Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

FS 2022

Prof. Dr. Roger Wattenhofer,
Dr. Michal Dory,
Dr. Goran Zuzic

Exam
Principles of Distributed Computing

Thursday, August 25, 2022
8:30 – 11:30

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. 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 - Multiple Choice		6
2 - Maximal Independent Set		27
3 - High-Capacity CONGEST		30
4 - Improved Network Decomposition		27
5 - Maximum Edge Labeling on Paths		18
6 - Shared Objects		18
7 - Bipartite Graphs		24
8 - Distributed Sorting		18
9 - Graph Neural Networks		12
Total		**180**

1 Multiple Choice (6 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. Every node of the network has at least one neighbor.

At the beginning, every node of the input network is either *red* or *blue*. We want to mark every node with a letter `A` or `B` such that every node has either:

- a neighbor with the same starting color and the same output letter,
- or a neighbor with the other starting color and the other output letter.

For example, if a node that is initially blue is marked with a letter `A`, we require that at least one of its neighbors is a blue node marked with `A` or a red node marked with `B`.

	true	**false**
The problem is solvable in $o(n)$ rounds.	☐	☐
The problem is solvable in $O(\log^* n)$ rounds.	☐	☐
The problem is solvable in $o(\log^* n)$ rounds.	☐	☐

B) [3] Consider an n-node *path* 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**
Remember that a C-edge coloring is an assignment of colors $\{1, 2, \ldots, C\}$ to edges such that no two edges with a common endpoint share a color. On a path graph, does there exist an $O(\log n)$-round algorithm which finds a 2-edge coloring?	☐	☐
On a path graph, does there exist an $O(\log^* n)$-round algorithm which finds a 10-edge coloring?	☐	☐
Suppose that the IDs are not necessarily unique. On a path graph, is there an algorithm which checks if there are two equal IDs in $O(\log^* n)$ rounds?	☐	☐

This page intentionally left (almost) blank.

Solutions

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

At the beginning, every node of the input network is either *red* or *blue*. We want to mark every node with a letter `A` or `B` such that every node has either:

- a neighbor with the same starting color and the same output letter,
- or a neighbor with the other starting color and the other output letter.

For example, if a node that is initially blue is marked with a letter `A`, we require that at least one of its neighbors is a blue node marked with `A` or a red node marked with `B`.

	true	**false**
The problem is solvable in $o(n)$ rounds. *Reason: Implied by the next question.*	✓	
The problem is solvable in $O(\log^* n)$ rounds. *Reason: Implied by the next question.*	✓	
The problem is solvable in $o(\log^* n)$ rounds. *Reason: Indeed, it is solvable in* 0 *rounds by red nodes marking themselves with* `A` *and blue nodes with* `B`.	✓	

B) [3] Consider an n-node *path* network in the synchronous model, where nodes have identifiers in $\{1, 2, \dots, n^{10}\}$, and where per round each node can send one unbounded-size message to its neighbors.

	true	**false**
Remember that a C-edge coloring is an assignment of colors $\{1, 2, \dots, C\}$ to edges such that no two edges with a common endpoint share a color. On a path graph, does there exist an $O(\log n)$-round algorithm which finds a 2-edge coloring? *Reason: No, this is a global problem: there is a linear lower bound similar to the vertex 2-coloring.*		✓
On a path graph, does there exist an $O(\log^* n)$-round algorithm which finds a 10-edge coloring? *Reason: Yes; the problem corresponds to a vertex coloring of the line graph, which can be efficiently simulated in* LOCAL. *The line graph is also a path graph.*	✓	
Suppose that the IDs are not necessarily unique. On a path graph, is there an algorithm which checks if there are two equal IDs in $O(\log^* n)$ rounds? *Reason: No; checking if the IDs are the same is a global problem that requires $\Omega(D)$, which is $\Omega(n)$ rounds on a path graph.*		✓

2 Maximal Independent Set (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 identifiers in $\{1, 2, ..., n\}$. Per round each node can send an unbounded-size message to each neighbor.

The objective is to compute a maximal independent set (MIS), that is, a set of nodes S such that no two nodes of S are neighbors and each node $v \in V$ has at least one neighbor in S.

A) [15] Prove or disprove: for graphs with maximum degree $\Delta = O(\log \log n)$, there is a deterministic distributed algorithm that solves the MIS problem in $O((\log \log n)^2)$ rounds. You can assume that all the nodes in the graph know the values of Δ and n at the beginning of the algorithm.

B) [12] Prove or disprove: for graphs with maximum degree $\Delta = O(1)$, there is a deterministic distributed algorithm that solves the MIS problem in $O(1)$ rounds. You can assume that all the nodes in the graph know the values of Δ and n at the beginning of the algorithm.

Solutions

A) The statement is correct. To see this, we compute MIS using coloring, as follows. First, we compute a $(\Delta+1)$-coloring in $O(\Delta \log \Delta + \log^* n)$ rounds, which is bounded by $O((\log \log n)^2)$ since $\Delta = O(\log \log n)$. Then, we go over the $O(\log \log n)$ color classes one by one, such that when we are in color class i we add to the MIS all the nodes in color i that don't already have a neighbor in the MIS. By the end of the algorithm, each node that is not in MIS, has a neighbor in the MIS. Additionally, nodes in the MIS are not adjacent because nodes in the same color class are not adjacent, and we cannot add to the MIS two neighbors from different color classes by the structure of the algorithm.

