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
Lucas Slot, Manuel Wiedmer, Hongjie Chen, Ding Jingqiu

20 November 2023

Algorithms & Data Structures Exercise sheet 9 HS 23

The solutions for this sheet are submitted at the beginning of the exercise class on 27 November 2023.

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 *Transitive graphs.*

Let $G = (V, E)$ be an undirected graph. We say that G 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 union** of $G_1 = (V_1, E_1), \dots, G_k = (V_k, E_k)$ iff $V = V_1 \cup \dots \cup V_k$, $E = E_1 \cup \dots \cup E_k$, and the $(V_i)_{1 \leq i \leq k}$ are pairwise disjoint.

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

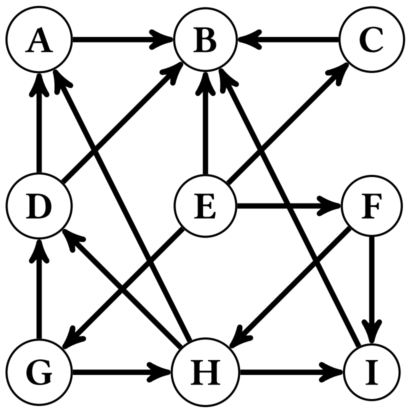
Exercise 9.2 *Short statements about graphs (cont'd)* **(1 point)**.

In the following, let $G = (V, E)$ be a directed graph. 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.

(a) If for every vertex $v \in V$ its in-degree $\deg_{\text{in}}(v)$ is even, then $|E|$ is even.

(b) For a longest directed path $P : v_0, \dots, v_\ell$ in G, the endpoint has to be a sink.

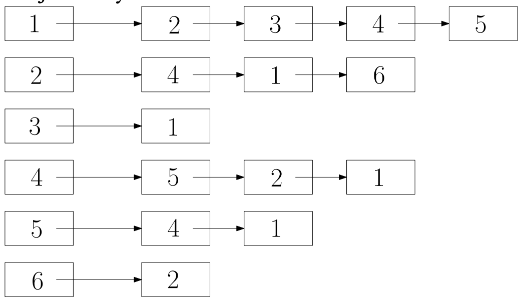
(c) The following graph has a topological sorting. If so, give a topological sorting; if not, prove why no topological sorting can exist.



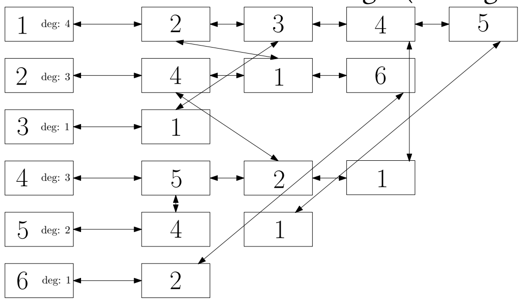
Exercise 9.3 *Data structures for graphs.*

Consider three types of data structures for storing an undirected graph G with n vertices and m edges:

1) Adjacency matrix.
2) Adjacency lists:



3) Adjacency lists, and additionally we store the degree of each node, and there are pointers between the two occurences of each edge. (An edge appears in the adjacency list of each endpoint).



For each of the above data structures, what is the required memory (in Θ-Notation)?

Which runtime (worst case, in Θ-Notation) do we have for the following queries? Give your answer depending on n, m, and/or $\deg(u)$ and $\deg(v)$ (if applicable).

(a) Input: A vertex $v \in V$. Find $\deg(v)$.

(b) Input: A vertex $v \in V$. Find a neighbor of v (if a neighbor exists).

(c) Input: Two vertices $u, v \in V$. Decide whether u and v are adjacent.

(d) Input: Two adjacent vertices $u, v \in V$. Delete the edge $e = \{u, v\}$ from the graph.

(e) Input: A vertex $u \in V$. Find a neighbor $v \in V$ of u and delete the edge $\{u, v\}$ from the graph.

(f) Input: Two vertices $u, v \in V$ with $u \neq v$. Insert an edge $\{u, v\}$ into the graph if it does not exist yet. Otherwise do nothing.

(g) Input: A vertex $v \in V$. Delete v and all incident edges from the graph.

For the last two queries, describe your algorithm.

Exercise 9.4 *Number of paths in DAGs* **(1 point)**.

Let $G = (V, E)$ be a directed graph without directed cycles[1] (i.e., a directed acyclic graph or short DAG). Assume that $V = \{v_1, \ldots, v_n\}$ (for $n = |V| \in \mathbb{N}$). Further assume that v_1 is a source and v_n is

[1]A directed cycle is a closed directed walk of length at least 2 for which all vertices are pairwise distinct except the endpoints.

a sink. The goal of this exercise is to find the number of paths from v_1 to v_n.

(a) Prove that there exists a topological sorting of G that has v_1 as first and v_n as last vertex.

Using part (a), we assume from now on that the sorting $v_1, v_2, \dots, v_n$ of the vertices is a topological sorting. We can achieve this by renaming the vertices. Part (a) tells us then that we do not need to rename v_1 and v_n.

(b) Prove that for any directed v_1-v_n-path $P : v_1 = v_{i_0}, v_{i_1}, \dots, v_{i_\ell} = v_n$ we have $i_0 < i_1 < \cdots < i_\ell$.

(c) Describe a bottom-up dynamic programming algorithm that, given a graph G with the property that $v_1, \dots, v_n$ is a topological sorting, returns the number of v_1-v_n paths in G in $O(|V| + |E|)$ time. You can assume that the graph is provided to you as a pair (n, Adj) of the integer $n = |V|$ and the adjacency lists Adj. Your algorithm can access $Adj[u]$, which is a list of vertices to which u has a direct edge, in constant time. Formally, $Adj[u] := \{v \in V \mid (u, v) \in E\}$.

   In your solution, address the following aspects:

   1. *Dimensions of the DP table*: What are the dimensions of the DP table?
   2. *Subproblems*: What is the meaning of each entry?
   3. *Recursion*: How can an entry of the table be computed from previous entries? Justify why your recurrence relation is correct. Specify the base cases of the recursion, i.e., the cases that do not depend on others.
   4. *Calculation order*: In which order can entries be computed so that values needed for each entry have been determined in previous steps?
   5. *Extracting the solution*: How can the solution be extracted once the table has been filled?
   6. *Running time*: What is the running time of your solution?

   Hint: *Define the entry of the DP table as $DP[i]$ = number of paths in G from v_i to v_n.*

(d)* What happens if the vertices v_1 and v_n are not a source respectively a sink? Can we still find the number of v_1-v_n paths using a similar approach as above?

Exercise 9.5 *Strongly connected vertices* **(1 point)**.

Let $G = (V, E)$ be a directed graph with n vertices and m edges. We say two distinct vertices $v, w \in V$ are *strongly connected* if there exists both a directed path from v to w, and from w to v.

Describe an algorithm which finds a pair $v, w \in V$ of strongly connected vertices in G, or decides that no such pair exists. The runtime of your algorithm should be at most $O(n + m)$. You are provided with the number of vertices n, and the adjacency list Adj of G.

Hint: *Use DFS as a subroutine.*