Eidgenössische
Technische Hochschule
Zürich

Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Federal Institute of Technology at Zurich

Departement of Computer Science
Markus Püschel, David Steurer
François Hublet, Goran Zuzic, Tommaso d'Orsi, Jingqiu Ding

21 November 2022

Algorithms & Data Structures — Exercise sheet 9 — HS 22

The solutions for this sheet are submitted at the beginning of the exercise class on 28 November 2022.

Exercises that are marked by * are "challenge exercises". They do not count towards bonus points.

You can use results from previous parts without solving those parts.

Exercise 9.1 *Party & Beer & Party & Beer.*

For your birthday, you organize a party and invite some friends over at your place. Some of your friends bring their partners, and it turns out that in the end everybody (including yourself) knows exactly 7 other people at the party (note that the relation of knowing someone is commutative, i.e. if you know someone then this person also knows you and vice versa). Show that there must be an even number of people at your party.

Exercise 9.2 *Transitive graphs* **(1 point)**.

We say that a graph $G = (V, E)$ is

- **transitive** when, for any two edges $\{u, v\}$ and $\{v, w\}$ in E, the edge $\{u, w\}$ is also in E;
- **complete** when its set of edges is $\{\{u, v\} \mid u, v \in V, u \neq v\}$;
- the **disjoint sum** of $G_1 = (V_1, E_1), \ldots, G_k = (V_k, E_k)$ iff $V = V_1 \cup \cdots \cup V_k$, $E = E_1 \cup \cdots \cup E_k$, and the $(V_i)_{1 \leq i \leq k}$ are pairwise disjoint.

Show that a graph is transitive if, and only if, it is a disjoint sum of complete graphs.

Exercise 9.3 *Star search, reloaded* **(1 point)**.

A *star* in an undirected graph $G = (V, E)$ is a vertex that is adjacent to all other vertices. More formally, $v \in V$ is a star if and only if $\{\{v, w\} \mid w \in V \setminus \{v\}\} \subseteq E$.

In this exercise, we want to find a star in a graph G by walking through it. Initially, we are located at some vertex $v_0 \in V$. Each vertex has an associated flag (a Boolean) that is initially set to `False`. We have access to the following constant-time operations:

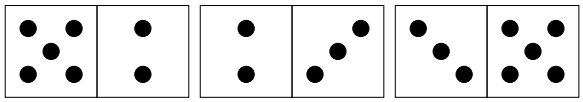
- `countNeighbors()` returns the number of neighbors of the current vertex
- `moveTo(`i`)` moves us to the ith neighbor of the current vertex, where $i \in \{1..$`countNeighbors()`$\}$
- `setFlag()` sets the flag of the current vertex to `True`
- `isSet()` returns the value of the flag of the current vertex
- `undo()` undoes the latest action performed(the movement or the setting of last flag)

Assume that G has exactly one star and $|G| = n$. Give the pseudocode of an algorithm that finds the star, i.e., your algorithm should always terminate in a configuration where the current vertex is a star in G. To obtain full points, your algorithm must have complexity $O(|V|+|E|)$, and must not introduce any additional datastructures (no sets, no lists etc.). Show that your algorithm is correct and prove its complexity. The behavior of your algorithm on graphs that do not contain a star can be disregarded.

Exercise 9.4 *Domino.*

(a) A domino set consists of all possible $\binom{6}{2} + 6 = 21$ different tiles of the form $[x|y]$, where x and y are numbers from $\{1, 2, 3, 4, 5, 6\}$. The tiles are symmetric, so $[x|y]$ and $[y|x]$ is the same tile and appears only once.

Show that it is impossible to form a line of all 21 tiles such that the adjacent numbers of any consecutive tiles coincide.



(b) What happens if we replace 6 by an arbitrary $n \geq 2$? For which n is it possible to line up all $\binom{n}{2} + n$ different tiles along a line?

