Locality

Locality Everywhere

Outline

- Locality everywhere.

- Locality in Computing — Distributed Comp.
- Local Coloring
- Coloring Trees
- Lower Bounds

Locality Everywhere

Locality Everywhere!

- Locality is everywhere:
  - Physics
  - Biology
  - Social Sciences
  - Mathematics
- They have differences and similarities.

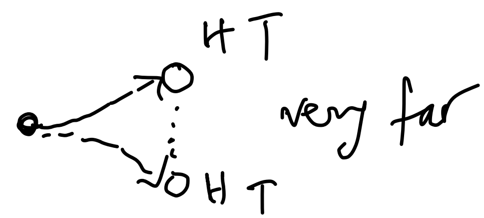
Locality in Physics

- An object is only directly influenced by its immediate surroundings.

- A theory using the principle of locality is said to be a "local theory".

Theory

- Relativity is a local theory
  - It limits the speed at which such influences can travel to the speed of light c.



- Quantum mechanics is not a local theory.
  - A measurement made on one of a pair of separated but entangled particles causes a simultaneous effect, the collapse of the wave function, in the remote particle (i.e. an effect exceeding the speed of light).

Locality in Biology

- Phenotypes might be influenced by local variations and effects.
  - Shape
  - Size
  - Color
  - Nature
  - Other environmental factors
- In turn, this affects the genotype.
- Quantum Biology is a newly developing field for the study of non-local biological phenomena.
  - Bird navigation

Salmon, Turtle

Locality in Social Sciences

Social

- Local Characteristics
  - Language
  - Behaviour
  - Culture
  - Food

- Global Phenomena
  - Cascades
  - Rumors
- How do certain events cascade?

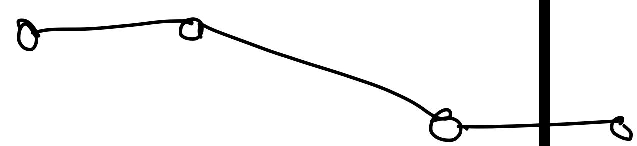
Locality in Mathematics

- It has a proximity interpretation.

- Related somehow to distance. (geometric distance)

- Concerns phenomena that are geometrically close to each other.

- Locality is influenced by distance but is not the same thing as location!



G :

paths between nodes

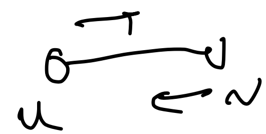
paths of a certain length

u sends messages to all nodes three hops away

Locality in Computing

Locality

- Usually it means:
  - the execution of a process depends on nearby processes.
  - there is no dependency between events that occur far away.
- It has a special role in computing and communication.
  - What can be computed globally if there is a restriction on how far information can propagate?
- Can you elect a leader
  - making use only of local information?



Locality

- Decision made at node u not affected by nodes far away from u.

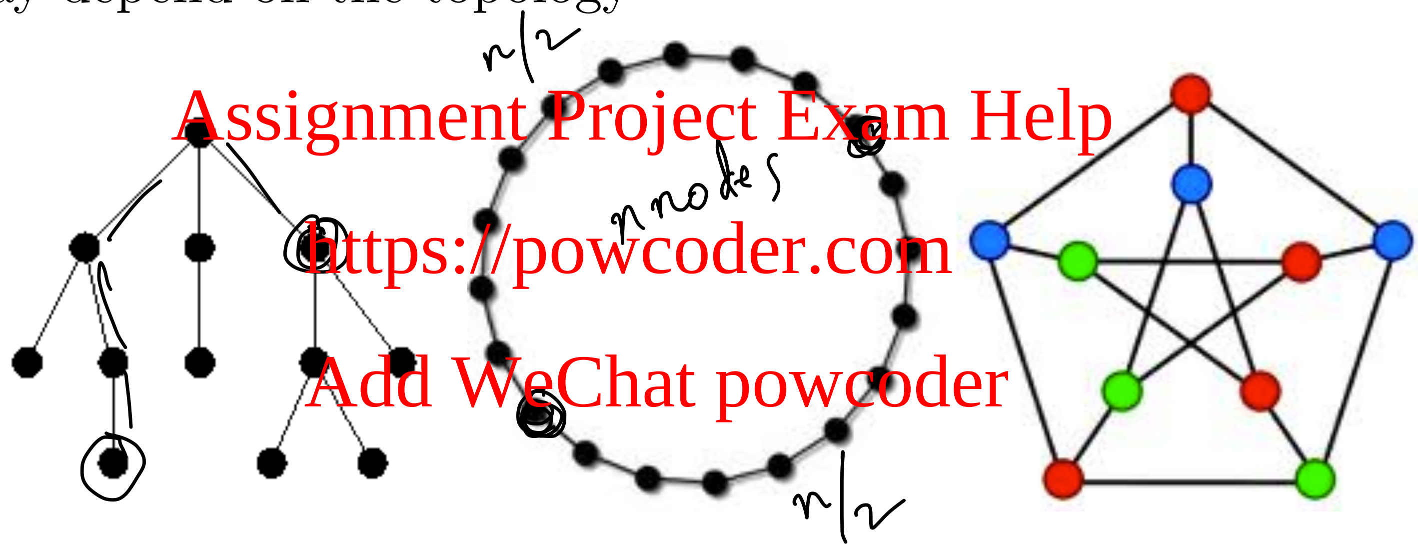
locality is somehow hop-distance

- How do we quantify "far away" from u?

How far is local?

- Given that locality is influenced by distance "how far is far away"?

- May depend on the topology



- How do you parametrize locality?
- Best to study specific problems!

