ECS-189A Special Topics in TCS May 23, 2023

Lecture 8

Lecturer: Slobodan Mitrović

1 Introduction

Homework 1, Problem 3, asks to develop an algorithm that in $O(\log n)$ MPC rounds finds the 3-vertex coloring problem of cycle graphs. This lecture will show an algorithm much faster than $O(\log n)$ MPC rounds. In fact, if we assume that each vertex has a unique ID in the range $[1, \text{poly}\, n]$, then the algorithm we will show is even deterministic. We note that this assumption on ID is not a very restrictive requirement, e.g., each device in a network has a unique serial number that can be used in place of an ID. If no ID is given, we can still assign ones randomly and obtain correctness whp probability instead of always. The ideas we will describe date back to [CV86]. This algorithm is typically explained in the context of coloring rooted trees with 3 colors, but the heart of the approach we describe here is the same as the ideas used for coloring trees.

2 Preliminaries

2.1 Notation

We will use the notation $\tilde{O}(f)$ to hide poly-logarithmic factors in f, i.e., $\tilde{O}(f) = O(f \cdot \text{poly} \log f)$.

By saying that an event $\mathcal{E}$ happens *with high probability* (whp), we refer that $\Pr[\mathcal{E}] \geq 1 - n^{-c}$ for some constant $c \geq 1$.

Definition 1 (Vertex coloring)**.** *Let $G = (V, E)$ be a graph and C be a coloring of the vertices in V. We say that C is a* vertex *coloring if for each $e = \{u, v\} \in E$ it holds that $C_u \neq C_v$. We say that C is a k-vertex coloring if C consists of k different colors.*

In the complexity analysis, we will use *iterated logarithm*, also called *log-star* and denoted by $\log^\star$. The value $\log^\star x$ represents the number of times it is needed to take nested logs of x so that the obtained value is at most 1. Formally,

$$\log^\star n = \begin{cases} 0 & \text{if } n \leq 1 \\ 1 + \log^\star(\log n) & \text{otherwise} \end{cases}$$

For instance, $\log^\star 65536 = 4$ because $\log\log\log\log 65536 = \log\log\log 16 = \log\log 4 = \log 2 = 1$. That is, 4 is the smallest number of nested logs of 65536 that yield value at most 1.

3 3-vertex coloring of cycles

The algorithm we present for this problem uses two procedures. The first one colors a cycle by $O(1)$ colors, while the second one reduces the number of colors from $O(1)$ to 3 in $O(1)$ steps.

3.1 Coloring a cycle by $O(1)$ colors in $O(\log^\star n)$ steps

We first assume that the cycle is oriented so that each vertex has one outgoing and one incoming arc. In this orientation, we say that a vertex p is the parent of vertex u if the arc outgoing arc from u goes to p. In Section 3.2, we discuss removing the assumption that such an orientation is given.

In the approach we will see in a moment, the main idea is to start with a coloring that uses poly n colors and then drastically reduce the number of colors. The starting point is trivial – each vertex uses its ID as a color to obtain a (poly n)-coloring of vertices. Recall that we assume that each vertex has a unique ID. A unique ID is not necessary; what matters is that the neighboring vertices have distinct IDs.

```
1 Function Recolor-Vertex($C_v, C_p$):
2     Let $i$ be the smallest integer such that $C_v$ and $C_p$ differ at the $i$-th bit; the bits are 0-indexed.
3     Let $b_i$ be the value of the $i$-th bit of the ID of $v$.
4     return $2 \cdot i + b_i$
```

Algorithm 1: A function that, given two distinct colors C_v and C_p, outputs a new color.

Let us consider an example to understand better Algorithm 1. Let v be a vertex and $p(v)$ be its parent. Assume that $C_v = 23$ and $C_{p(v)} = 15$. In the binary representation, $23 = (10111)_2$ and $15 = (1111)_2$. Hence, $i = 3$ as 23 and 15 differ at the 3-rd bit; this bit corresponds to the value $2^3 = 8$. So, on input v, Algorithm 1 outputs $2 \cdot 3 + 1$. It is also convenient to think of this output as a pair $(3, 1)$, so we will adopt this notation and refer to the output of Algorithm 1 as a pair.

What can we say about the properties of new colors? The following claim is crucial.

Lemma 2. *Let $C : V \to \mathbb{N}$ be a vertex coloring; no two adjacent vertices have the same color. Let $C' : V \to \mathbb{N}$ be a vector of vertex colors such that $C'_v = \text{Recolor-Vertex}(C_v, C_{p(v)})$. Then, C' is a vertex coloring, i.e., no two adjacent vertices have the same color.*

Proof. To prove this claim, it suffices to show that $C'_v \neq C'_{p(v)}$.