Exercise 9.5 *Introduction to Trees* **(1 point)**.

We start with a few definitions:

Definition 1. Let $G = (V, E)$ be a graph.

- A sequence of vertices $(v_0, v_1, \ldots, v_k)$ (with $v_i \in V$ for all i) is a **simple path** iff all the vertices are distinct (i.e., $v_i \neq v_j$ for $0 \leq i < j \leq k$) and $\{v_i, v_{i+1}\}$ is an edge for each $0 \leq i \leq k-1$. We say that v_0 and v_k are the **endpoints** of the path.
- A sequence of vertices $(v_0, v_1, \ldots, v_k)$ (with $v_i \in V$ for all i) is a **simple cycle** iff (1) $v_0 = v_k$, (2) all other vertices are distinct (i.e., $v_i \neq v_j$ for $0 \leq i < j < k$), and (3) $\{v_i, v_{i+1}\}$ is an edge for each $0 \leq i \leq k-1$.
- A graph G is **connected** iff for every two vertices $u, v \in V$ there exists a simple path with endpoints u and v.
- A graph G is a **tree** iff it is connected and has no simple cycles.

In this exercise the goal is to prove a few basic properties of trees.

(a) A **leaf** is a vertex with degree 1. Prove that in every tree G there exists a leaf.

Hint: *Consider the longest simple path in G. Prove that its endpoint is a leaf.*

(b) Prove that every tree with n vertices has exactly $n-1$ edges.

Hint: *Prove by using induction on n. In the inductive step, use part (a) to find a leaf. Disconnect the leaf from the tree and argue the remaining subgraph is also a tree. Apply the inductive hypothesis and conclude.*

(c) Prove that a graph with n vertices is a tree iff it has $n-1$ edges and is connected.

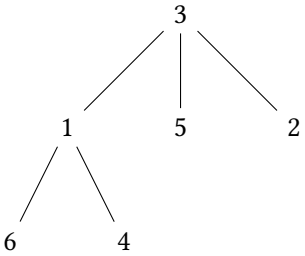
Hint: *One direction is immediate by part (c). For the other direction (every connected graph with $n-1$ edges is a tree), use induction on n. First, prove there always exists a leaf by considering the average degree. Then, disconnect the leaf from the graph and argue the remaining graph is still connected and has exactly one less edge. Apply the inductive hypothesis and conclude.*

(d) Write the pseudocode of an algorithm that is given a graph G as input and checks whether G is a tree.

As input, you can assume that the algorithm has access to the number of vertices n, the number of edges m, and to the edges $\{a_1, b_1\}, \{a_2, b_2\}, \ldots, \{a_m, b_m\}$ (i.e., the algorithm has access to $2m$ integers $a_1, \ldots, a_m, b_1, \ldots, b_m$, where each edge of G is given by its endpoints a_i and b_i). You can assume that the graph is valid (specifically, $1 \leq a_i, b_i \leq n$ and $a_i \neq b_i$). The algorithm outputs "YES" or "NO", corresponding to whether G is a tree or not. Your algorithm must always complete in time polynomial in n (e.g., even $O(n^{10}m^{10})$ suffices).

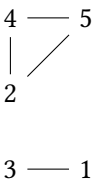
Hint: *Use part (c). There exists a (relatively) simple $O(n+m)$ solution. However, the official solution is $O(n \cdot m)$ for brevity and uses recursion to check if G is connected.*

Example 1: $n = 6$
$m = 5$
$a_1, b_1 = 1, 3$
$a_2, b_2 = 6, 1$
$a_3, b_3 = 3, 5$
$a_4, b_4 = 2, 3$
$a_5, b_5 = 4, 1$



Output: YES

Example 2: $n = 5$
$m = 4$
$a_1, b_1 = 1, 3$
$a_2, b_2 = 4, 5$
$a_3, b_3 = 5, 2$
$a_4, b_4 = 2, 4$



Output: NO