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, Markus Püschel, David Steurer
Kasper Lindberg, Kostas Lakis, Lucas Pesenti, Manuel Wiedmer

17 November 2025

Algorithms & Data Structures — Exercise sheet 9 — HS 25

The solutions for this sheet are submitted on Moodle until 23 November 2025, 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.

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 $G_1 = (V_1, E_1), \ldots, G_k = (V_k, E_k)$ iff $V = V_1 \cup \cdots \cup V_k$, $E = E_1 \cup \cdots \cup E_k$, and $V_1, \ldots, V_k$ are pairwise disjoint.

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

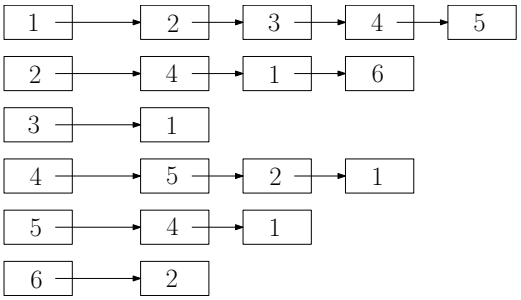
Exercise 9.2 *Data structures for graphs.*

Consider the following three types of data structures for storing an undirected graph $G = (V, E)$ with $V = [n]$ and $|E| = m$: (the following three instances of data structures are for a graph with $n = 6$ and $m = 7$)

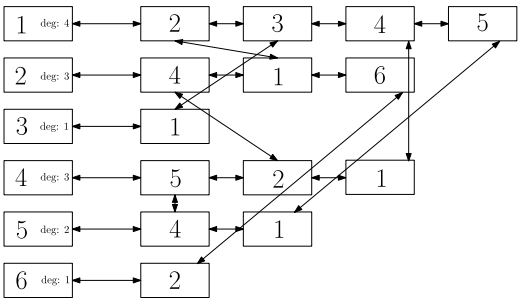
1) Adjacency matrix:

0	1	1	1	1	0
1	0	0	1	0	1
1	0	0	0	0	0
1	1	0	0	1	0
1	0	0	1	0	0
0	1	0	0	0	0

2) Adjacency lists: For each vertex v, we store its neighbors in a singly linked list $\mathrm{Adj}[v]$. Given a vertex $v \in V$, we can access the head of $\mathrm{Adj}[v]$ in constant time.



3) Improved adjacency lists: For each vertex, we store its neighbors in a doubly linked list $\mathrm{Adj}[v]$. Given a vertex $v \in V$, we can access the head of $\mathrm{Adj}[v]$ in constant time. We also store the degree of each vertex $v \in V$ in the head of $\mathrm{Adj}[v]$. Additionally, for each edge $\{u, v\}$, there are pointers between the element corresponding to v in $\mathrm{Adj}[u]$ and the element corresponding to u in $\mathrm{Adj}[v]$.



Question: For each of the above data structures, what is the required memory in Θ-notation?

Question: Which worst-case runtime do we have for the following queries? Give your answer in Θ-notation 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: An edge $\{u, v\} \in E$. Delete the edge $\{u, v\}$ 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 all edges incident to v from the graph.

Question: For the last two queries, describe your algorithm.

Exercise 9.3 *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 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.

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

Using part (a), we assume from now on that the sorting $v_1, v_2, \ldots, v_n$ 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 $P : v_1 = v_{i_0}, v_{i_1}, \ldots, v_{i_\ell} = v_n$ 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 $v_1, \ldots, v_n$ 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? Describe the calculation order in pseudocode.
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?

Definition 1. We say that a graph G is *Eulerian* if it contains a closed Eulerian Walk (Eulerzyklus).

Definition 2. A graph $G = (V, E)$ is *bipartite* if it is possible to partition the vertices in two sets V_1 and V_2 (i.e. $V_1 \cap V_2 = \emptyset$ and $V_1 \cup V_2 = V$) such that every edge $\{u, v\} \in E$ has one endpoint in V_1 and the other in V_2.

Theorem 1. *A graph is bipartite if and only if it does not contain any cycle of odd length.*

Exercise 9.4 *Bipartite graphs, Eulerian graphs and painting rooms* **(2 points)**.

In this exercise, you can use Theorem 1 above.

(a)* Prove Theorem 1.

(b) Prove or disprove the following statements:

(1) Every graph G that is bipartite and Eulerian must have an even number of edges.

(2) Every Eulerian graph G that has an even number of vertices must also have an even number of edges.

(c) You recently moved in with your best friend (see floor plan below) and you would like to repaint the room walls. Every room should be painted either in red or in purple (as these are your favorite colors), and you also would like that whenever you walk from a room to another room through a door, the color changes. Is that possible?

Note that there are 7 rooms (i.e. the Hallway, the Bathroom and the Kitchen are counted as rooms).