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

28 November 2022

Algorithms & Data Structures — Exercise sheet 10 — HS 22

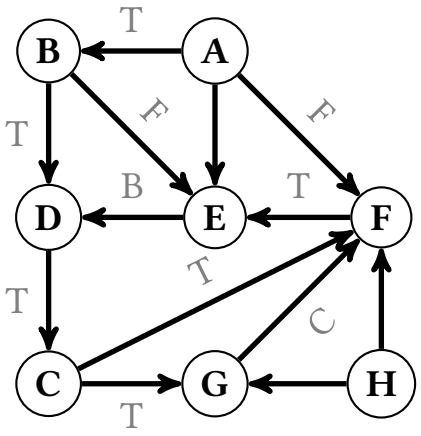
The solutions for this sheet are submitted at the beginning of the exercise class on 5 December 2022.

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 10.1 *Depth-First Search* **(1 point)**.

Execute a depth-first search (*Tiefensuche*) on the following graph starting from vertex A. Use the algorithm presented in the lecture. When processing the neighbors of a vertex, process them in alphabetical order.



(a) Mark the edges that belong to the depth-first tree (*Tiefensuchbaum*) with a "T" (for tree edge).

(b) For each vertex in the depth-first tree, give its *pre*- and *post*-number.

(c) Give the vertex ordering that results from sorting the vertices by pre-number. Give the vertex ordering that results from sorting the vertices by post-number.

(d) Mark every forward edge (*Vorwärtskante*) with an "F", every backward edge (*Rückwärtskante*) with an "B", and every cross edge (*Querkante*) with a "C".

(e) Does the above graph have a topological ordering? How can we use the above execution of depth-first search to find a directed cycle?

(f) Draw a scale from 1 to 16, and mark for every vertex v the interval I_v from pre-number to post-number of v. What does it mean if $I_u \subset I_v$ for two different vertices u and v?

(g) Consider the graph above where the edge from E to D is removed and an edge from A to H is added. How does the execution of depth-first search change? Which topological sorting does the

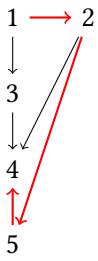
depth-first search give? If you sort the vertices by *pre-number*, does this give a topological sorting?

Exercise 10.2 *Longest path in DAGs* **(1 point)**.

Given a directed graph $G = (V, E)$ without directed cycles (i.e., a DAG), the goal is to find the number of edges on the **longest path** in G.

Describe a dynamic-programming algorithm that, given G, returns the length of the longest path in G in $O(|V| + |E|)$ time. You can assume that $V = \{1, 2, \ldots, n\}$, and that the graph is provided to you as a pair (n, Adj) of an integer $n = |V|$ and an adjacency list Adj. Your algorithm can access $Adj[u]$, which is a list of vertices to which u has a direct edge, in constant time. Formally, $Adj[u] := \{v \in V \mid (u, v) \in E\}$.

Example: $n = 5$



Output: 3

(the path is highlighted in red.)

In your solution, address the following aspects:

1. *Dimensions of the DP table*: What are the dimensions of the DP table?
2. *Definition of the DP table*: What is the meaning of each entry?
3. *Computation of an entry*: How can an entry be computed from the values of other entries? Specify the base cases, i.e., the entries 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?

Exercise 10.3 *Subtree sum* **(1 point)**.

Definition 1. Given a directed graph $\vec{G} = (V, \vec{E})$ we define its **undirected version** as $\overleftrightarrow{G} = (V, \overleftrightarrow{E})$ with each directed edge $u \to v$ being transformed to an undirected $u \leftrightarrow v$. Formally, $\overleftrightarrow{E} := (\{u, v\} \mid (u, v) \in \vec{E}\}$.

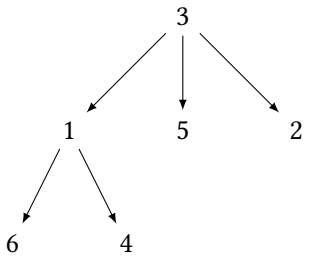
Definition 2. A directed graph $G = (V, E)$ is a **tree rooted at** $r \in V$ if G's undirected version $\overleftrightarrow{G}$ is an undirected tree (see Exercise 9.5 for a definition) and every node is reachable from r via a directed path.