Coloring

- Global vs Local Algorithms

- On a Line
- On a Tree

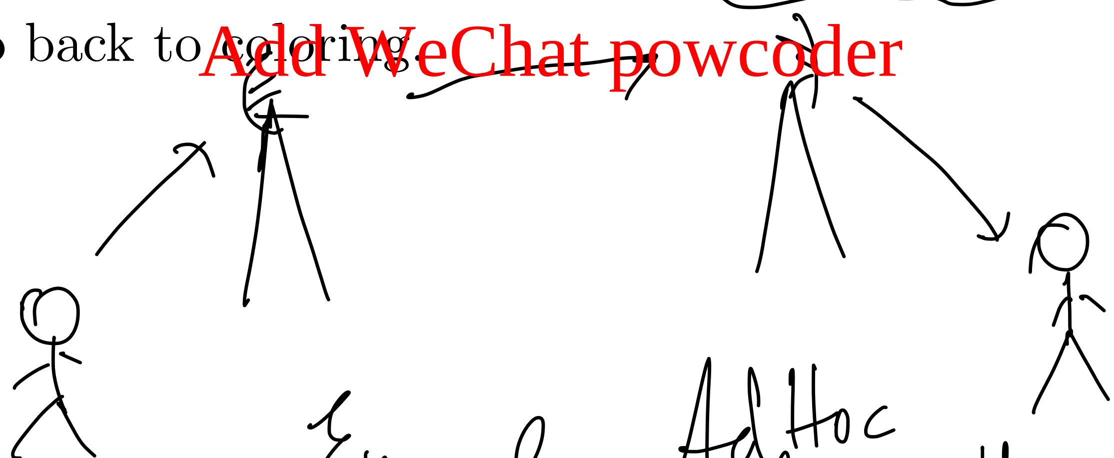
Local Algorithms in DC

- An algorithm is local if messages initiated by the nodes do not propagate too far from their originator.

  - How can you ensure correctness of the algorithm?
  - Which problems can you solve this way?
  - How far is too far?

- Local approach is important for wireless communication!

- Lets go back to coloring.

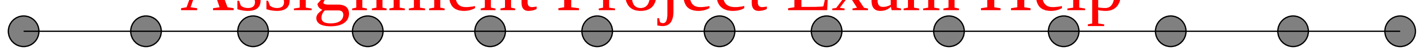


Example: Ad Hoc Bluetooth

Coloring

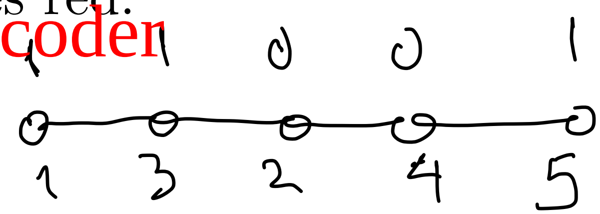
- A vertex coloring is an assignment of colors to vertices of a graph so that any two adjacent vertices are assigned different colors.

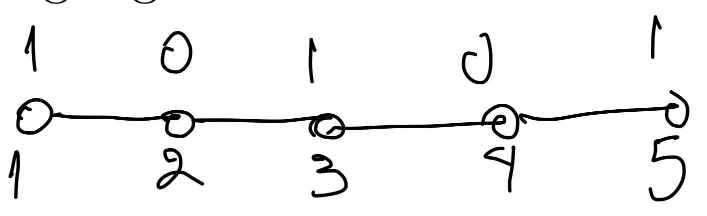
- How do you color a set of points on a line?



- If nodes have identities $1, 2, \ldots, n$ then color nodes with even identities blue, and with odd identities red.

  – Is the algorithm correct?
  – Is this a local algorithm?
  – Is there a local colouring algorithm?





Global vs Local Coloring

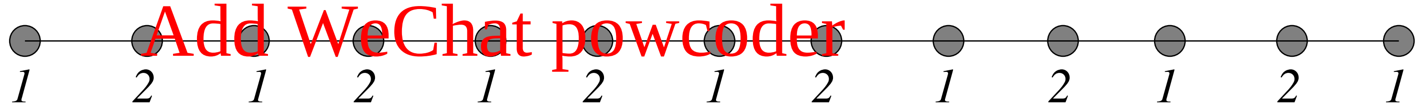
- Before a node decides on its colour it must collect information about its neighboring nodes.

- There are two ways to do this depending on how far this information collection can spread

1. Globally
2. Locally

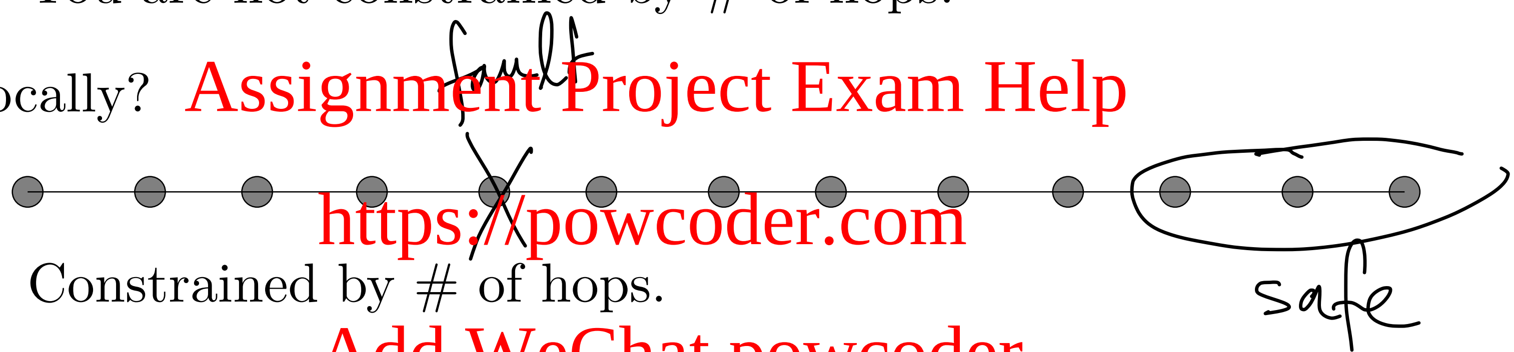
Globally/Locally

- Globally?



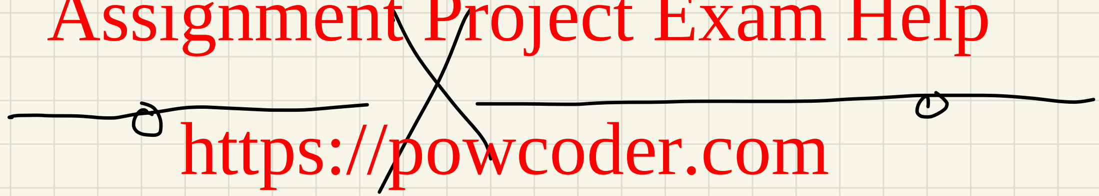
  - You are not constrained by # of hops.

- Locally?



  - Constrained by # of hops.
  - In a distributed setting, the difficulty lies in keeping the assignment of colors consistent throughout the graph despite the fact that propagation is limited!

