ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

FS 2023

Prof. Dr. Roger Wattenhofer

Exam
Principles of Distributed Computing

Wednesday, August 9, 2023
14:00 – 16:00

Do not open or turn until told to by the supervisor!

The exam lasts 120 minutes, and there is a total of 120 points. The maximal number of points for each question is indicated in parentheses. Your answers must be in English. Be sure to always justify (prove) your answers. Algorithms can be specified in high-level pseudocode or as a verbal description. You do not need to give every last detail, but the main aspects need to be there. Big-O notation is acceptable when giving algorithmic complexities. Some questions will ask you to fill in answers in a template. If you decide to start over you will find fill-in replacements at the end of the examination booklet. Please write legibly, illegible answers will not be graded.

Please write down your name and Legi number (your student ID) in the following fields.

Name	Legi-Nr.

Exercise	**Achieved Points**	**Maximal Points**
1 - Tree Algorithms MC		8
2 - Ivy MC		9
3 - Ivy		13
4 - Wireless		16
5 - Graph Neural Networks		19
6 - Labeling		16
7 - Self-Stabilization		23
8 - Stone Age		16
Total		**120**

1 Tree Algorithms MC (8 points)

For each of the following statements, indicate whether they are **TRUE** or **FALSE**. Ticking the correct box is worth 1 point. A correct justification of the correct answer is worth 1 additional point.

[8] In the synchronous setting, consider a graph with n nodes where the diameter $D = O(1)$ is constant. We want to construct a spanning tree T, rooted at a node s that is marked as the source.

	TRUE	**FALSE**
There exists an algorithm that computes a spanning tree rooted at s and has time complexity $O(1)$. Justification:	☐	☐
There exists an algorithm that computes a spanning tree rooted at s and has message complexity $O(1)$. Justification:	☐	☐
Given a precomputed spanning tree T rooted at s. We can do a broadcast from s using only edges of T and reach every node in the graph in $O(1)$ rounds. Justification:	☐	☐
Given a precomputed BFS spanning tree T rooted at s. We can do a broadcast from s using only edges of T and reach every node in the graph in $O(1)$ rounds. Justification:	☐	☐

Solutions

A) [8] In the synchronous setting, consider a graph with n nodes where the diameter $D = O(1)$ is constant. We want to construct a spanning tree T, rooted at a node s that is marked as the source.

	TRUE	**FALSE**
There exists an algorithm that computes a spanning tree rooted at s and has time complexity $O(1)$. Justification: *Reason: We can use the Dijkstra BFS construction that gives us the spanning tree in $O(D^2) = O(1)$ rounds.*	✓	
There exists an algorithm that computes a spanning tree rooted at s and has message complexity $O(1)$. Justification: *Reason: No, every node must at least know its parent. Therefore, every node has to receive at least one message.*		✓
Given a precomputed spanning tree T rooted at s. We can do a broadcast from s using only edges of T and reach every node in the graph in $O(1)$ rounds. Justification: *Reason: No, the tree T can have very large diameter, even if the original graph has constant diameter.*		✓
Given a precomputed BFS spanning tree T rooted at s. We can do a broadcast from s using only edges of T and reach every node in the graph in $O(1)$ rounds. Justification: *Reason: Yes, the tree T also has constant diameter.*	✓	

2 Ivy MC (9 points)

A) [9] Consider a 9-node complete graph on which we are executing the Ivy protocol. The initial configuration of the graph is shown in Figure 1.

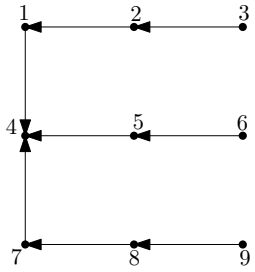
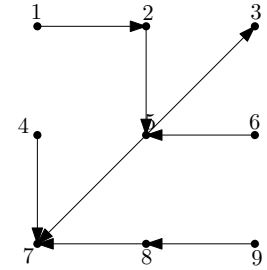
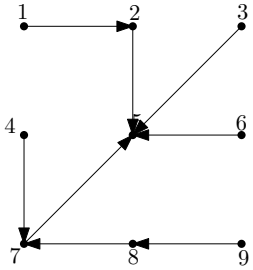


Figure 1: Initial state of the network

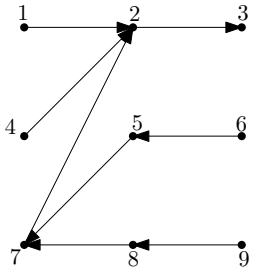
Suppose the nodes $2, 3, 5$ and 7 each request the token. The nodes can request the token simultaneously or sequentially in any order. For each of the following three configurations, decide whether it could be the final state of the network (select yes if it could be the final state, and no otherwise). Briefly explain why.



