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, Ulysse Schaller

07. December 2020

Algorithms & Data Structures — Exercise sheet 12 — HS 20

Exercise Class (Room & TA): _____

Submitted by: _____

Peer Feedback by: _____

Points: ____

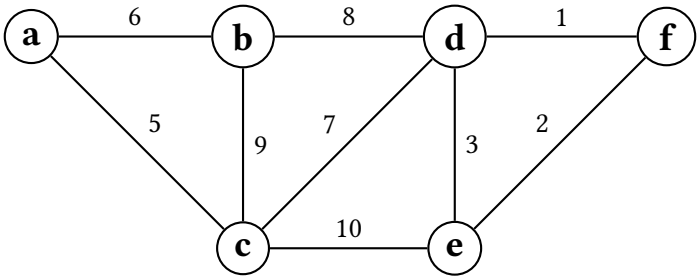
Submission: On Monday, 14. December 2020, hand in your solution to your TA *before* the exercise class starts. Exercises that are marked by * are challenge exercises. **Execeptionnally, they will also count toward bonus points.**

Remark. Let $G = (V, E)$ be a weighted graph with nonnegative weights ($w(e) \geq 0 \quad \forall e \in E$) such that all edge-weights are different ($\forall e \neq e'$ in E, $w(e) \neq w(e')$). Then the minimum spanning tree of G is unique.

You can use this fact without further justification for solving this exercise sheet.

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 $\{a, c\}, \{a, b\}, \{d, f\}, \{e, f\}$. At the second step we add $\{c, d\}$.

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

Solution: $\{d, f\}, \{e, f\}, \{a, c\}, \{a, b\}, \{c, d\}$.

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

Solution: $\{a, c\}, \{a, b\}, \{c, d\}, \{d, f\}, \{e, f\}$.

Exercise 12.2 *Ancient Kingdom of Macedon.*

The ancient Kingdom of Macedon had N cities and M roads connecting them, such that from one city, you can reach all other $N-1$ cities. All roads were *roman roads* i.e. stone-paved roads that did not require any maintenance and no two roads were of the same length. With the technological developments in the Roman Kingdom, a new type of carriage was developed, called the *Tesla Carriage*, which was much faster than all the alternatives in the Ancient Macedon Kingdom. However, the Tesla Carriage required *asphalt roads* to operate, and such roads had to be maintained every year, or otherwise the asphalt would wear off, rendering the road unusable as if it was a roman road.

With the effort to modernise the kingdom, Phillip II promised the Ancient Macedonians that he will provide them with asphalt roads by paving some of the existing roman roads, such that every two cities can be reached through a Tesla Carriage. The price to pave a roman road or maintain an asphalt road is equal, and is proportional to the length of the road. To save money Phillip II decided to pave sufficient roman roads to fulfil his promise, while minimizing the overall yearly maintenance price.

Even in the first years, the new Tesla Carriages improved the lives of the average Ancient Macedonians, but at the same time, they also provided means for robbers to commit crimes and escape to another city. To resolve this, Phillip II decided to create checkpoints the second year, one at each asphalt road. Each of the checkpoint will have a fixed cost for both building and maintenance.

Assuming a fixed price k for each checkpoint, does Phillip II have to consider paving new roman roads, or he can maintain the same set of roads in order to make sure that the overall maintenance price of the roads and the checkpoints is still minimal? Prove your reasoning, or provide a counter example.

Note: For simplicity, assume that the roman roads were paved all at once, on the first day of the year, and maintenance will be done the same day next year, again all at once. Also assume that checkpoints can also be built at once for all roads, as well a they can be maintained all at once in a day.

Solution.

Let's think of all cities in the Ancient Macedon Kingdom as vertices in a graph G and all roads as edges. In order to minimize the overall maintenance price, Philip II must pave only roman roads that from a minimum spanning tree T in G. As a result, we can rephrase the problem into a graph problem: Let T be a minimum spanning tree of a weighted graph G. Construct a new graph G' by adding a weight of k to every edge of G. Do the edges of T form a minimum spanning tree of G'?

In a graph with N vertices, every spanning tree has $N-1$ edges. Thus the weight of every spanning tree is increased by exactly $(N-1)\cdot k$. Therefore, the minimum spanning trees remains the same. In other words, Phillip II does not have to consider paving new roman roads: keeping the current asphalt roads and building a checkpoint on each of them still guarantees minimal maintenance cost.

*__Exercise 12.3__ *Spanning Forest with 2 components* ***(2 points)***.

Let $G=(V,E)$ be a connected edge-weighted graph in which all weights of the edges are different and positive. Consider the following two algorithms, which take G, the weights of all edges, and two different vertices $u,v\in V$ as input.

