transform a 16-coloring of a tree into a 3-coloring using $O(1)$ many rounds, see Theorem 3.

Theorem 2 (Cole-Vishkin)**.** *In the CONGEST model, an oriented tree can be 16-colored in time $O(\log^* n)$ in a deterministic way.*

Proof. We describe Algorithm 2 that performs this task and analyze it's runtime. Let us use the notation c_v^i to denote the color of node v after i rounds. For the initial coloring, for each node $v \in V$, we let $c_v^0 := id_v$, i.e., the starting colors are the IDs of the nodes. Here we assume w.l.o.g. that $id_r = 1$.

Definition 4 (Most Significant Differing Bit)**.** *Given two bit strings $a = a_k, \dots, a_1$ and $b = b_k, \dots, b_1$ of k bits each, the most significant bit index where they differ is the largest $l \in \mathbb{N}$ such that $a_l \neq b_l$ an for each $l', k \geq l' > l$ it is $a_{l'} = b_{l'}$. W.l.o.g. We say that b_l is the most significant differing bit.*

Now we iteratively compute c_v^{i+1} based on the values c_v^i and c_p^i, where p is v's parent. In particular we compute the index l of the most significant differing bit between c_v^i and c_p^i an set $c_v^{i+1} := (l, b_l)$.

As the root does not have a parent, the root keeps color 1.

Algorithm 2 Fast 16-Coloring of an Oriented Tree - executed by node v

Assume:

- The graph is an undirected oriented tree.
- UIDs in the range $\{1, \dots, n\}$.
- Each node v knows its ID id_v, its parent and its children.
- Each node knows n.
- The root has ID 1.

Required:

- Legal coloring $C = \{c_1, \dots, c_n\} \subset \{1, \dots, 16\}^n$.
- Each node v should output its new color c_v.

Pseudocode:

```
//** The algorithm computes one temporary coloring $C^i = \{c_u^i \mid u \in V\}$ for each iteration $i$;
1:  $c_v^0 := id_v$;
2:  for $i = 0, \dots, \lceil \log^* n \rceil$ do
3:      send $c_v^i$ to children;
4:      send $c_v^i$ to children; //** if children exist
5:      if $v$ is not the root then
6:          receive $c_p^i$ from parent;
7:          interpret colors $c_v^i$ and $c_p^i$ as bit strings;
8:          $l$ := index of the most significant bit where $c_v^i$ and $c_p^i$ differ;
9:          $b_l$ := the $l$'th bit of $c_p^i$;
10:         $c_v^{i+1} := (l, b_l)$;
11:     end if
12: end for
```

Example 1. *Algorithm 2 executed on the following part of a tree:*

$$\begin{array}{llll} \texttt{Grandparent } pp & c^i_{pp} = 0010110000 & & \\ \texttt{Parent } p & c^i_p = 1010010000 & \rightarrow & c^{i+1}_p = \underbrace{1010}_{l=10}\underbrace{0}_{b_l=0} \\ v & c^i_v = 1011110000 & \rightarrow & c^{i+1}_v = \underbrace{0111}_{l=7}\underbrace{1}_{b_l=0} \quad \rightarrow \quad c^{i+2}_v \underbrace{101}_{l=5}\underbrace{0}_{b_l=0} \end{array}$$

Theorem 3. *If C^i is a legal coloring, then is C^{i+1}.*

Proof. Consider any two neighboring nodes v and p. W.l.o.g. p is the parent of v. To show that $c^{i+1}_v \neq c^{i+1}_p$ if $c^i_v \neq c^i_p$, we distinguish two cases: 1) p is the root, and 2) p is not the root.

Case 1) p is the root: It is $c^i_p = c^{i+1}_p =$ "*color* 1" $= 00\dots01$ for all i. All colors c^{i+1}_v are of type (l, b_l), where $l \geq 1$ is an index. Thus the binary encoding of l contains an 1 at some point and thus $c^{i+1}_v = (l, b_l)$ differs from $00\dots01 = c^{i+1}_p$.

Case 1) p is not the root: Consider also the grandparent pp of p and colors c^i_v, c^i_p and c^i_{pp}. It is $c^i_v \neq c^i_p$ and $c^i_v \neq c^i_{pp}$, as C^i is a legal coloring. Therefore there must be an index l_p indicating the most significant bit where c^i_v and c^i_p differ, and an index l_{pp} indicating the most significant bit where c^i_p and c^i_{pp} differ. This results in two further cases: 1) $l_p \neq l_{pp}$, and 2) $l_p = l_{pp}$. If $l_p \neq l_{pp}$, then c^{i+1}_v and c^{i+1}_p differ in their l-component. If $l_p = l_{pp}$, then the l_p'th bit of c^i_p differs from the $l_{pp} = l_p$'th bit of c^i_{pp}. As c^{i+1}_v and c^{i+1}_p take over these bits from c^i_p and c^i_{pp}, colors c^{i+1}_v and c^{i+1}_p differ in this bit as well. □

