Distributed computing

D

1) The issue with the GHS algorithm arises during the selection of the **Minimum Outgoing Edge (MOE)**. The algorithm assumes that the MOE is unique for each fragment, ensuring that only one edge is chosen per fragment at each step. If edges do not have unique weights, multiple edges with the same weight may be considered as MOEs, leading to ambiguity and inconsistency.

Problems Caused by Non-Unique Weights:

1. **Ambiguity in Selecting the MOE:**
   - Multiple fragments may select the same edge as their MOE, causing duplicate merges or fragment conflicts.
   - Alternatively, fragments may fail to select the correct MOE, violating the algorithm's assumption of progress toward the MST.
2. **Violation of MST Properties:**
   - Without unique weights, it becomes possible to select edges that:
     - Introduce cycles.
     - Fail to merge fragments correctly.
3. **Break in the Proof:**
   - The proof of correctness assumes that the MOE is always unique and part of some MST. Non-unique weights break this assumption, invalidating the guarantee that merging fragments preserves acyclic behavior in the MST.

By the way we can solve this really easliy if we assume that each node have a unique name just the the minimum name…

---

2) Suppose there exists a **uniform synchronous algorithm** AAA that computes the AND of the input bits x1,x2,...,xnx_1, x_2, ..., x_nx1,x2,...,xn in an anonymous ring.

Case 1: All x_i = 1:

- If all xi=1x_i = 1xi=1, every node behaves identically because the algorithm is **uniform** and the ring is **anonymous**.
- The algorithm must compute AND (x1,x2,...,xn)=1 correctly in this scenario.

Case 2: Larger Ring with One x_j=0:

- Now, consider a larger ring where all x_i=1 except for a single node j with x_j=0.

- Since the algorithm is uniform and nodes have no unique identifiers, they cannot distinguish their roles or positions in the ring.
- This makes the behavior of nodes in the larger ring **indistinguishable** from the ring in Case 1 where all x_i=1.

Contradiction:

- In the larger ring, the AND function should compute 0, but due to uniformity and anonymity, the algorithm cannot distinguish between the two cases.
- This leads to an incorrect computation of the AND function in the larger ring.

There is no uniform synchronous algorithm for computing the AND function in an anonymous ring, as the nodes cannot reliably distinguish between configurations with identical behavior.

This prof come for the lecture where we showed that we can not select a leader in an anonymos ring because there is no way to distingish between nodes.

By the way I assumed you can not get the answer only from some of the nodes, because if you could then it is easy and the question is not true, because I can say on round one (sync settings),
If your value if 0: then send 0 on round 1,
If your value is 1: don't do anything,
Then you can assume that if you did not get any message after round 1 then the result it 1.

b) Each node i has an input x_i∈{0,1} and knows its neighbors (clokwise and counter clockwise) in the ring.

- **Algorithm Steps:**

  - Each node i sends its input x to its clockwise neighbor.
  - If the message is 0 all forward message from now on be 0
  - The receiving node forwards the received value
  - This process repeats until every node has received inputs from all other n−1 nodes in the ring (which is a circle so we wait until we recive n – 1 meesage).

- **Termination:**

  - After n−1 rounds, each node computes the AND of all received inputs (AND(x1,x2,...,xn)) and outputs the result.
  - If any x_i=0 is detected during message passing, nodes can terminate early, output 0, and stop forwarding.

- **Message Complexity:**

- In the worst case, every node sends n−1 messages to propagate all inputs to all other nodes.
- Total messages: n·(n−1)=O(n^2).

Note: we can also just make the 0 nodes a zombie node,
That mean they send a message to both nighbors to change thire value to 0 and they will recive acknolagment when the nigbors sent a meesage to the nigbors to change the state from 0 to one.

c) In an asynchronous ring, each node has to learn the input of every other node to correctly compute the AND function. This requires propagating information around the ring, and in the worst case, this leads to a message complexity of Ω(n^2).

---

1. **Information Propagation:**
   - Each node must determine whether any input xi=0 exists in the ring to compute the AND function.
   - Since nodes are unaware of the global structure of the ring or the inputs of other nodes, the only way to gather this information is through message exchanges.
2. **Worst-Case Scenario:**
   - In the worst case, the asynchronous nature of the system means that messages may take the longest path through the ring.
   - Each node must communicate its input x_i to all other n−1 nodes, and every node must receive all inputs to compute the AND function.
3. **Message Complexity:**
   - Each of the n nodes sends n−1 messages (one to each other node).
   - Total message complexity = n·(n−1)=Ω(n^2).

