

Lecture 10: June 30

*Lecturer: Vijay Garg**Scribe: Pankaja A*

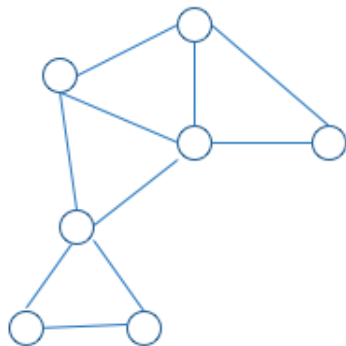
10.1 Introduction

This is a lecture on Breaking Symmetry and we discussed the sequential and parallel algorithm for the graph coloring problem.

- Graph coloring problem
- Graph as a ring
- Parallel algorithm for graph coloring

This set of lecture notes will briefly re-examine the topics covered in this lecture, in the order in which they appeared during class.

10.2 Graph Coloring Problem

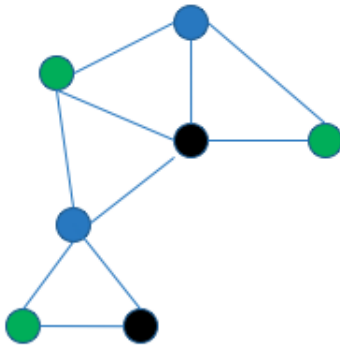


Consider a graph as shown above with a set of vertices V and a set of edges E . The problem statement is to color all the vertices of the graph such that no two adjacent vertices have the same color.

Prob statement: color the vertices u, v s.t. $(u, v) \in E \Rightarrow \text{color}(u) \neq \text{color}(v)$

The goal here is to use as few colors as possible.

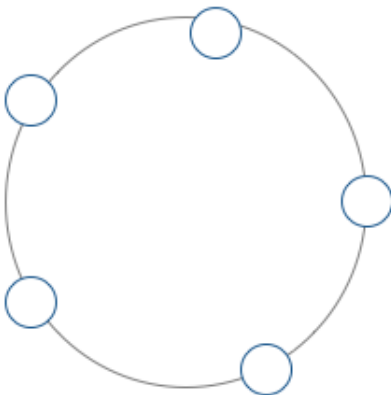
The below graph addresses this problem statement using 3 colors.



Solution: We start at one vertex and color it using one color(Blue), then we move to its neighboring vertex and color it using a different color(Green). Then we move to the neighbor of vertex(Green), this is the neighbor of vertex(Blue) too, thus now we color it using a third color(black).Repeat this process until all vertices are colored. After coloring all vertices we see that we can solve this problem using 3 colors.

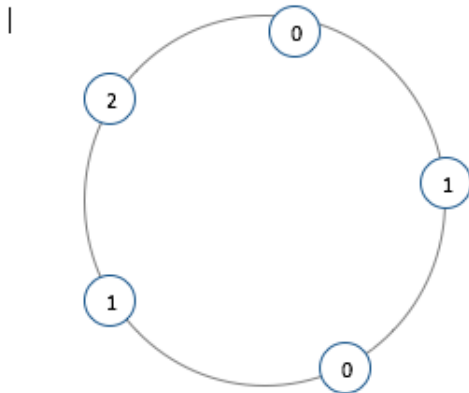
10.2.1 Graph as a Ring

Now lets look at a more restricted problem when the graph is in the form of a ring as shown below.



Think of the nodes in this graph as stations and they use frequencies to talk to each other.

From the list ranking problem, we know that the nodes can be divided into 2 classes(even and odd). So the goal here is to divide the nodes into 2 groups, such that even nodes does some task, and the odd nodes do other tasks.



To color this ring using sequential algorithm, we would start at any node, label it 0 (0 - represents the color here), the next neighboring node will be 1, then 0, then 1 and so on until all the nodes are colored.

When the number of nodes in the ring is even, this can be done using just 2 colors (0,1).

When the number of nodes in the ring is odd, we need 3 colors (0,1,2)

The time and work complexity for the sequential approach is :

$$T(n) = O(n)$$

$$W(n) = O(n)$$

10.2.2 Parallel Algorithm

Now lets look at how the same problem can be solved in parallel.

The professor asked us to think and come up with some solution before presenting the actual solution.

The problem is really hard if we have to come up with a deterministic strategy.

The Parallel algorithm is given below:

Given: some initial coloring(possibly different for every one)

```

1 for all i in parallel do
2   c[i] = i
3   // Begin with n colors and then reduce colors
4   repeat m times
5     for all i in parallel do
6       k := least significant bit where c[i] and c[next[i]] differ
7       d[i] := 2*k + c[i]_k

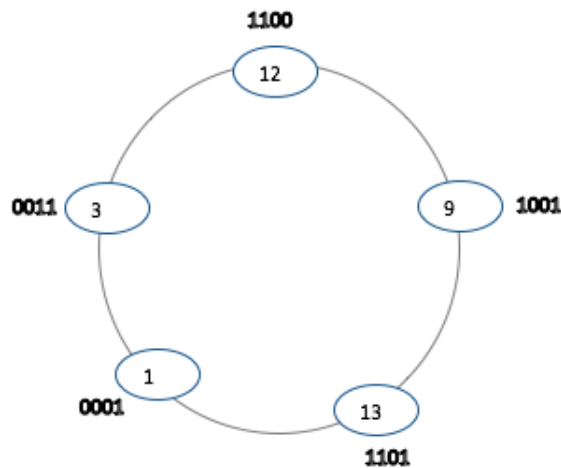
```

k is the first bit by which the two numbers differ.

