1 Task 5.11

To prove that the problem of deciding whether a graph is a tree is in AC_1 complexity class, we first need to understand what the AC_1 complexity class is. AC_1 is the set of problems that can be solved by a family of circuits with a polynomial number of gates and a constant, logarithmic, depth.

First, let's define the problem clearly: given a graph G, we need to check whether it is a tree or not. To be a tree, a graph must meet two conditions:

1. The graph is connected. This means that there is a path between every pair of vertices.
2. The graph is acyclic. This means that there are no cycles in the graph.

Next, we can present an algorithm to decide this problem:
1. Check if the graph is connected. This can be done using a breadth-first search (BFS) or a depth-first search (DFS) algorithm starting from any vertex. If the graph is connected, every vertex should be visited once and only once. If any vertex is not visited, then the graph is not connected and thus not a tree. Checking this condition can be done in AC_1, as BFS and DFS can be implemented with a queue or stack respectively, and each operation (enqueuing or dequeuing, pushing or popping) can be done with a constant number of operations on the binary representation of the index and the array representing the adjacency matrix of the graph, which are all operations in AC_0 and hence in AC_1. This check has a Boolean output.
2. Check if the graph has n-1 edges where n is the number of vertices. A tree with n vertices must have exactly n-1 edges. This condition checks that the graph is acyclic. If the graph has more or less than n-1 edges, it cannot be a tree. This can also be done in AC_1. We just need to count the number of edges in the graph, which is half of the sum of the degrees of the vertices. Each degree can be computed in AC_1, by iterating over an adjacency list and performing a bit-wise AND operation followed by a bit-wise OR operation which are also in AC_0, hence in AC_1.

Therefore, since we can decide whether a given graph is a tree or not using only operations in AC_1 and producing a Boolean output, we conclude that this problem is in AC_1.

2 Task 5.12

1) To show that every sparse language is in NC_1.

Let L be a sparse language and $p(n)$ be the polynomial such that $|L \cap \Sigma^n| \leq p(n)$. For $x \in \Sigma^n$, decide if x is in L or not by checking every string in $L \cap \Sigma^n$. The number of such strings is bounded by $p(n)$, hence each string can be checked in NC_1. Since the union of P-complete problems results in a P-complete problem, we can conclude that L is in NC_1.

2) To show that for each c, there exists a sparse language that does not have circuits with fan-in 2 of depth $c \log n$ for infinitely many n.

For this, we can refer to the diagonalization argument. Assume for the sake of contradiction that for a specific c, every sparse language has circuits of depth $c \log n$. We can construct a new language based on a counterexample as follows:

For each n, construct a string s_n of length n such that it differs from the n-th string of length n that has a description less than $c \log n$ in the lexicographic order. Now, consider the language L made up of all such strings s_n.

L is a sparse language (it has one string of each length) but it does not have circuits with fan-in 2 of depth $c \log n$ for infinitely many n. This contradicts our assumption, proving the claim.

3 Task 6.9

We shall show that the MAXIMUM-INTER-CLUSTER problem is NP-complete by reducing the problem of GRAPH-COLORING, which is known to be NP-complete, to the current problem.

We begin by briefly stating the decision problem for GRAPH-COLORING.

GRAPH-COLORING: Given an undirected graph $G = (V, E)$ and an integer k, can the vertices of the graph be colored using at most k colors such that no two adjacent vertices share the same color?

This problem is known to be NP-complete. Now we show that this problem can be reduced to MAXIMUM-INTER-CLUSTER.

Reduction: Given an instance $G = (V, E)$ of GRAPH-COLORING with k colors, we can construct an instance of MAXIMUM-INTER-CLUSTER as follows:

(i) The number of points is the same as the number of vertices in G.

(ii) Set $t = 1$ and construct the pairwise distance between points as $D_{i,j} = 1$ if $(i, j) \in E$ and $D_{i,j} = 0$ otherwise.

Hence if the graph G is k-colorable, then we can color the points according to the coloring of G such that for all $D_{i,j} >= t$, points i and j have different colors. Conversely, if we have a coloring of points, such that for all $D_{i,j} >= t$, points i and j have different colors then we can color the vertices of G according to the coloring of points. Thus, the coloring of the graph G exists if and only if such a coloring of points exists.

The remaining tasks are to show that this problem is in NP and the reduction operation can be performed in polynomial time.

This problem is in NP because given a particular coloring we can easily check, in polynomial time, whether the property for all $D_{i,j} >= t$, points i and j have different colors holds.

Since the reduction operation involves only the basic operation of constructing the $n \times n$ matrix, which is a polynomial operation, the reduction can be performed in polynomial time. Therefore, the MAXIMUM-INTER-CLUSTER problem is NP-complete.

4 Task 6.10

First, we construct a polynomial-time algorithm to solve the problem.

Step 1: Check if d is less than or equal to b and greater than 2. If not, proceed to step 2.
If d is greater than b or less than 2, then d cannot be a factor of N, so return

¨No¨.
If d is greater than 2 and less than or equal to b, then try dividing N by d. If N can be divided by d without leaving a remainder, then return ¨Yes¨. Otherwise, return ¨No¨.

Step 2: If N is a prime number, then return ¨No¨. A prime number cannot be divided by any number other than 1 and itself without leaving a remainder. Therefore, if N is a prime number, then it cannot have a factor d where $2 \leq d \leq b$.

Step 3: If N is a composite number, then return ¨Yes¨. A composite number must have a factor d where $2 \leq d \leq b$.

This algorithm can be executed in polynomial time because each step takes polynomial time.

Next, we need to show that SMALL-FACTOR is in coNP.
A problem is in coNP if its complement is in NP. The complement of SMALL-FACTOR is the problem of determining whether a number N is not divisible by any number d with $2 \leq d \leq b$. If we can find a witness d such that N cannot be divided by d without leaving a remainder, then we know that N is not divisible by any number with $2 \leq d \leq b$. This witness can be found in polynomial time, so the complement of SMALL-FACTOR is in NP. Therefore, SMALL-FACTOR is in coNP.

Lastly, we show that if SMALL-FACTOR is NP-complete, then NP=coNP. SMALL-FACTOR is in coNP, as we have shown. If it is also NP-complete, then it is in NP $\cap$ coNP. As a result, we have NP $\subset$ coNP and coNP $\subset$ NP, which means NP = coNP.

Hence, if SMALL-FACTOR is NP-complete, then NP = coNP.