Write the pseudocode of an algorithm that, given a rooted tree $G = (V, E)$, computes, for each vertex v, the total number of vertices that are reachable from v (via directed paths). The algorithm should have a runtime of $O(|V| + |E|)$. You an assume $V = \{1, 2, \ldots, n\}$. The graph will be given to the algorithm as access to n, the root $r \in V$, and an adjacency list. Namely, the algorithm can access $Adj[u]$, which is a list of vertices to which u has a direct edge. Formally, $Adj[u] := \{v \in V \mid (u, v) \in E\}$.

Explain in a few sentences why your algorithm achieves the desired runtime.

Hint: *If needed, you can use the fact that "in G, there is a unique path from the root to each vertex" without proof.*

Example:
$n = 6$
$r = 3$



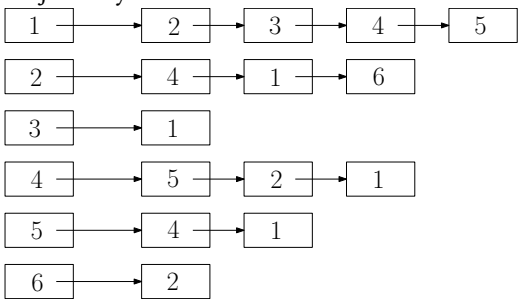
Output: [3, 1, 6, 1, 1, 1].

Exercise 10.4 *Data structures for graphs.*

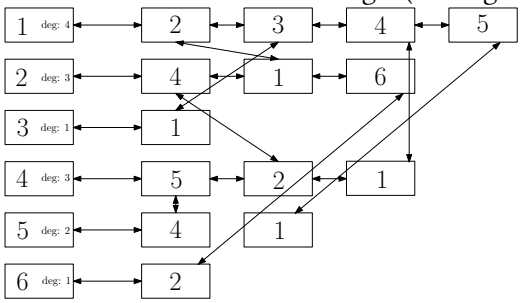
Consider three types of data structures for storing a graph G with n vertices and m edges:

a) Adjacency matrix.

b) Adjacency lists:



c) 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 neighbour of v (if a neighbour 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 10.5 *Maze solver.*

You are given a maze that is described by a $n \times n$ grid of blocked and unblocked cells (see Figure **??**). There is one start cell marked with 'S' and one target cell marked with 'T'. Starting from the start cell your algorithm may traverse the maze by moving from unblocked fields to adjacent unblocked fields. The goal of this exercise is to devise an algorithm that given a maze returns the best solution (traversal from 'S' to 'T') of the maze. The best solution is the one that requires the least moves between adjacent fields.

Hint: *You may assume that there always exists at least one unblocked path from 'S' to 'T' in a maze.*

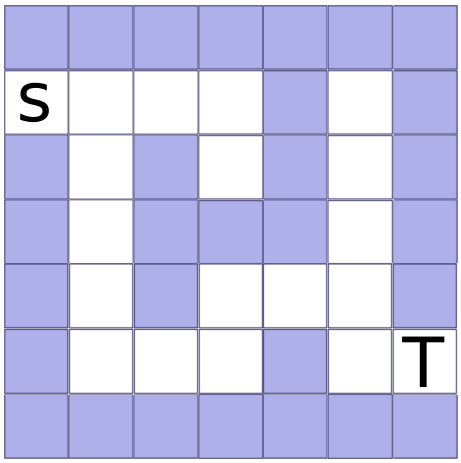


Figure 1: An example of 7×7 maze in which purple fields are blocked, white fields can be traversed (are unblocked). The start field is marked with 'S' and the target field with a 'T'.

(a) Model the problem as a graph problem. Describe the set of vertices V and the set of edges E in words. Reformulate the problem description as a graph problem on the resulting graph.

(b) Choose a data structure to represent your maze-graphs and use an algorithm discussed in the lecture to solve the problem.

Hint: *If there are multiple solutions of the same quality, return any one of them.*

(c) Determine the running time and memory requirements of your algorithm in terms of n in Θ notation.