$c[i]_k$ is the kth bit of c[i]

Now lets apply this algorithm on a ring graph. Lets say in this graph color is represented using bits, and

two colors differ by one bit.



This is the initial round of colors(applying the first for loop).

Lets start with node 12 (binary rep: 1100), the adjacent node which we take is node 9 (binary rep : 1001).

1100 and 1001 - the first bit that differs in these two numbers is 0^{th} bit. Thus $k=0$, and $c(i)_k = 0$

so, $d(i) := 2 * 0 + 0 = 0$

The professor gave us some more examples so that this is clear.

Consider 9 (bin rep : 1001) and 13(binary rep : 1101) - the first bit that differs in these two numbers is 2^{nd} bit. Hence $k=2$ and $c(i)_k = 0$. so, $d(i) := 2 * 2 + 0 = 4$

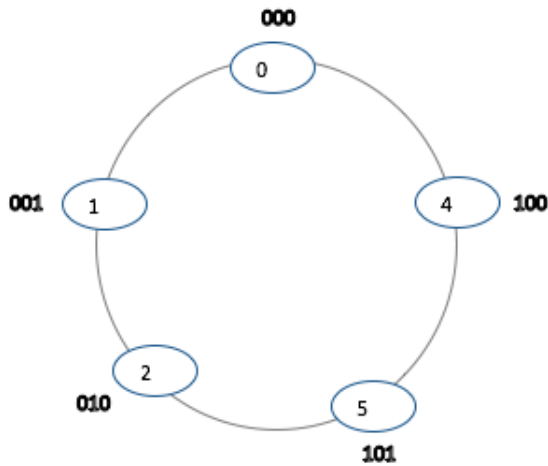
Consider 13 (bin rep : 1101) and 1(binary rep : 0001) - the first bit that differs in these two numbers is 2^{nd} bit. Hence $k=2$ and $c(i)_k = 1$. so, $d(i) := 2 * 2 + 1 = 5$

Consider 1(bin rep : 0001) and 3(binary rep : 0011) - the first bit that differs in these two numbers is 1^{st} bit. Hence $k=1$ and $c(i)_k = 0$. so, $d(i) := 2 * 1 + 0 = 2$

To summarize,

vertex	c	k	d
12	1100	0	0
9	1001	2	4
13	1101	2	5
1	0001	1	2
3	0011	0	1

So, with this calculation, the ring colors will be as below.



Now after the first round, max number is 5, and 5 can be represented using 3 bits. So, we can run this algorithm until we get 6 colors.


What is the deduction here?

If we have 2^t colors in c , after 1 iteration there will be $2t + 1$ colors.

If we start with n colors, the number of colors after 1 iteration will be $O(\log n)$


If we take n and reduce by $\log n$, the length will be

$$n \rightarrow n/2 \rightarrow n/4 \rightarrow n/8 \dots$$

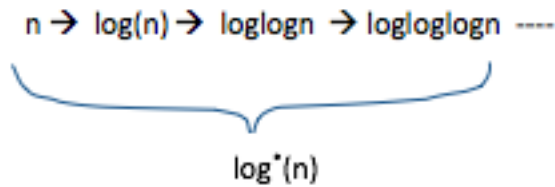

 $\log(n)$

if we start we n colors and reduce it by $\log \log n$, the length will be

$$n \rightarrow n^{1/2} \rightarrow n^{1/4} \rightarrow n^{1/8} \dots$$


 $\log \log(n)$

Note that now we are going from



The above sequence is called $\log^* n$

$$\log^* n = \min[i | \log^{(i)}(n) \leq 1]$$

$\log^{(i)}(n)$ expression means $\log n$ iterated i times.

Lets analyze the work and time complexity of this algorithm.

The inner loop is $O(1)$ The outer loop is repeating m times, and m is $\log^* n$.

$$T(n) = O(\log^* n)$$

$$W(n) = O(n \log^* n)$$

10.2.3 How can we reduce the number of colors further?

Now we have a ring with 6 colors - 0,1,2,3,4,5. So, how can we further reduce the number of colors?

Our aim is to use only colors 0,1,2. To start with, lets not bother about colors 4 and 5, lets only get rid of color 3.

To remove color 3, all nodes with color 3 chooses in Parallel color 0,1 or 2 (whichever is available). Every node choose a color different from its predecessor and successor. Using this approach we have removed 3.

So, using the same approach, remove 4 and 5. Thus after 3 parallel steps, there will be only 3 colors.

For this algorithm to work, the assumption is that the initial coloring works and colors are still valid i.e no two neighbours have the same color.

The algorithm takes $O(n \log^* n)$ work, as there are n processors and each takes $O(\log^* n)$ time. This is not work optimal but close to work optimal as $\log^* n$ is a very slow growing function.

There is another algorithm for this which is $O(1)$ complexity, but this was not discussed in the class.