(a) ☐ yes ☐ no
Reason:



(b) ☐ yes ☐ no
Reason:



(c) ☐ yes ☐ no
Reason:

Solutions

A) The only correct answer is c) which corresponds to executing the sequence 5,7,2,3 sequentially. Answer a) is incorrect, since the final tree has 2 sinks. Answer b) is incorrect because if 5 is the final owner of the token, then there must be an edge connecting 7 with either 2 or 3, as these requests were served before 5 was served and thus their requests cannot go through node 5 given the initial configuration.

3 Ivy (13 points)

Tree topologies

In the exercise class, we saw that when n is a power of 2, there exists a tree consisting of n nodes and a sequence of requests such that the tree topology remains the same after each request in the Ivy protocol. In this exercise, we are interested in finding a sequence for a tree with 5 nodes for which the tree topology of the underlying undirected graph remains the same.

Given a directed graph, the underlying undirected graph is the graph we obtain when replacing every arc with an undirected edge. If there are arcs in both directions between two nodes, we only add one undirected edge. For example, Figure 3 shows four directed tree topologies at the bottom which all have the same underlying undirected graph which is displayed at the top.

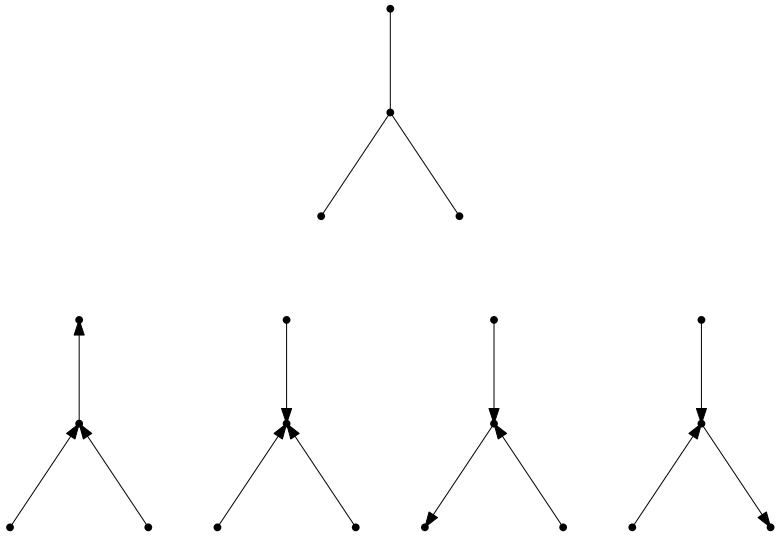


Figure 3: Four valid Ivy-tree topologies at the bottom that all have the same undirected underlying tree displayed at the top.

B) [3] Draw all non-isomorphic undirected trees with $n = 5$ nodes.

C) [10] Find an initial configuration of a tree consisting of 5 nodes, and a sequence of requests in which the topology of the underlying undirected tree does not change at any step. In your sequence, every node in the system must request the token at least once, and a node can only request the token if it is not already holding the token. To get full points, your sequence must have a length of exactly 5. That is, every node requests the token exactly once.

Draw the initial graph, and the state of the network after each request.