Why do you want a local coloring algorithm

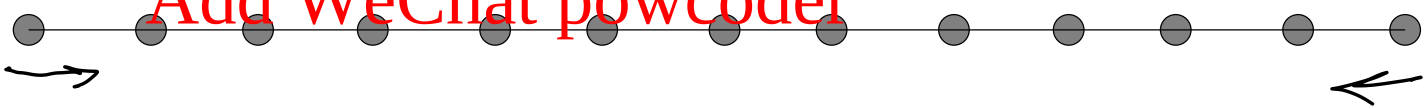


If algorithm is global, then it is not fault tolerant

Local algorithms are more robust to failures (node or link)

Coloring with Restricted Number of Hops

- Consider nodes “independently” initiating coloring.

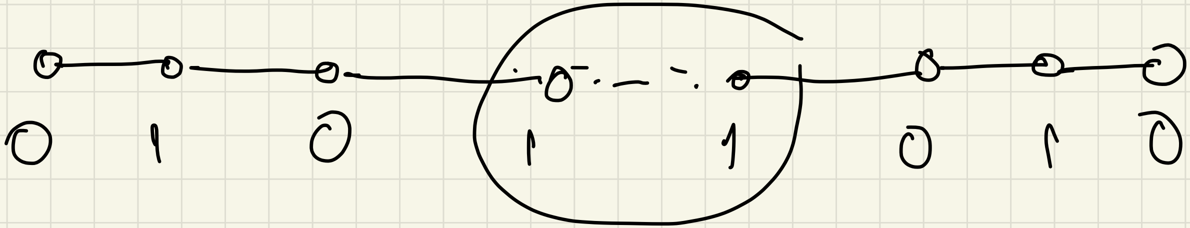


- If the number of hops a message can propagate is restricted you may not be able to complete the coloring!

- If a given set of nodes start coloring at the same time how do you ensure consistent coloring?

  – Nodes will start with their own identifiers.

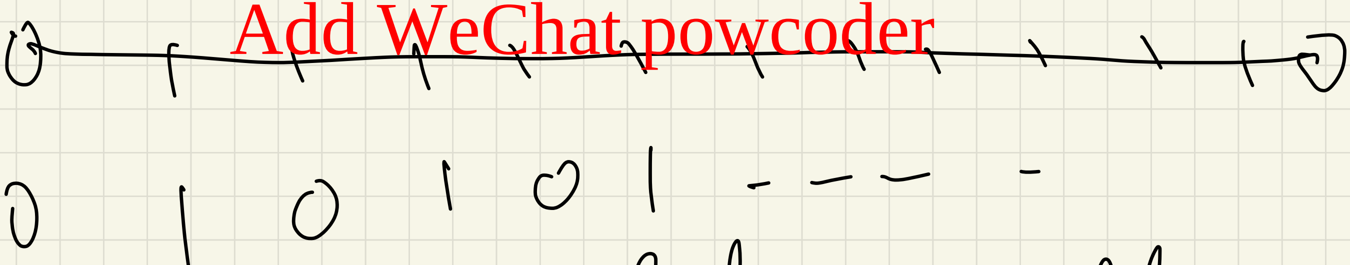
- More than that, you may have to use more than the minimum required number of colors so as to achieve a correct coloring!
- Regardless of the number of colors you use
  – can you achieve a proper coloring, and
  – at the same time restrict the number of hops?



Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder



Limit on the # of hops will
affect the # of colors used!

Quantifying Locality: Network

- Consider a class $\mathcal{N}$ of networks.

- A typical network $G = (V, E)$ in $\mathcal{N}$ is a graph with n vertices.
  - Line,
  - Ring,
  - Tree,
  - etc.

- The concept should be applicable to all classes of graphs (networks).

Quantifying Locality: Distance

- Locality should depend on distance.

- Let $n \to h(n)$ be an integer valued function:
  - $h(n)$ is the number of hops allowed in a network of size n.
- Examples:
  - $n \to h(n) = 1$,
  - $n \to h(n) = c$, c is some constant,
  - $n \to h(n) = \log n$,
  - $n \to h(n) = \sqrt{n}$,
  - $n \to h(n) = n$,
  - $n \to h(n) = \log^* n$, etc

h-local is an algorithm which for every node in a graph of size n nodes messages are propagated at most h(n) hops

Quantifying Locality: Problems

- Consider a problem $\mathcal{P}$ (e.g., colouring), and a class $\mathcal{A}$ of synchronous, distributed algorithms solving $\mathcal{P}$ for $\mathcal{N}$.
  - The class $\mathcal{A}$ of distributed algorithms is h-local if during the execution of an algorithm $A \in \mathcal{A}$ on a network $G \in \mathcal{N}$ (on n vertices), a message emanating from a node will never propagate more than $h(n)$ hops from its originator.

Propagation distance (hop-distance) is determined by the function h.

Which Problems in DC are Local?

- Not all problems are going to be h-local, for a given function h.

- Which ones are h-local, for a function $n \to h(n) = c$, where c a constant?
  - Leader Election
  - Spanning Tree
  - Maximum Independent Set {maximal: maybe}
  - Coloring
  - Minimum Dominating Set

$\log^* n$ algorithm

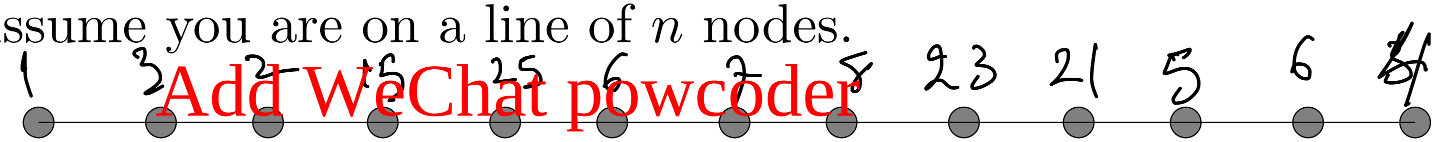
- For which topologies?

$\log^* 10^{100} \leq 5$

Local Coloring

Coloring a Line Graph: Assumptions

- Assume you are on a line of n nodes.