B) The statement is incorrect, and contradicts the $\Omega(\log^* n)$ lower bound for 3-coloring paths we saw in the lectures. To see this, assume to the contrarty that there is an algorithm for computing MIS in $O(1)$ rounds when $\Delta = O(1)$, and we will get an $O(1)$ round algorithm for 3-coloring paths. Recall that in the lectures we saw that in order to compute a $(\Delta + 1)$-coloring in the graph G, we can compute an MIS in a related graph H, that is obtained from G by taking $(\Delta + 1)$ copies of G, and connecting all the copies of a node in a clique. Note that if G is a path, the maximum degree in H is constant, hence by our assumption we can compute an MIS in the graph H in $O(1)$ rounds. As we saw in the lectures, this translates to an $O(1)$-round algorithm for computing a 3-coloring of the path G, a contradiction.

3 High-Capacity CONGEST (30 points)

We define a variant of the CONGEST model called CONGEST(B) in which one is allowed to send, larger, $O(B \log n)$-bit messages. Formally, we are given an n-node graph $G = (V, E)$, where nodes have unique identifiers in $\{1, \ldots, n\}$, and in each round every pair of neighboring nodes exchanges a $O(B \log n)$-bit message, where $B \geq 1$ is the parameter of the model.

A) [15] Suppose we are given a designated "leader" node $r \in V$ and there are k $O(\log n)$-bit messages distributed among the nodes (some nodes can have multiple or zero messages). Design an algorithm to deliver all k messages to r within $O(D + k/B)$ rounds, where D is the diameter of G. Prove that the algorithm completes in the stated number of rounds.

B) [15] Design a minimum spanning tree (MST) algorithm in the CONGEST(B) model that utilizes the higher capacity B of the model. The algorithm should be a natural generalization of the standard $(D + \sqrt{n})\mathrm{poly}(\log n)$-round MST algorithm learned in class, but should be significantly faster whenever $B \gg 1$ and $D \ll \sqrt{n}$. Describe the algorithm. What is the round complexity of the algorithm (you can ignore polylogarithmic factors)?

Solutions

A) We say a "block" is a set of exactly B messages residing in the same node. Note that in each round each node can send one block of messages to its parent. Our algorithm consists of two phases: (1) in $O(D)$ rounds, each message is either delivered to the root r or joins a block, then (2) in $O(D + k/B)$ rounds, all blocks are delivered to the root. Therefore, the total runtime is $O(D + D + k/B) = O(D + k/B)$ rounds.

First phase. At any time, if a node v has at least B messages outside of a block, it repeatedly takes B of them and makes a new block out of them. The messages left outside of a block (at most $B - 1$ of them) are sent to v's parent. Note that this algorithm, by definition, suffers no congestion issues, hence it completes $O(D)$ rounds.

Second phase. Once all messages are in blocks, it is sufficient to deliver all the blocks to the root by sending a single block on an edge per round. This is equivalent to the standard CONGEST model in which we want to deliver at most k/B messages along a tree of depth D. As explained in the lecture, this can be done in $O(D + k/B)$ rounds.

B) We follow the general outline of the MST algorithm from class and textbook in which we are given (disjoint and connected) components of nodes $S_1, \ldots, S.$ and the goal is to compute, simultaneously, what is the minimum outgoing edge from each component. Following the textbook, if this can be implemented in X rounds, then the entire MST algorithm can be implemented in $\tilde{O}(X)$ rounds where $\tilde{O}$ hides polylogarithmic factors (hence, also constants).

To solve this, we will say a component is "small" if it has less than $Q > 0$ nodes (to be determined later) and "large" otherwise. Note that, computing the minimum outgoing edge requires $O(Q)$ rounds in small components (build a BFS tree internally in each component). Furthermore, if there are k large components, then computing the minimum outgoing edge takes $O(D + k/B)$ time by computing a global BFS tree, and using the BFS tree to deliver the minimum outgoing weight of each large component to the root. In more detail, any node v that receives two different messages about the minimum outgoing edge that came from the same component are aggregated into a single message that only contains the smaller weight, hence effectively there are at most k different messages (we can directly generalize the proof from part A to this setting). Moreover, there are at most n/Q many large components. Therefore, our algorithm requires $O(Q + D + (n/Q)/B)$ rounds. Setting $Q := \sqrt{n/B}$, we get a runtime of $\tilde{O}(D + \sqrt{n/B})$. Note that $D + \sqrt{n/B} \ll D + \sqrt{n}$ when $D \ll \sqrt{n}$ and $B \gg 1$.

4 Improved Network Decomposition (27 points)

For this question, consider an n-node network $G = (V, E)$ in the synchronous LOCAL model, where nodes have unique identifiers in $\{1, 2, \dots, n^{10}\}$, and where per round each node can send one unbounded-size message to its neighbors. Every node of the network has at least one neighbor.

In the lecture notes, you saw that there is a deterministic distributed $O(\log^6 n)$-round algorithm $\mathcal{A}$, which solves the following problem: Given a set of living vertices $S \subseteq V$, it finds a subset $S' \subseteq S$ of living vertices, where $|S'| \geq |S|/2$, such that the subgraph $G[S']$ induced by the set S' is partitioned into non-adjacent disjoint clusters, each of weak-diameter $O(\log^3 n)$ in the graph G.

In the exercises, you also saw that there is a deterministic sequential algorithm $\mathcal{B}$, which given a graph $H = (V', E')$ finds a subset of vertices $T \subseteq V'$, where $|T| \geq |V'|/2$ such that the subgraph $H[T]$ induced by the set T is partitioned into non-adjacent clusters, each of strong-diameter $O(\log n)$ in the graph H.

A) [12] Show how to combine these two algorithms to get a deterministic distributed $O(\log^6 n)$-round algorithm, which given a set of living vertices $S \subseteq V$, finds a subset $S' \subseteq S$ of living vertices, where $|S'| \geq |S|/4$, such that the subgraph $G[S']$ induced by the set S' is partitioned into non-adjacent clusters, each of strong-diameter $O(\log n)$ in the graph G.

