ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

FS 2021

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

Exam
Principles of Distributed Computing

Thursday, August 19, 2021
15:00 – 18:00

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

The exam lasts 180 minutes, and there is a total of 180 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		24
2 - Center		20
3 - Matching		27
4 - Arrow and Ivy on Grids		16
5 - Maximal Paths		27
6 - Shared Graph		18
7 - Subtree Size		27
8 - Distance Labeling		21
Total		**180**

1 Multiple Choice (24 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] For this question, consider an n-node network in the synchronous model, where nodes have identifiers in $\{1, 2, \ldots, n^{10}\}$, and where per round each node can send one unbounded-size message to its neighbors.

	true	**false**
There is a one round deterministic algorithm that colors any graph with constant maximum degree using at most $O(\log n)$ colors.	☐	☐
There is a one round deterministic algorithm that colors any bipartite graph with constant maximum degree using at most $O(\log^* n)$ colors. We assume that the nodes are not aware of which part of the bi-partition they belong to.	☐	☐
There is a one round deterministic algorithm that colors any graph with constant maximum degree using at most $O(\log^* n)$ colors.	☐	☐

B) [3] Consider the Maximal Independent Set (MIS) problem and Luby's algorithm for solving it.

	true	**false**
In Luby's MIS algorithm, in each iteration, in expectation, at least $1/4$ of edges are removed.	☐	☐
In Luby's MIS algorithm, in each iteration, in expectation, at least $1/4$ of nodes are removed.	☐	☐
In Luby's MIS algorithm, in each iteration, in expectation, at least $1/4$ of nodes of nearly maximum degree are removed. Here, we call a node v a node of nearly maximum degree if node v has degree at least $\Delta/2$ where Δ denotes the maximum degree in the graph in that iteration.	☐	☐

C) [3] The following questions discuss sorting networks where all inputs are either 0 or 1.

	true	**false**
If we feed a monotonically increasing sequence into a half cleaner, then the output will be sorted.	☐	☐
If we feed a bitonic sequence into a half cleaner, then the output will be sorted.	☐	☐
If we feed a monotonically decreasing sequence into a half cleaner, then the output will be sorted.	☐	☐

D) [3] Alice has an n-bit string A and Bob has an n-bit string B. Assume that Alice and Bob are only allowed to communicate $\mathcal{O}(\log n)$ bits.

	true	**false**
Alice and Bob can decide if there is an index i such that the i^{th} bit of A and B are the same.	☐	☐
Alice and Bob can decide whether the number of 0-bits in A is larger than in B.	☐	☐
Alice and Bob can decide if there is a bit-string of length 4 that appears as a substring in both A and B.	☐	☐

E) [3] For this question, consider an n-node network $G = (V, E)$ in the synchronous model with diameter D and maximum degree Δ, where nodes have identifiers in $\{1, 2, \ldots, n^{10}\}$, and where per round each node can send one $O(\log^2 n)$-bit message to its neighbors.

	true	**false**
There is an $O(D)$ round deterministic algorithm that determines whether the minimum cut size of the graph has size equal to 1 or at least 2.	☐	☐
There is a one round randomized algorithm that computes an $O(\Delta \log n)$ coloring of the graph, with high probability.	☐	☐
Consider an input spanning subgraph $H = (V, E')$ of G where $E' \subseteq E$ and each node knows its incident edges in E'. We can determine whether H is connected or not in $O((D + \sqrt{n})poly(\log n))$ rounds.	☐	☐

F) [3] Consider the uniform leader election algorithm (Algorithm 13.10), and assume for convenience that n is a power of 2.

	true	**false**
A successful transmission can never happen while we have $k < \log n$ in the main loop.	☐	☐
With high probability, the algorithm never reaches $k = \log n + 1$.	☐	☐
If the algorithm reaches $k = 2 \cdot \log n$, then the probability of having a successful transmission with this k value is less than $\frac{1}{2}$.	☐	☐

G) [3] The following questions are about collision detection in wireless networks.

	true	**false**
Collision detection means that nodes in the network can transmit and receive simultaneously.	☐	☐
Collision detection provides no extra information if the network consists of $n = 2$ nodes.	☐	☐
Collision detection provides no extra information if the network consists of $n = 3$ nodes.	☐	☐

H) [3] A path graph is a graph on n nodes $v_1, v_2, ..., v_n$, consisting of the edges (v_i, v_{i+1}) for $i \in \{1, ..., n-1\}$.

	true	**false**
There is an $O(\log^2 n)$ labeling scheme for distance in path graphs.	☐	☐
There is an $O(\log n)$ labeling scheme for distance in path graphs.	☐	☐
There is an $O(\log \log n)$ labeling scheme for distance in path graphs.	☐	☐

Solutions

A) [3] For this question, consider an n-node network in the synchronous model, where nodes have identifiers in $\{1, 2, \ldots, n^{10}\}$, and where per round each node can send one unbounded-size message to its neighbors.