Solutions

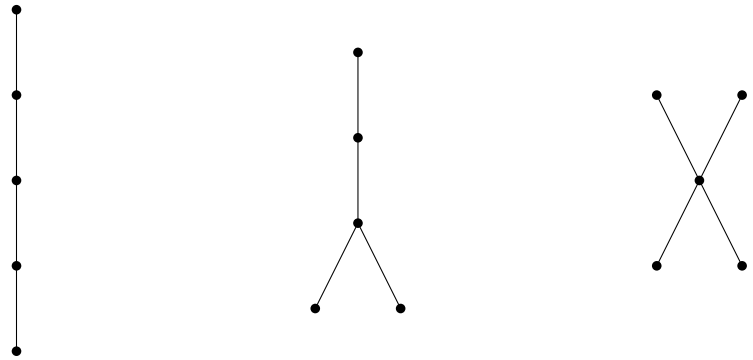


Figure 4: There are three non-isomorphic undirected trees on 5 nodes.

A)

B) There are many solutions that involve sequences of length longer than 5 because in the Ivy protocol, the topology of the underlying undirected graph does not change when a neighbor of the token holder requests the token. A possible solution of a sequence of length 5 that involves all 5 nodes is displayed in Figure 5.

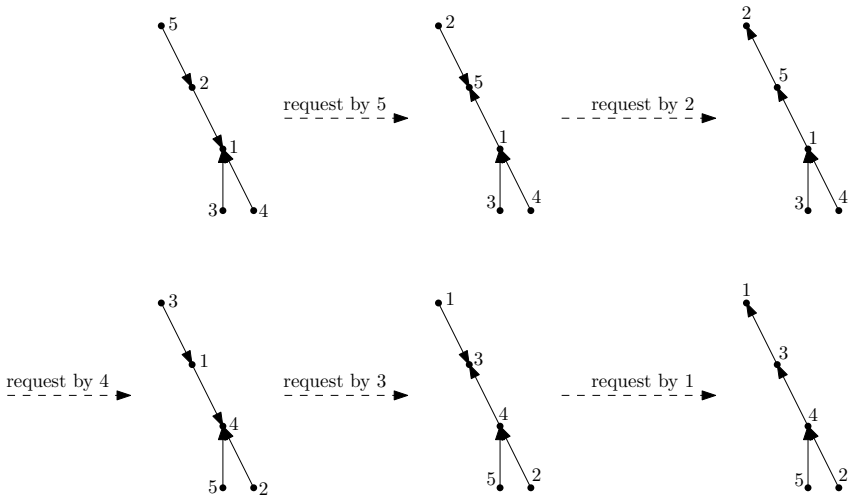


Figure 5: A possible solution to task **C)**, that scores full points.

4 Wireless (16 points)

We consider the wireless network model. However, in this question nodes are arranged on a graph, and their transmissions only reach and conflict with their 1-hop neighborhood. We assume collision detection (CD) is available to all nodes, accordingly, nodes observe a collision when two or more of their neighbors transmit simultaneously.

Consider the example where three nodes are connected on a line: $u_1 \leftrightarrow u_2 \leftrightarrow u_3$. If u_1 and u_3 transmit simultaneously, u_2 will observe a collision, while u_1 and u_3 cannot detect anything wrong with their transmission.

A) [6] Consider the extension of the example above, where n nodes lie on a path graph. The initiating node u_r wants to find out the number of nodes in the graph, while all other nodes have no information. u_r is situated at one end of the path, and aware of being at the end. Design an algorithm that runs in as few time slots as possible and succeeds deterministically. For full points, your algorithm should run in $2n$ time slots.

B) [10] Consider nodes connected on an arbitrary tree. The initiating node u_r wants to find the height of the tree, i.e., the radius of u_r. All other nodes have no information. Design an algorithm that runs in as few time slots as possible and succeeds deterministically. For full points, your algorithm should run in $2r + c$ time slots, where r is the radius of u_r, and c is a constant.

This page is intentionally left (almost) blank.

Solutions

A) **Nodes other than** u_r: When receiving a transmission they transmit themselves in the next time slot. They then check if they receive a transmission in the following time slot.

- If they don't, they transmit again and terminate.
- If they do, they wait for another transmission. Upon reception, they transmit in the following time slot and terminate.

Node u_r: Transmits during the first slot. If no message is received in the following slot, output 1. If a transmission is received, wait for another transmission. Upon reception thereof at time slot j, $\frac{j-1}{2}$ is the number of nodes on the path.