Lemma 1. *For all $v \in V$, c^i_v has $\left\lceil \log^{(i+1)} n \right\rceil + 2$ bits for every i satisfying $\log^{(i)} \geq 2$.*

Proof. We prove this by induction on i. In case $i = 1$, each c^1_v can be encoded using $\lceil \log n \rceil = \left\lceil \log^{(1)} n \right\rceil$ bits, as we assume UIDs in $\{1, \dots, n\}$. Assume c^i_v is represented with $\left\lceil \log^{(i)} n \right\rceil + 2$ bits. The procedure guarantees, that c^{i+1}_v has $\lceil \log\left(\lceil \log^i n \rceil + 2\right) \rceil + 1 = \left\lceil \log^{(i+1)} n \right\rceil + 2$ bits. □

We conclude: after $\log^* n$ iterations, each node v has a color $c^{\log^* n}_v$ that can be represented with $\left\lceil \log^{(\log^* n)} n \right\rceil + 2 \leq 4$ bits. As $C^0 := C$ is a legal coloring, $C^{\log^* n}$ is a legal 16-coloring. This proves Theorem 2. □

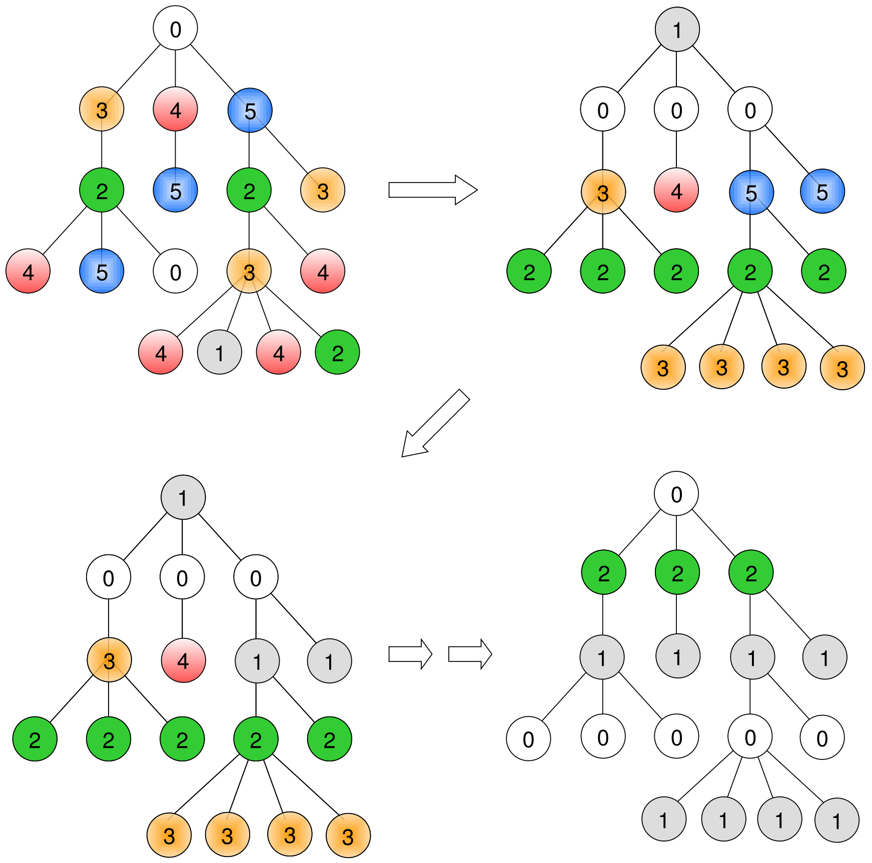
Lemma 2. *In the CONGEST model, an 16-coloring of an oriented tree can be transformed into a 3-coloring in $O(1)$ rounds in a deterministic way.*

Proof. We reduce the number of colors by one color every two rounds. To remove color 16, in the first round the root r picks a color in $\{1, 2, 3\}$ different from its previous color and each non-root node adopts the color of its parent. Then the children of each node have the same color. As in the first part of the lecture we conclude that the neighbors of each node occupy at most 2 colors out of $\{1, 2, 3\}$. Now we apply the same trick as in Algorithm 1: each color-16 node picks a color in $\{1, 2, 3\}$ that is not occupied by its neighbors such that only colors $\{1, \cdots, 7\}$. Here we use a similar argument as in the proof of Theorem 1.