Let $(c_v, b_v) = \text{Recolor-Vertex}(C_v, C_{p(v)})$ and $(c_{p(v)}, b_{p(v)}) = \text{Recolor-Vertex}(C_{p(v)}, C_{p(p(v))})$. On one hand, if v and $p(v)$ differ at the same bit position as $p(v)$ and $p(p(v))$, then $b_v \neq b_{p(v)}$. On the other hand, if v and $p(v)$ differ at a different bit position than $p(v)$ and $p(p(v))$, then $c_v \neq c_{p(v)}$. In either case, we have $C'_v \neq C'_{p(v)}$. □

So, Lemma 2 shows that if an input to Recolor-Vertex is a vertex coloring, then its output across all the vertices is also a vertex coloring. How does it help us? The clever idea of this approach is that if C contains colors 0 through $k-1$, then by the design of Algorithm 1 C' contains $2\lceil \log k \rceil$ different colors numbered 0 through $2\lceil \log k \rceil - 1$. This is the case as i defined in Algorithm 1 takes a value in the range $[0, \lceil \log k \rceil - 1]$ and $b_i \in \{0, 1\}$.

Also, observe that if $k = 6$, then $2\lceil \log k \rceil - 1 = 5 = k - 1$. Moreover, for any $k > 6$ we have $2\lceil \log k \rceil - 1 < k - 1$. In other words, as long as C contains more than 6 colors, Algorithm 1 reduces the number of colors in a new vertex coloring almost exponentially. This discussion motivates the following algorithm.

```
  Input : A cycle graph $G = (V, E)$
1 Define coloring $C^0$ such that $C^0_v$ equals the ID of $v$.
2 $i \leftarrow 0$
3 while $C^i$ has more than 6 colors do
4     Let $C^{i+1}$ be a coloring such that $C^{i+1}_v = \text{Recolor-Vertex}(C^i_v, C^i_{p(v)})$.
5     $i \leftarrow i + 1$
6 return $C^i$
```

Algorithm 2: A function that outputs a 6-vertex coloring of a cycle graph.

Following our observations that the number of colors in each while-loop of Algorithm 2 drops almost exponentially, we can prove the following claim.

Lemma 3. *The while-loop of Algorithm 2 performs $O(\log^\star n)$ iterations.*

One can show that Algorithm 2 performs at most $\log^\star n$ iterations. We will not provide formal proof of this claim in these notes, but an interested reader is referred to lecture notes https://jukkasuomela.fi/dda-2010/lecture-2.pdf, Page 27, for a complete proof.

3.2 Removing the assumption on orientation

In our provided analysis, we assumed that the input cycle is oriented. There are at least two ways to remove this assumption. We now outline those two ways.

Output a pair of colors. Assume that vertex v's neighbors are x and y. When we were given orientation, we did output $\textsc{Recolor-Vertex}(C_v, C_x)$ or $\textsc{Recolor-Vertex}(C_v, C_y)$, depending on whether x or y is the parent of v. Instead, since we are not given an orientation, we now output the pair $(\textsc{Recolor-Vertex}(C_v, C_x), \textsc{Recolor-Vertex}(C_v, C_y))$. It follows directly from our prior discussion that this coloring is a proper vertex coloring with at most $(2\lceil\log k\rceil)^2$ colors. Note that $2\left(\log(2\lceil\log k\rceil)^2\right)^2 = 8(1+\log\lceil\log k\rceil)^2 \le 2\lceil\log k\rceil$ for a sufficiently large constant k. Hence, outputting pairs of colors as described also yields $O(1)$-vertex coloring of a cycle graph in $O(\log^\star n)$ many iterations.

Orient arbitrary. Orient each edge arbitrarily, e.g., from a vertex with a smaller to the vertex with a larger ID. Three situations can happen with this kind of orientation:

(1) A vertex has no parent: this can be handled by outputting the 0-th bit from the corresponding vertex.

(2) A vertex has a single parent: This is handled as before, i.e., by invoking $\textsc{Recolor-Vertex}$.

(3) A vertex v has two parents: color v by color $2\lceil\log k\rceil$.

Verifying that this coloring is a proper vertex coloring with at most $2\lceil\log k\rceil + 1$ colors is an easy exercise.

3.3 From $O(1)$ to 3 colors

Assume now that a cycle is colored by $k \in O(1)$ colors. We aim to recolor this cycle so that we use 3 colors. Since k is a constant, this can be done in $k-3$ iterations by invoking Algorithm 3 for $k-3$ times.

```
  Input :
          A cycle graph $G = (V, E)$
          A $k$-vertex coloring $C$, for $k > 3$
  // We assume that $C$ contains colors 0 through $k-1$.
1 Let $C'$ be a coloring.
2 For each $v$ such that $C_v < k-1$ set $C'_v = C_v$.
3 For each $v$ such that $C_v = k-1$, vertex $v$ performs the following steps: Let $x$ and $y$ be the
    colors in $C$ of $v$'s two neighbors. Set $C'_v$ to be the smallest non-negative integer different
    than $x$ and $y$.
4 return $C'$
```

Algorithm 3: A function that takes on input k-vertex coloring for $k > 3$ and outputs a $(k-1)$-vertex coloring.

Observe that C'_v set on Line 3 of Algorithm 3 is at most $k-2$ as we are assuming $k > 3$.

References

[CV86] Richard Cole and Uzi Vishkin. Deterministic coin tossing with applications to optimal parallel list ranking. *Information and Control*, 70(1):32–53, 1986.