Algorithm 1

Run Kruskal's algorithm to get a minimum spanning tree $T = (V, E_T)$.
Find the unique path π from u to v in T.
Find the edge e of maximal weight among edges in π.
Remove e from T to get a graph $H = (V, E_T \setminus \{e\})$.
return H

Algorithm 2

$M \leftarrow \{u, v\}$
$E_M \leftarrow \emptyset$
while $M \neq V$ **do**
  $\Delta M = \{\{w_0, w_1\} \in E : w_0 \in M, w_1 \in V \setminus M\}$
  Find the edge $\{w_0, w_1\} \in \Delta M$ of minimal weight.
  $E_M \leftarrow E_M \cup \{\{w_0, w_1\}\}$
  $M \leftarrow M \cup \{w_1\}$
return $H = (M, E_M)$

Prove that the two algorithms return the same graph.

Hint: *Consider the graph G' which is obtained from G by adding an edge e_0 of weight 0 between u and v (if the edge $\{u, v\}$ already exists, then simply decrease its weight to 0). Try to relate the two given algorithms on G to algorithms that you know from the lecture on G'.*

Solution.

Note that G' is still a weighted graph with all edge-weights being different (since we assume that $w(e) > 0$ for all $e \in E$). Clearly the edges of G' all have nonnegative weights, so by the remark at the beginning of the sheet we know that the MST of G' is unique. In order to prove that the two algorithms return the same graph, we will show that they both yield the MST T' of G' minus the edge e_0 (note that since e_0 is the edge of minimal weight in G', it is clear that it is part of T').

Algorithm 1:

We claim that Algorithm 1 yields the same graph as running Kruskal's algorithm on G', minus e_0. Since e_0 has minimal weight in G', the first step of Kruskal's algorithm on G' is to add e_0 to T'. It remains to show that after this step, the only edge added to T in Algorithm 1 that is not added to T' is precisely e, and that every edge added to T' is also added to T. Let T_n denote the set of edges added to T by Kruskal's algorithm after n steps, and T'_n the set of edges added to T' by Kruskal's algorithm after $n + 1$ steps (i.e. after going through e_0 and n additional edges). We will show by induction that $T_n \setminus \{e\} = T'_n \setminus \{e_0\}$, where e denotes the edge of maximal weight on the path from u to v in T.

The base case $n = 0$ clearly holds since $T_0 \setminus \{e\} = \emptyset \setminus \{e\} = \emptyset$ and $T'_0 \setminus \{e_0\} = \{e_0\} \setminus \{e_0\} = \emptyset$. Let $n \in \mathbb{N}$ and suppose by induction that $T_n \setminus \{e\} = T'_n \setminus \{e_0\}$. We will prove that $T_{n+1} \setminus \{e\} = T'_{n+1} \setminus \{e_0\}$. Let f be the $n + 1$-th edge conmsidered (i.e. in this step Kruskal's algorithm needs to decide if it adds f to T_n and/or to T'_n).

Suppose first that f is not added to T_n. This means that $T_n \cup \{f\}$ contains a cycle. If this cycle doesn't contain the edge $e := ww'$, then clearly there is also a cycle in $T'_n \cup \{f\}$ and thus f is also not added to T'_n. If the cycle contains e this means in particular that $e \in T_n$. Since e is the edge of maximal weight on the path π from u to v in T and Kruskal's algorithm considers edges in order of increasing

weights, we must have $\pi \subset T_n$. Since $T_n \setminus \{e\} = T'_n \setminus \{e_0\}$, replacing e in the cycle by the path from w to u (on π), concatenated with e_0 and the path from v to w' (on π), we also obtain a closed walk in $T'_n \cup \{f\}$. Therefore f is also not added to T'_n. So in this case we have shown that indeed $T_{n+1} \setminus \{e\} = T_n \setminus \{e\} = T'_n \setminus \{e_0\} = T'_{n+1} \setminus \{e_0\}$.

Now suppose that f is added to T_n but not to T'_n. Since $T_n \setminus \{e\} = T'_n \setminus \{e_0\}$, this is only possible if the cycle in $T'_n \cup \{f\}$ contains e_0. In particular, there must be a path from u to v that uses f in

$$(T'_n \setminus \{e_0\}) \cup \{f\} = (T_n \setminus \{e\}) \cup \{f\} \subseteq T_n \cup \{f\}.$$

On the other hand, since f was added to T_n, there was no such path in T_n. This is only possible if f is the last edge on this path from u to v that was added by Kruskal's algorithm, i.e. if f has maximal weight on this path. Thus, we must have $f = e$, which implies

$$T_{n+1} \setminus \{e\} = (T_n \cup \{e\}) \setminus \{e\} = T_n \setminus \{e\} = T'_n \setminus \{e_0\} = T'_{n+1} \setminus \{e_0\}.$$

The only remaining case is when f is added both to T_n and to T'_n. But in this case it is clear that $T_n \setminus \{e\} = T'_n \setminus \{e_0\}$ implies $T_{n+1} \setminus \{e\} = T'_{n+1} \setminus \{e_0\}$, so the proof for Algorithm 1 is complete.