B) [15] Show how the algorithm from part **A)** can be used to get an $O(\log^7 n)$-round deterministic distributed algorithm that computes an $(O(\log n), O(\log n))$ strong-diameter network decomposition of any n-node graph G.

Note: If you did not solve part ***A)****, you can assume that such an algorithm exists.*

Solutions

A) As a first step, we execute algorithm $\mathcal{A}$ with the same inputs as given by the exercise. Then, by broadcasting in each cluster separately for $O(\log^6 n)$ rounds, every node $v \in S'$ learns the topology of its cluster. Now, every node in S' can execute or simulate the algorithm $\mathcal{B}$ on its local cluster to achieve a clustering with diameter $O(\log n)$ and at least half of the nodes executing the algorithm will be clustered. Finally, we remove all nodes that were not clustered in the previous step from S' to find a set that satisfies the conditions. We have that $S' \geq |S|/4$, as we remove at most half of the nodes in each of the steps.

B) We can use the algorithm from **A)** once to get a clustering that clusters at least $1/4$ of all vertices. This will be the first graph G_1 of our network decomposition. Then, we recursively cluster the remaining $3/4$ vertices in the same way, obtaining G_2. We repeat this process until all vertices are clustered, which is done after $O(\log n)$ recursions. Thus, we have $O(\log n)$ color classes in our network decomposition and each cluster has diameter $O(\log n)$, by part **A)**.

5 Maximum Edge Labeling on Paths (18 points)

In this exercise we study undirected weighted paths. Every edge e has a positive integer weight w_e. We want to answer the following type of query: Given two vertices u, v of the path, compute the maximum edge weight on the sub-path between u and v. We want to design a labeling scheme that answers such queries using only the labels of u and v. Throughout this exercise you can assume that $u \neq v$.

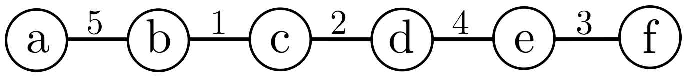


Figure 1: A path with 6 nodes. To get from vertex c to vertex e, the edges with weights 2 and 4 need to be traversed. Therefore, the maximum edge weight on the sub-path is 4. Similarly, from e to a the maximum edge weight is 5.

A) [7] Assume that the path has n vertices and each edge weight is in $\{0, 1, ..., k\}$, where $k \ll n$. Design a labeling scheme that, given the labels of two distinct vertices u and v, determines the maximum edge weight on the path between the vertices u and v. Full points are awarded for any labeling scheme using at most $O(k \log n)$ bits per label.

B) [4] Let s be a segment given by two integer points $s_a, s_b \in \{1, 2, ..., n\}$, with $s_a < s_b$, where the s_a is the start and s_b the end of the segment. Segment s has length $s_b - s_a$. We say that two segments c, c' *match* a segment s if and only if the union of c and c' is equal to s.

Show that for every segment s with $1 \le s_a < s_b \le n$ there exist two integers l and l', where $0 \le l, l' \le \log_2 n$, such that c has length 2^l, c' has length $2^{l'}$, and c, c' *match* s.

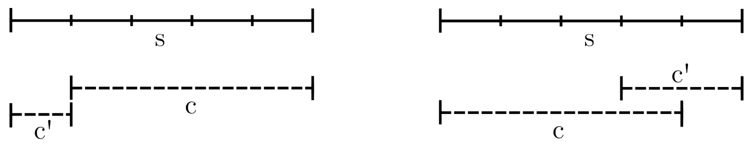


Figure 2: In this example, the segment s has length 5. It can be matched by the two dashed segments of length 2^0 and 2^2 shown on the left. Another possible way would be to use two segments of lengths 2^2 and 2^1, as shown on the right.

For the following part, you may assume part **B)** without proof.

C) [7] We want to consider larger edge weights. Each edge weight is now in $\{0, 1, ..., n\}$, where n is the number of nodes of the path. Design a labeling scheme that, given the labels of two distinct vertices u, v determines the maximum edge weight on the path between the vertices u and v. Full points awarded for any labeling scheme with at most $O(\log^2 n)$ bits per label.

Solutions

A) First, we enumerate the vertices on the path from left to right with 1 to n to assign them ID's. This can be done with $O(\log n)$ bits per label. Let u be the vertex with ID i. For each possible edge weight $w \in \{0, ..., k\}$ we store the largest ID j ($j \geq i$) that we can reach using edges of weight at most w in $R_{i,w}$. This can be done using $O(k \log n)$ bits per label as we store one ID for each possible weight.

Given the labels of arbitrary vertices u, v we can calculate the maximum weight on the sub-path as follows: Let u be the vertex with the smaller ID i and v have ID j. Our answer is the smallest w that satisfies $R_{i,w} \geq j$. This corresponds to the maximum weight on the sub-path as $w - 1$ does not cover the whole path, but w does.

B) Let s be an arbitrary segment with $1 \leq s_a < s_b \leq n$. Furthermore, let $l_s = s_b - s_a$. Choose k to be the largest integer so that $2^k \leq l_s$. Note that $l_s \leq n$, meaning that $2^k \leq n$ and, as a consequence, that $k \leq \log_2 n$. Let c_1 be a segment with $c_a = s_a, c_b = s_a + 2^k$ and c' be a segment with $c'_b = s_b, c'_a = s_b - 2^k$. Segments c and c' have length 2^k, c starts at the same place as s and c' ends at the same place as s. Moreover, they fulfill the length requirements. We now show that they also *match* s. It is sufficient to show that $c'_a \leq c_b$ as neither c_1 nor c_2 cover any parts outside s and are aligned with the endpoints of s. Recall that we choose k to fulfill $l_s = s_b - s_a < 2^{k+1}$, which we can rewrite as $s_b - 2^k < s_a + 2^k$ which is by definition the same as $c'_a < c_b$, giving the conclusion.

