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.

Solution:

Let n denote the number of people at your party. We can model the situation by a graph $G = (V, E)$, where the vertices V are the people who came to your party, and two vertices are connected by an edge whenever they know each other. Since everybody knows exactly 7 other people at the party, we have $\deg(v) = 7$ for all vertices $v \in V$. Therefore, $\sum_{v \in V} \deg(v) = 7n$ since there are n vertices. On the other hand, by the Handshaking lemma (Handschlaglemma) we also know that $\sum_{v \in V} \deg(v) = 2|E|$, and in particular the sum of the degrees must be an even number. In other words, $7n$ is an even number, which implies that n must be even as well.

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 $G_1 = (V_1, E_1), \dots, G_k = (V_k, E_k)$ 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.

Solution:

We first show that disjoint sums of complete graphs are transitive ($\Leftarrow$), and then that any transitive graph is a disjoint sum of complete graphs ($\Rightarrow$).

$\Leftarrow$ Let $G = (V, E)$ be a disjoint sum of complete graphs $G_1 = (V_1, E_1), \dots, G_k = (V_k, E_k)$. Let $\{u, v\}, \{v, w\} \in E$. Since G is a disjoint sum, there exists $i \in \{1..k\}$ such that $v \in V_i$, $\{u, v\} \in E_i$, and

$\{v, w\} \in E_i$. Since $E_i \subseteq V_i \times V_i$, we get $u \in V_i$ and $w \in V_i$. From the assumption that G_i is complete, we finally get $\{u, w\} \in E_i \subseteq E$.

$\Rightarrow$ Let $G = (V, E)$ be a transitive graph. We can decompose G into its connected components $G_1 = (V_1, E_1), \ldots, G_k = (V_k, E_k)$. Clearly, G is the disjoint sum of its connected components. Let us now show that each connected component is complete. Let $i \in \{1..k\}$. Consider $u, v \in V_i$ with $u \neq v$. As u and v are in the same connected component of G, there exists a path $u = w_1 \to w_2 \to \cdots \to w_p = v$ from u to v in G_i.

We will now show by induction on $j \in \{2, \ldots, p\}$: $P(j)$: "the edge $\{u, w_i\}$ is in E_i."

Base case: $j = 2$. As $u = w_1, \ldots, w_p$ forms a path in G_i, we have $\{u, w_2\} = \{w_1, w_2\} \in E_i$, which immediately yields $P(2)$.

Induction step: Let $j \in \{2, \ldots, p-1\}$ such that $P(j)$ holds. This means that we have an edge $\{u, w_j\} \in E_i$. Now, as $w_1, \ldots, w_p$ is a path in G_i, we also have an edge $\{w_j, w_{j+1}\} \in E_i$. Using the transitive property of G, we obtain $\{u, w_{j+1}\} \in E$, and since G is a disjoint sum, we also have $\{u, w_{j+1}\} \in E_i$. This shows $P(j+1)$.

Conclusion: $P(j)$ holds for all $j \in \{2, \ldots, p\}$.

For $j = p$, we obtain $P(p)$: "the edge $\{u, w_p\}$ is in E_i". As $w_p = v$ and $E_i \subseteq E$, we get $\{u, v\} \subseteq E$. Hence G_i is complete, which concludes the proof.

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.

Solution:

Consider the following algorithm:

In the following, we say that a vertex is *marked* iff its flag is set to `True`. In each iteration of the while loop, a new, previously unmarked vertex is explored (if the vertex was already marked, the movement towards this vertex would have been undone). Hence, in each iteration, either the current vertex has $n-1$ neighbors and the algorithm terminates (case 1), or the number of vertices to be explored decreases

Algorithm 1 Star-finding algorithm

```
while countNeighbors() $\neq n-1$ do
    setFlag()
    for $i = 1$ to countNeighbors() do
        moveTo($i$)
        if isSet() then
            undo()
        else
            break
```

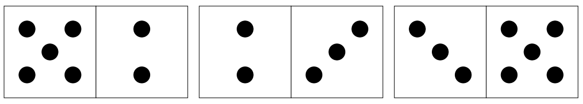
by exactly one (case 2), or the current vertex has no unmarked neighbors and we loop forever on this vertex (case 3). Whenever the algorithm reaches the star $s \in V$, it successfully terminates (case 1), since a vertex is a star if and only if it has $n-1$ neighbors. Now, the star s is, by definition, a neighbor of all vertices; in particular, s is always a neighbor of the current vertex. Hence, for case 3 to occur, the star s must have been previously marked. But this never occurs, since the algorithm always terminates when reaching the star. Hence, only cases 1 and 2 can happen, and the number of unmarked vertices decreases by exactly one in each iteration until the star is eventually reached. This proves the correctness of the algorithm.