	true	**false**
There is a one round deterministic algorithm that colors any graph with constant maximum degree using at most $O(\log n)$ colors. *Reason: Use Linial's coloring algorithm which only requires the IDs of each neighbor.*	✓	
There is a one round deterministic algorithm that colors any bipartite graph with constant maximum degree using at most $O(\log^* n)$ colors. We assume that the nodes are not aware of which part of the bi-partition they belong to. *Reason: Suppose there exists such an algorithm and consider directed paths (which are bipartite). Then, Observation 1.9 from [DGA], there is a 3-ary $O(\log^* n)$-coloring, which implies that there is a 1-ary $2^{2^{O(\log^* n)}}$-coloring, which is a contradiction to Lemma 1.10 of [DGA].*		✓
There is a one round deterministic algorithm that colors any graph with constant maximum degree using at most $O(\log^* n)$ colors. *Reason: Implied by the above question.*		✓

B) [3] Consider the Maximal Independent Set (MIS) problem and Luby's algorithm for solving it.

	true	**false**
In Luby's MIS algorithm, in each iteration, in expectation, at least $1/4$ of edges are removed. *Reason: Shown in Theorem 1.51 of the [DGA] book.*	✓	
In Luby's MIS algorithm, in each iteration, in expectation, at least $1/4$ of nodes are removed. *Reason: Consider a complete bipartite graph $K_{a,b}$ with $a \ll b$. With probability $a/(b+1) = o(1)$ no node on the smaller side joins the MIS. When this is the case, all nodes on the larger side are preserved.*		✓
In Luby's MIS algorithm, in each iteration, in expectation, at least $1/4$ of nodes of nearly maximum degree are removed. Here, we call a node v a node of nearly maximum degree if node v has degree at least $\Delta/2$ where Δ denotes the maximum degree in the graph in that iteration. *Reason: Each neighbor of v joins the MIS with probability at least $1/\Delta$. Therefore, by summing the values over the $\Delta/2$ neighbors, we expect $1/2$ of v's neighbors to join the MIS. Therefore, with constant probability, some neighbor joins the MIS and removes v.*	✓	

C) [3] The following questions discuss sorting networks where all inputs are either 0 or 1.

	true	**false**
If we feed a monotonically increasing sequence into a half cleaner, then the output will be sorted. *Reason: A monotonically increasing sequence is already sorted.*	✓	
If we feed a bitonic sequence into a half cleaner, then the output will be sorted. *Reason: E.g. the input sequence* 0010 *is transformed into* 0010.		✓
If we feed a monotonically decreasing sequence into a half cleaner, then the output will be sorted. *Reason: E.g. the input sequence* 1110 *is transformed into* 1011.		✓

D) [3] Alice has an n-bit string A and Bob has an n-bit string B. Assume that Alice and Bob are only allowed to communicate $\mathcal{O}(\log n)$ bits.

	true	**false**
Alice and Bob can decide if there is an index i such that the i^{th} bit of A and B are the same. *Reason: If one of them inverts the string, this would allow them to decide if there is a bit that is different, i.e. if the strings are equal.*		✓
Alice and Bob can decide whether the number of 0-bits in A is larger than in B. *Reason: They can both communicate the number of* 0*'s in their bitstring in* $\mathcal{O}(\log n)$ *bits.*	✓	
Alice and Bob can decide if there is a bit-string of length 4 that appears as a substring in both A and B. *Reason: There are only* $2^4 = 16$ *possible substrings; they can check them all in their own string, and communicate the results in* 16 *bits.*	✓	

E) [3] For this question, consider an n-node network $G = (V, E)$ in the synchronous model with diameter D and maximum degree Δ, where nodes have identifiers in $\{1, 2, \ldots, n^{10}\}$, and where per round each node can send one $O(\log^2 n)$-bit message to its neighbors.

	true	**false**
There is an $O(D)$ round deterministic algorithm that determines whether the minimum cut size of the graph has size equal to 1 or at least 2. *Reason: Construct a BFS tree, then check if any tree edge is a cut edge. Checking whether e is a cut edge corresponds to checking if all edges in the subtree of e have both endpoints in the subtree, which can be done for all edges in* $O(D)$ *rounds.*	✓	
There is a one round randomized algorithm that computes an $O(\Delta \log n)$ coloring of the graph, with high probability. *Reason: Each node randomly chooses* $k := O(\log n)$ *potential colors from a palette of size* $2\Delta k$*. With high probability, there is at least one potential color not chosen by any neighbor.*	✓	
Consider an input spanning subgraph $H = (V, E')$ of G where $E' \subseteq E$ and each node knows its incident edges in E'. We can determine whether H is connected or not in $O((D + \sqrt{n})poly(\log n))$ rounds. *Reason: Mark edges with probability* $1/\sqrt{n}$*. Aggregate all maximal components of unmarked edges in the root and determine whether they are connected.*	✓	