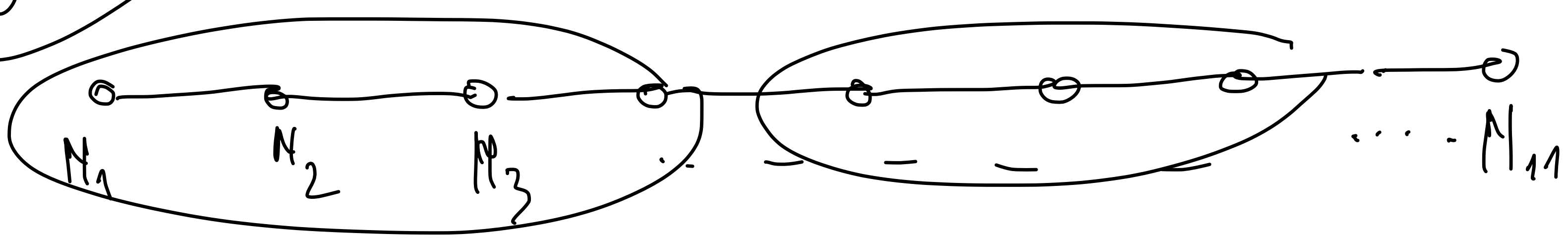


- To start, assume that each node v has a distinct identity id_v (for example, either their location or the network interface card would do).

  - Identity selection is much easier than the coloring problem...besides we also know several ways to solve this problem!

$\log^*(111) \leq 3$

All students 111



Local Coloring Algorithm

- Our main goal is to show

- **Theorem 1** *There is a coloring algorithm which can 3-color any line in $O(\log^* n)$ time, where*
  - $\log^* n$ *is the iterated lograithm of* n

    $\underbrace{\log(\log(\ldots(\log n)))}\leq 2$

  - *in the algorithm all nodes start with their identifiers.*

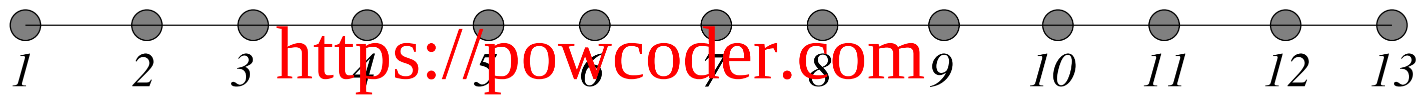
- This result is important in certain types of networks (like wireless) where messages should not propagate too far!

- **NB:** Note the important parameters taken into account:
  - Final number of colors in the graph.
  - Termination time of the coloring algorithm.

Assumptions for Coloring

- Let $v \to c_v$ be an arbitrary coloring of the vertices.
  - Observe that $c_v := id_v$ is a legal coloring!

- For example,
  - the identity assignment below is a colouring using n colors,



  - and so is any permutation of the identities.

line of n vertices

L_n = : can be colored with n colors :

$2^0 = 1, 2^1, 2^2, \ldots, 2^n$

$$1^2, 2^2, 3^2, \ldots, n^2$$

$$n^2, 3^2, 5^2, \ldots, 7^2$$

Assumptions for Coloring

Every vertex v is initially colored c_v

- Represent each c_v as a sequence of bits.
  - Let $|c_v|$ be the number of bits in c_v, and
  - $c_v(i)$ the i-th bit of c_v.

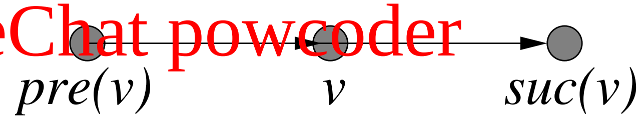
- **Example:** 576 512 594
  - $c_u = 594 = 512 + 64 + 16 + 2 = 2^9 + 2^6 + 2^4 + 2^1$.
  - In binary $c_u = 1001010010$
  - $c_u(i)$ is the ith bit where counting starts from $i = 0$ from left to right: $c_u(0) = 1, c_u(2) = 0$.

- The **concatenation**
  - of two sequences s, s' of bits is the sequence ss'.
  - **Example:** if $s = 1010$ and $s' = 110$ then $ss' = 1010110$

$\underbrace{10110 \quad 1011011}$

Idea for an Algorithm on a Line

- Assume an ordering of the vertices (left to right would do).



- Starting Rule:
  - Start with any legal coloring,
    - for example $c_v := id_v$, for all v.
  - Color "leftmost vertex" with the bit 0.[a]

- Any other starting coloring would do.

[a] This is a starting condition and we will need to justify it: will do this later!

Recoloring Rule

- Since nodes $u \to v$ are neighbors (with u preceding v), their current colors must be different: $c_u \neq c_v$.

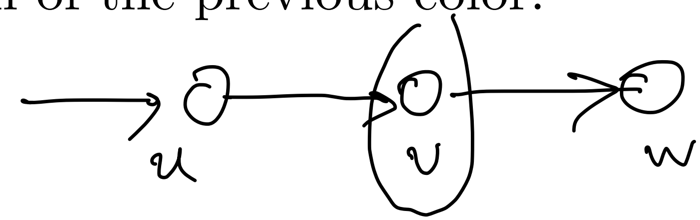
- Produce a new "legal" coloring for a vertex v from the current one, say c_v, as follows:

  - Find the first index $1 \leq i \leq |c_v|$ such that v's color differs from the colour of its predecessor.

  - Set new color to "$\overset{\text{index}}{i}$ concatenated with $\overset{\text{bit}}{c_v(i)}$": $c_v \to ic_v(i)$;

- Recoloring rule guarantees that neighbors will get new **different** colors.

- **NB:** Bit representation of each new color is of length logarithmic of the length of the previous color!



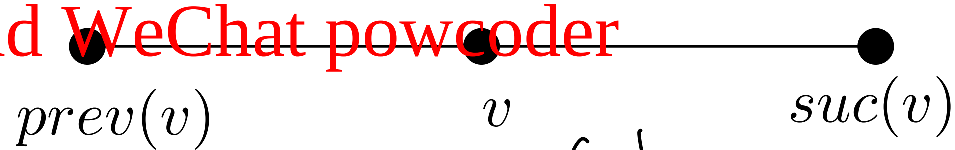
$\tilde{v}$

color : c_v $\qquad 1 \le i \le |c_v|$

$i \quad c_v(i) \qquad (\log c_v) + 1$

Coloring Algorithm for Vertex v

- Assume an ordering of the vertices (left to right would do).



