ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

FS 2018

Prof. Dr. Mohsen Ghaffari,
Prof. Dr. Roger Wattenhofer

Exam
Principles of Distributed Computing

Monday, August 13, 2018
09:00 – 11:00

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. Please write legibly. If we cannot read your answers, we cannot grade them.

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

Name	Legi-Nr.

Exercise	**Achieved Points**	**Maximal Points**
1 - Multiple Choice		12
2 - Worst-case Spanning Trees		18
3 - Sorting Networks		16
4 - Ivy on Trees		20
5 - LCL Problems on Directed Cycles		12
6 - Orienting a Minimum Spanning Tree		20
7 - Maximal 10-matching		22
Total		**120**

1 Multiple Choice (12 points)

Evaluate each of the following statements in terms of correctness. Indicate whether a statement is true or false by ticking the corresponding box. Each correct answer gives one point. Each wrong answer and each unanswered question gives 0 points.

A) [3] It is possible to design a labeling scheme for testing node adjacency...

	wahr	**falsch**
...in trees, with label size $O(\log n)$.	☐	☐
...in general graphs, with label size $O(\log n)$.	☐	☐
...in general graphs, with label size $O(\log^2 n)$.	☐	☐

B) [3] Given a tree and a node x in the tree. Consider a labeling scheme such that based on labels of any nodes a and b, it can be decided if b lies on the unique path between a and x.

	wahr	**falsch**
Such labeling scheme can be designed with the label size of $O(\sqrt{n})$.	☐	☐
Such labeling scheme can be designed with the label size of $O(\log^2 n)$.	☐	☐
The smallest label size possible for such a scheme is $O(\log^2 n)$.	☐	☐

C) [3] Consider the distributed model of computation where per round each node can send one unbounded-size message to each of its neighbors.

	wahr	**falsch**
There is a distributed algorithm that colors any unrooted tree with 2 colors in $O(\log n)$ rounds.	☐	☐
There is a distributed algorithm that colors any unrooted tree with 6 colors in $O(\log^* n)$ rounds.	☐	☐
There is a distributed algorithm that colors any unrooted tree with 4 colors in $O(\log n)$ rounds.	☐	☐

D) [3] Consider the distributed model of computation where per round each node can send one unbounded-size message to each of its neighbors. Suppose we are given an algorithm that computes a maximal independent set in $f(n)$ rounds in any graph with at most n nodes. Then,

	wahr	**falsch**
There is an algorithm that computes a matching with size at least $\lceil k/2 \rceil$ in $f(n^2)$ rounds, in any n-node graph that has a matching of size k.	☐	☐
There is an algorithm that colors the nodes with 10 colors in any n-node graph with maximum degree 9 in $f(n^2)$ rounds.	☐	☐
There is a distributed algorithm that colors the edges with 21 colors in any n-node graph with maximum degree 10 in $f(30n)$ rounds.	☐	☐

Solutions

A) [3] It is possible to design a labeling scheme for testing node adjacency...

	wahr	falsch
...in trees, with label size $O(\log n)$. *Reason: Theorem 14.1 in the notes.*	✓	
...in general graphs, with label size $O(\log n)$. *Reason: Theorem 14.3 in the notes.*		✓
...in general graphs, with label size $O(\log^2 n)$. *Reason: Theorem 14.3 in the notes.*		✓

B) [3] Given a tree and a node x in the tree. Consider a labeling scheme such that based on labels of any nodes a and b, it can be decided if b lies on the unique path between a and x.

	wahr	falsch
Such labeling scheme can be designed with the label size of $O(\sqrt{n})$. *Reason: The problem is a paraphrase of testing ancestry for the tree rooted at x. Theorem 14.4 in the notes shows it can be solved with the label size of $O(\log n)$.*	✓	
Such labeling scheme can be designed with the label size of $O(\log^2 n)$. *Reason: See above.*	✓	
The smallest label size possible for such a scheme is $O(\log^2 n)$. *Reason: See above.*		✓

C) [3] Consider the distributed model of computation where per round each node can send one unbounded-size message to each of its neighbors.

