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

02. December 2019

Algorithms & Data Structures — Exercise sheet 11 — HS 19

Exercise Class (Room & TA): ____________

Submitted by: ____________

Peer Feedback by: ____________

Points: ____

Submission: On Monday, 09. 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 11.1 *Shortest paths by hand* **(1 Point)**.

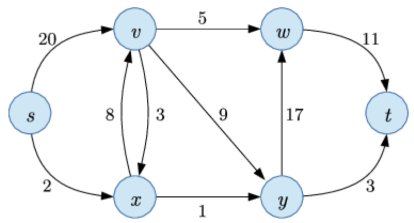
At the end of the lecture on November 28, we discussed an algorithm for finding shortests paths when all edge costs are nonnegative. Here is the pseudo-code for that algorithm, which is typically called Dijkstra's algorithm:

```
function DIJKSTRA(G, s)
    d[s] ← 0                                          ▷ upper bounds on distances from s
    d[v] ← ∞ for all v ≠ s
    S ← ∅                                             ▷ set of vertices with known distances
    while S ≠ V do
        choose v* ∈ V \ S with minimum upper bound d[v*]
        add v* to S
        update upper bounds for all v ∈ V \ S:
            d[v] ← min_{predecessor u∈S of v} d[u] + c(u, v)
            (if v has no predecessors in S, this minimum is ∞)
```

We remark that this version of Dijkstra's algorithm focuses on illustrating how the algorithm explores the graph and why it correctly computes all distances from s. You can use this version of Dijkstra's algorithm to solve exercises from this sheet.

In order to achieve the best possible running time, it is important to use an appropriate data structure for efficiently maintaining the upper bounds $d[v]$ with $v \in V \setminus S$, which will be discussed during the lecture on December 5. In the next sheets and the exam you should use the efficient version of the algorithm (not the pseudocode described above).

Consider the following weighted directed graph:



a) Execute the Dijkstra's algorithm described above by hand to find a shortest path from s to each node in the graph. After each step (i.e. after each choice of v^*), write down:

1) the upper bounds $d[u]$, for $u \in V$, between s and each node u computed so far,

2) the set M of all nodes for which the minimal distance has been correctly computed so far,

3) and the predecessor $p(u)$ for each node in M.

Solution: When we choose s: $d[s] = 0$, $d[x] = d[v] = d[w] = d[y] = d[t] = \infty$, $M = \{s\}$, there is no $p(s)$.

When we choose x: $d[s] = 0$, $d[x] = 2$, $d[v] = 20$, $d[w] = d[y] = d[t] = \infty$, $M = \{s, x\}$, there is no $p(s)$, $p(x) = s$.

When we choose y: $d[s] = 0$, $d[x] = 2$, $d[v] = 10$, $d[y] = 3$, $d[w] = d[t] = \infty$, $M = \{s, x, v, y\}$, there is no $p(s)$, $p(x) = s$, $p(v) = x$, $p(y) = x$.

When we choose t: $d[s] = 0$, $d[x] = 2$, $d[v] = 10$, $d[y] = 3$, $d[t] = 6$, $d[w] = 20$, $M = \{s, x, v, y, t\}$, there is no $p(s)$, $p(x) = s$, $p(v) = x$, $p(y) = x$, $p(t) = y$.

When we choose v: $d[s] = 0$, $d[x] = 2$, $d[v] = 10$, $d[y] = 3$, $d[t] = 6$, $d[w] = 20$, $M = \{s, x, v, y, t\}$, there is no $p(s)$, $p(x) = s$, $p(v) = x$, $p(y) = x$, $p(t) = y$.

When we choose w: $d[s] = 0$, $d[x] = 2$, $d[v] = 10$, $d[y] = 3$, $d[t] = 6$, $d[w] = 15$, $M = \{s, x, v, y, t, w\}$, there is no $p(s)$, $p(x) = s$, $p(v) = x$, $p(y) = x$, $p(t) = y$, $p(w) = v$.

b) Change the weight of the edge (x, y) from 1 to -1 and execute Dijkstra's algorithm on the new graph. Does the algorithm work correctly (are all distances computed correctly)? In case it breaks, where does it break?

Solution: The algorithm works correctly.

When we choose s: $d[s] = 0$, $d[x] = d[v] = d[w] = d[y] = d[t] = \infty$.

