Eidgenössische
Technische Hochschule
Zürich

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

Departement of Computer Science
Johannes Lengler, David Steurer
Kasper Lindberg, Lucas Slot, Hongjie Chen, Manuel Wiedmer

11 November 2024

Algorithms & Data Structures — Exercise sheet 8 — HS 24

The solutions for this sheet are submitted on Moodle until 17 November 2024, 23:59.

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.

We first recall some definitions from the lecture and introduce some new ones.

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

- For $v \in V$, the **degree** $\deg(v)$ of v (german "Knotengrad") is the number of edges that are incident to v.
- A sequence of vertices $(v_0, v_1, \ldots, v_k)$ (with $v_i \in V$ for all i) is a **walk** (german "Weg") if $\{v_i, v_{i+1}\}$ is an edge for each $0 \leq i \leq k-1$. We say that v_0 and v_k are the **endpoints** (german "Startknoten" and "Endknoten") of the walk. The **length** of the walk $(v_0, v_1, \ldots, v_k)$ is k.
- A sequence of vertices $(v_0, v_1, \ldots, v_k)$ is a **closed walk** (german "Zyklus") if it is a walk, $k \geq 2$ and $v_0 = v_k$.
- A sequence of vertices $(v_0, v_1, \ldots, v_k)$ is a **path** (german "Pfad") if it is a walk and all vertices are distinct (i.e., $v_i \neq v_j$ for $0 \leq i < j \leq k$).
- A sequence of vertices $(v_0, v_1, \ldots, v_k)$ is a **cycle** (german "Kreis") if it is a closed walk, $k \geq 3$ and all vertices (except v_0 and v_k) are distinct.
- An **Eulerian walk** (german "Eulerweg") is a walk that contains every edge exactly once.
- A **closed Eulerian walk** (german "Eulerzyklus") is a closed walk that contains every edge exactly once.
- A **Hamiltonian path** (german "Hamiltonpfad") is a path that contains every vertex.
- A **Hamiltonian cycle** (german "Hamiltonkreis") is a cycle that contains every vertex.
- For $u, v \in V$, we say u **reaches** v (or v is **reachable** from u; german "u erreicht v") if there exists a walk with endpoints u and v, or equivalently, there exists a path with endpoints u and v.
- A **connected component** of G (german "Zusammenhangskomponente") is an equivalence class of the (equivalence) relation defined as follows: Two vertices $u, v \in V$ are equivalent if u reaches v.
- A graph G is **connected** (german "zusammenhängend") if for every two vertices $u, v \in V$, u reaches v, or equivalently, if there is only one connected component.
- A graph G is a **tree** (german "Baum") if it is connected and has no cycles.

Exercise 8.1 *Introduction to graphs* **(1 point)**.

A group of $n \geq 3$ people wants to play the following *telephone game*: A random player in the group is given a message. The goal is to communicate this message to each member of the group. Each player is allowed to make one phone call and receive one phone call. Furthermore, a player can only make calls to other players who are in their *contact list* (you may assume that if player a is in the contact list of player b, then player b is also in the contact list of player a).

In this exercise, we care about the following question: *under what circumstances is it possible for the group to win the game, regardless of the starting player?* You may assume the group communicated a strategy beforehand, and each player is aware of the contents of each other player's contact list.

(a) Model the telephone game using a graph. Indicate carefully what the vertices and edges of this graph are. Then, give a necessary and sufficient condition for the game to be winnable (regardless of the starting player) using terminology from the lecture. Briefly argue the correctness of your condition.

(b) Give an example of a situation where the game is winnable for some, but not all starting players. Describe your example by drawing the graph that models it according to part (a).

(c) Someone claims the game is always winnable if the following conditions hold:

- Each player has at least two other players in their contact list;
- For any two players a and b, it is possible for the message to reach player b if player a was the starting player.

Translate these conditions to your graph model of part (a) using terminology from the lecture. Then show the claim is false when $n = 5$.

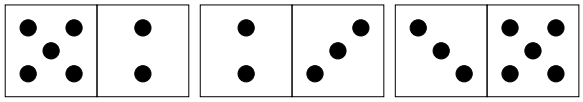
(d)* In a variant of the game for advanced players, the last person to learn the message has to call back the starting player to let them know everything went according to plan. Model this *advanced telephone game* using a graph as in part (a). Then, show that even if the (normal) telephone game is winnable regardless of the starting player, this does not mean the advanced telephone game is also winnable.

Hint: Look up the *Petersen graph*.

Exercise 8.2 *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 like in the example below.



(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 8.3 *Star search, reloaded.*

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 $|V| = n$. 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. 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 or contain several stars can be disregarded.

Exercise 8.4 *Introduction to Trees.*

In this exercise the goal is to prove a few basic properties of trees (for the definition of a tree, see Definition 1).

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

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

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

Hint: *One direction is immediate by part (b). 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 that the remaining graph is still connected and has exactly one edge less. Apply the induction hypothesis and conclude.*

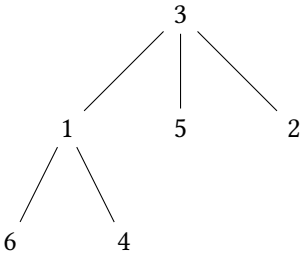
(d) Write the pseudocode of an algorithm that is given a graph G as input and checks whether G is a tree.

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). You do not have to show the correctness of your algorithm or what the running time of your algorithm is.

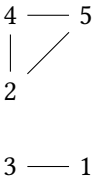
Hint: *Use part (c). There exists a (relatively) simple $O(n+m)$ solution. However, the official solution is $O(n \cdot m)$ for brevity and uses recursion to check if G is connected.*

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

Exercise 8.5 *Short questions about graphs* **(2 points)**.

In the following, let $G = (V, E)$ be a graph, $n = |V|$ and $m = |E|$.

(a) Let $v \neq w \in V$ and suppose that G is a tree. Prove that if P_1 and P_2 are paths that both start at v and end at w, then $P_1 = P_2$.

For each of the following statements, decide whether the statement is true or false. If the statement is true, provide a proof; if it is false, provide a counterexample.

(b) If every vertex of G has at least $\lceil n/2 \rceil$ neighbors, then G is connected.

(c) If G contains a Hamiltonian cycle C, then any other Hamiltonian cycle of G must contain an edge from C.

(d) For every graph G with $n \geq 2$, there must be at least two vertices with the same degree.

(e) Suppose in a connected graph G, for every path of length at least 2, the sum of the degrees of the vertices in the path is even. Then G has an Eulerian walk.

(f) Let G be a connected graph. Suppose that deleting any edge of G does not disconnect the graph. Then deleting any vertex of G does not disconnect the graph. (When deleting a vertex, we also remove all edges incident to the vertex.)