	wahr	falsch
There is a distributed algorithm that colors any unrooted tree with 2 colors in $O(\log n)$ rounds. *Reason:*		✓
There is a distributed algorithm that colors any unrooted tree with 6 colors in $O(\log^* n)$ rounds. *Reason:*		✓
There is a distributed algorithm that colors any unrooted tree with 4 colors in $O(\log n)$ rounds. *Reason:*	✓	

D) [3] Consider the distributed model of computation where per round each node can send one unbounded-size message to each of its neighbors. Suppose we are given an algorithm that computes a maximal independent set in $f(n)$ rounds in any graph with at most n nodes. Then,

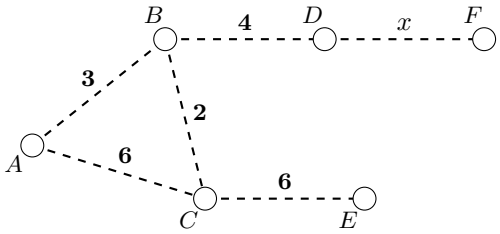
	wahr	**falsch**
There is an algorithm that computes a matching with size at least $\lceil k/2 \rceil$ in $f(n^2)$ rounds, in any n-node graph that has a matching of size k. *Reason:*	✓	
There is an algorithm that colors the nodes with 10 colors in any n-node graph with maximum degree 9 in $f(n^2)$ rounds. *Reason:*	✓	
There is a distributed algorithm that colors the edges with 21 colors in any n-node graph with maximum degree 10 in $f(30n)$ rounds. *Reason:*		✓

2 Worst-case Spanning Trees (18 points)

In this task, we study the Broadcast and Echo algorithms in the asynchronous model. We refer to the Broadcast and Echo processes together as the BC/E algorithm, and define the running time of BC/E as the sum of the running time of Broadcast and the running time of Echo.

We use a dashed line and a value k in the figures to denote a path of k consecutive edges connecting two specific nodes. Since we are in the asynchronous model, a message can take within $[0, k]$ time units to travel through such a path.

Consider the following graph, consisting of nodes connected by paths of different length. Node A of the figure is the root node.



There is no need justify your answers in this exercise; specifying the correct numerical answer (or selecting the correct paths from the list) is enough.

A) [3] What is the running time of the BC/E algorithm on this graph for $x = 1$?

[3] Consider the spanning tree formed by the algorithm in the worst case for $x = 1$. Which of the paths AB, AC, BC are contained entirely in this spanning tree?

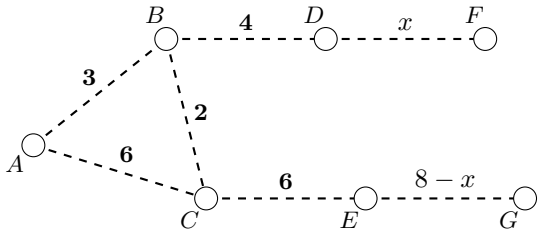
B) [3] What is the running time of the BC/E algorithm on this graph for $x = 4$?

[3] Consider the spanning tree formed by the algorithm in the worst case for $x = 4$. Which of the paths AB, AC, BC are contained entirely in this spanning tree?

C) Assume that after choosing some integer value $x \in [0, 8]$, we append a path of length $8 - x$ to node E, as shown in the figure below.

[3] Which value x produces the graph in which the running time of the BC/E algorithm is minimal?

[3] How many time units is this minimal running time?

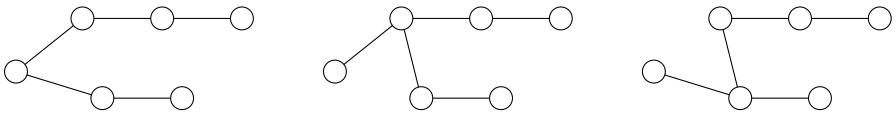


Solutions

Let us start with some general remarks. Each internal node in the path AC is at most at distance 5 from node C, while on the other hand, node E is 6 hops away from E, and only reachable through E. This means that in a worst-case run of the BC/E algorithm, node E is always reached later than any internal point of the path AC. We can argue similarly about the internal nodes of path AB and BC, which are all closer to B than node F.

