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
Johannes Lengler, Gleb Novikov, Chris Wendler

25. November 2019

Algorithms & Data Structures — Exercise sheet 10 — HS 19

Exercise Class (Room & TA): _____

Submitted by: _____

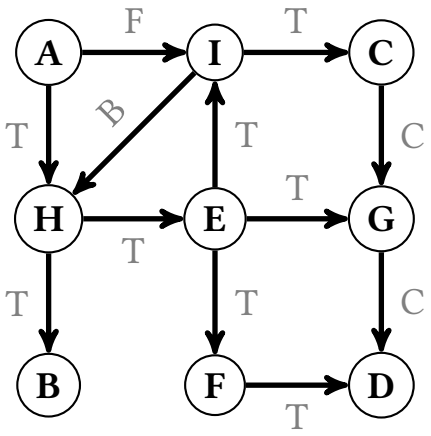
Peer Feedback by: _____

Points: ____

Submission: On Monday, 02 December 2019, hand in your solution to your TA *before* the exercise class starts. Exercises that are marked by * are challenge exercises. They do not count towards bonus points.

Exercise 10.1 *Depth-First Search and Breadth-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, give its *pre*- and *post*-number.

Solution: A (1,18), H (2,17), B (3,4), E (5,16), F (6,9), D (7,8), G (10,11), I(12,15), C (13,14)

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.

Solution: Pre-ordering: A, H, B, E, F, D, G, I, C. Post-ordering: B, D, F, G, C, I, E, H, A

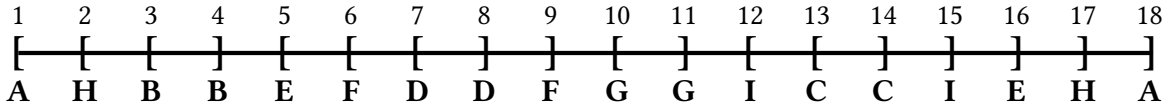
d) Mark every forward edge (Vorwärtskante) with an "F", every backward edge (Rückwärtskante) with an "B", and every cross edge (Querkante) with a "C".

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?

Solution: The decreasing order of the post-numbers gives a topological ordering, whenever the graph is acyclic. This is the case if and only if there are no back edges. If there is a back edge, then together with the tree edges between its end points it forms a directed cycle. In our graph, the only back edge is $I \to H$, and the tree edges from H to I are $H \to E$ and $E \to I$. Together they form the directed cycle $(H \to E \to I \to H)$.

f) Draw a scale from 1 to 18, 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?

Solution:



If $I_u \subset I_v$ for two different vertices u and v, then u is visited during the call of DFS-Visit(v).

g) Consider the graph above with the edge from I to H removed. 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 topogical sorting?

Solution: The execution of depth-first search doesn't change. The topological sorting is: A, H, E, I, C, G, F, D, B. The pre-ordering is A, H, B, E, F, D, G, I, C; it does not give a topological ordering, since there is an edge (G, D) in the graph.

h) Consider the graph above with the edge from I to H removed. Give the order in which the vertices are enqueued by a breadth-first search starting in A. You should list vertices several times if they are enqueued more than once. When processing the neighbors of a vertex, process them in alphabetical order.

The BFS-order is the order in which the vertices are enqueued for the first time in a breadth-first search. What is the BFS-order of the above execution? Does it give a topological sorting?

Solution: The order in which the vertices are enqueued by a breadth-first search: A, H, I, B, E, C, F, G, G, D, D. The BFS-order: A, H, I, B, E, C, F, G, D. it does not give a topological ordering, since there is an edge (E, I) in the graph.

Exercise 10.2 *Graph Coloring* **(1 point)**.

A k-coloring of a graph $G = (V, E)$ is a mapping $c : V \to \{1, \ldots, k\}$ that assigns a color to each vertex such that $c(v_i) \neq c(v_j)$ if $\{v_i, v_j\} \in E$. A two-coloring and three-coloring is a coloring with 2 and 3 colors, respectively.

For $n > 2$ consider the cycle graph G with $V = \{v_1, \ldots, v_n\}$ and edges

$$E = \{\{v_i, v_{i+1}\}, \text{ for } i \in \{1, \ldots, n-1\}\} \cup \{\{v_1, v_n\}\}.$$

a) Show that for even $n > 2$ there exists a two-coloring of the cycle.

Solution: We assign 1 to v_i if i is odd and 2 if i is even. Obviously, $c(v_i) \neq c(v_{i+1})$. Since n is even, $c(v_n) \neq c(v_1)$, hence it is indeed a two-coloring.

b) Show that for odd $n > 2$ there does not exist a two-coloring of the cycle.

Solution: Assume that there exists a two-coloring c. Let's prove by induction that $c(v_i) = c(v_1)$ for all odd $i \leq n$. Base case $i = 1$ is obvious. Assume that $c(v_i) = c(v_1)$ for some $i < n$. Let's show that $c(v_{i+2}) = c(v_1)$. Since c is a two-coloring, $c(v_{i+2}) \neq c(v_{i+1})$ and $c(v_{i+1}) \neq c(v_i)$. Since there are only two colors, $c(v_{i+2}) = c(v_i)$. Hence $c(v_i) = c(v_1)$ for all odd $i \leq n$. Therefore, $c(v_n) = c(v_1)$, which contradicts the assumption that c is a two-coloring.

c) Show that for arbitrary $n > 2$ there exists a three-coloring of the cycle.

Solution: Consider a coloring such that $c(v_1) = 1$, $c(v_i) = 2$ for even i and $c(v_i) = 3$ for odd $i > 1$. Obviously, $c(v_i) \neq c(v_{i+1})$. Moreover, $c(v_1) = 1 < c(v_n)$.

d) Come up with an example graph for which there does not exist a three-coloring.