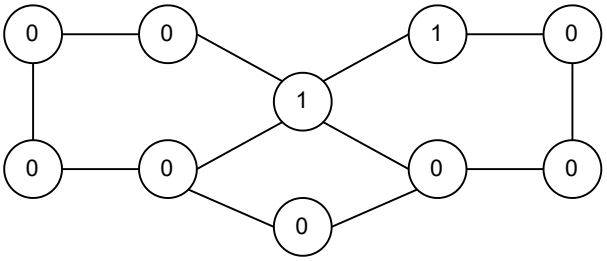
In essence, the idea is that a wave traverses the path, and by exploiting the synchronization of nodes the length of the path can be measured. The algorithm terminates after $2n$ time slots.

B) **Nodes other than** u_r: When receiving a transmission, nodes transmit themselves in the next time slot. They keep transmitting every second time slot until only one transmission is received during the time slot after its own transmission (this transmission stems from the parent, and the lack of collision indicates that all children terminated).

Node u_r: Transmits during the first slot and every second subsequent slot. If no transmission is received in the in-between slot $i+1$, output $\frac{i}{2}$.

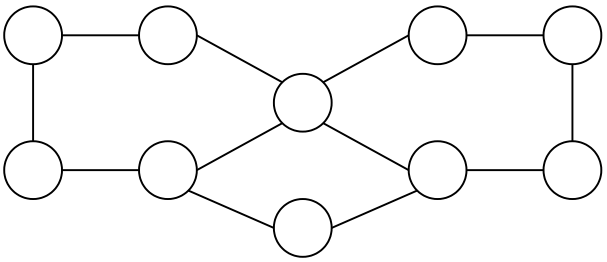
The algorithm requires exactly $2(h+1)$ time slots.

5 Graph Neural Networks (19 points)



You are given the graph above with initial node labels 0 and 1.

A) [3] Run one round of WL and draw color classes into the template (for example by assigning numbers to colors).



B) [2] How many rounds does WL need to finish on this graph?

C) [3] Consider a tree with a 2-coloring. Show that this does not suffice to distinguish all nodes in the graph with WL.

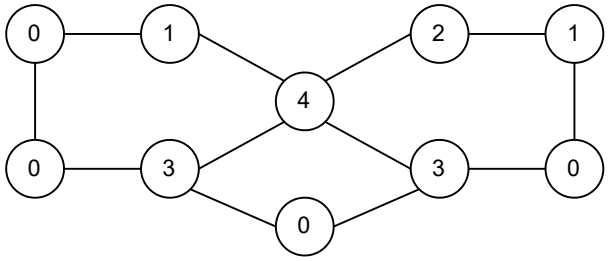
D) [5] A k-hop-coloring is an assignment of colors to nodes such that all nodes with distance k or less from each other have different colors. Is there a constant k such that in all graphs (not just trees), given an initial k-hop-coloring, all nodes can be distinguished from each other with WL?

E) [6] Now assume the tree is labeled with a 2-hop-coloring. Show that this suffices to distinguish all nodes in the tree with WL.

Hint: You may assume that you are given an initial marking on the tree that marks exactly one node.

Solutions

A) The color classes can be assigned as follows.



B) 3

C) Consider a star graph: the coloring will assign one color to the center and the other to the leaves. The leaves will get the same WL-labels here.

D) No. For fixed k, consider the $2(k+1)$-cycle with color classes assigned $1, 2, \ldots k+1, 1, 2, \ldots k+1$ around the cycle. This is a valid k-hop-coloring, but WL will not be able to distinguish nodes with the same initial color.

E) We first show that given the coloring we are able to compute a "leader", a node that we can distinguish from the rest. This can be done by computing the center of the tree and breaking a possible tie between two center nodes by using the coloring. Next, we can think of assigning every node a unique identifier sequence (or "address") by listing the colors on the path from the root to the node. This will always be unique when a 2-hop-coloring is used.

6 Labeling (16 points)

Given an undirected graph G on n nodes. We want a labeling scheme l to find the length of the shortest cycle that contains two given nodes in the graph. More precisely: Given only the labels $l(u)$, $l(v)$ of two vertices u, v you have to compute the length of the shortest cycle in G that contains both u and v. Cycles may not repeat nodes. If there is no such cycle that contains both u and v, you should report the length as ∞.