This implies that in any worst-case run of Broadcast, the last node to receive the message is either E or F, and in any worst-case run of Echo, the furthest point from A in the tree is either E or F. Therefore, it suffices to concentrate on how the BC/E algorithm reaches these two nodes in the worst case.

Essentially, there are 3 different spanning trees that the broadcast algorithm can build to reach node E and F:



(1) (2) (3)

Furthermore, the behavior of tree (1) and (2) is very similar. For both of them, a worst case Broadcast takes 11 time units to reach E and $7 + x$ time units to reach F. On the other hand, Echo takes $\text{MAX}(12, 7 + x)$ units in tree (1), and $\text{MAX}(11, 7 + x)$ units in tree (2). This shows that the running time of BC/E on tree (2) is always at most as much as on tree (1), and thus, we can ignore tree (2). The worst case running time is always realized on tree (1) or (3).

A) Let us consider tree (1). In Broadcast, E is reached in 11 units, while F is reached in 8 units. In Echo, the message from E arrives in 12 units, and the message from F arrives in 8 units. Thus the runtime of BC/E is 11+12=23.

On the other hand, to from the tree (3) in Broadcast, node B must be reached from C after 3 time units, so C is also reached in 3 time units at most. Then E is reached in 9 units, and F is reached in 8 units. In Echo, the message from E arrives to A in 12 time units, while the message from F arrives in 13 units. The runtime of BC/E is 9+13=22.

This shows that the worst-case running time is 23, and the worst case tree is tree (1). This tree contains paths AB and AC.

B) In tree (1) in Broadcast, E is reached in 11 units and F is also reached in 11 units. In Echo, the message of E travels for 12 units, the message from F travels for 11 units. The running time of BC/E is 11+12=23.

In tree (3), node B once again must be reached from C in 3 time units, thus C is also reached in 3 time units. This implies that E is reached in 9 units, while F is reached in 11 units. In Echo, the message from E takes 12 time units to arrive, the message from F takes 16 units. This adds up to a BC/E runtime of 11+16=27 units.

Thus the running time is 27, the worst case tree is (3). This tree contains paths AC and BC.

C) Note that similarly to before, tree (2) still produces consistently smaller running times even after extending the graph with the new path connected to E.

Generally, Broadcast in tree (1) reaches G in $5+6+(8-x) = 19-x$ units, and F in $3+4+x = 7+x$ units. In Echo, G reaches A in $20-x$ units and F reaches A in $7+x$ units. With that, the running time is $\text{MAX}(19-x, 7+x)+\text{MAX}(20-x, 7+x) = 14+\text{MAX}(x, 12-x)+\text{MAX}(x, 13-x)$ units.

In tree (3), node G is reached in $3 + 6 + (8 - x) = 17 - x$ units, and F is reached in $3+4+x = 7+x$ units. In Echo, the message from G arrives in $20-x$ time units, the message from F arrives in $12 + x$ units. This adds up to $\text{MAX}(17 - x, 7 + x) + \text{MAX}(20 - x, 12 + x) = 19 + \text{MAX}(x, 10 - x) + \text{MAX}(x, 8 - x)$ units.

Thus our job is to find the minimal value for

$$\text{MAX}(14 + \text{MAX}(x, 12 - x) + \text{MAX}(x, 13 - x),\ 19 + \text{MAX}(x, 10 - x) + \text{MAX}(x, 8 - x))$$

for $x \in [0, 8]$. The first argument equals $39 - 2x$ on $x \in [0, 6]$ and $14 + 2x$ on $x \in [7, 8]$. The second argument is $37 - 2x$ on $x \in [0, 4]$ and $19 + 2x$ on $x \in [5, 8]$. This already shows that the minimal value of the maximum is within $[4, 7]$. The simplest solution is to calculate these values one by one, giving the following maximums: 31 for $x = 4$, 29 for $x = 5$, 31 for $x = 6$ and 33 for $x = 7$.