When we choose x: $d[s] = 0$, $d[x] = 2$, $d[v] = 20$, $d[w] = d[y] = d[t] = \infty$.

When we choose y: $d[s] = 0$, $d[x] = 2$, $d[v] = 10$, $d[y] = 1$, $d[w] = d[t] = \infty$.

When we choose t: $d[s] = 0$, $d[x] = 2$, $d[v] = 10$, $d[y] = 1$, $d[t] = 4$, $d[w] = 18$.

When we choose v: $d[s] = 0$, $d[x] = 2$, $d[v] = 10$, $d[y] = 1$, $d[t] = 4$, $d[w] = 18$.

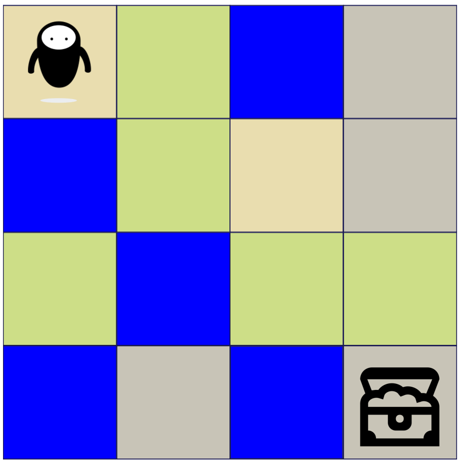
When we choose w: $d[s] = 0$, $d[x] = 2$, $d[v] = 10$, $d[y] = 1$, $d[t] = 4$, $d[w] = 15$.

c) Now, additionally change the weight of the edge (v, y) from 9 to -10. Show that in this case the algorithm doesn't work correctly, i.e. there exists some $u \in V$ such that $d[u]$ is not equal to a minimal distance from s to u after the execution of the algorithm.

Solution: The algorithm doesn't work correctly, for example, the distance from s to y is 0, but the algorithm computes exactly the same values of $d[\cdot]$ as in part b), so $d[y] = 1$.

Exercise 11.2 *Robot* **(1 Point).**

Consider the following game:



The game is played on a $d \times d$ board with four different types of fields (grassland, water, desert and mountain). You start at top left field and have to move to the bottom right field. At each turn you may move to an adjacent field (a field that shares a border with your current field). Moving through a grassland field requires 3 minutes, moving through a desert field requires 5 minutes, moving through a mountain field requires 7 minutes and you cannot swim (which makes water fields impassable). You spend zero time in the starting and target field. You may assume that there is always at least one possible way from top left to bottom right. The goal of this exercise is to find the fastest way from the top left field to the bottom right field.

a) Model the problem as a graph problem:

1) Describe your graph. What are the vertices, what are the edges and what are the weights on the edges?

   Solution: The graph $G = (V, E, w)$ is as follows: V is a set of non-water fields, and there are two directed edges (u, v) and (v, u) if and only if fields $u \in V$ and $v \in V$ share a border. The weight $w((u, v))$ of an edge $(u, v) \in E$ is the time that is required to move through the field $v \in V$ (so for a target field t and for each edge $(u, t) \in E$, $w((u, t)) = 0$).

2) What is the graph problem that we are trying to solve?

   Solution: Finding the shortest path between the starting field $s \in V$ and the target field $t \in V$ in the graph $G = (V, E, w)$.

3) Solve the problem using an algorithm discussed in the lecture (without modification).

   Solution: We can apply Dijkstra's algorithm to (G, s) to find the shortest path between s and t.

b)* Now, we modify the game a little bit: You learned how to swim and, thus, moving through a water field requires 11 minutes. However, swimming is very exhausting for you and you cannot swim through more than w water fields. Again, you may assume that the game boards are generated in a way that they are solvable using w of the water fields. Model the modified problem as a graph problem. Find a description as graph problem such that you can directly apply one of the algorithms in the lecture, without modifications to the algorithm.

1) Describe your graph. What are the vertices, what are the edges and what are the weights on the edges?

   Solution: The graph G is as follows: for each non-water non-target field a, V contains $w+1$ vertices $a_0, \dots a_w$, and for each water field b, V contains w vertices $b_1, \dots, b_w$. For a target field V contains one vertex t.