F) [3] Consider the uniform leader election algorithm (Algorithm 13.10), and assume for convenience that n is a power of 2.

	true	**false**
A successful transmission can never happen while we have $k < \log n$ in the main loop. *Reason: A successful transmission can happen for any k, even $k = 1$, it only has a very small probability.*		✓
With high probability, the algorithm never reaches $k = \log n + 1$. *Reason: Indeed, when $k = \log n$, we get a successful transmission with high probability.*	✓	
If the algorithm reaches $k = 2 \cdot \log n$, then the probability of having a successful transmission with this k value is less than $\frac{1}{2}$. *Reason: The probability of a successful transmission with this k in each round is $n \cdot \frac{1}{n^2} \cdot \left(1 - \frac{1}{n^2}\right)^{n-1} < \frac{1}{n}$, so the total probability of a successful transmission is less than $\frac{2 \cdot \log n}{n} \ll \frac{1}{2}$.*	✓	

G) [3] The following questions are about collision detection in wireless networks.

	true	**false**
Collision detection means that nodes in the network can transmit and receive simultaneously. *Reason: Collision detection is only used when a node is receiving, and its purpose is to tell apart silence from collision.*		✓
Collision detection provides no extra information if the network consists of $n = 2$ nodes. *Reason: Indeed, when a node listens, then there is only 1 other node that can transmit, so a collision cannot happen.*	✓	
Collision detection provides no extra information if the network consists of $n = 3$ nodes. *Reason: When a node listens, then the other two nodes can either both transmit (collision) or both stay silent. Collision detection separates these two cases.*		✓

H) [3] A path graph is a graph on n nodes $v_1, v_2, ..., v_n$, consisting of the edges (v_i, v_{i+1}) for $i \in \{1, ..., n-1\}$.

	true	**false**
There is an $O(\log^2 n)$ labeling scheme for distance in path graphs. *Reason: A path graph is a tree.*	✓	
There is an $O(\log n)$ labeling scheme for distance in path graphs. *Reason: We can simply assign IDs $1, ..., n$ to the nodes, and consider the difference of the two IDs.*	✓	
There is an $O(\log \log n)$ labeling scheme for distance in path graphs. *Reason: This is not possible anymore, since we need to produce $n - 1$ different kinds of results.*		✓

2 Center (20 points)

In this exercise, we work with a tree T, where each edge e is assigned a positive length l_e, which does not affect the speed of the message delivery. We denote the distance between any two nodes $u, v \in T$ by $d(u, v)$, and we define it as the sum of the lengths of the edges on the path connecting u and v in T.

A center of the tree is defined as a node u that minimizes $\max_{v \in T} d(u, v)$. Note that a tree can have more than one center. You are going to help the nodes decide whether they are a center of the tree or not.

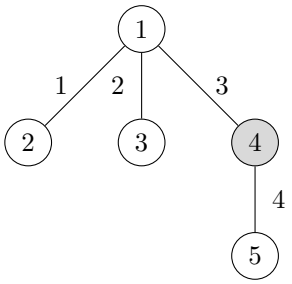

Figure 1: The tree in this example has a single center: node 4. The maximum distance from every node to the other nodes in the tree is computed as follows: $\max_{v \in T} d(1, v) = 7$, $\max_{v \in T} d(2, v) = 8$, $\max_{v \in T} d(3, v) = 9$, $\max_{v \in T} d(4, v) = 5$, and $\max_{v \in T} d(5, v) = 9$. Hence, the only center of this tree is node 4.

A) [7] We assume that the network is **synchronous** and each edge e has length $l_e = 1$. However, every node is only allowed to send **at most one message** to each of its neighbors. Design a distributed algorithm in which each node outputs "I am a center" if it is a center of the tree, and "I am not a center" otherwise.

B) [13] We now assume that the network is **asynchronous**. Again, design a distributed algorithm in which each node outputs "I am a center" if it is a center of the tree, and "I am not a center" otherwise. Your algorithm should have $\mathcal{O}(n)$ message complexity.

Solutions

A) In the first round, we remove the leaves of T: each leaf sends a "remove me" message to its neighbor u. Afterwards, in each round, we remove the leafs of the remaining subtree similarly, until we only have one node (and that is the single center of the tree) or one edge (and the two nodes incident to it are the centers of the tree).

B) In the first part of our solution, each node v computes $d_{\max}(v) = \max_{u \in T} d(u, v)$.