Therefore, the minimal worst case running time is obtained with a choice of $x = 5$, and the running time is 29 time units.

3 Sorting Networks (16 points)

A transposition sorting network is a special case of a sorting network in which each comparator connects only adjacent wires.

A) [8] Show that a transposition sorting network with n inputs has $\Omega(n^2)$ comparators.

B) [8] An odd/even sorting network is a special case of a transposition sorting network. Show that an odd/even sorting network is not necessarily a counting network.

Solutions

A) Suppose n is even w.l.o.g. If the transposition network is a sorting network, then it must sort the input:

$$(\frac{n}{2}+1, \frac{n}{2}+2, ..., n, 1, 2, ..., \frac{n}{2}).$$

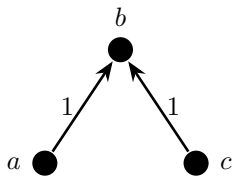
Each input has to be moved across exactly $\frac{n}{2}$ lines. In a transposition network, each comparator moves two inputs by exactly one line. Thus, each input must be involved in at least $\frac{n}{2}$ comparators. Since there are n inputs, and each comparator can only move two inputs, we need at least $\frac{n \cdot n}{4} \in \Omega(n^2)$ comparators.

B) Any counter example. Model counter-example: assume we have 3 wires with comparators $1-2$, $2-3$, $1-2$ (in this order) and we input values only to the 3rd wire. Then, the first value will exit at the 1st wire, but the second value will exit at the 3rd wire which is incorrect (should exit at the 2nd wire).

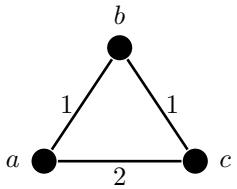
4 Ivy on Trees (20 points)

In the lecture, you saw the Ivy protocol for unweighted complete graphs. In this task we apply the Ivy protocol to weighted trees: you start with a weighted tree, and the weight of any pointer is the distance between the corresponding nodes in the tree. In other words, we will consider the performance of the Ivy algorithm from the lecture on a complete weighted graph where the distance between any two nodes is defined by the distance on an underlying tree.

A) [8] Show that Ivy on the tree given in Figure **??**, with the corresponding complete graph in Figure **??**, has a *competitive ratio* of at least $2 - \varepsilon$. Give therefore a recurring sequence of requests so that the ratio of *ivy cost* and the *optimal cost* is at least $2 - \varepsilon$. For a sequence of requests, the *ivy cost* is defined as the sum of distances traversed by the Ivy algorithm between the consecutive requests, where as the *optimal cost* is the sum of the shortest distances between the consequetive requests. You will obtain **8 points** if your ε is arbitrarily small.



(a) Underlying Tree



(b) Complete Network

Figure 1: A small example

B) [4] Show that Ivy is always 2-competitive for the tree in Figure **??**, i.e., the ratio of *ivy cost* and *optimal cost* is at most 2 for any finite sequence of requests.

C) [8] Show that Ivy is $\Omega(n)$ competitive on general weighted trees where n is the number of nodes in the tree.
Hint: You only need edge weights of 1 and ε in the tree.

Solutions

A) Consider the infinite request sequence that starts with c, a and repeats the pattern b, c, b, a infinitely as shown in Figure **??**. After the first two requests, the Ivy tree has an edge of weight 2. The next requesting nodes make sure that the edge of weight 2 is always kept in the Ivy tree. Such requests should be considered in pairs: the first request always has the cost 1 which is also optimal, while the second request produces a cost of 3, the optimal cost is however 1. Thus, the competitive ratio is at least $\frac{1+2+(3+1)k}{1+2+(1+1)k}$ for arbitrarily large $k \geq 1$ or $2 - \varepsilon$.