If we could get a leader or a non uniform ring we could have done it better, by for example Leader send a message to his clockwise node, and just compute the AND until get get the message from the other side which will take O(n) with O(n) messages.

d) if a node a zero has it value, then send to both nighbords thire value and they do the same,
at round n/2 + 1, either all nodes are 0 or 1, and we know the ring and.

1) For every node check
   - Self.incomeMessageLeft == 0 or self.incomeMessageRight == 0: self.v = 0
   - If self.v === 0 : self.sendLeft(Self.v), self.sendRight(Self.v)

a meesage will travel exactly once (or zero) on each node, and alert if a value is zero.
Message complexsity is n-1 => O(n)
Time n/2 => O(n)

3) We showed in the class the color reducation algorithm of a Graph G with n nodes (the identifier of the nodes of log(n) bits) took

$$O(\Delta \cdot \log(n))$$

Assume we have Δ^2 coloring , that mean that each color can be represented by log (Δ^2) bits,
That's mean if we first color the graph at t time with (Δ^2) colors then the total run time

$$T_{\Delta^2\ coloring} + T_{reducation} = O(t) + O(\Delta \cdot \log(\Delta^2)) = O(t) + O(\Delta \cdot 2 \cdot \log(\Delta))$$
$$= O(t) + O(\Delta \cdot \log(\Delta)) = O(t + \Delta \cdot \log(\Delta))$$

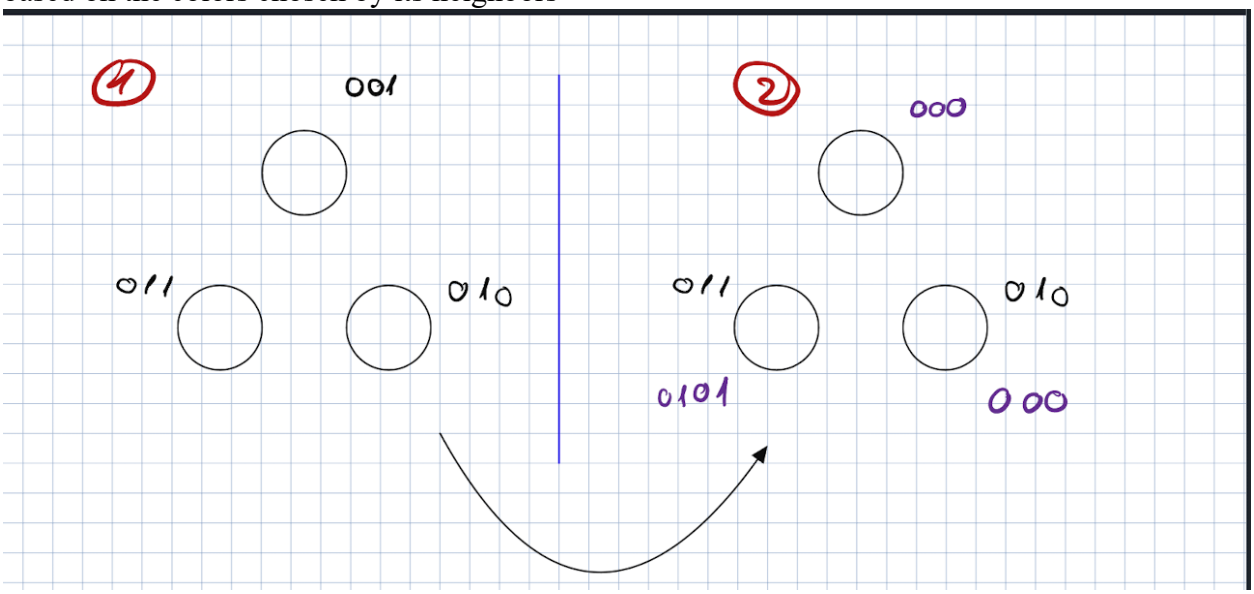
The same prof with more formal text ....

We know from the class that the **color reduction algorithm** for a graph G with n nodes (where the identifiers are O(logn) bits) takes O(Δ·log(n)) time.