Each node v first computes a "partial" $d_{\max}(v)$, denoted by $d_{\text{partial}}(v, w)$, based on partial distances received from all of v's neighbors excepting one neighbor w. We define $d_{\text{partial}}(v, w)$ as the distance between v and its furthest node in the maximal subtree containing v, but not w. This computation starts from the leaves: every leaf v sends $d_{\text{partial}}(v, w) := 0$ to its neighbor w. When a non-leaf node v receives d_{partial} from all its neighbors excepting one neighbor w, it computes $d_{\text{partial}}(v, w) := \max\{d_{\text{partial}}(u, v) + l_{uv} | u \in N(v) \setminus \{w\}\}$, and sends $d_{\text{partial}}(v, w)$ to w.

Eventually, each node v receives $d_{\text{partial}}(w, v)$ from its remaining neighbor w, and computes $d_{\max}(v) := \max(d_{\text{partial}}(w, v) + l_{wv}, d_{\text{partial}}(v, w))$. Afterwards, node v sends $d_{\text{partial}}(v, u)$ to each neighbor $u \neq w$: if $d_{\max}(v)$ is not based on $d_{\text{partial}}(u, v)$, then $d_{\text{partial}}(v, u) := d_{\max}(v)$, and otherwise, $d_{\text{partial}}(v, u) := \max\{d_{\text{partial}}(u, v) + l_{uv} | u \in N(v) \setminus \{u\}\}$.

In the second part of our solution, each node decides if it is a center. This can be done by computing the minimum $d_{\max}$ in $O(n)$ message complexity.

3 Matching (27 points)

For this problem, we use the LOCAL model of distributed algorithms: the network is abstracted as an n-node undirected graph $G = (V, E)$, where nodes have unique $O(\log n)$-bit identifiers. Per round each node can send an unbounded-size message to each neighbor.

The objective is to compute a maximal matching, that is, a set of edges $M \subset E$ such that no two edges of M share an endpoint and moreover, for each edge $e \in E$, at least one of the endpoints of e has an incident edge in M.

A) [15] Devise a deterministic algorithm that computes a maximal matching in $O(\log^* n)$ rounds, in any network where the maximum degree is at most 100.

B) [12] Argue that the $O(\log^* n)$ round complexity is optimal asymptotically. That is, for any deterministic algorithm for maximal matching, there is a network with maximum degree at most 100 where the algorithm takes at least $\Omega(\log^* n)$ rounds to finish.

Solutions

A) Let Δ denote the maximum degree of G. Let G' be the line graph of G, i.e., the nodes of G' are the edges of G and there is an edge between two nodes in G' if they are distinct and the two corresponding edges share an endpoint. We start by coloring G' with $O(\Delta^2)$ colors using Linial's coloring algorithm. We simulate the execution of Linial's algorithm on G' in G by assigning to each edge of G (= node of G') the concatenation of the IDs of the two endpoints as the edge ID and the endpoint with the higher ID as the responsible node (which will take care of sending the messages the corresponding edge would send in G'). Since for any two neighboring nodes of G' the responsible nodes in G' have a distance of at most 2, we can perform the simulation by spending 2 rounds in the simulation for every round the execution would take on G'. Hence, we incur a multiplicative overhead of 2, and by Theorem 1.16 and the facts that the degree of each node in G' is at most 2Δ and the number of nodes in G' is at most Δ^2, it follows that we will obtain a proper coloring of the edges of G with $O(\Delta^2)$ colors in $O(\log^* n)$ rounds.

Now, we simply iterate through the color classes one by one, in each iteration processing the edges of the chosen color class. When processing an edge, we add it to the (initially empty) matching if no neighboring edge has been added so far. Both endpoints can compute this condition in 2 rounds, hence each iteration can be performed in 2 rounds and the runtime of all iterations together is in $O(\Delta^2)$. Since $\Delta \in O(1)$, we obtain an overall runtime of $O(\log^* n)$.

The condition specifying when to add an edge to the matching clearly ensures that no two neighboring edges are both added to the matching. Moreover, for any edge e that is not in the computed matching, there must be a neighboring edge that is in the matching as otherwise e would have been added to the matching according to the above condition.

B) For the sake of contradiction, assume that there is a deterministic algorithm $\mathcal{A}$ that computes a maximal matching in any network with maximum degree at most 100 in $o(\log^* n)$ rounds. Then, algorithm $\mathcal{A}$ also computes a maximal matching on paths, and therefore also on directed paths (as $\mathcal{A}$ can simply ignore the additional information given by the edge orientations).

Now, consider an algorithm $\mathcal{A}'$ on directed paths that first executes $\mathcal{A}$ (obtaining maximal matching M) and then outputs (in additional $O(1)$ rounds), for each (directed) edge e in M, color 1 for the tail of e and color 2 for the head of e, and, for each node without an incident edge in M, color 3. Note that the definition of a maximal matching and the consistent orientation of the input path together immediately imply that there are no two neighboring nodes which are 1) both the head of a matching edge, or 2) both the tail of a matching edge, or 3) both nodes without an incident edge in M. Hence, the output of $\mathcal{A}'$ is a proper 3-coloring.