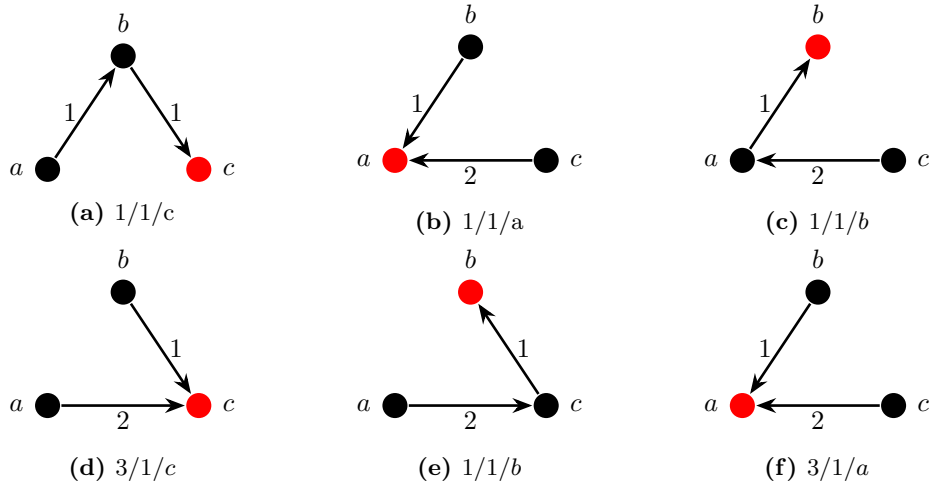


Figure 2: The captions show the ivy cost/optimal cost/new root

B) Note that the only request which produces costs larger than the optimal is the request which travels over the edge $\{a, c\}$ and over one other edge. Such a request always produces costs of 3. Since this request travels over a line, the resulting graph for the next request will be a star, with either the edges $a - b$ and $c - b$ or the edges $a - c$ and $b - c$. As all these edges are shortest paths, any request would produce optimal cost. Since every request that produces a cost of 3 but an optimal cost 1 is interleaved with requests that have cost x and also optimal cost x for $x \geq 1$, the competitive ratio is at most $\max_{x \geq 1} \frac{3+x}{1+x} = 2$.

C) Consider the network of Figure **??** and assume that the algorithm starts with the object in the node v_n and the first request comes from the node v_1.

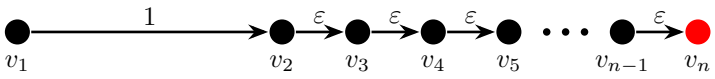
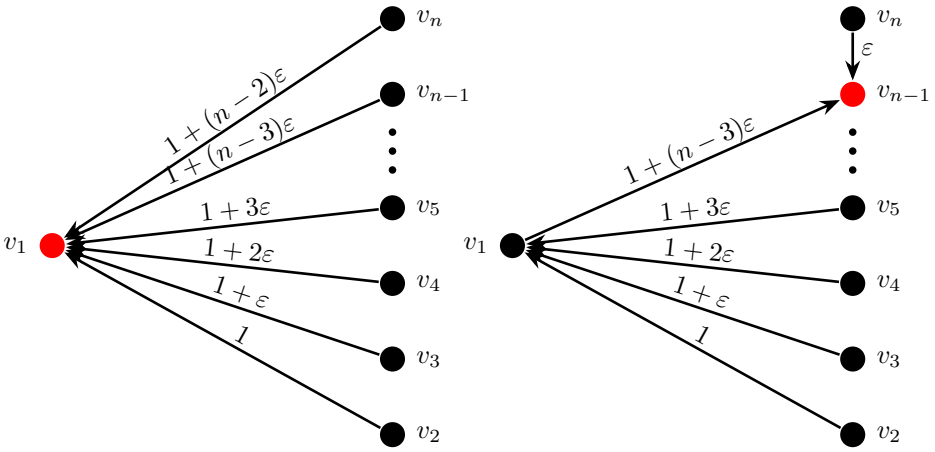