$Ic_v(I) \rightarrow$

- **Coloring Algorithm:**

1. $c_v \leftarrow id_v$;
2. **Repeat:**

   (a) $\ell \leftarrow |c_v|$;

   (b) **if** v is "leftmost vertex" then set $I \leftarrow 0$
   else set $I \leftarrow \min\{i : c_v(i) \neq c_{pre(v)}(i)\}$;
   (c) Set $c_v \leftarrow Ic_v(I)$; /* concatenation */
   (d) Inform the successor $suc(v)$ of v of this choice;
3. **Until** $|c_v| = \ell$; /*Until length does not change */

Example (1/2)

- Given two nodes $u \to v$.

- Lets show how the color of node v changes from the old color c_v to a new color c_v.
  - A similar change occurs to the color of u, but this is influenced from the predecessor of u.

$u \longrightarrow v$

- Let their current colors be $c_u = 594$ and $c_v = 631$.
- Convert to binary:

  $c_u = 512 + 64 + 16 + 2 = 2^9 + 2^6 + 2^4 + 2^1$
  $c_v = 512+64+32+16+4+2+1 = 2^9+2^6+2^5+2^4+2^2+2^1+2^0$
- $c_u = 1001010010$ and $c_v = 1001110111$

Example (2/2)

- Consider the two nodes with colors

$$c_u = 1001010010 \text{ and } c_v = 1001110111$$

- What is the smallest i such that $c_u(i) \neq c_v(i)$?

Assignment Project Exam Help

- Line up the bits

https://powcoder.com

$$1001010010$$

Add WeChat powcoder

$$\uparrow$$

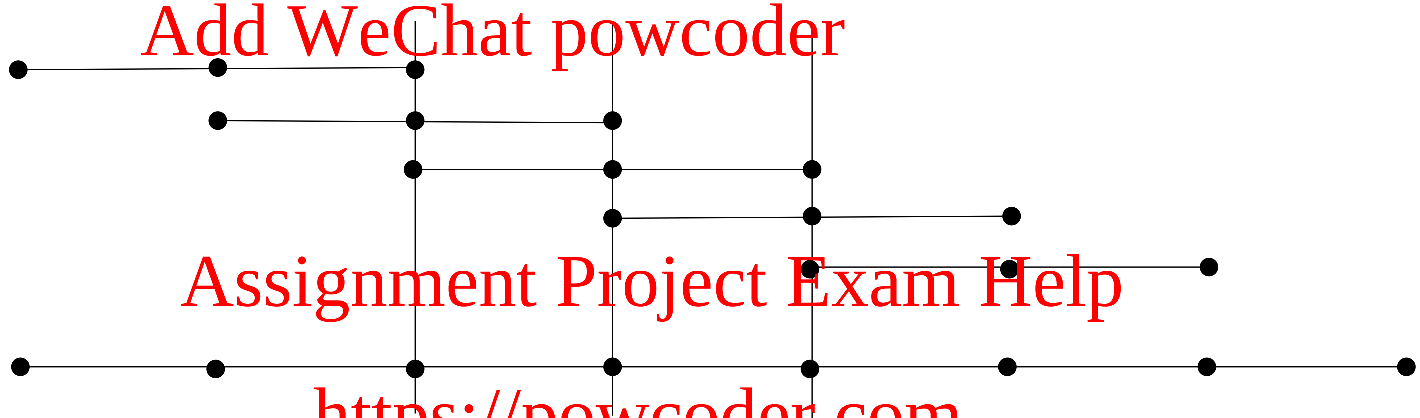
$$1001110111$$

- So $i = 4$ (counting starts from 0); in binary 4 is 100 and the new colour of v in binary representation is

$$ic_v(i) = 1001 = 9$$

Execution of Coloring Algorithm

- A node receives input from its predecessor...

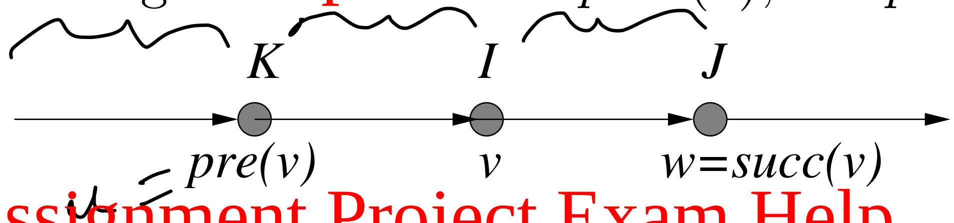


- ...and provides input to its successor.

$\log^* n$

Correctness: Legal Coloring (1/2)

- Consider three consecutive neighboring nodes u, v, w at some iteration of the algorithm with $u = prev(v), v = pre(w)$.



- Let I, J be the indices picked by v, w in Step 2(b), respectively.
  - $I := \min\{i : c_u(i) \neq c_v(i)\}$ and $J := \min\{j : c_v(j) \neq c_w(j)\}$
  - v, w receive the new colours:

$$c_v \leftarrow I c_v(I)$$

and

$$c_w \leftarrow J c_w(J)$$

Correctness: Legal Coloring (2/2)

- We need to show that $Ic_v(I) \neq Jc_w(J)$.

- There are two cases to consider:

1. If $I \neq J$ then rule 2(b) ensures that the new labels $Ic_v(I)$, $Jc_w(J)$ as defined in 2(c) differ in a bit
   - because I, J do

2. If $I = J$ then rule 2(b) ensures that the new labels as defined in 2(c) differ in the last bit
   - Recall that $c_u(I) \neq c_v(I)$ and $c_v(J) \neq c_w(J)$
   - Since $I = J$ we have that $c_u(I) \neq c_v(I)$ and $c_v(I) \neq c_w(I)$
   - The new labels for v, w will be $Ic_v(I)$ and $Ic_w(I)$ and by choice of I we have that $c_v(I) \neq c_w(I)$.

Number of Rounds

5

- At the start, $K_0 = K = O(\log n)$ is the max number of bits of a node in the original ID coloring.

$5 \log n$: n^5

- Let K_r denote the number of bits in the color representation after the rth iteration.

r-th iteration c_v

r+1-st '' $\lceil \log K_r \rceil + 1$

- Observe that $K_{r+1} = \lceil \log K_r \rceil + 1$.
  - Therefore the second coloring will be of roughly $\log \log n$ bits, the third of roughly $\log \log \log n$ bits, etc.

- As a matter of fact the "sizes of the colours" shrink very rapidly!
  - The size of the colour (measured in bits) in the new step is the logarithm of the size of the colour in the previous step!

Iterated Logarithm: $\log^*$

- $\log^* n$ is not really a logarithm:
  - it is rather the number of iterations of the log function on a number n until it stops having an effect!