Since the runtime of $\mathcal{A}'$ is asymptotically the same as the runtime of $\mathcal{A}$, i.e., $o(\log^* n)$ rounds, we obtain a contradiction to Theorem 1.7 in the lecture notes. It follows that the $O(\log^* n)$-round complexity of part A) is asymptotically optimal.

4 Arrow and Ivy on Grids (16 points)

We look at the performance of Arrow and Ivy when the underlying communication network is a grid. Figure 2 shows a specific state of the parent pointers on a 3×3 grid.

We are interested in the competitive ratio of an algorithm: What is the ratio on the message complexity of an algorithm compared to the message complexity of an optimal solution that knows the request sequence upfront.

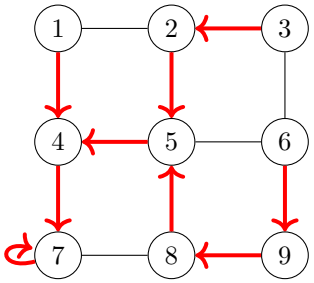

Figure 2: Parent pointers of the Arrow algorithm on a 3×3 grid.

A) [4] Consider the Arrow algorithm starting in the state depicted in Figure 2. Give a sequential request sequence (every request is fully resolved before a new request starts) with high competitive ratio.

From now on, we consider the Ivy algorithm. The message cost (and time cost) of a message is equal to the shortest distance in the underlying grid, so sending a message from node 3 to node 7 in the 3×3 grid costs 4. From now on, you may give an asymptotic bound, omitting constant factors.

B) [6] You are given an $m \times m$ grid with all nodes initially directly pointing to the node in the left lower corner as their parent. What is the competitive ratio of Ivy, if the sequential requests are issued by the boundary nodes in clockwise order, starting at the lower left node (inner nodes never issue any request)? So, in the 3×3 grid in Figure 2, the requests would come from nodes $7, 4, 1, 2, 3, 6, 9, 8, 7, 4, \ldots$.

C) [6] Again for Ivy, we are starting in a state where all nodes directly point to the same node in the center of an $m \times m$ grid. Now, all boundary nodes of the grid concurrently request the token (at time 0). Assume asynchronous communication. In the worst case, after how much time will the token reach the lower left node?

Solutions

A) For Arrow, as the directory tree stays constant, the worst case sequence is given by looping between the points which have the longest path in the directory relative to the shortest path in the underlying graph. In the given example this would mean looping between node 3 and node 6 (which yields asymptotically an overhead ratio of 5)

B) As none of the inner nodes issues any request, we can focus our analysis on the outer ring. The optimal algorithm simply requests the object from the neighbour before which holds it, requiring $\theta(m)$ messages for one loop. In Ivy, since all nodes initially point to the lower left corner, every initial request of a node has to make a detour over the lower left corner. The number of messages needed to serve a request therefore linearly grows until we reach the top right corner, which gives an overall message complexity of $\theta(m^2)$ for one loop. Next we note that Ivy turns the directory into a path along the outer ring in this first loop. The next loop starts by a request from the end of that path, redirecting all nodes on the outer ring to point towards the node just above the lower left corner node. While the cost of the next loop becomes cheaper, we note that every loop has a cost $\theta(m^2)$, which compared to the $\theta(m)$ cost of an optimal algorithm shows that Ivy is $\theta(m)$-competitive on grids.

C) The requests lead more and more to congestion as they approach the center of the grid. Further, the requests might arrive at the center node in such an order that they are served alternating from one side of the grid towards the other, each request costing $\Omega(m)$ token moves. As there are $\Omega(m)$ outer nodes, the cost is in $\Omega(m^2)$ in this case.

5 Maximal Paths (27 points)

For this problem, we use the LOCAL model of distributed algorithms, where the network is abstracted as an n-node undirected graph $G = (V, E)$, where nodes have unique $O(\log n)$-bit identifiers. Per round each node can send an unbounded-size message to each neighbor.

As input, we are given two disjoint sets of nodes $S, T \subseteq V$ as well as a positive integer k. In particular, each node v knows the value of k and also knows whether v is in S, in T, or in neither. Our task is to find a maximal collection of node-disjoint paths of length at most k where each path starts in a node of S and ends in a node of T. Here, by "node-disjoint" we mean that no two of the paths share a node. Moreover, by "maximal", we mean that if we remove all the vertices on the chosen paths, then there is no remaining path of length at most k between vertices in S and vertices in T.

A) [15] Devise a deterministic algorithm that solves this problem in $k \cdot \text{poly}(\log n)$ rounds.

B) [12] Devise a randomized algorithm that solves this problem in $O(k^2 \log n)$ rounds, with high probability.

Solutions

A) Let G^t be a graph with the same nodes as G but two nodes are connected if there is a path between them of length at most t. Note that one round of communication in G^t can be implemented in t rounds of communication in G (every node sends the message to its entire t-hop neighborhood).