Figure 3: Counter example

This generates a star graph as shown in Figure **??**. Now consider the rest of the sequence, which is $v_n, v_{n-1}, \ldots, v_5, v_4, v_3, v_2, v_1$. Figure **??** shows the graph after the first two requests. The total cost of the Ivy algorithm is $1 + (n-2)\varepsilon$ for the request from v_n to v_1 and then $\Sigma_{i=n-1}^{2}(2(1 + (n - i + 2)\varepsilon) + \varepsilon)$. Thus, the total Ivy cost is $c_1 n + c_2 n^2 \varepsilon$ for constants c_1 and c_2. The optimal cost is $1 + \varepsilon(n-2)$ for the first request, a cost of $1 + (n-2)\varepsilon$ for the second request from v_1 to v_n, a cost of $(n-1)\varepsilon$ for the subsequent requests until v_2, and a cost of 1 for the final request from v_2 to v_1. Thus, the total optimal cost is $c_1' + c_2' n\varepsilon$ and the competitive ratio is $\Omega(n)$.



(a) Star graph after the first request **(b)** Graph after the requests v_n, v_{n-1}.

Figure 4: Star graph

5 LCL Problems on Directed Cycles (12 points)

Consider the LOCAL model of distributed computing where we have an n-node graph $G = (V, E)$, where nodes have unique identifiers with $O(\log n)$ bits, and per round each node can send one unbounded-size message to each of its neighbors. In this task, all input graphs will be directed cycles and contain at least 100 nodes.

A) [6] Determine the asymptotically tight deterministic complexity of the following LCL problem: Select a subset of the nodes of the directed cycle such that (1) there are no two adjacent nodes that are both selected, and (2) any sequence of 100 consecutive nodes on the cycle contains at least one selected node.

Argue about the correctness of the claimed complexity, i.e., argue why the claimed asymptotic complexity is both an upper and a lower bound.

B) [6] Determine the asymptotically tight deterministic complexity of the following LCL problem: Each node has to output a number from the palette $\{14, 15, 22, 33, 35\}$ such that (1) the outputs of any two consecutive nodes of the directed cycle are coprime, i.e., their greatest common divisor is 1.

Argue about the correctness of the claimed complexity, i.e., argue why the claimed asymptotic complexity is both an upper and a lower bound.

Solutions

A) The complexity is $\Theta(\log^* n)$.

For the upper bound, recall from the lecture that an MIS can be computed in time $O(\log^* n)$ on a directed cycle. Now, observing that selecting the nodes of an (arbitrary) MIS satisfies the constraints of the given problem yields the desired upper bound of $O(\log^* n)$.

For the lower bound, observe that if there is an algorithm $\mathcal{A}$ that solves the given problem in time $o(\log^* n)$, then we could also solve MIS (or, e.g., 3-coloring) in $o(\log^* n)$ rounds by first collecting all nodes that are selected by $\mathcal{A}$ (call this set of nodes S) and then letting each node $s \in S$ select some nodes from the part of the cycle between s and the next node from S (in direction of the cycle). Clearly, the nodes from S can select the additional nodes in a way that ensures that all selected nodes together form an MIS and they can do so in 100 rounds, i.e., in time $O(1)$, which implies that MIS is solvable in time $o(\log^* n)$. As we know from the lecture, computing an MIS requires $\Omega(\log^* n)$ rounds, which yields the desired lower bound.

B) The complexity is again $\Theta(\log^* n)$. Recall from the reading assignment that for LCL problems on directed cycles, there are only three possible time complexities, namely $\Theta(1)$, $\Theta(\log^* n)$ and $\Theta(n)$.

For the upper bound, observe that in the output neighborhood graph there is a flexible node since, e.g., node $(14, 15)$ has paths to itself of coprime length, e.g., $(14, 15), (15, 14), (14, 15)$ (length 2) and $(14, 15), (15, 22), (22, 35), (35, 33), (33, 14), (14, 15)$ (length 5). By Claim 1 of the reading assignment, this implies that the complexity of our problem is in $O(\log^* n)$.

For the lower bound, it is sufficient to show that there is no self-loop in the neighborhood graph, by Claim 1 of the reading assignment. If there is a self-loop it has to be a self-loop for a node of the form (x, x) for $x \in \{14, 15, 22, 33, 35\}$ since each node in the input graph clearly has to have the same output; however, there is no such node in the neighborhood graph because none of the possible values of x is coprime to itself. Hence, there is no self-loop in the neighborhood graph.