- **Log-Star** (in base 2) of n:
  - Is the number of logarithms in base 2 needed so that starting from n you get down to ≤ 2.

- Can be defined in any base! Here we look only on base 2.

$\lceil \log 3 \rceil + 1 = 3$

2

4

$\log 4 = 2$

15

$\lceil \log 15 \rceil + 1 = 5$

$\lceil \log 5 \rceil + 1 = 4$

$\lceil \log 4 \rceil + 1 = 3$

Definition of $\log^*$

- **Iterated Definition of $\log^* n$:** Let
  - $\log^{(1)} n = \log n$, and
  - $\log^{(x+1)} n = \log(\log^{(x)} n)$, for $x \geq 1$.

  Then $\log^* n =$ first integer x such that $\log^{(x)} n \leq 2$. [a]

- **Recursive definition of $\log^* n$:**

$$\log^* x = \begin{cases} 1 & \text{if } x \leq 2 \\ 1 + \log^*(\log x) & \text{if } x > 2 \end{cases}$$

[a] $\log^{(x)} n$ should not be confused with $\log^x n$: the logarithm to the power x.

Example

- Log-star is a very slowly growing function.

- Consider the number $n = 2^{2^5}$.

$$\log(2^{2^5}) = 2^5$$

$$\log(2^5) = 5$$

$$\log(5) \approx 2.32192809489$$

$$\log(2.32) < 2.$$

Hence, $\log^*(2^{2^5}) = 4$.

- Log-star of all the atoms in the observable universe (estimated to be 10^{80}) is 5.

The Starting Nodes: Something Wrong?

- Recall the leftmost node was given the color 0.

- It is not clear from the description of the algorithm why the identities of the nodes "located" at the beginning of the line are reduced to constant size.

  0 c $\log^* n$

  – By beginning we mean the first $O(\log^* n)$ nodes.

- Observe that the identities of the nodes after location $O(\log^* n)$ are indeed reduced to constant size.

  $\log^*(\log^* n)$

- Can remedy this by adding an additional step at the end of the algorithm:

  – The first $O(\log^* n)$ nodes run a recoloring algorithm to reduce their colors to constant size.

- Note that this step takes additional time $O(\log^* n)$.

  $O(\log^* n) + \log^* n$

Six Coloring in $\log^* n$ Iterations

- If K_i = number of bits in the coloring after i iterations then
  - $K_{r+1} = \lceil \log K_r \rceil + 1$.
  - $K_{r+1} < K_r$ as long as $K_r \geq 4$.

If $x \geq 4$

$1 + \lceil \log x \rceil < x$

- In the final iteration r we have that $K_r = K_{r-1} \leq 3$.

$K_r \leq 3$

- Therefore in the final coloring you have
  - at most three choices for an index to a bit in the $(r-1)$ st coloring, and
  - two choices for the value of the bit,

which gives a total of six colors.

Can color the line in 6 colors. But we promised can do it in 3!

- It turns out,
  - we can improve on # of colors from six to three, but
  - cannot improve on the $\log^* n$.

Three Colors Suffice

- How do we reduce the number of colors from six to three?

- Suppose that the algorithm we discussed before has colored a line with the six colors $0, 1, 2, 3, 4, 5$ as follows

$$0\ 5\ 4\ 2\ 5\ 3\ 0\ 3\ 1\ 5\ 4\ 2\ 3\ 0\ 1\ 4\ 3\ 2\ 4\ 0\ 1\ 0\ 2\ 4\ 5$$

- How do you color it using only the colors $0, 1, 2$?

0, 1, 2, 3, 4, 5

Three Colors Suffice

- Start with the sequence

0 5 4 2 5 3 0 3 1 5 4 2 3 0 1 4 3 2 4 0 1 0 2 4 5   2

- Eliminate 5: by choosing a color from 0, 1, 2

0 1 4 2 0 3 0 3 1 0 4 2 3 0 1 4 3 2 4 0 1 0 2 4 0   2

- Eliminate 4: by choosing a color from 0, 1, 2

0 1 0 2 0 3 0 3 1 0 1 2 3 0 1 0 3 2 1 0 1 0 2 1 0

- Eliminate 3: by choosing a color from 0, 1, 2   2

0 1 0 2 0 1 0 ~~0~~ 1 0 1 2 1 0 1 0 1 2 1 0 1 0 2 1 0

2

Coloring Rings

- **Theorem 2** *There is an algorithm which can 3-color any ring of size n in $\log^* n$ time.*

- Same algorithm.