We start by constructing the network decomposition with $C = O(\log n)$ and $D = O(\log n)$ of G^{2k} with a deterministic algorithm from the lecture. Since we need $2k$ rounds to perform one step of the algorithm, the decomposition is constructed in time $O(k \cdot \text{poly}\log(n))$ if we use the algorithm from Section 1.5 in the script. Note that if the diameter of a constructed cluster is $O(\log n)$ in G^{2k}, its diameter in G is $O(k \log n)$.

We now iterate over the $O(\log n)$ classes of the decomposition. In the i-th iteration, each cluster C of color i gathers information about the k-hop neighbourhood of every node in C to its root. This can be done in $O(k \log n)$ rounds, as this is the diameter of each cluster in G. After gathering the information, the root of C finds a maximal set of paths between S and T of length at most k such that those paths cover each node of C that is in $S \cup T$. This can be done by a simple sequential algorithm where we add a path whenever there is a node in $S \cap C$ that can still be connected with some other node in T, or vice versa. After finding the solution, the root of C sends it back to all nodes in C. The nodes covered by paths are then marked as covered and will not participate in future iterations. This finishes the description of the algorithm.

After building the decomposition, our algorithm needs additional $O(\log n \cdot k \log n)$ steps, since we need to iterate over all $O(\log n)$ color classes and spend $O(k \log n)$ rounds per each one. Hence, the total time of our algorithm is $O(k\text{poly}\log(n))$.

Note that whenever two clusters compute a solution during the same iteration, the sets of nodes they operate with do not intersect, since any two clusters are not adjacent in G^{2k}, hence they are at least $2k$ hops apart. Hence, the paths we found do not intersect. On the other hand, we found a maximal set, since for any node in $S \cup T$, we consider covering it by a path at some point during our algorithm.

B) Consider a graph H where every node corresponds to a path of length at most k connecting a node from S with a node from T. Moreover, two nodes in H are connected if the corresponding paths in G intersect in some node. One round of communication in H can be simulated in $O(k)$ communication rounds of G: each node $u \in S$ will be responsible for all paths that start in u and starts by learning its $2k$-hop neighborhood so that it knows what other paths are intersected by those that start in u. Then, simulating one round of communication in H can be done by each node sending the relevant information to everybody in its $2k$-hop neighborhood.

Now we will run Luby's algorithm in H to find its maximal independent set. This set exactly corresponds to the solution we need to find.

The round complexity of Luby's algorithm is $O(\log |V(H)|) = O(\log n^{k+1}) = O(k \log n)$, where we used that the number of paths of size at most $k+1$ (=length at most k) in a n-node graph is $O(n^{k+1})$. In fact, number of all $k+1$-sized subsets of nodes is equal to $\binom{n}{k+1} = O(n^{k+1})$. Moreover, recall that simulating one step of Luby's algorithm in H requires $O(k)$ rounds in G, hence the final complexity is $O(k^2 \log n)$.

6 Shared Graph (18 points)

Alice and Bob share the knowledge of a simple graph $G = (V, E)$ with $n \geq 3$ vertices. Alice and Bob both know all the vertices of G, and the unique vertex IDs from 0 to $n-1$. Alice knows a subset of the edges $A \subseteq E$ and Bob knows a subset of the edges $B \subseteq E$, such that $A \cup B = E$. A and B are disjoint, that is $A \cap B = \emptyset$. Alice and Bob want to communicate as little as possible. Help Alice and Bob to show the following:

A) [6] Check whether every connected component in G is a cycle with $\mathcal{O}(n)$ communication complexity.

B) [6] Let Alice and Bob use public randomness: They both have access to the same random bit string $z \in \{0,1\}^{2n}$. Discover whether not every connected component in G is a cycle with at least probability $\frac{1}{2}$ and $\mathcal{O}(1)$ communication complexity.

C) [6] Check whether G is a cycle (graph that consists of a single cycle) with $\mathcal{O}(n \log n)$ communication complexity.

Solutions

A) Alice/Bob locally check if no node $v \in V$ has $\deg(v) > 2$ in their subgraph, else they can stop. Alice/Bob each compute and exchange strings $s \in \{0,1\}^{2n}$, where $s_{2v}s_{2v+1}$ is the binary representation of $\deg(v)$ in their subgraphs. They now check locally whether every $v \in V$ has $\deg(v) = 2$, only then every connected component in G is a cycle graph.