6 Orienting a Minimum Spanning Tree (20 points)

Consider the CONGEST model of distributed computing where we have an n-node graph $G = (V, E)$, where nodes have unique identifiers in $\{1, \ldots, n\}$, and per round each node can send one $O(\log n)$-bit message to each of its neighbors. In the class, we saw an algorithm that computes a minimum spanning tree (MST) T of the graph G in $O((D + \sqrt{n}) \log n)$ rounds, with high probability. Suppose we are given a root node r and we would like to orient all the edges of the tree T outwards from the root. That is, each edge $e = \{v, u\}$ of the MST T is oriented from the node v who is closer in T to the root r towards the node u who is further away.

A) [5] Devise an algorithm that in $O(h_T)$ extra rounds, after having the MST T, orients its edges outwards from the root. Here, h_T denotes the distance in T of any node from the root r.

B) [5] Devise an algorithm that in $O(h_T)$ extra rounds, after having the MST T, makes each node v learn the number of nodes u for which the shortest path in T from r to u includes v.

C) [10] Devise an algorithm that in $O((D + \sqrt{n}) \log n)$ extra rounds, after having the MST T, orients its edges outwards from the root.

Comment: Any randomized algorithm that succeeds with high probability (i.e., probability at least $1 - 1/n$) would be accepted.

Solutions

A) We perform a simple broadcast (or flooding) in the MST T, starting from the given root r. In the first round, root r sends messages to all of its neighbors and as a result, orients all of its edges outward from itself. From there on, whenever a node v receives a message from a neighbor u in round t, we do as follows: the edge (u, v) is oriented from u to v and node u is considered the parent of v. Moreover, in round $t + 1$, node v sends a message to all of its neighbors except u (if there is any such neighbor). All such neighbors that receive a message from v are out-neighbors of v and are considered as the children of v. The process terminates in h_T rounds.

B) The question is asking each node v to learn its number of descendants. We perform a simple echo with respect to the flooding in the previous question on the MST T (or a convergecast in the reverse direction of orientation). Any node w that has no child starts the echo with sending a 1 back to its parent. Notice that parent/child relation is according to the orientation outward from the root r (i.e., as we computed in the answer to the previous question). Any node v that has one or more children waits until it receives messages from all of its children. Then, it sums up all these numbers, add a +1 to the sum, and sends this summation to its own parent. The number passed to the parent is in fact the number of descendants of v (including itself), which is exactly the number that v should output.

C) Recall the $L = \Theta(\log n)$ iterations of merges in the Boruvka-style computation of MST described in the class. We work with the same structure of merges, but this time in the reverse direction. That is, we first process the merge of iteration L, then the merges of iteration $L-1$, and so on. Let us focus on the merge of iteration L. Recall that each merge has a star shape. That is, some components $C_1, C_2, \ldots, C_d$ which had a tail coin toss where being merged with a component C_0 which had a head coin toss, along edges (u_1, v_1), (u_2, v_2), $\ldots$, (u_d, v_d) where $u_i \in C_i$ and $v_i \in C_0$. One of these components contains the root r. For all the other components, our goal is to identify the node w that is closest to r (notice that this closest node will be one of $\{u_1, \ldots, u_d, v_1, \ldots, v_d\}$, which are the endpoints of the merge edges); we will consider this node w the root of its component as we go to the lower merge levels. Moreover, the merge edge connected to w, which connect w to the component that contains r, will be oriented inward to w. We next explain how we identify these roots, one for each component of the merge.