Removing colors 7, 6, 5 and 4 using the same idea transforms an 16-coloring into a 3-coloring within $O(1)$ rounds of communication. Figure 2 displays this process for the case where we remove color 5. □

As a conclusion from Theorem 2 and Lemma 2 we state:

Theorem 4. *In the CONGEST model, an oriented tree can be 3-colored in time $O(\log^* n)$ in a deterministic way.*



Example 2.

Figure 2: Possible execution of the algorithm described in Lemma 2.

Finally, we show how to use this 3-coloring algorithm to compute an MIS on an oriented tree fast.

Lemma 3. *An MIS can be computed on oriented trees in time $O(\log^* n)$.*

Proof. In lecture 4 we showed how to compute a 3-coloring of an oriented tree. An MIS can be obtained based on a 3-coloring in three rounds: In the first round all nodes of color 1 join a set M and notify their neighbors of this. In the next round all nodes of color 2 who do not have a neighbor in M join M and notify their neighbors of this. In the final round all nodes of color 3 who do not have a neighbor in M join M. At any time M is an independent set. At the end each node has a neighbor in M, such that M is an MIS. □

3 Lower Bound for Coloring a Ring

We finish this lecture with a proof that no deterministic algorithm can compute an MIS on a ring in time less than $\frac{1}{2}\log^* n - 1$, see Theorem 6. Before we start with this, we want to point out a significant difference in runtime between computing a 2-coloring and 3-coloring on the ring.

Theorem 5. *In the CONGEST model, a ring cannot be 2-colored in time less than $\frac{n}{2} - 1$ in a deterministic way.*

Proof. Similar to the proof for 2-coloring a line, which you will do as a homework. □

This is much slower than computing a 3-coloring using the algorithm from the previous section when adapted to a ring (as claimed before).

Theorem 6. *In the CONGEST model, an oriented ring cannot be 3-colored in time less than $\frac{1}{2}\log^* n - 1$ in a deterministic way.*

Proof. If we have a distributed algorithm with a running time of T communication rounds, then each node has to pick its own color based on the information that is available within distance T from it; see Figure 3. Moreover, two nodes that are adjacent to each other must pick different colors. Hence

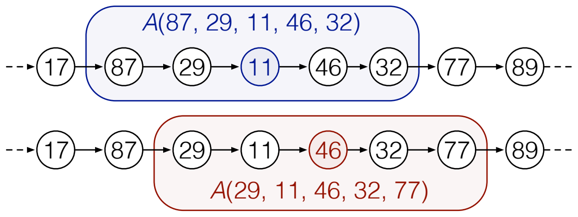


Figure 3: Coloring directed cycles in time $T = 2$. For example, the output of node 11 only depends on its radius-T neighborhood, $(87, 29, 11, 46, 32)$. We can interpret algorithm A as a k-ary function, $k = 2T+1 = 5$, that maps each local neighborhood to a color. As it is possible that adjacent nodes have neighborhoods $(87, 29, 11, 46, 32)$ and $(29, 11, 46, 32, 77)$, function A must satisfy $A(87, 29, 11, 46, 32) \neq A(29, 11, 46, 32, 77)$.

the algorithm is a function A with $2T + 1$ arguments that satisfies

$$A(x_1, x_2, \ldots, x_{2T+1}) \in \{1, 2, 3\},$$
$$A(x_1, x_2, \ldots, x_{2T+1}) \neq A(x_2, x_3, \ldots, x_{2T+2})$$

whenever $x_1, x_2, \ldots, x_{2T+2}$ are distinct identifiers from the set $\{1, 2, \ldots, n\}$.

We now define and study a more abstract mathematical object that we call *coloring function*, prove a key theorem about it, and then apply that theorem to get the final result about the ring.

Definition 5 (Coloring function)**.** *For $n, k, c \in \mathbb{N}$, we say that A is a k-ary c-coloring function if*

$$A(x_1, x_2, \ldots, x_k) \in \{1, 2, \ldots, c\} \qquad \textit{for all } 1 \leq x_1 < x_2 < \ldots < x_k \leq n, \tag{1}$$
$$A(x_1, x_2, \ldots, x_k) \neq A(x_2, x_3, \ldots, x_{k+1}) \qquad \textit{for all } 1 \leq x_1 < x_2 < \ldots < x_{k+1} \leq n. \tag{2}$$

Any deterministic distributed algorithm A that finds a proper 3-coloring of an n-cycle defines a k-ary 3-coloring function for $k = 2T + 1$ (the converse is not necessarily true).

Lemma 4. *For any k-ary c-coloring function, it is $k \geq \log^* n - \log^* c$.*

Proof. (of Lemma 4). The proof is by induction on k; the base case is trivial. If a coloring function only sees one identifier, it cannot do much. We continue with proving Lemmas 5 which is used in the induction step (Lemma 6).