B) Alice/Bob locally check if no node $v \in V$ has $\deg(v) > 2$ in their subgraph, else they can stop. Alice computes $a \in \{0,1\}^{2n}$ and Bob computes $b \in \{0,1\}^{2n}$ for their subgraphs, for $v \in V$:

$$a_{2v}a_{2v+1} = \begin{cases} 01 & \text{if } \deg(v) = 1 \\ 00 & \text{if } \deg(v) = 0 \\ 11 & \text{if } \deg(v) = 2 \end{cases} \quad \text{and} \quad b_{2v}b_{2v+1} = \begin{cases} 01 & \text{if } \deg(v) = 1 \\ 00 & \text{if } \deg(v) = 2 \\ 11 & \text{if } \deg(v) = 0 \end{cases}.$$

Now, they compute and exchange $\tilde{a} = \sum_{i \in 2n} a_i z_i \pmod 2$ and $\tilde{b} = \sum_{i \in 2n} b_i z_i \pmod 2$. With probability $\frac{1}{2}$, $\tilde{a} \neq \tilde{b}$ for $a \neq b$ (Lemma 9.26), and if $\tilde{a} \neq \tilde{b}$, they discover that not every connected component in G is a cycle; the opposite holds iff $a = b$.

C) Alice/Bob locally sum the number of edges they see. If either has more than n edges, they can stop. Otherwise, they exchange all their edges, by sending the IDs of both endpoints. Upon, exchanging all edges, Alice/Bob can check locally whether G is a cycle graph.

7 Subtree Size (27 points)

For this problem, we use the CONGEST model of distributed algorithms, where the network is abstracted as an n-node undirected graph $G = (V, E)$. We use D to denote the diameter of graph G. Moreover, nodes have unique $O(\log n)$-bit identifiers, and per round each node can send one $O(\log n)$-bit message to each neighbor.

Let T be an arbitrary spanning tree of graph G, which is provided as input in the format of each node knowing which of its edges is in T. We are also informed about a special node $r \in V$, which we will call the root of T (however, no orientation of T is provided and for instance a node does not know which of its neighbors is its parent in tree T). The task is to design an algorithm so that each node learns the number of its descendants. Recall that we call a node u a descendant of node v if the shortest path in T connecting node u to the root r includes node v.

A) [15] Devise a deterministic algorithm that solves this problem in $O(d_T)$ rounds where d_T denotes the diameter of tree T, i.e., the maximum distance in the tree T between any two nodes.

B) [12] Devise a randomized algorithm that solves this problem in $O((D + \sqrt{n}) \cdot \text{poly}(\log n))$ rounds, with high probability. Notice that here D is the diameter of the entire graph G.

Solutions

A) We start a broadcast procedure from the root. Initially, the root sends a message to its neighbors and all other nodes are idle. Each node v, upon hearing a message from its w, notes that v's parent is w and proceeds to broadcast a message to all of its neighbors except its parent w. After $O(d_T)$ rounds the process stops and each node learns about its parent node.

We now run the second phase of the algorithm. In the first round, each leaf v (nodes where the only neighbor is their parent) sends a message containing its number of descendants (one, since it's a leaf) to its parent. In the following rounds, if a node v has a child (i.e., a non-parent) that it did not hear from, v is idle in the current round. Otherwise, a node v learned about the number of descendants from each of its children, hence v can calculate its number of descendants (sum of all incoming messages plus one) and sent this number to v's parent. After $O(d_T)$ rounds, each node learns its number of descendants. Note that each message is a number between 1 and n, hence it can be stored within $O(\log n)$ bits, as required.

B) Each edge "*marks*" itself independently with probability $p := 1/\sqrt{n}$ (this can be achieved, for example, by flipping a random coin at the endpoint of an edge e with the higher ID and transmitting the coin outcome to the other endpoint of the edge). We say that a "*component*" is every maximal connected component comprised of unmarked edges.

We now argue that every component has diameter $L := O(\sqrt{n}\log n)$ with probability at least $1 - 1/n$. A simple calculation shows that every path with at least $L = O(\sqrt{n}\log n)$ edges will contain at least one marked edge with probability at least $1 - 1/n^3$: the probability that all $O(\sqrt{n}\log n)$ edges are unmarked is $(1-p)^L \leq \exp(p\sqrt{n} \cdot O(\log n)) \leq 1/n^3$ (where we used $1 - p \leq \exp(p)$). Furthermore, there are at most n^2 simple paths in a tree, hence using a union bound we conclude that with probability at least $1 - 1/n$, every path with at least L edges has at least one marked edge on it. Suppose now one of the components has diameter at least L. Then, there exists a simple path in T of length at least L without any marked edge. And we already proved that this is false with probability at least $1 - 1/n$.

Furthermore, the expected number of marked edges is $p \cdot (n-1) < \sqrt{n}$. Since the edges are marked independently, a standard Chernoff bound shows that the total number of marked edges is $O(\sqrt{n})$ with probability at least $1 - \exp(c \cdot \sqrt{n}) \geq 1 - 1/n$. This also implies that the number of components is $O(\sqrt{n})$.

