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

09. December 2019

# Algorithms & Data Structures — Exercise sheet 12 — HS 19

Exercise Class (Room & TA): ____________________

Submitted by: ____________________

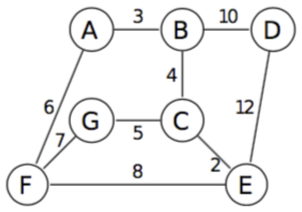
Peer Feedback by: ____________________

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



a) Compute the minimum spanning tree (MST) using Boruvka's algorithm. For each step, provide the set of edges that are added to the MST.

**Solution:** At the first step we add edges $\{A, B\}, \{C, E\}, \{D, B\}, \{F, A\}, \{G, C\}$. At the second step we add an edge $\{B, C\}$.

b) Provide the order in which Kruskal's algorithm adds the edges to the MST.

**Solution:** $\{C, E\}, \{A, B\}, \{B, C\}, \{C, G\}, \{A, F\}, \{B, D\}$.

c) Provide the order in which Prim's algorithm (starting at vertex A) adds the edges to the MST.

**Solution:** $\{A, B\}, \{B, C\}, \{C, E\}, \{C, G\}, \{A, F\}, \{B, D\}$.

**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 $C_e$.

**Solution:** Assume that $T$ is a minimal spanning tree and that there is an edge $e \in E \setminus T$ such that its weight is strictly smaller than the weight of some edge $e' \in C_e$. Then we can replace $e'$ by $e$ in $T$ and obtain a graph $\tilde{T}$. $\tilde{T}$ has $n-1$ edges and is connected, since any two vertices $u$ and $v$ can be connected by a path in $T$, and we can replace $e'$ in this path by $C_e \setminus \{e'\}$ to get a walk from $u$ to $v$ in $\tilde{T}$. Hence $\tilde{T}$ is a tree and its weight is strictly smaller than the weight of $T$, which contradicts the assumption that $T$ is a minimal spanning tree.

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 $G$.

**Solution:** Assume that the weight of each edge $e \in E \setminus T$ is the maximal weight in $C_e$. Consider some MST $T_0$ of $G$. If $T_0 = T$, then $T$ is obviously MST. Otherwise $T_0$ contains an edge $e = \{u, v\}$ that is not contained in $T$. If we remove $e$ from $T_0$, we get two different connected components. Notice that $C_e$ contains an edge $e' \in T$ that connects two vertices from these components (since $C_e$ contains a path from $u$ to $v$ in $T$). Moreover, the weight of $e$ is maximal in $C_e$, so the weight of $e'$ is not greater than the weight of $e$. If we replace $e$ by $e'$ in $T_0$, we get a spanning tree $T_1$ (it is a tree since it has $n-1$ edges and remains connected) and its weight is not greater than the weight of $T_0$, so $T_1$ is MST. Notice that the number of common edges in $T_1$ and $T$ is strictly greater than the number of common edges in $T_0$ and $T$. So we can repeat this procedure and get minimal spanning trees $T_2$, $T_3$ and so on until we get $T_k = T$ for some $k \geq 0$, which implies that $T$ is a minimal spanning tree.

## 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?

   **Solution:** $G = (V, E, w)$ is defined as follows: $V$ is a set of all friends, $E$ is a set of all unordered pairs of elements of $V$, $w(\{u, v\})$ is a cost of installing a tube between $u$ and $v$.

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

   **Solution:** Finding MST of $G$.

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

**Solution:** We can solve the problem using Kruskal's algorithm or Prim's algorithm.

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

**Solution:** The running time of Kruskal's algorithm is $\Theta(|E| \log |V|)$, the running time of Prim's algorithm depends on the priority queue implementation, if we use a binary heap, the running time of Prim's algorithm is $\Theta(|E| \log |V|)$.

Since $E$ is a set of all unordered pairs of elements of $V$, $|E| = \Theta(|V|^2)$. Hence the running time is $\Theta(|V|^2 \log |V|)$.

*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]

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 $T(n) + \mathcal{O}(n^2)$.

**Solution:** We change the sign of each weight and apply the shortest path algorithm to a new graph. The shortest path in the new graph is the longest path in the initial graph.

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).

**Solution:** We assign weight 1 to each edge and apply the longest path algorithm to each pair of vertices. If among the longest paths there exists a path of length (weight) $n - 1$, it is a Hamiltonian path, and if all longest paths between any pair of vertices have length strictly less than $n - 1$, there is no Hamiltonian path in the graph.

c) Combining parts a) and b), conclude that if there exists a polynomial time algorithm that can find

[1]One 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.

the shortest (vertex-disjoint) path between any two vertices in any weighted graph, then the Hamiltonian path problem can be solved in polynomial time.

**Solution:** If there exists an algorithm that can find the shortest vertex-disjoint path between any two vertices in any weighted graph in time $\mathcal{O}(n^c)$ for some constant $c > 0$, then by a) there exists an algorithm that finds the longest path between any two vertices in time $\mathcal{O}(n^c + n^2) \leq \mathcal{O}(n^{c+2})$. So by b) there exists an algorithm that solves the Hamiltonian path problem in time $\mathcal{O}(n^2 \cdot n^{c+2}) = \mathcal{O}(n^{c+4})$, which is polynomial in $n$.