Can you reduce the number of rounds any further

$\Omega(\log^* n)$

e.g. inverse Ackerman function

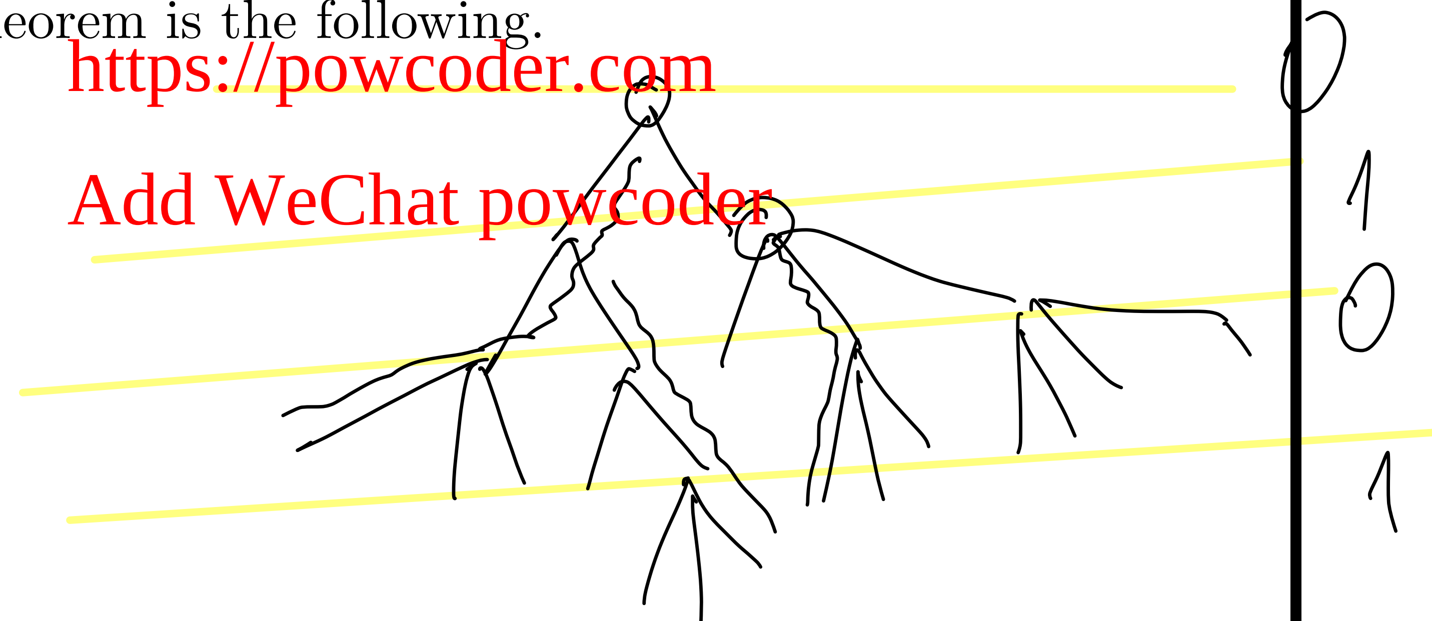
Coloring Trees

From Lines to Trees

- The line colouring algorithm also works on trees!

- The basic assumption is that you must have a node of the tree designated as the root!
- Further, other nodes must have a parent (i.e., a predecessor)!

- The main theorem is the following.



6-Coloring Theorem

- **Theorem 3** *There is an algorithm which can* 6*-color any tree in* $\log^* n$ *time.*

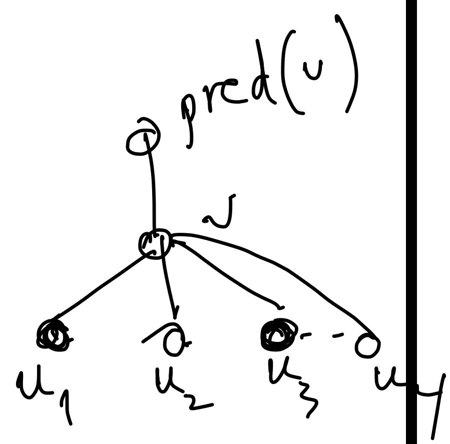
|

3-color

6-Coloring Algorithm for Trees: Vertex v

- **Algorithm: 6-Color**

1. $c_v \leftarrow id_v$;
2. **Repeat:**
   (a) $\ell \leftarrow |c_v|$;
   (b) **if** v is "the root" then set $I \leftarrow 0$
   else set $I \leftarrow \min\{i : c_v(i) \neq c_{parent(v)}(i)\}$;
   (c) Set $c_v \leftarrow I c_v(I)$; /* concatenation */
   (d) Inform all children of v of this choice;
3. **Until** $|c_v| = \ell$;



- Why is the algorithm correct?

3-Coloring Theorem for Trees

- **Theorem 4** *There is an algorithm which can* 3*-color any tree in* $O(\log^* n)$ *time.*

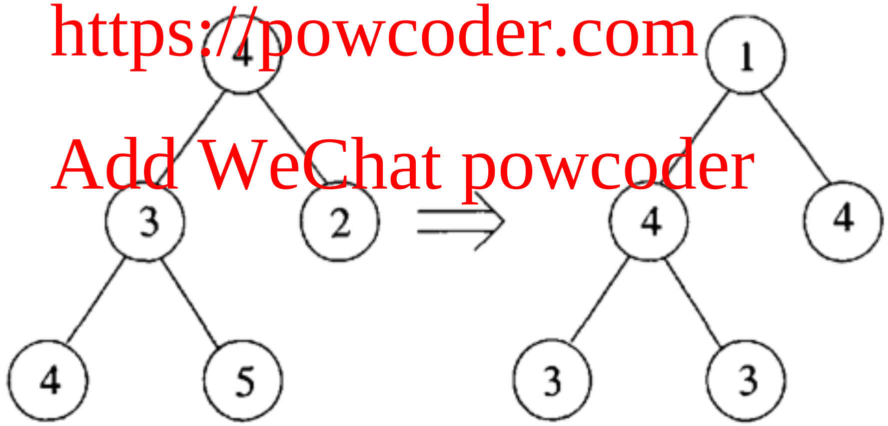
- The reason is that the coloring on the descendants of a given node is independent when done on disjoint paths.

Shift-Down Algorithm

- The color reduction method is called "shift-down".

- **Algorithm Shift-Down**

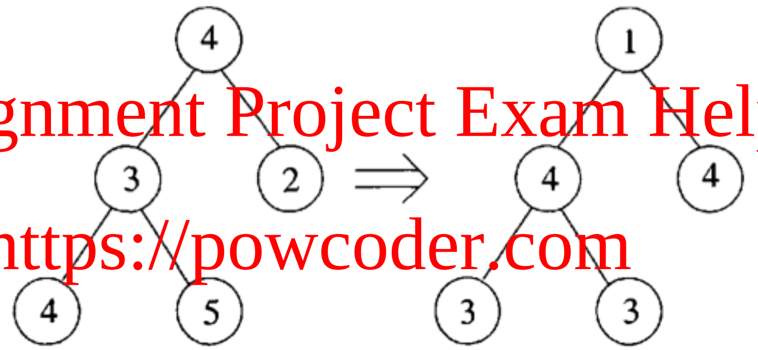
1. Concurrently at all vertices:
2. Recolor each non-root vertex by the color of its parent.
3. Recolor root by a new color, different from its current one.



- Why is "shift-down" correct?
- Colors (of the original coloring) are shifted down.

Analysis of Shift-Down Algorithm

- **Lemma 1 (Analysis of Algorithm Shift Down)**
  Algorithm Shift Down preserves coloring legality; also siblings are monochromatic.



- Two vertices $v = parent(w), w$ are recolored by $c_{parent(v)}$ and c_v, which are different since c was a legal colouring.

- If $v = root$, then the new colors are x and c_v, where x is some color different from c_v.

- Also, all children of some vertex v get the same new color c_v.

Final Color Reduction

- Now assume the six colors employed in the tree are

$$0, 1, 2, 3, 4, 5$$

- The final three reduction steps involve cancelling colors

$$3, 4, 5$$

one at a time.