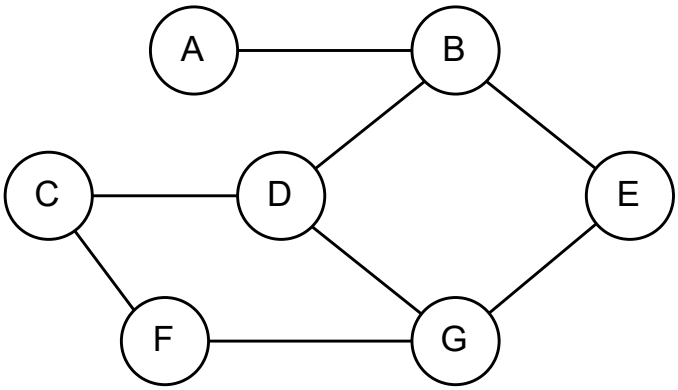


Figure 6: In this example graph, the shortest cycle that contains both C and G is the cycle of length 4 (C-D-G-F-C). Whereas the shortest cycle that contains both B and C is of length 6 (C-D-B-E-G-F-C). Note that A cannot form a cycle with any other node in the graph.

Note: You have to prove your statements, including correctness and length of the labeling scheme.

A) [6] Assume that k nodes in the graph are *golden*. Give a labeling scheme for the length of the shortest cycle with $\mathcal{O}(k \log n)$ bits when at least one of the two queried nodes is golden.

B) [10] Assume $n \geq 10$, design a labeling scheme that uses at most $\frac{n}{2} - 1$ bits or prove that no such scheme exists. You may use existing results from the script.

Solutions

A) For each possible query node precompute the solution for each case when the other queried node is one of the k golden nodes. The length of each cycle is at most n (fits in $\mathcal{O}(\log n)$ bits) and we can use the special value 0 to denote the nonexistence of a cycle. Hence, we can store all answers for a node in $\mathcal{O}(k \log n)$ bits. We also need to store node ids to identify the queried (golden) nodes. This can be done with $\mathcal{O}(\log k) = \mathcal{O}(\log n)$ extra bits per node.

B) **No** such scheme exists.
Assume there is a labeling scheme l^* for the shortest cycle between two vertices that uses at most $\frac{n}{2} - 1$ bits. We can use l^* to get a labeling scheme l' for adjacency on graphs. Given a graph G on n vertices, we construct a graph G' by adding a vertex s to G and connecting all other vertices of G' to s. Now, observe that u, v are adjacent if and only if u and v lie on a 3-cycle in G'. Therefore, l' can use the labels of l^* applied to G' and output u, v adjacent iff l^* outputs a 3-cycle. Moreover, the length of l' will be at most $\frac{n+1}{2} - 1 = \frac{n-1}{2}$, but we know from the script that for adjacency $l' > \frac{n-1}{2}$ has to hold, a contradiction.

7 Self-Stabilization (23 points)

Given an undirected simple graph $G = (V, E)$, a *matching* is a set of edges $M \subseteq E$ such that no two edges in M are incident (i.e., have any common endpoints). A matching M is *maximal* if there is no edge $e \in E \setminus M$ such that $M \cup \{e\}$ is a matching.

This question concerns a self-stabilizing distributed algorithm for computing a maximal matching of the communication network, which is the undirected graph $G = (V, E)$. Given a node $v \in V$, write $N(v) = \{u \mid (v, u) \in E\}$ for the neighborhood of v. The state of node $v \in V$ throughout the algorithm is denoted by $S_v \in N(v) \cup \{\perp\}$, where $S_v = \perp$ signifies that node v is unmatched, and $S_v = u$ means that v is matched to $u \in N(v)$. In order for the state of the whole system $(S_v)_{v \in V}$ to represent a matching, we require:

(Consistency) If $S_v = u$, then $S_u = v$. This implies that $M = \{(v, S_v) \mid S_v \neq \perp\}$ is a matching.

If Consistency holds, we further want that the matching is maximal:

(Maximality) There is no edge $(u, v) \in E$ such that $S_u = S_v = \perp$.