Lemma 5. *If A is a 1-ary c-coloring function, then we have $c \geq n$.*

Proof. If $c < n$, then by the pigeonhole principle there exist two identifiers $x_1 < x_2$ with $A(x_1) = A(x_2)$. This contradicts (2). ☐

The key part of the proof is the inductive step. Given any coloring function A, we can always construct another coloring function B that is "faster" (smaller number of arguments, i.e. smaller value of k) but "worse" (larger number of colors, , i.e. smaller value of c).

Lemma 6. *If A is a k-ary c-coloring function, then we can construct a $(k-1)$-ary 2^c-coloring function B.*

Proof. We define B as follows:

$$B(x_1, x_2, \dots, x_{k-1}) = \{A(x_1, x_2, \dots, x_{k-1}, x_k) \,:\, x_k > x_{k-1}\} \subseteq 2^{\{1,\dots,c\}} = \{U \,:\, U \subseteq \{1, \dots, c\}\}.$$

This means that B is the set of all values of A when evaluated on the same values $x_1, x_2, \dots, x_{k-1}$ as B plus any possible value of x_k, where we require that $x_k > x_{k-1}$. Although there can be up to $n-k$ choices for x_k, there are at most 2^c possible values of B: all possible subsets of $\{1, 2, \dots, c\}$. These can be represented as integers $\{1, 2, \dots, 2^c\}$, and hence property (1) of the definition of a coloring function holds.

The interesting part is to prove (2). Let $1 \le x_1 < x_2 < \ldots < x_k \le n$. By way of contradiction, suppose that

$$B(x_1, x_2, \dots, x_{k-1}) = B(x_2, x_3, \dots, x_k). \tag{3}$$

Let

$$\alpha = A(x_1, x_2, \dots, x_k).$$

From the definition of B we have $\alpha \in B(x_1, x_2, \dots, x_{k-1})$. By assumption (3), this implies $\alpha \in B(x_2, x_3, \dots, x_k)$. But then there must exist a x_{k+1} such that $x_k < x_{k+1} \le n$ such that

$$\alpha = A(x_2, x_3, \dots, x_{k+1}).$$

That is, A cannot be a coloring function, as $A(x_1, x_2, \dots, x_k) = A(x_2, x_3, \dots, x_{k+1})$ for this choice of x_{k+1}. □

To complete the proof of Lemma 4, we will need power towers. Define

$$2 \upuparrows i = \underbrace{2^{2^{\cdot^{\cdot^{2}}}}}_{i \text{ times}}$$

with i twos in the power tower. For example, $2 \upuparrows 2 = 4$ and $2 \upuparrows 3 = 16$. Now assume that A_1 is a k-ary c-coloring function. Certainly it is also a k-ary $(2 \upuparrows (\log^*(c) + 1))$-coloring function. We can apply Lemma 6 iteratively to obtain

- a $(k-1)$-ary $(2 \upuparrows (\log^*(c) + 2))$-coloring function A_2,
- a $(k-2)$-ary $(2 \upuparrows (\log^*(c) + 3))$-coloring function A_3,
  ...
- a 1-ary $(2 \upuparrows (\log^*(c) + k))$-coloring function A_k.

By Lemma 5, we must have $2 \upuparrows (k + \log^*(c)) \ge n$, which implies $k \ge \log^*(n) - \log^*(c)$. □

By plugging in $k = 2T + 1$ an $c = 3$ in Lemma 4, we obtain the main result $T \ge \frac{1}{2} \log^*(n) - 1$ for 3-coloring. This concludes the proof of Theorem 6. □

4 Bibliographical Notes

Sources - see reading list Section 1 follows Peleg's book [2], Chaper 7.2. Section 2 follows Peleg's book, Chapter 7.3 and uses examples and figures from [3]. Note that [2] performs a more detailed analysis to obtain a 6-coloring. Our analysis is simplified and produces an 16-coloring. However, an 6-coloring is not needed here, as we reduce the 16-coloring to a 3-coloring in the same way as [2] reduces a 6-coloring to a 3-coloring, such that the final result is the same. Section 3 is a modification of the write-up of [1] adapted to our notation.

References

[1] Juhana Laurinharju and Jukka Suomela. Brief announcement: linial's lower bound made easy. In *Proceedings of the 2014 ACM symposium on Principles of distributed computing*, pages 377–378. ACM, 2014.

[2] David Peleg. Distributed computing. *SIAM Monographs on discrete mathematics and applications*, 5, 2000.

[3] Roger Wattenhofer. Lecture notes - principles of distributed computing, lecture 2 - eth zurich. In *http://dcg.ethz.ch/lectures/podc_allstars/lecture/chapter1.pdf*.