- In the end, there will be three colors left $0, 1, 2$.
  - This is done by Algorithm Six2Three

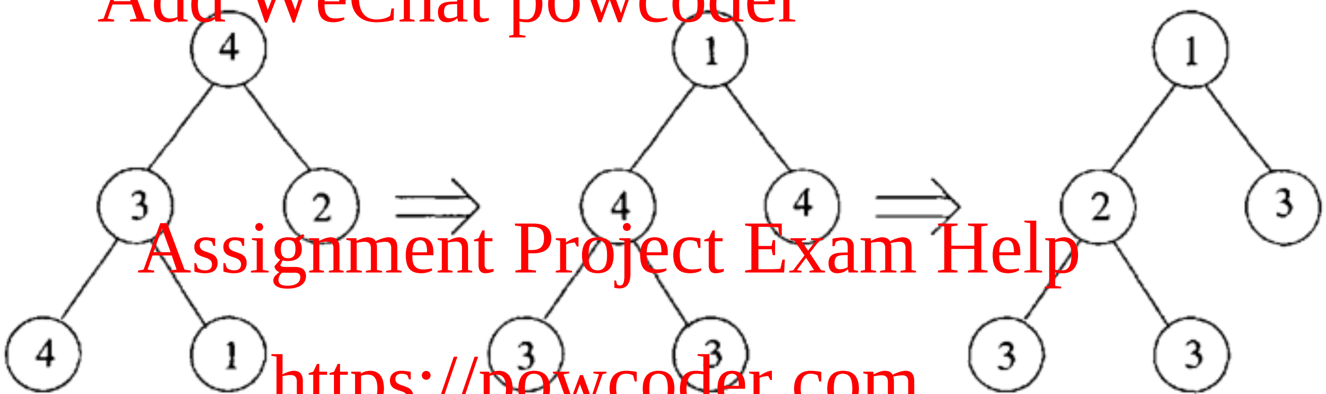
Six2Three Algorithm

- **Algorithm Six2Three**

1. **for** $x = 5, 4, 3$ **do** /* Cancel color x */
2. Perform subroutine **Shift-Down** on the current colouring;
3. **if** $c_v = x$ **then**
4. v chooses new color $c_v \in \{0, 1, 2\}$ not used by any of the neighbors.
5. **endif**
6. **endfor**

Example of Six2Three

- Recolouring method



- Example discarding color 4.

Analysis of Six2Three

- **Theorem 5 (Analysis of Algorithm Six to Three)**
  Algorithm Six2Three colors a tree with three colors in time $O(\log^* n)$.

- Each vertex colored x will find an available color from the set $\{1, 2, 3\}$,

  – since by the Shift-Down Lemma at most two of these colors are occupied, one by its parent and one by its children.

- Now note that recoloring the x colored vertices simultaneously creates no problem since they are all mutually nonadjacent.

Optimality

- Fast tree-coloring with only 2 colors is more than exponentially more expensive than coloring with 3 colors.
  - In a tree degenerated to a line, nodes far away need to figure out whether they are an even or odd number of hops away from each other in order to get a 2-coloring.
  - To do that one has to send a message to these nodes. This costs time linear in the number of nodes.

Lower Bounds

Can anything be better than $\log^* n$?

- The only thing better than $O(\log^* n)$ running time is $O(1)$ running time!
  - A 2-coloring is possible with $O(1)$ running time in a distributed system with GPS!

- It turns out that we can prove a lower bound of $\Omega(\log^* n)$ on the time required to color the n-vertex line (ring) by three colors.
  - This implies a tight bound of $\Theta(\log^* n)$ on the time required for 3-coloring the line (ring).

$\Omega(\log^* n)$ Lower Bound

- **Theorem 6** *Every deterministic, distributed algorithm to color a directed ring with 3 or less colors needs at least* $(\log^* n)/2 - 1$ *rounds.*

- The proof uses a theorem of Frank P. Ramsey.

(22 February 1903 19 January 1930).

- We will not prove Theorem 6 here.

Generalizations and Additional Results

- Linial (1992) proves that

  - in rooted d-regular tree $T_{d,r}$ of radius r, any synchronous distributed algorithm running in time $\leq \frac{2}{3}r$ cannot color $T_{d,r}$ by fewer than $\frac{1}{2}\sqrt{d}$ colors.

  - an arbitrary graph G of order n and max degree Δ, can be colored with $5\Delta^2 \log n$ colors in one time unit distributively.

  - for G labeled, in time $O(\log *n)$ it is possible to color G with $O(\Delta^2)$ colors in a distributive synchronous algorithm.

- There exists a deterministic distributed algorithm for coloring arbitrary graphs with max degree Δ;
  - can be colored with $\Delta + 1$ colors in $O(\Delta \log^* n)$ time.

Exercises[a]

1. For any graph $G = (V, E)$ define the chromatic numbers

$$\chi_{centralized}(G), \chi_{distributed}(G), \chi_{local}(G)$$

   for centralized, distributed, and local computation.

   (a) How do they differ?

   (b) Is there a natural order of these three quantities?

2. Define the concepts of centralized, distributed and local for any algorithmic computation and make a comparison.

3. Let $n \to h(n)$ be an integer valued function, where $h(n)$ is the number of hops allowed in a network of size n to complete the computation. Formulate the various types of computation discussed above in terms of the function $h(n)$.

4. **($\star\star$)** Consider Exercise 3. If $h(n) = n$ then the number of

[a] Do not submit!

colors is 2. If $h(n) = 1$ then the number of colors is 3. For which threshold value of $h(n)$ does the number of colors jumps from 2 to 3?

5. Compute $\log^*(10^{1000})$.

6. Compute $\log^*(2^{2^{2^{16}}})$.

7. Explain in more detail (than the slide presented in class) that the local coloring algorithm (before the six $\to$ three reductions) reduces to a six coloring.

8. Show in detail that on the line graph three colors suffice.

9. Prove that a $\log^*$ coloring algorithm is possible on a ring. How many colors does it require?

10. Prove in detail the correctness of the $\log^*$ tree coloring algorithm.

Sources

- L. Barenboim, and M. Elkin. Distributed graph coloring: Fundamentals and recent developments. Synthesis Lectures on Distributed Computing Theory 4.1 (2013): 1-171.
- N. Linial. Locality in distributed graph algorithms. SIAM Journal on Computing 21.1 (1992): 193-201.
- D. Peleg, Distributed Computing: a Locality Sensitive Approach, SIAM, 2000.