When Consistency and Maximality hold, we say that the system is in a *legitimate* state.

A) [2] Consider the path graph G with vertices $V = \{1, 2, 3, 4\}$ and edges $E = \{(1, 2), (2, 3), (3, 4)\}$. There are two legitimate states for the system, write them down.

S_1	S_2	S_3	S_4

We make the following simplifying assumptions. First, the state S_v of each node v can be written to by v and read by $v \cup N(v)$. Communication between adjacent nodes is implicit, by reading the state of neighbors and updating one's own state. Moreover, we assume that the system is controlled by a scheduler that, in each round, chooses a node $v \in V$ and executes one "step" of v. For our algorithm, the exact step logic is exhibited in Algorithm 1 below. The scheduler guarantees that every node $v \in V$ is picked to perform a step infinitely often.

Algorithm 1 One Step of the Self-Stabilizing Maximal Matching Algorithm

Whenever chosen by the scheduler, node v executes the following code.

1: **if** $S_v = \perp$ **then**
2: **if** $\exists u \in N(v)$ such that $S_u = v$ **then**
3: $S_v := u$ for an arbitrary such u. **(Line 3)**
4: **else if** $\exists u \in N(v)$ such that $S_u = \perp$ **then**
5: $S_v := u$ for an arbitrary such u. **(Line 5)**
6: **else if** $S_{S_v} \neq \perp$ and $S_{S_v} \neq v$ **then**
7: $S_v := \perp$ **(Line 7)**

The goal of a self-stabilizing distributed algorithm is to converge back to a legitimate state within finitely many rounds whenever a transient fault occurs (i.e., nodes having their state S_v corrupted to arbitrary values). Convergence means that a legitimate state is eventually reached, and afterward $(S_v)_{v \in V}$ no longer changes. Equivalently, one can think that the algorithm is started from an arbitrary state and no transient faults occur, which we will assume for this question. We will now show that our algorithm is self-stabilizing. For all parts, assume that no transient faults occur, but the initial system state is arbitrary.

Consider a node $v \in V$ with $S_v \neq \perp$. Call v *finished* if $S_{S_v} = v$. Call v *good* if $S_{S_v} = \perp$. Call v *bad* if $S_{S_v} = u$ for $u \in N(S_v) \setminus \{v\}$. Note that any node v with $S_v \neq \perp$ is exactly one of finished, good or bad. Moreover, note that nodes v with $S_v = \perp$ are neither finished, good, nor bad. Now, define the quantity $Q = (n+1)f + g - b$, where f is the number of finished nodes, g is the number of good nodes, and b is the number of bad nodes. As is customary, $n = |V|$ denotes the number of nodes in graph G.

B) [12] The following table specifies how the quantities f, g, b and Q change whenever one of the lines **Line 3**, **Line 5** and **Line 7** execute. Complete the entries in the empty cells by writing in each either $+c$, $-c$ (where c can be any positive constant), 0, < 0, > 0, ≤ 0, ≥ 0, or ?, depending on how the respective quantities change.

	f	g	b	Q
Line 3				
Line 5				
Line 7				

For instance, "$+5$" means "increases by 5", while "-4" means "decreases by 4", and ≤ 0 means "does not increase", while "?" means "other". **Be as precise as possible in your answer!** E.g., if the most precise correct answer is -4 and you write < 0, partial marks will be awarded, but if -3 and -4 are both possible, then < 0 earns full marks (≤ 0 does not).

C) [3] Argue why $(S_v)_{v \in V}$ can only change finitely many times.
Hint: use your answer to part **B)**.

D) [6] Assume part **C)** and consider the state after which $(S_v)_{v \in V}$ no longer changes, show that it is legitimate.

Solutions

A) The completed table is below:

S_1	S_2	S_3	S_4
2	1	4	3
$\perp$	3	2	$\perp$

B) The completed table is below:

	f	g	b	Q
Line 3	2			> 0
Line 5	0	1	0	1
Line 7	0	≥ 0	< 0	> 0

C) By part **B)**, whenever a node changes its state, Q increases by at least one. Since Q is bounded by $(n+1)n + n - 0 = (n+2)n$, it follows that $(S_v)_{v \in V}$ can only change finitely many times.