C) First, we enumerate the vertices on the path from left to right with 1 to n to assign them ID's. This can be done with $O(\log n)$ bits. For each node of ID i and each $k \in \{0, 1, ..., \log_2 n\}$ we calculate $L_{i,k}$ and $R_{i,k}$. We define $R_{i,k}$ as the maximum weight on the path between the vertex with ID i and $\min\{n, i + 2^k\}$. Similarly, $L_{i,k}$ is the maximum weight on the path between the vertex with ID i and $\max\{1, i - 2^k\}$. This can be done using $O(\log^2 n)$ bits as we store one ID for R and L of size $O(\log n)$ for each possible k (which there are $O(\log n)$).

To answer a query we can reuse part **B)**. Let u, v be arbitrary vertices with ID's i, j ($i < j$). To each segment we associate the maximum edge weight contained within the segment as the maximum of the segment. The answer to the query is then the maximum of the segment s with $s_a = i$ and $s_b = j$. Note that the maximum of s is the same as the maximum of two segments c, c' that *match* s. We can then chose an appropriate k so that the segments corresponding to $R_{i,k}$ and $L_{j,k}$ *match* s. Therefore, the answer to the query is $\max\{R_{i,k}, L_{j,k}\}$ with k the largest power of 2 that is less than or equal to $j - i$.

6 Shared Objects (18 points)

We consider a sequential execution of the Arrow protocol on the graph shown in Figure 3. Initially, the token is located at node 0 in the center of the graph. The initial Arrow tree is depicted by bold edges. The other edges of the graph are represented by thin dotted lines. For example, the edge $\{9, 10\}$ is an Arrow tree edge and the edge $\{2, 3\}$ is in the graph but is not part of the Arrow tree.

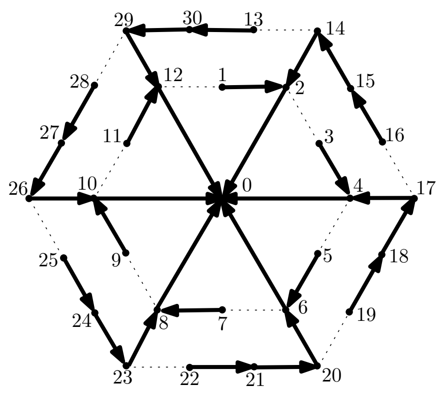


Figure 3: Graph used in parts **A)** and **B)**.

A) [5] Give a sequence of length at least 10 requests consisting of entirely *different* nodes such that if the sequence is executed sequentially by the Arrow protocol, then the cost is optimal.

B) [5] What is the worst-case competitive ratio the displayed tree can have when using the Arrow protocol sequentially? Give an example of a sequence that has this competitive ratio. Argue why it is the worst competitive ratio.

C) [8] Suppose now that the underlying graph is a complete 13-node graph and that we are executing the Ivy protocol. What does the graph look like if the sequence $1, 2, 3, 4, 5, 6, 7, 8$ is executed sequentially by the Ivy protocol, assuming that the starting tree is the one depicted in Figure 4. Show the graph after each request. You may use the skeleton graphs below to present your work. Clearly indicate which graph represents which step of the algorithm. If needed, there are additional skeleton graphs at the end of the exam.

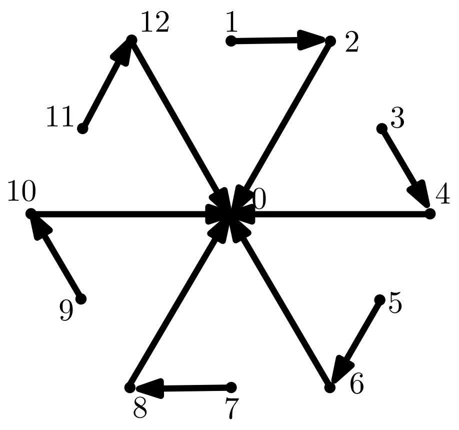
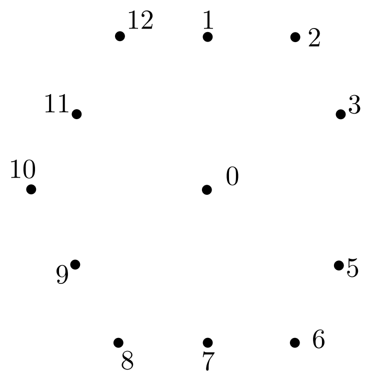
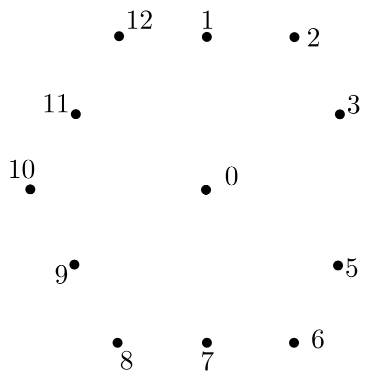
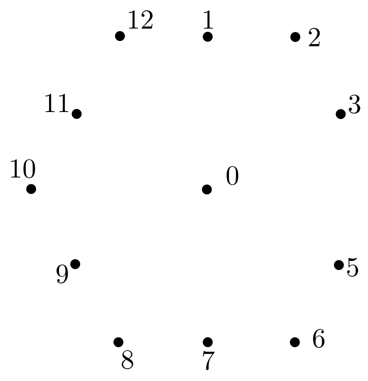
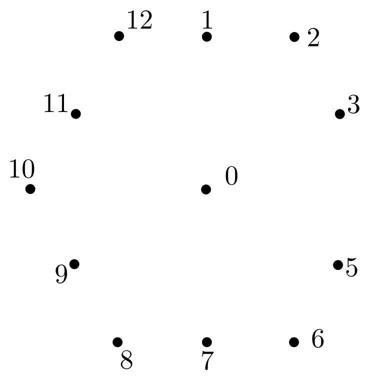