We first perform one step of communication, in $O(D + \sqrt{n})$ rounds, to make sure that each node of each component in $C_0, C_1, \ldots, C_d$ knows whether its component contains the given root r or not. This can be done via a simple broadcast inside the component for small components (those of size at most $\sqrt{n}$) and via communication on the global BFS for large components (those of size greater than $\sqrt{n}$). This step is similar to the communications per component that we did in computing MST. Now, each of the endpoint nodes of the merges — i.e., those in set $\{u_1, \ldots, u_d, v_1, \ldots, v_d\}$— that knows its component has the root r informs the other endpoint of the merge. For instance, u_1 might inform v_1 that the component C_1 has the root. In that case, we orient the edge (u_1, v_1) from u_1 to v_1. Moreover, node $v_1 \in C_0$ becomes the root of component C_0. We repeat another step of communication in $O(D+\sqrt{n})$ rounds to inform all nodes of the components that now have a root, similar to above. Then, we use one more round of communication along the merge edges where each node v_i in a component that now has a root sends a message to its pair $u_i \in C_i$ in a component that does not have a new root already. In that case, edge (v_i, u_i) gets oriented as v_i to u_i and node u_i becomes the root of C_i. This completes the description of the process for identifying the root for each component, all in $O(D + \sqrt{n})$. Then, we remove all the merge edges (u_1, v_1), (u_2, v_2), $\ldots$, (u_d, v_d) of this level from the MST, and we continue to one level down, in the hierarchy of the merges that we had in computing MST.

At the very bottom level of the merges, the above process is even easier to describe and it identifies the orientation. Here, we are dealing with star merges of simple nodes. For instance, we might have a component made of nodes $\{v_0, v_1, \ldots, v_d\}$, which was formed by merging them all around node v_0. By recursion, from the previous levels, one of these nodes $\{v_0, v_1, \ldots, v_d\}$ is identified as the root of this component. Let it be r'. Then, in one round of

communication, we can identify the child (or children) of r' and in one extra round, we can identifiy the children of that node. This determines the orientation inside this component.

7 Maximal 10-matching (22 points)

Consider the LOCAL model of distributed computing where we have an n-node graph $G = (V, E)$, where nodes have unique identifiers in $\{1, \ldots, n\}$, and per round each node can send one unbounded-size message to each of its neighbors.

In the maximal 10-matching problem, we want to select a subset $S \subseteq E$ of the edges with the following properties: (1) each node $v \in V$ has at most 10 edges in S incident on it, and (2) for each edge $e = \{u, v\} \notin S$, either u or v hast at least 10 edges in S incident on it.

A) [10] Devise a distributed algorithm that computes a maximal 10-matching of the graph in any n-node graph with maximum degree at most 100. Argue about the correctness of your algorithm, and analyze its round complexity.

Comment: Any randomized algorithm that succeeds with high probability (i.e., probability at least $1 - 1/n$) would be accepted. Any algorithm with a round complexity of $\Theta(\log n)$ or higher would receive zero points.

B) [12] Devise a distributed algorithm that computes a maximal 10-matching of the graph in any n-node graph. Argue about the correctness of your algorithm, and analyze its round complexity.

Comment: Any randomized algorithm that succeeds with high probability (i.e., probability at least $1 - 1/n$) would be accepted. Any algorithm with a round complexity of $\Theta(\log^3 n)$ or higher would receive zero points.

Solutions

A) Compute a 2Δ-1-Edge-Coloring in $O(log^* n + \Delta)$ rounds, using the algorithm of Theorem . Then go through the color classes one by one, and add all remaining edges of that color class to the matching. If a node has reached its limit of 10 incident edges in the matching, all the other incident edges are removed from the graph. This takes $O(log^* n + \Delta) = O(log^* n)$ rounds. Due to the definition of edge-coloring, no two incident edges are processed at the same time. This guarantees that at no point the condition is violated.

B) We compute a $(C, D) = (O(\log n), O(\log n))$ network decomposition in $O(log^2 n)$ rounds, w.h.p. Then we sequentially go through all C color classes, in each color class solving the remaining problem locally (trivially) and removing all the edges from the remaining graph if an incident node has reached its limit of having 10 added edges in the graph. This takes $O(log^2 n)$ rounds, w.h.p. Since we are solving the problem offline within connected components of one color class, it is correct there by construction, and since we are processing the colors sequentially, each time removing all edges that cannot be part of the matching anymore, there cannot be any conflicts between color classes.