D) Assume for a contradiction that $(S_v)_{v \in V}$ is not legitimate. If Consistency is violated, this can happen in two ways: (i) $S_v = u$ and $S_u = \perp$, then waking up node u would perform $S_u := v$ (or equivalent for some v' with the same properties) contradicting that $(S_v)_{v \in V}$ no longer changes (the scheduler guarantees that waking up u would eventually happen); (ii) $S_v = u$ and $S_u = x$ with $x \neq v$, then waking up v would similarly perform $S_v := \perp$ and give a contradiction. Finally, if $(S_v)_{v \in V}$ represents a matching, but Maximality is violated, say for edge $(u, v) \in E$ satisfying $S_u = S_v = \perp$, then waking up either of them would make them change their state, again giving a contradiction.

8 Stone Age (16 points)

You are given a star graph with n nodes which has an odd number of leaves. In the beginning, every leaf-node is assigned either the color black or white.

Assume you are in the Stone Age model of the following type:

- Communication is done in synchronous rounds.
- Every node has a state and can broadcast the same message to all its neighbors or decide to not send anything. Nodes do not have IDs.
- We use the one-two-many model. This means a node can only distinguish between receiving 0, 1 or more than 1 message of a given type .
- Each node can use randomization (but does not know n, the size of the graph).
- Initially, each leaf-node starts in either a black or white state, while the center starts in the special center state.
- Overall, there can only be $\mathcal{O}(1)$ different node states and $\mathcal{O}(1)$ different message types. Therefore, you cannot store non-constant numbers.

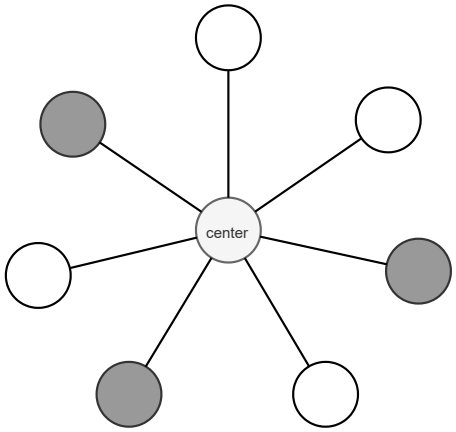


Figure 7: A star graph with 7 leaves (3 black and 4 white) and a center node. In this case, the majority color would be white.

For the following tasks it is sufficient to outline the protocols, rather than explicitly defining all state transitions. However, it should be argued why the protocol complies with the Stone Age model.

A) [3] Design a protocol so that the center node can decide if there is at least one black and one white node in the graph.

B) [5] Design a protocol so that with non-zero probability exactly one black and one white node send a message. After the protocol, both nodes should know that they were the only ones to send a message.

C) [8] Design a protocol so that each node in the star graph knows whether there are more black or white nodes in the graph. Moreover, all nodes should terminate at the same time. Your protocol must be correct, but you do not have to optimize your protocol to minimize the number of rounds (the time complexity could even be exponential).

Solutions

A) All leafs just send their initial state as the message. If there are at least one black and at least one white node in the graph the center node will receive either 1 or 1+ (by the one-two-many principle) for each message type.

B) Each node sends his own initial state with probability $\frac{1}{2}$. With probability $\frac{1}{2}^{n-3}\frac{1}{4} \cdot \text{black} \cdot \text{white} > 0$ exactly one white and one black node will send their message. The center node knows if this was successful if he receives exactly 1 message of each type (1+ is a failure as at least two sent a message). The center node can then acknowledge if the protocol was successful in the next round of communication.

C) We can reuse part A) and B) as building blocks. The idea is to cancel out white and black nodes and make them grey until one of the colors is gone and we know the majority color. So at the beginning of each phase the center node checks if there is at least one black and one white node in the graph. If that is not the case it announces the surviving color to all nodes and the protocol terminates. Otherwise, we make use of part B) and repeat it until exactly one white and one black node transmits a message. If successful, the center node will announce that they should become gray now (from now on they will no longer actively participate in the protocol) and a new phase starts, otherwise repeat (which might take exponentially many rounds).