   If non-water fields a and a' share a border, then E contains directed edges (a_i, a'_i) and (a_i, a'_i) for all $i = 0, 1, \dots, w$.

   If the target field t shares a border with some non-water field a, then E contains directed edges (a_i, t) for all $i = 0, 1, \dots, w$. If the target field t shares a border with some water field b, then E contains directed edges (b_i, t) for all $i = 1, \dots, w$.

   If a non-water non-target field a and a water field b share a border, then E contains directed edges (a_i, b_{i+1}) and (b_i, a_i) for all $i = 0, 1, \dots, w$. If two water fields b and b' share a border, then E contains directed edges (b_j, b'_{j+1}) and (b'_j, b_{j+1}) for all $j = 1, \dots, w-1$.

   The weight of an edge $(u, v) \in E$ is the time that is required to move through the field $v \in V$ (so for a target field t and for each edge $(u, t) \in E$ its weight is 0).

2) What is the graph problem that we are trying to solve?

   Solution: Finding the shortest path between the vertex $s_0 \in V$ that corresponds to the starting field s and the target vertex $t \in V$ in the graph G.

3) Solve the problem using an algorithm discussed in the lecture (without modification).

   Solution: We can apply Dijkstra's algorithm to (G, s_0) to find the shortest path between s_0 and t.

Exercise 11.3 *Negative Cycles.*

Show that the Bellman-Ford algorithm can be used to find out whether negative cycles exist. Recall that for the analysis of the algorithm we considered values $d(s,v)^{\leq \ell}$, which is the minimum weight of an s-v walk with at most ℓ edges. Show that a graph contains a negative cycle that can be reached from s if and only if there exists a vertex v such that $d(s,v)^{\leq n-1} \neq d(s,v)^{\leq n}$, where n is the number of vertices.

Solution.

Let's denote $d(s,v)^{\leq n}$ by $T(v,n)$.

$\Leftarrow$ Assume there exists a node j such that $T(j, n-1) \neq T(j,n)$. Then $T(j,n) < T(j,n-1)$, and there is a walk with exactly n edges that attains $T(j,n)$. By a pigeonhole principle, we must have visited one vertex at least twice. The walk can be written as:

$$s \to \dots \to v \to \dots \to v \to \dots \to j.$$

The walk $v \to \dots \to v$ must form a negative cycle. Let $w_{s \to v}$ be the sum along the walk from s to v, $w_{v \to v}$ be the sum along the walk from the first time v is visited to the last time, and $w_{v \to j}$ be the sum along the walk from v to j. Then:

$$w_{s \to v} + w_{v \to v} + w_{v \to j} = T(j,n). \quad (1)$$

If we remove the cycle from the first time we visit v to the second, we obtain the walk $s \to \ldots \to v \to \ldots \to j$. This is a walk from s to j with at most $n-1$ edges [1], which gives us:

$$T(j, n-1) \leq w_{s \to v} + w_{v \to j}. \tag{2}$$

From (1), (2), and $T(j,n) < T(j,n-1)$, we obtain:

$$w_{s \to v} + w_{v \to v} + w_{v \to j} = T(j,n) < T(j,n-1) \leq w_{s \to v} + w_{v \to j} \quad \Rightarrow \quad w_{v \to v} < 0.$$

So the vertices visited along the walk from v to v form a negative cycle.

$\Rightarrow$ Assume $v_0 \to v_1 \to \ldots \to v_k$ is a cycle of negative weight, where $v_k = v_0$. Then, assume for the purposes of contradiction that

$$\forall v_i, i \in \{1, \ldots, n\}, T(v_i, n-1) = T(v_i, n)$$

We know that there is a walk from s to v_{i+1} with cost $T(v_i, n-1) + w(v_i, v_{i+1})$.

$$T(v_{i+1}, n) \leq T(v_i, n-1) + w(v_i, v_{i+1}).$$

$$T(v_{i+1}, n-1) \leq T(v_i, n-1) + w(v_i, v_{i+1}).$$

Sum over all vertices in the cycle:

$$\sum_{i=0}^{k-1} T(v_{i+1}, n-1) \leq \sum_{i=0}^{k-1} T(v_i, n-1) + \sum_{i=0}^{k-1} w(v_i, v_{i+1}).$$

