

Departement of Computer Science
Markus Püschel, David Steurer
Johannes Lengler, Gleb Novikov, Chris Wendler

09. December 2019

Algorithms & Data Structures

Exercise sheet 12

HS 19

Exercise Class (Room & TA): _____

Submitted by: _____

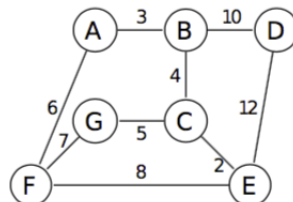
Peer Feedback by: _____

Points: _____

Submission: On Monday, 16. 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. You will obtain feedback for this sheet (in particular the number of bonus points) by email from your TA.

Exercise 12.1 *MST practice (1 point).*

Consider the following graph



- Compute the minimum spanning tree (MST) using Boruka's algorithm. For each step, provide the set of edges that are added to the MST.
- Provide the order in which Kruskal's algorithm adds the edges to the MST.
- Provide the order in which Prim's algorithm (starting at vertex A) adds the edges to the MST.

Exercise 12.2 *MST theory (1 point).*

Let $G = (V, E)$ be a connected edge-weighted graph in which all edge-weights are different. Let $T \subseteq E$ be a spanning tree of G . You already know (exercise 9.2.) that adding an edge $e \in E \setminus T$ to a tree T creates a cycle C_e in $T \cup \{e\}$. Moreover, this cycle is unique. (You may use this fact without further justification.)

In this exercise, we show that T is the minimum spanning tree (MST) of G if and only if for every edge $e \in E \setminus T$, the weight of e is larger than the weight of all other edges in C_e . Note that this consists of the following two parts.

a) Prove the *only if* part:

T is the MST of $G \implies$ the weight of each edge $e \in E \setminus T$ is the maximal weight in C_e .

b)* Prove the *if* part:

The weight of each edge $e \in E \setminus T$ is the maximal weight in $C_e \implies T$ is the MST of G .

Exercise 12.3 *Tube mail (1 point).*

You and your friends decide to install a tube mail system to enable high speed communication and transfer of reasonably small goods between each other. You don't insist on direct connections – you are fine with sending messages via relaying it through one or several friends. You only want that from each friend to each other there is a sequence of friends connecting them that can forward the message via tubes.

Your best friend knows that you are a computer scientist. Thus, he*she gives you a document containing the installation costs and tells you to come up with the most cost efficient tube network that connects your group of friends. The installation-cost-document contains for each pair of residences the cost of installing a tube between them. Your friend obtained this document by calling the tube mail installation company.

a) Model the problem as a graph problem:

1) What are the vertices, the edges and the weights of the graph?

2) What is the graph problem your friend wants you to solve?

b) Solve the problem using an algorithm from the lecture (without modification).

c) Discuss the running time of your solution in terms of $|V|$ and $|E|$.

The following exercise is a challenge exercise and it will not be relevant for the exam.

***Exercise 12.4** *Shortest paths in graphs with negative cycles.*

From lectures you know the Bellman-Ford algorithm that finds the shortest path between any two vertices in any graph that does not contain negative cycles. A natural question is whether there exists an efficient algorithm that finds the shortest (vertex-disjoint) path between any two vertices in arbitrary weighted graphs (possibly with negative cycles).

Hint: *Note that this problem is specifically about paths, not about walks. So in this exercise we are only interested in vertex-disjoint walks, i.e., walks do not contain any cycles. This differs from the sloppy use of the word paths that is otherwise used for shortest-path algorithms.*

This guided exercise shows that it is unlikely that such algorithm exists. Specifically, the existence of such an algorithm implies the existence of an efficient algorithm that solves the Hamiltonian path problem.

Recall that a Hamiltonian path in a graph is a path that visits each vertex exactly once. The Hamiltonian path problem is a problem of determining whether a Hamiltonian path exists in a given graph. It is strongly believed that this problem can not be solved in polynomial time.¹

¹One can show that this problem satisfies a property called *NP-hardness*, and a polynomial algorithm for this problem would also lead to a polynomial algorithm for a large number of other problems.

- a) Show that if there exists an algorithm that finds the shortest (vertex-disjoint) path between any two vertices in any weighted graph (possibly with negative cycles) with n vertices in time $T(n)$, then there exists an algorithm that finds the longest (vertex-disjoint) path between any two vertices in any weighted graph in time $T(n) + \mathcal{O}(n^2)$.
- b) Show that if there exists an algorithm that finds the longest (vertex-disjoint) path between any two vertices in any weighted graph with n vertices in time $T(n)$, then there is an algorithm that can find a Hamiltonian path in any (unweighted) graph in time $\mathcal{O}(n^2 \cdot T(n))$ (or output that such a path does not exist).
- c) Combining parts a) and b), conclude that if there exists a polynomial time algorithm that can find the shortest (vertex-disjoint) path between any two vertices in any weighted graph, then the Hamiltonian path problem can be solved in polynomial time.