Solution: For example, a complete graph with 4 vertices K_4: there cannot be two vertices with the same color, since every two vertices are connected by an edge. Hence there is no k-coloring of K_4 if $k < 4$.

Exercise 10.3 *Maze solver* **(1 point)**.

You are given a maze that is described by a $n \times n$ grid of blocked and unblocked cells (see Figure 1). 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.*

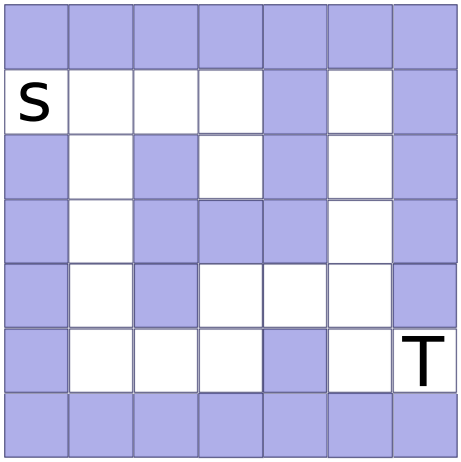


Figure 1: An example of 7×7 maze in which purple fields are blocked, white fields can be traversed (are unblocked). The start field is marked with 'S' and the target field with a 'T'.

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 graphs.

Solution: V is a set of unblocked fields, there is and edge between v_i and v_j if and only v_i and v_j are adjacent unblocked fields. The corresponding graph problem is to find a shortest path between

vertices ‘S’ and ‘T’ in $G = (V, E)$.

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

Solution: The data structure is adjacency list, the algorithm is BFS starting from ‘S’. Once we know all the distances from ‘S’, we append vertices to a sequence starting from ‘T’ using the following rule: if the last appended vertex is v, we append some neighbour u of v such that $d_G(\text{‘S’}, v) = d_G(\text{‘S’}, u) + 1$. We stop after appending ‘S’. Then we return a reverse sequence.

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.

Solution: Adjacency list requires $\Theta(|V| + |E|)$ memory, where V is a number of vertices and $|E|$ is a number of edges in the graph. BFS requires $\Theta(|V| + |E|)$ time and appending procedure also requires $\Theta(|V| + |E|)$ time, so the total running time is $\Theta(|V| + |E|)$. Since each vertex has degree at most 4, $|E| = \mathcal{O}(|V|)$, so the running time and memory are $\Theta(|V|)$ which is $\Theta(n^2)$ in the worst case.

Exercise 10.4 *Soapbox race.*

You participate in a soapbox race. In contrast to a conventional race you are allowed to use the entire street network. The street network is modeled as a directed graph $G = (V, E)$. Each vertex $v \in V$ corresponds to a crossroad and there is an edge (v, w) in E if there is a street from v to w with no crossroads in between. Additionally, for each edge (v, w) we know that v's altitude is strictly higher than w's. Your soapbox is not motorized. Using the street from (v, w) requires $f((v, w))$ seconds, for each $(v, w) \in E$.

In this exercise you can assume that the directed graph is represented by a data structure that allows you to traverse the direct successors and direct predecessors of a vertex u in time $\mathcal{O}(\deg_+(u))$ and $\mathcal{O}(\deg_-(u))$ respectively, where $\deg_-(u)$ is the in-degree of vertex u and $\deg_+(u)$ is the out-degree of vertex u.

a) Provide a *dynamic programming* algorithm that, given the graph G, the time function $f : E \to \mathbb{R}^+$, a start node s and a target node t, determines the time required by the fastest path from s to t. If there is no path from s to t, you can assume that the time is ∞.

   (a) *Dimension of the DP table: What is the dimension of the table $DP[\ldots]$?* What range do you have in each dimension?

   (b) *Definition of the DP table*: What is the meaning of each entry?

   (c) *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.

   (d) *Calculation order*: In which order can entries be computed so that values needed for each entry have been determined in previous steps?

   (e) *Extracting the solution*: How can the final solution be extracted once the table has been filled?

   (f) *Running time*: What is the runtime of your solution?

Dimensions of the table:

The DP table is linear, its dimension is $n = |V|$.

Meaning of a table entry (in words): $DP[v]$ is the distance between s and v. If there is no path from s to v, $DP[v] = \infty$.

Computation of an entry (initialization and recursion): $DP[s] = 0$. $DP[v] = \infty$ if $v \neq s$ and there is no u such that $(u, v) \in E$. For any other v:

$$DP[v] = \min_{u:(u,v)\in E} \Big[DP[u] + f((u, v))\Big].$$

Order of computation: Topological order. It exists since "for each edge (v, w) we know that v's altitude is strictly higher than w's" implies that G is a directed acyclic graph.

Computing the output: $DP[t]$ contains the result.

Running time in $\mathcal{O}$-notation in terms of $|V|$ and $|E|$: Topological sorting takes time $\mathcal{O}(|V| + |E|)$, filling each $DP[v]$ takes $\mathcal{O}(\deg\_(v))$ time, so filling the whole DP table takes time $\mathcal{O}(|V|+|E|)$. So the running time is $\mathcal{O}(|V| + |E|)$.

b) Finally, assume you have already filled out the DP table. Describe how you can find the fastest path using the DP table.

Hint: *If there are multiple fastest paths return only one of them. If there is no paths from s to t, you should output the message that there is no path from s to t.*

Solution: If $DP[t] = \infty$ we just output the message that there is no path from s to t. Otherwise we append vertices to a sequence starting from t using the following rule: if the last appended vertex is v, we append some u such that $(u, v) \in E$ and $DP[v] = DP[u]+f((u, v))$. We stop after appending s. Then we return a reverse sequence. The running time of this procedure is $\mathcal{O}(|V| + |E|)$.