Algorithm 2:

Consider Prim's algorithm on the graph G' with starting vertex u. Since e_0 has minimal weight in the whole graph and is incident to u, this will be the first edge added by Prim's algorithm, creating a connected component $\{u, v\}$. After that, it is clear that Algorithm 2 and Prim's algorithm are doing the exact same operations (basically Algorithm 2 is simply Prim's algorithm in G' with the first step hidden in the initialization $M \leftarrow \{u, v\}$). So Algorithm 2 will also return T' minus the edge e_0. This concludes the whole proof.

Remark. Algorithm 1 and Algorithm 2 actually both return the optimal solution of the following problem. Given $u, v \in V$, find two trees $T_u = (V_u, E_u)$, $T_v = (V_v, E_v)$ that are subgraphs of G such that:

- they span the graph (i.e. $V_u \cup V_v = V$).
- $u \in V_u$ and $v \in V_v$.
- their total weight $\sum_{e \in E_u \cup E_v} w(e)$ is minimized subject to the above conditions.

Exercise 12.4 *Counting walks.*

Recall that a *walk of length* k in a graph $G = (V, E)$ is a sequence of vertices $v_0, v_1, \ldots, v_k$ such that for $1 \leq i \leq k$ there is an edge $\{v_{i-1}, v_i\} \in E$.

Consider the adjacency matrix representation of a graph $G = (V, E)$, with $V = \{v_1, \ldots, v_n\}$. The adjacency matrix A contains number of walks of length 1 (= edges), i.e., $A_{ij} = 1$ if $\{v_i, v_j\} \in E$ and $A_{ij} = 0$ otherwise. Similarly, A^k contains the number of walks of length k in its entries (see Exercise 9.5.).

a) Provide an efficient algorithm that given an adjacency matrix A and an integer $k \geq 1$ computes A^k using only matrix multiplications.

Solution:

Algorithm 3 Power(A, k)

```
if k = 1 then
    return A
else
    if k is odd then
        X ← Power(A, (k − 1)/2)
        return X · X · A
    else
        X ← Power(A, k/2)
        return X · X
```

b) Determine the number of matrix multiplications required by your algorithm in Θ notation. Justify your answer.

Solution: Let $T(k)$ be the number of matrix multiplications that the algorithm Power performs on input A, k. We are going to show that $T(k) \in \Theta(\log_2(k))$. In order to do so, we show that it is in $\mathcal{O}(\log_2(k))$ and in $\Omega(\log_2(k))$.

Let's prove by induction that $T(k) \leq 2\log_2 k$:

- **Base Case.**
  Let $k = 1$. Then $T(k) = 0 \leq 2\log_2 k$.
- **Induction Hypothesis.**
  Assume that the property holds for all positive integers $m < k$. That is, $T(m) \leq 2\log_2 m$.
- **Inductive Step.**
  We must show that the property holds for k. If k is even,

$$T(k) = T(k/2) + 1 \overset{\text{I.H.}}{\leq} 2\log_2(k/2) + 1 < 2\log_2 k.$$

  If k is odd,

$$T(k) = T((k-1)/2) + 2 \overset{\text{I.H.}}{\leq} 2\log_2((k-1)/2) + 2 < 2\log_2 k.$$

  By the principle of mathematical induction, this is true for any integer $k \geq 1$.

Thus, $T(k) \leq \mathcal{O}(\log_2(k))$. Let's prove by induction that $T(k) \geq \log_2 k$ for $k \geq 2$:

- **Base Case.**
  Let $k = 2$. Then $T(k) = 1 \geq \log_2 k$.
- **Induction Hypothesis.**
  Assume that the property holds for all positive integers $m < k$. That is, $T(m) \geq \log_2 m$.
- **Inductive Step.**
  We must show that the property holds for k. If k is even,

$$T(k) = T(k/2) + 1 \overset{\text{I.H.}}{\geq} \log_2(k/2) + 1 = \log_2 k.$$

  If k is odd,

$$T(k) = T((k-1)/2) + 2 \overset{\text{I.H.}}{\geq} \log_2((k-1)/2) + 2 = \log_2(2(k-1)) \geq \log_2 k, \text{ for } k \geq 2.$$

  By the principle of mathematical induction, this is true for any integer $k \geq 2$.

Thus, $T(k) \geq \Omega(\log_2(k))$ and $T(k) \in \Theta(\log_2(k))$.