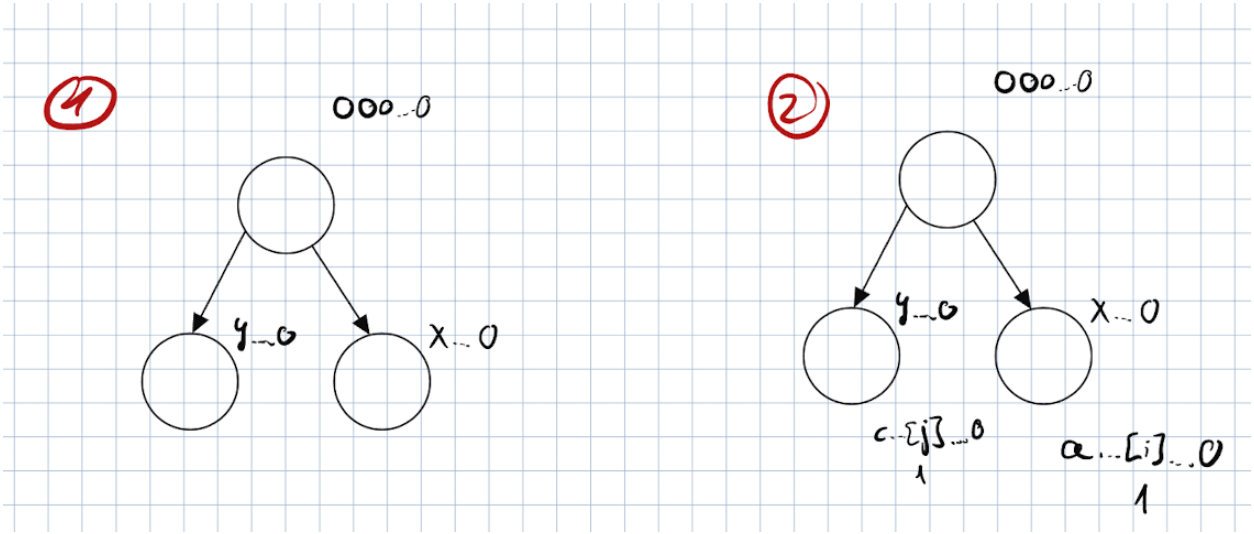
1. **Assume Initial Δ^2-Coloring:**
   - If we initially color the graph with Δ^2 colors in ttt synchronous rounds, each color can be represented using $O(\log(\Delta^2)) = O(2 \cdot \log(\Delta)) = O(\log(\Delta))$ bits.
2. **Apply Color Reduction:**
   - Using the color reduction algorithm, we reduce the Δ^2 colors to $(\Delta + 1)$ colors in $O(\Delta \cdot \log(\Delta^2))$ time.
3. **Total Runtime:**
   - Total runtime is:

$$T_{\Delta^2\ coloring} + T_{reducation} = O(t) + O(\Delta \cdot \log(\Delta^2)) = O(t) + O(\Delta \cdot 2 \cdot \log(\Delta))$$
$$= O(t) + O(\Delta \cdot \log(\Delta)) = O(t + \Delta \cdot \log(\Delta))$$

4) a) Forcing the root to always choose 000000000 prevents it from adapting to the colors of its neighbors, causing conflicts in the tree. To avoid this, the root must dynamically adjust its color based on the colors chosen by its neighbors



b) we can do this easily by setting the id of the root to 000…0 and the children id at [0] must be 0, so the color will not collide. Then even if the root stays 00…0 there is no issue.
We will show this example



If the root begins with color 000...0000...0000...0 and keeps updating its color to 000...0000...0000...0, the algorithm succeeds because:

1. **Root ID and Children IDs:**
    - If the root is assigned an ID of 000...0000...0000...0, the children can dynamically choose IDs such that their first bit differs from the root's color. This ensures no collision between the root and its children.
2. **No Conflicts in Coloring:**
    - As the children and grandchildren adjust their colors based on the root and their parent nodes, there will be no collisions. Even if the root continuously updates to 000...0000...0000...0, it dynamically adapts to its neighbors' colors.
3. **Example Illustration:**
    - Suppose the root RRR has color 000...0000...0000...0.
    - Its child C1C1C1 chooses a color 001001001, and the grandchildren choose colors distinct from C1C1C1 and their neighbors. The dynamic update ensures the tree maintains valid coloring.
4. **Dynamic Adjustment Ensures Success:**
    - The algorithm allows the root to repeatedly update to 000...0000...0000...0, avoiding conflicts due to the tree's acyclic structure and the dynamic nature of color assignments.