Now, each component has diameter $O(\sqrt{n}\log n)$, hence we can elect a leader in each component (each node broadcasts the minimum ID it ever heard along unmarked edges for $O(\sqrt{n}\log n)$ rounds). We refer to the leader ID of a component as the component ID. Similarly, using the procedure described in part A), we compute the number of nodes in each component. Furthermore, we construct a BFS tree of the entire graph G from the root in $O(D)$ rounds (as shown in the lecture). Using pipelining, we send to the root of the BFS tree (1) the component IDs and number of nodes in all $O(\sqrt{n})$ components, and (2) for all $O(\sqrt{n})$ marked edges we send the component IDs of its two endpoints. This information transfer can be performed in $O(D + \sqrt{n})$ since the depth of the BFS tree is $O(D)$ and the amount of bits sent is $O(\sqrt{n})$.

This information allows the BFS root to locally compute a graph with all components contracted, allowing it to calculate the orientations of all marked edges, as well as the number of descendants for both endpoints of every marked edge. This $O(sqrtn)$ pieces of information is broadcast from the root to all nodes in G. Finally, using this information, each component now uses the algorithm similar to the one in part A) to compute the number of descendants in time proportional to its diameter, i.e., $O(\sqrt{n}\log n)$ rounds.

8 Distance Labeling (21 points)

Call graph G a *Manhattan graph* if it satisfies the following: Each node has two integer coordinates $v = (x, y)$; Two nodes are adjacent if they are equal in one coordinate and differ by 1 in the other coordinate; Whenever $(x, y_1) \in G$, $(x, y_3) \in G$ and $y_1 < y_2 < y_3$, then $(x, y_2) \in G$. Also, G is connected.

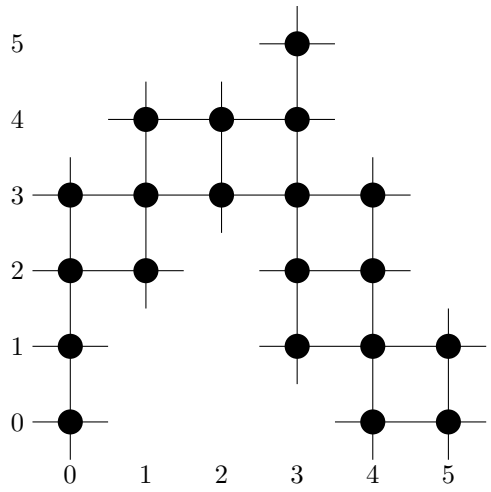

Figure 3: Example Manhattan graph where $n = 5$ and $m = 4$.

Let $n = \max_{(x_1,y_1),(x_2,y_2)\in G} |x_1 - x_2|$ and $m = \max_{(x,y_1),(x,y_2)\in G} |y_1 - y_2|$ for any x.
Let $L(u, v)$ denote the length of the shortest path between nodes u and v.

A) [4] Consider a Manhattan graph G and nodes u, $v = (x, y_1)$ such that for all $w = (x, y_2) \in G$, $L(u, v) \leq L(u, w)$. Write an equation for $L(u, w)$ in terms of $L(u, v)$, y_1, y_2.

B) [17] Describe a labeling scheme for $L(u, v)$ in a Manhattan graph. The decoder receives the labels and coordinates of u and v. Make the label size as small as possible in terms of n, m. Compute the label size.

Solutions

A) $L(u,(x,y_2)) = L(u,v) + |y_1 - y_2|$.

B) Clearly if $|x_1 - x_2| \leq 1$ then $L((x_1,y_1),(x_2,y_2)) = |x_1 - x_2| + |y_1 - y_2|$. Hence if $n \leq 1$, no labels are needed.

Consider $n > 1$ and construct the labeling recursively. Let x_h be the middle x coordinate of G: $x_h = \lfloor \frac{\min_{(x,y)\in G} x + \max_{(x,y)\in G} x}{2} \rfloor$. For each node u consider the node $u' = (x_h, y_{u'})$ such that $L(u,u')$ is minimal. Add $L(u,u')$ and the coordinates of u' to the label of u. By extension of A), with such labels, for nodes $u = (x_1,y_1)$ and $v = (x_2,y_2)$ where $x_1 \leq x_h \leq x_2$, we can compute $L(u,v) = L(u,u') + |y_{u'} - y_{v'}| + L(v,v')$.

Left to encode are $L((x_1,y_1),(x_2,y_2))$ where $x_1, x_2 < x_h$ or $x_h < x_1, x_2$. Proceed recursively for the subgraphs of all nodes (x,y) where $x < x_h$ and all nodes where $x_h < x$ respectively. Each recursive step adds to the label a distance to, and coordinates of a certain node, increasing the label by no more than $\log(n(m+1)) + \log n + \log m = O(\log n + \log m)$ bits. Each recursive step halves n, hence the recursion depth is $\log n$ and we obtain labels of size $O(\log^2 n + \log n \log m)$.