The cost of each iteration of the while loop is $O(1) + O(1) + \sum_{i=1}^{\deg v}(O(1) + O(1) + O(1)) = O(1) + O(\deg v)$, which sums up to at most $\sum_{v \in V}(O(1)+O(\deg v)) = O(|V|)+O\left(\sum_{v\in V} \deg v\right) = O(|V|)+O(2|E|) = O(|V|+|E|)$ as every vertex is explored at most once.

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?

Solution:

We directly solve the general problem.

First we note that we may neglect tiles of the form $[x|x]$. If we have a line without them, then we can easily insert them to any place with an x. Conversely, if we have a line with them then we can just remove them. Thus the problem with and without these tiles are equivalent.

Consider the following graph G with n vertices, labelled with $\{1, \ldots, n\}$. We represent the domino tile $[x|y]$ by an edge between vertices x and y. Then the resulting graph G is a complete graph K_n, i.e., the graph where every pair of vertices is connected by an edge. A line of domino tiles corresponds to a walk in this graph that uses every edge at most once, and vice versa. A complete line (of *all* tiles) corresponds to an Eulerian walk in G. Thus we need to decide whether $G = K_n$ has an Euler walk or not.

K_n is obviously connected. If n is odd then all vertices have even degree $n-1$, and thus the graph is Eulerian. On the other hand, if n is even then all vertices have odd degree $n-1$. If $n \geq 4$ is even, then there are more than 3 vertices of odd degree, and therefore K_n does not have an Euler walk. Finally, for $n = 2$, the graph K_n is just an edge and has an Euler walk. Summarizing, there exists an Euler walk if $n = 2$ or n is odd, and there is no Euler walk in all other cases. Hence, it is possible to line up the domino tiles if $n = 2$ or n is odd, and it is impossible otherwise.

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 with at least two vertices there exists a leaf. (post-publication correction marked in red)

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

Solution:

Consider the longest simple path $P = (v_0, v_1, v_2, \ldots, v_{k-1}, v_k)$ in G. Let $a := v_0$ be an endpoint of P. We claim a is a leaf. Suppose for the sake of contradiction that this is not true, i.e., the degree of a is at least 2. Hence, there exists a neighbor $b \neq v_1$ of a. Now, consider, the path $P' = (b, v_0, v_1, \ldots, v_k)$. This is a longer path, hence by choice of P, it cannot be simple. Therefore, since b is the only new addition, there must exist an index i such that $b = v_i$. But now, $(b, v_0, v_1, \ldots, v_i)$ is a simple cycle in G, a contradiction.

(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.

Solution:

We proceed by induction on n.
Base case. When $n = 1$, there can only be $0 = n-1$ edges. When $n = 2$, there exists a unique tree (two vertices connected by an edge), and that one has $1 = n-1$ edges. This completes the

base case.
Induction hypothesis. Assume that the hypothesis is true for every tree with $n \geq 2$ vertices: it contains $n-1$ edges.
Inductive step. We now show the property holds for every tree $G = (V, E)$ with $|V| = n+1$ vertices.

Let u be a leaf in G (it must exist by part (a)), and let v be u's only neighbor in the tree $G = (V, E)$. Consider the graph $G' := (V \setminus \{u\}, E \setminus \{u, v\})$. We first argue that G' is a tree.

Claim: G' is connected. Proof of Claim: Let $a, b \in V \setminus \{u\}$. Since G is a tree, there exists a simple path P in G with endpoints a, b. It is immediate that no simple path can contain a leaf except on its endpoints (or the leaf's only incident edge). Hence, P is also a simple path in G'. Hence, a and b are connected in G'. Hence, G' is connected. This completes the claim.