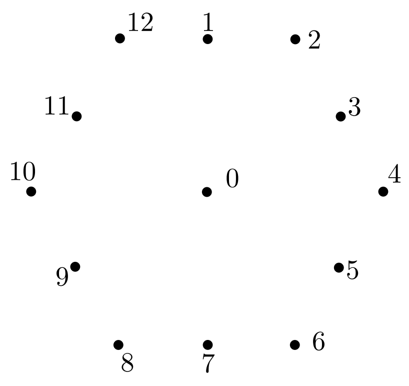
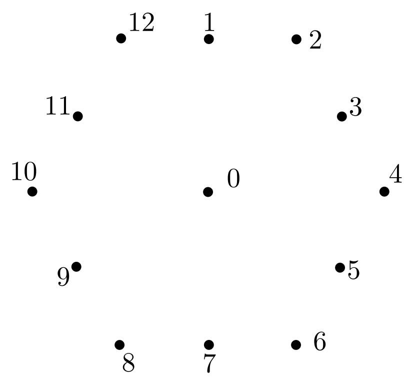
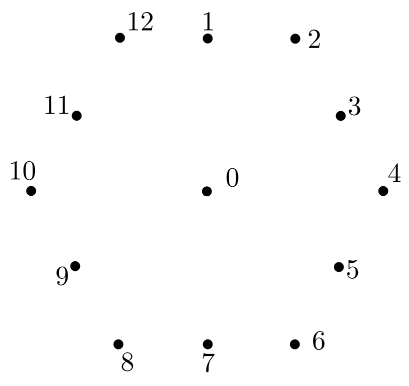
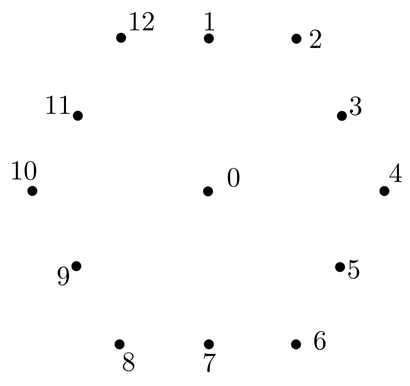
Figure 4: Graph used in part **C)**.










12
1
2
11
3
10
0
4
9
5
8
7
6

12
1
2
11
3
10
0
4
9
5
8
7
6

12
1
2
11
3
10
0
4
9
5
8
7
6

12
1
2
11
3
10
0
4
9
5
8
7
6

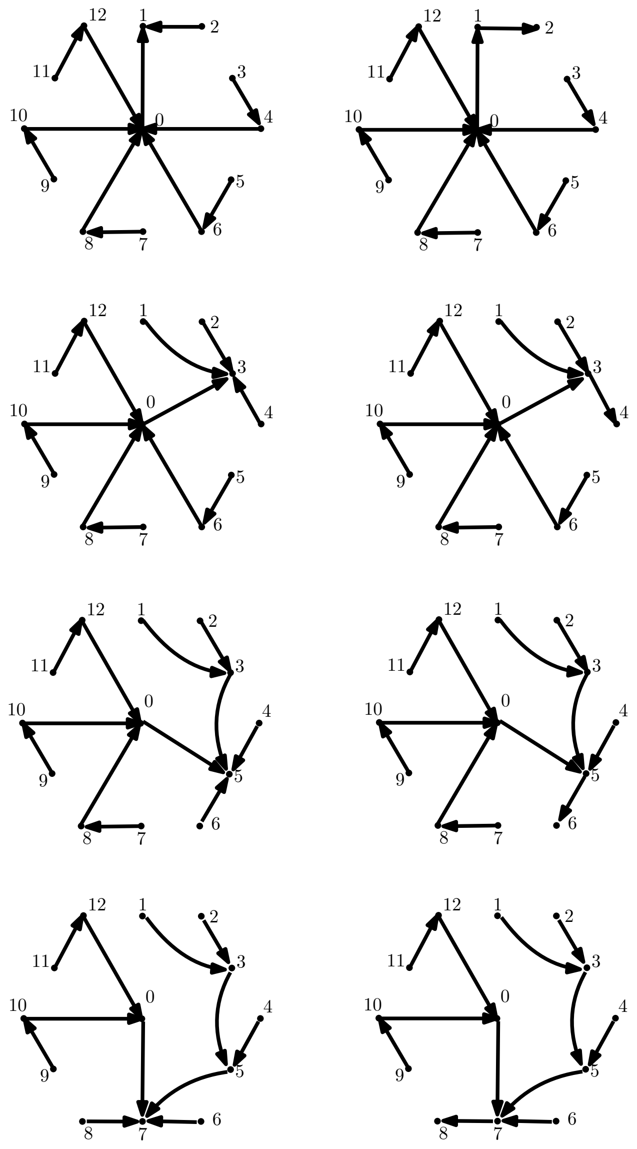
This page intentionally left (almost) blank.

Solutions

A) Any sequence of length at least 10 consisting of entirely different nodes, where any path in the graph connecting consecutive nodes in the sequence is at least as long as the path connecting the nodes in the Arrow tree will be accepted. For example, sequence 1,2,14,15,0,17,19,18,4,3,6 would satisfy the requirements.

