Dominik Wawszczak
student id number: 440014
group number: 3

Computational Complexity
assignment 2

Task 1

To begin, we will show that verifying whether an input contains a tree is indeed in **L**. A graph is a tree if and only if it meets the following two conditions:

1. The number of edges is one less than the number of vertices.
2. The graph is connected.

Checking the first condition is straightforward. We can iterate over the edges while maintaining a binary counter and then verify if it equals $n - 1$. This counter will require no more than $\mathcal{O}(N)$ space, where N is the size of the input.

The second condition is more complex. We can iterate over all pairs $(s, t) \in \{1, \ldots, n\}^2$ using two binary counters. For each pair, we need to determine if a simple path exists from s to t. This problem is in **L**, as discussed in the lecture.

A *root* of a tree with radius at most 2 is a vertex $r \in V(T)$ such that every other vertex in T is within distance 2 of r.

We will use a slightly different definition of an *induced subgraph* than the one in the problem statement. A graph G_1 is an induced subgraph of G_2 if and only if there exists an injection $\mu : V(G_1) \to V(G_2)$ such that

$$(u, v) \in E(G_1) \iff (\mu(u), \mu(v)) \in E(G_2),$$

which we will refer to as the *map function*.

Our approach will follow these steps:

1. Find a root r of T_1.

   To do this, iterate over all vertices of T_1, treating each vertex as a candidate for r. To verify a candidate, check for each $v \in V(T_1) \setminus \{r\}$ whether either $(v, r) \in E(T_1)$ or there exists a vertex $u \in V(T_1) \setminus \{v, r\}$ with $(v, u) \in E(T_1)$ and $(u, r) \in E(T_1)$.

   This requires three binary counters (for r, v, and u). If no root is found, we can reject the input.

2. For each $s \in V(T_2)$, check if there exists a valid map function μ such that $\mu(r) = s$.

   The remaining part of the solution will focus on this check.

<u>Lemma 1</u> Let T_1 be a tree of radius 2 with root r, and let T_2 be any tree. Let $v_1, \ldots, v_k$ denote the children of r, ordered so that

$$\underset{i \in \{2, \ldots, k\}}{\forall} (\deg(v_{i-1}), v_{i-1}) \geqslant (\deg(v_i), v_i),$$

where

$$(a_1, b_1) \geqslant (a_2, b_2) \iff a_1 > a_2 \vee (a_1 = a_2 \wedge b_1 \geqslant b_2).$$

Let s be any vertex of T_2, and denote $u_1, \ldots, u_l$ as the children of s when T_2 is rooted at s, ordered similarly. Then a map function μ with the property that $\mu(r) = s$ exists if and only if

$$k \leqslant l \quad \wedge \quad \underset{i \in \{1, \ldots, k\}}{\forall} \deg(v_i) \leqslant \deg(u_i). \tag{1}$$

Proof of Lemma 1 If condition 1 holds, we can define $\mu(r) = s$ and set $\mu(v_i) = u_i$ for each $i \in \{1, \ldots, k\}$, ensuring there are enough vertices in T_2 to accommodate each grandson of r.

To derive a contradiction, assume there is a map function μ, but condition 1 does not hold. Since $k \leqslant l$ must be true, there exists an $i \in \{1, \ldots, k\}$ such that $\deg(v_i) < \deg(u_i)$. Take the smallest such i. There is a unique j such that $\mu(v_i) = u_j$, which must be less than i, or there would not be enough vertices to match the children of v_i; thus $i > 1$. For each $i' < i$, let j' be the index with $\mu(v_{i'}) = u_{j'}$; similarly, j' must be less than i because $\deg(i') \geqslant \deg(i)$. This creates a contradiction by the Pigeonhole principle, since μ was assumed to be injective, completing the proof.

Returning to the second step, once we have the root r, finding the number k of its children requires only one binary counter. For a given $s \in V(T_2)$, we can similarly find the number of its children l. By Lemma 1, if $k > l$, we can discard s as a candidate. Otherwise, we need to verify for each $i \in \{1, \ldots, k\}$ whether $\deg(v_i) \leqslant \deg(u_i)$. To calculate the degree of a vertex, we can iterate over edges while using a single binary counter. However, finding v_i and u_i in logarithmic space is more challenging.

We can determine $(\deg(v_1), v_1)$ by iterating over the sons of r, comparing each $(\deg(v_{\text{cur}}), v_{\text{cur}})$ with the best pair found so far, and updating as needed. We can calculate $(\deg(u_1), u_1)$ in a similar way. After this, we compare $\deg(v_1) \leqslant \deg(u_1)$ and reject if the condition does not hold; otherwise, we move on to the next pair. Once $(\deg(v_i), v_i)$ is calculated, obtaining $(\deg(v_{i+1}), v_{i+1})$ is straightforward – simply ignore any $(\deg(v_{\text{cur}}), v_{\text{cur}})$ strictly greater than $(\deg(v_i), v_i)$. The same process applies to $(\deg(u_{i+1}), u_{i+1})$.

The entire algorithm requires only a constant number of binary counters, concluding the solution.