Claim: G' has no simple cycles. Proof of Claim: Suppose for the sake of contradiction that P is a simple cycle in G'. But since G' is a subgraph of G, P is also a simple cycle in G. However, G is a tree and this is impossible. This completes the claim.

We proven that G' is a tree. It contains $|V \setminus \{u\}| = (n+1) - 1 = n$ vertices. Hence, by induction, $|E \setminus \{u, v\}| = n - 1$. Therefore, $|E| = n$. This completes the inductive step and the proof.

(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.

Solution:

Suppose G is a tree. By definition, G is connected. By part (b), it has $n-1$ edges. This completes one direction of the implication.

We now prove the other direction. Suppose G is connected and has $n-1$ edges. We proceed by induction on n.
Base case. Let $n = 1$. The graph with a single vertex and 0 edges is trivially a tree. Let $n = 2$. There exists one unique graph with two vertices and 1 edge, and that one graph is also obviously a tree. This completes the base case.
Induction hypothesis. Assume the hypothesis: every connected graph with $n \geq 2$ vertices and $n-1$ edges is a tree.
Inductive step. We now show the property holds for $n+1$. Let $G = (V, E)$ be a connected graph with $n+1$ vertices and n edges. The average degree in this graph is $2|E|/|V| = 2n/(n+1) < 2$. Hence, there must exist a vertex u with degree 1 (no connected graph with at least 2 vertices can have 0-degree vertices).

In other words, u is a leaf and let v be u's only neighbor in G. Consider the graph $G' := (V \setminus \{u\}, E \setminus \{u, v\})$. Clearly, G' has $n-1$ edges.

Claim: G' is connected. Proof of Claim: Let $a, b \in V \setminus \{u\}$. Since G is connected, there exists a simple path P in G with endpoints a, b. It is immediate that no simple path can contain a leaf except on its endpoints (or the leaf's only incident edge). Hence, P is also a simple path in G'. Hence, a and b are connected in G'. Hence, G' is connected. This completes the claim.

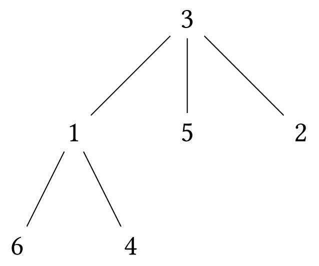
Therefore, we can apply the induction hypothesis on G' and conclude G' is a tree. It is simple to conclude that then G is also a tree: any simple cycle in G must be fully contained in G' (since it cannot contain a leaf), and this is impossible since G' is a tree.

(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 $\{a_1, b_1\}, \{a_2, b_2\}, \dots, \{a_m, b_m\}$ (i.e., the algorithm has access to $2m$ integers $a_1, \dots, a_m, b_1, \dots, b_m$, 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).

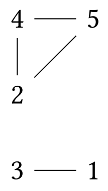
Hint: *Use part (c). There exists a (relatively) simple $O(n + m)$ solution. However, the official solution is $O(n \cdot m)$ for brevity and uses recursion to check if G is connected.*

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

Solution:

Algorithm 2

```
Input: integers $n, m$. Collection of integers $a_1, b_1, a_2, b_2, \ldots, a_m, b_m$.

Let $visited[1 \ldots n]$ be a global variable, initialized to $False$.

**function** $walk(u)$                    ▷ Find all neighbors of $u$ that have not been visited and walk there.
    $visited[u] \leftarrow True$
    **for** $i \leftarrow 1 \ldots m$ **do**                    ▷ Iterate over all edges.
        **if** $a_i = u$ and not $visited[b_i]$ **then**
            $walk(b_i)$
        **if** $b_i = u$ and not $visited[a_i]$ **then**
            $walk(a_i)$

$walk(1)$                    ▷ Find all vertices connected to 1.
$connected \leftarrow True$ if $visited[\cdot] = [True, True, \ldots, True]$ and $connected \leftarrow False$ otherwise
**if** $connected = True$ and $m = n - 1$ **then**                    ▷ Use the characterization from part (c).
    Print("YES")
**else**
    Print("NO")
```