$v_0 = v_k$, so $\sum_{i=0}^{k-1} T(v_{i+1}, n-1) = \sum_{i=0}^{k-1} T(v_i, n-1)$.

$$0 \leq \sum_{i=0}^{k-1} w(v_i, v_{i+1}).$$

This implies that $v_0 \to v_1 \to \cdots \to v_k$ is a cycle of positive weight, which is a contradiction. Therefore,

$$\exists v_i, i \in \{1, \ldots, n\}, T(v_i, n-1) \neq T(v_i, n).$$

Exercise 11.4 *Arbitrage* ***(1 Point).***

When trading currencies, *arbitrage* means to exploit price differences in order to profit by exchanging currencies multiple times. For example, on June 2nd, 2009, 1 US Dollar could be exchanged for 95.729 Yen, 1 Yen for 0.00638 Pound sterling, and 1 Pound sterling for 1.65133 US Dollars. If a trader exchanged 1 US Dollar for Yen, exchanged the obtained amount for Pound sterling and finally exchanged this amount back to US Dollars, he would have obtained $95.729 \cdot 0.00638 \cdot 1.65133 \approx 1.0086$ US Dollars, corresponding to a gain of 0.86%.

1. You are given n currencies $\{1, \ldots, n\}$ and an $(n \times n)$ exchange rate matrix R with positive rational number entries. For two currencies $i, j \in \{1, \ldots, n\}$ one unit of currency i can be exchanged for $R(i,j) > 0$ units of currency j. The goal is to decide whether an arbitrage opportunity exists, i.e., if there exists a sequence of k different currencies $W_1, \ldots, W_k \in \{1, \ldots, n\}$ such that $R(W_1, W_2) \cdot R(W_2, W_3) \cdots R(W_{k-1}, W_k) \cdot R(W_k, W_1) > 1$ holds.

[1] In a graph with no loops, it will have strictly fewer than $n-1$ edges.

Model the above problem as a graph problem. Show how the input can be transformed into a directed, weighted graph $G = (V, E, w)$ that contains a cycle with negative weight *if and only if* an arbitrage activity is possible. Justify why G contains a negative cycle if and only if an arbitrage opportunity exists.

Hint: *Using logarithms might be beneficial because of the property* $\ln(a \cdot b) = \ln(a) + \ln(b)$.

Solution. We create a complete graph $G = (V, E)$ with the vertices $V = \{1, \ldots, n\}$. An edge $(i, j) \in E$ gets the weight $w(i, j) = -\log R(i, j)$. Then suppose an arbitrage opportunity with the sequence of currencies $W_1, \ldots, W_k$ is possible. Then it must be the case that

$$
\begin{aligned}
& R(W_1, W_2) \cdot R(W_2, W_3) \cdot \ldots \cdot R(W_{k-1}, W_k) R(W_k, W_1) > 1 \\
\Leftrightarrow\ & \log\left(R(W_1, W_2) \cdot R(W_2, W_3) \cdot \ldots \cdot R(W_{k-1}, W_k) R(W_k, W_1)\right) > 0 \\
\Leftrightarrow\ & \log R(W_1, W_2) + \log R(W_2, W_3) + \ldots + \log R(W_{k-1}, W_k) + \log R(W_k, W_1) > 0 \\
\Leftrightarrow\ & -\log R(W_1, W_2) - \log R(W_2, W_3) - \ldots - \log R(W_{k-1}, W_k) - \log R(W_k, W_1) < 0 \\
\Leftrightarrow\ & w(W_1, W_2) + w(W_2, W_3) + \ldots + w(W_{k-1}, W_k) + w(W_k, W_1) < 0
\end{aligned}
$$

consequently G contains a cycle of negative weight. Because we only used equivalence transformations, the argument applies in both directions.

2. Use the previous part of this exercise and the criterion from exercise 11.3 to design an algorithm that decides if an arbitrage opportunity exists. What is the best running time you can get (in terms of n)?

   Solution. Since the graph is complete, any negatively-weighted cycle can be reached from any vertex, we can run Bellman-Ford to detect a cycle starting from any vertex in G. There are $|V| = n$ vertices and $|E| = n(n-1) \in \Theta(n^2)$ edges. The algorithm therefore has a running time of $\Theta(|V||E|) = \Theta(n^3)$.