B) The competitive ratio is $\frac{\text{cost of Arrow}}{\text{optimal cost}}$. The ratio is maximized if the sequence consists of nodes whose distance in the tree compared to the distance in the underlying graph is maximal. In the given graph, the maximal competitive ratio is 6. One sequence which achieves this competitive ratio is to alternate between nodes 20 and 19. For no two nodes which are neighbors in G is the path connecting them in the Arrow tree longer than 6. The diameter of the Arrow tree is 8, hence we do not need to consider nodes of distance two or greater, because for two nodes we would have cost of Arrow ≤ 8 and optimal cost ≥ 2. This would give a competitive ratio of less than 8.

C) The intermediate steps and final solution looks as follows. Reading from left to right top to bottom.



7 Bipartite Graphs (24 points)

Alice and Bob share the knowledge of a simple graph $G = (V, E)$ with $n \geq 3$ vertices; i.e. an undirected graph without self-loops and/or parallel edges. Alice and Bob both know all the vertices of G, and the unique vertex IDs in $\{0, 1, \ldots, n-1\}$. Alice knows a subset of the edges $A \subseteq E$ and Bob knows a subset of the edges $B \subseteq E$, such that $A \cup B = E$. Additionally, A and B are disjoint, that is $A \cap B = \emptyset$. Alice and Bob need to figure out whether the graph G is bipartite with as little communication as possible.

Definition (Bipartite Graph)**.** *A graph is bipartite iff a 2-coloring exits.*

A) [4] Show how Alice and Bob can decide whether graph G is bipartite with $O(n^2 \log n)$ communication complexity.

B) [6] Prove that if G is connected and bipartite, then exactly one valid 2-coloring (up to interchanging the two colors) exists for G and all its connected subgraphs.

For the following parts, you may assume part **B)** without proof.

C) [6] Assume that the graph that Alice sees, i.e., $G_A = (V, A)$, is connected. Show how Alice and Bob can decide whether the graph G is bipartite with $O(n)$ communication complexity.

D) [8] Let the number of connected components in graph G_A be denoted by k_A and let the number of connected components in graph $G_B = (V, B)$ be denoted by k_B. Show how Alice and Bob can decide whether the graph G is bipartite with $O(n(1 + \log k))$ communication complexity, where $k = \min\{k_A, k_B\}$.

Solutions

A) Alice/Bob exchange all their edges, by sending the IDs of both endpoints (each edge can be encoded in $O(\log n)$ bits). There are at most n^2 edges in graph G. Thus, the communication complexity is $O(n^2 \log n)$. Upon, exchanging all edges, Alice/Bob can check locally whether G is a bipartite graph.

B) We use $c_0 = 0$ and $c_1 = 1$ for the two colors. Since G is connected, let T be a spanning tree of G. WLOG, assume T is rooted at a node $r \in V$, which we call the root of T. WLOG, start with r colored with color 0. Now, proceed in a top-down manner: whenever a vertex is colored and has an uncolored children in the tree, color that children with the opposite color. After $n-1$ steps, the whole tree will be colored. Observe that the produced coloring is a valid 2-coloring of G, and that at no point we had a choice of what color to assign to a vertex, so the coloring is unique up to the original choice of coloring r with 0. If, instead, we colored r with 1, we would get the opposite coloring. The argument can easily be extended to all connected subgraphs of G.

C) Alice locally tries to compute a 2-coloring for G_A with colors $c_0 = 0$ and $c_1 = 1$. If Alice does not find a valid 2-coloring, she informs Bob that the graph is not bipartite. Otherwise, she sends a string $s \in \{0,1\}^n$ to Bob, where s_i is is the color of vertex i. Bob locally checks whether the 2-coloring sent by Alice is also valid for the graph he sees; i.e. G_B. By **B)**, the 2-coloring is unique, so the graph is bipartite iff the received 2-coloring is also valid for Bob. Thus, if the coloring is also valid for G_B, Bob informs Alice that the graph is bipartite.

D) Alice & Bob locally compute k_A & k_B respectively and inform each other. As $k_A \leq n$ & $k_B \leq n$, they can be encoded in $O(\log n)$ bits. If $k_A \leq k_B$, Alice (Bob otherwise) locally tries to compute a 2-coloring of their graph with colors $c_0 = 0$ & $c_1 = 1$.

Without loss of generality, assume $k_A \leq k_B$. If Alice finds no 2-coloring, she informs Bob and stops. Else, she numbers all connected components with $0, 1, \ldots, k-1$ and sends Bob a string $s \in \{0,1\}^{n(\lceil \log k \rceil + 1)}$. For $i = j \cdot (\lceil \log k \rceil + 1)$, s_i encodes node j's color and $s_{i+1} \ldots s_{i+\lceil \log k \rceil}$ its connected component. Bob then locally checks if the graph is 2-colorable. Within components, Bob cannot change the color of the vertices with respect to each other by **B)**. However, Bob can choose to interchange the two colors in any subset of connected components. If Bob finds a 2-coloring with these restrictions, the graph is bipartite & he informs Alice.

8 Distributed Sorting (18 points)

In this task we look at *transposition* sorting networks, a special case of sorting networks in which each comparator connects only adjacent wires. Prove or disprove each of the following statements:

A) [6] For every n there exists a correct transposition sorting network with depth $O(n)$.

B) [6] For every n there exists a correct transposition sorting network with depth $O(\sqrt{n})$.

C) [6] Every correct transposition sorting network with n inputs uses $\Omega(n^2)$ comparators.

Solutions

A) The statement is true. Take the even-odd sorting algorithm from the lecture as a sorting network of depth $2n$, where n is the number of inputs. The sorting network has comparators between i and $i+1$ for even (odd) values of i at each even (odd) depth. This sorting network is a transposition sorting network and has depth $O(n)$. We showed in the lecture that the even-odd sorting algorithm is correct and terminates within n rounds.

B) The statement is false. Consider the worst-case input $(1, 0, 0, 0, \ldots)$ for an arbitrary number of inputs n. This needs n sequential transpositions to move the 1 to the end. Therefore, for every n there is an input such that a transposition sorting network needs to have depth at least $O(n)$ to sort it.

C) The statement is true. Consider the input $(1, 1, \ldots, 1, 0, \ldots, 0, 0)$ with equal number of ones and zeroes for an arbitrary even number of inputs n (for odd n the argument is similar). Because we are only allowed to swap adjacent wires, moving all the ones by one spot takes $\frac{n}{2}$ swaps. We need to move all the ones by $\frac{n}{2}$ spots, thus requiring $\frac{n^2}{4} \in \Omega(n^2)$ swaps in total.

9 Graph Neural Networks (12 points)

In this exercise we investigate the abilities of GNNs to detect if a given connected graph is a clique (fully connected graph). For this task, you are given a *connected* graph $G = (V, E)$ such that $|V| > 2$ with unlabeled nodes (you can assume that all initial states are 0: $h_v^{(0)} = 0$ for all $v \in V$). After the execution of the GNN every node has to output whether it thinks that the given graph is a clique or not.
More precisely, if the graph is a clique, *all* nodes have to output 1. If it is not, at least *one* node has to output 0. The size of the clique or graph is *not* known beforehand.

A) [4] Prove that it is not possible to solve the clique problem with a GNN on connected graphs. To do this, first draw a pair of connected graphs as a counterexample and then argue why the clique problem cannot be solved by a GNN.

B) [8] We now adapt the input slightly: Instead of all nodes being initially labeled 0, we use two-dimensional states $h_v^{(i)} \in \mathbb{R}^2$ (for any round i) and mark exactly *one* (arbitrary) node v_m in the initial states. We do so by setting $h_{v_m}^{(0)} = (1, 1)$ and $h_w^{(0)} = (1, 0)$ for $w \in V \setminus \{v_m\}$. In this case we say that we have a one-marking. The graph G is still guaranteed to be connected.

The GNN has to finish in 2 rounds and the output $h_v^{(2)}$ has to be either $(0, 1)$ if the node thinks it is in a clique or $(0, 0)$ in case that it thinks it is not. Similarly to the previous part, if the graph is not a clique at least one node has to output $(0, 0)$, otherwise all nodes have to output $(0, 1)$. The GNN has to work with any initial one-marking. Complete the following functions for a GNN that solves the problem with a one-marking. The functions used in the two rounds can be different.

Round 1:

$a_v^{(1)} = \text{AGGREGATE}^{(1)}(\{\{h_u^{(0)} \mid u \in N(v)\}\}) =$

$h_v^{(1)} = \text{UPDATE}^{(1)}(h_v^{(0)}, a_v^{(1)}) =$

Round 2:

$a_v^{(2)} = \text{AGGREGATE}^{(2)}(\{\{h_u^{(1)} \mid u \in N(v)\}\}) =$

$h_v^{(2)} = \text{UPDATE}^{(2)}(h_v^{(1)}, a_v^{(2)}) =$

Solutions

A) We know that k-regular graphs cannot be distinguished by the WL algorithm and thus not by a GNN. Consequently, a k-regular graph cannot be distinguished from a clique of size $k+1$ as node embeddings will always be the same, making it impossible to solve the problem for a GNN.

For example, take a triangle and a 4-cycle. The labels for all nodes will always be the same but one graph is a clique while the other is not.

B) **Round 1:**

$$a_v^{(1)} = \text{AGGREGATE}^{(1)}(\{\{h_u^{(0)} \mid u \in N(v)\}\}) = \textstyle\sum_{u \in N(v)} h_u^{(0)} \text{ (element-wise sum)}$$

$$h_v^{(1)} = \text{UPDATE}^{(1)}(h_v^{(0)}, a_v^{(1)}) = h_v^{(0)} + a_v^{(1)}$$

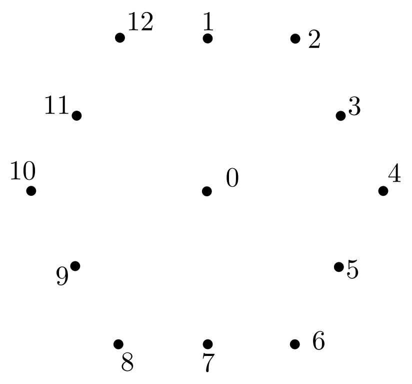
Round 2:

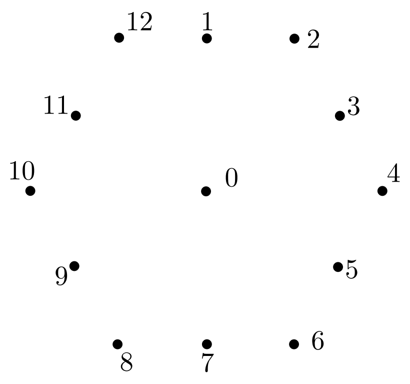
$$a_v^{(2)} = \begin{cases} h_u^{(1)}, & \text{if all } h_u^{(1)} \text{ are the same} \\ (0,0), & \text{else} \end{cases}$$

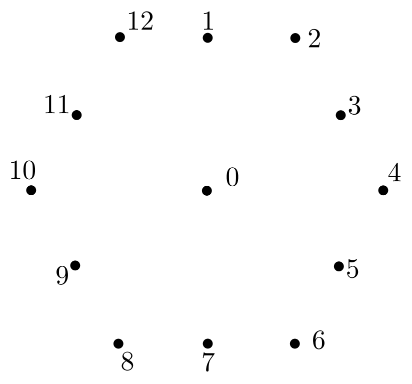
$$h_v^{(2)} = \begin{cases} (0,0), & \text{if } h_v^{(1)} \neq a_v^{(2)} \\ (0,1), & \text{else} \end{cases}$$

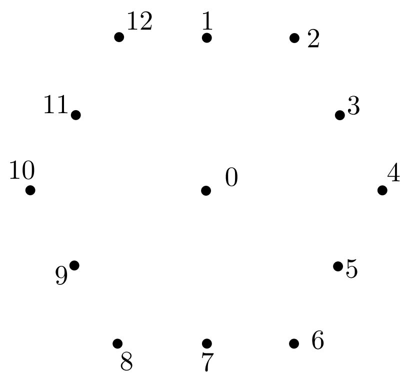
Replacements for Question 6, Part C)

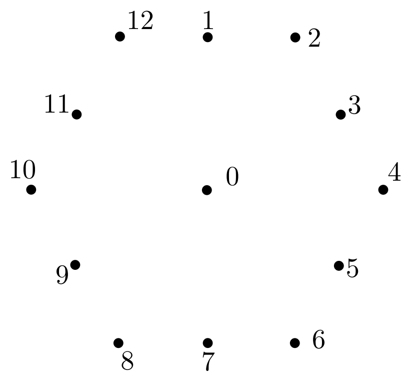
Here you can find replacement templates in case your original solution becomes too messy. If you use these, please *clearly* indicate which one represents which step of the algorithm.

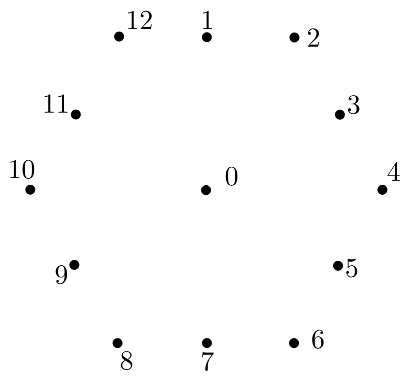


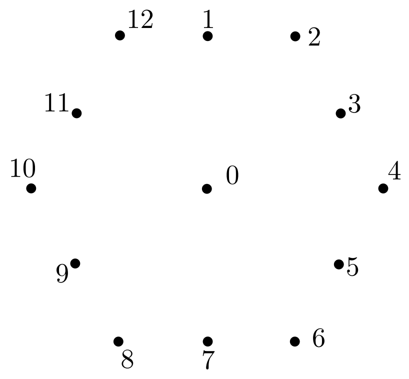


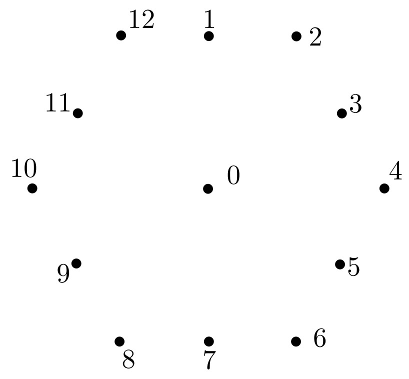


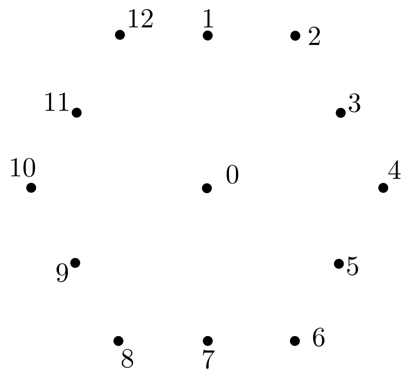


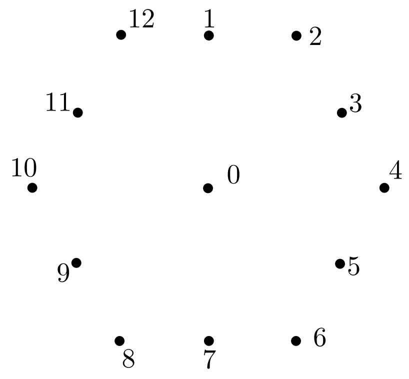


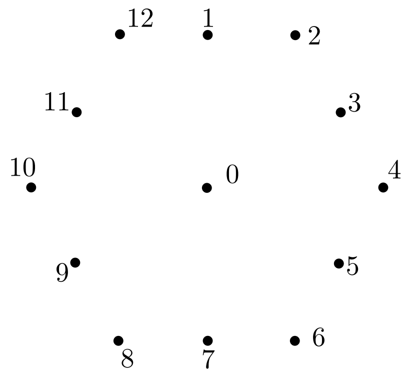


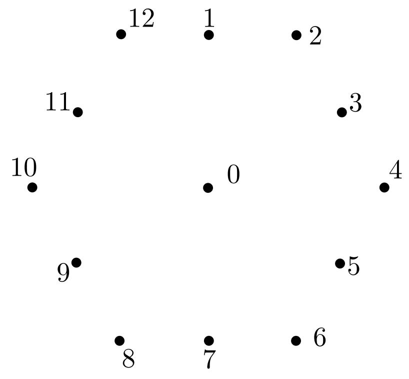

12
1
2
11
3
10
0
4
9
5
8
7
6


12
1
2
11
3
10
0
4
9
5
8
7
6


12
1
2
11
3
10
0
4
9
5
8
7
6


12
1
2
11
3
10
0
4
9
5
8
7
6


12
1
2
11
3
10
0
4
9
5
8
7
6


12
1
2
11
3
10
0
4
9
5
8
7
6