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

3. Dezember 2018

Algorithms & Data Structures — Homework 11 — HS 18

Exercise Class (Room & TA): _____

Submitted by: _____

Peer Feedback by: _____

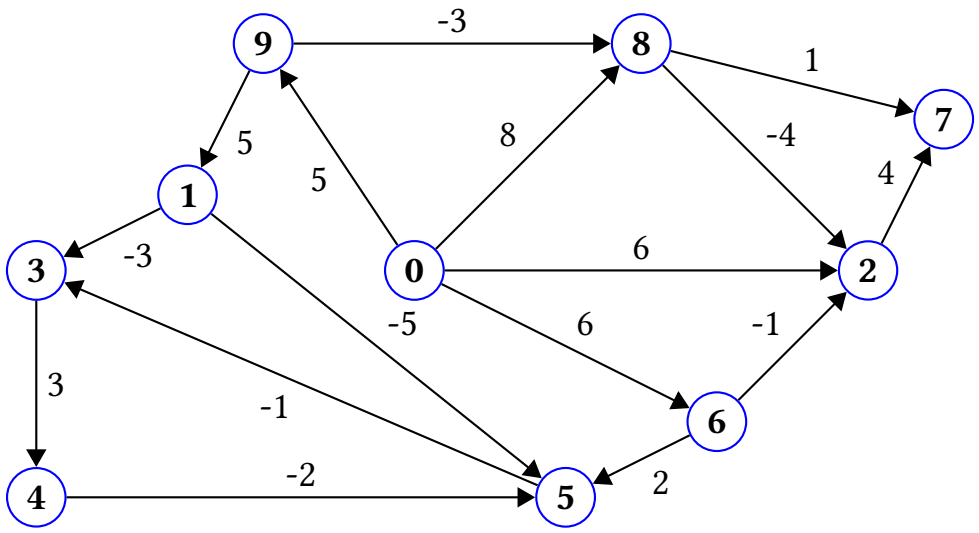
Points: _____

Hint: This exercise sheet is concerned with *dynamic programming*. A complete description of a dynamic program always includes the following aspects (important also for the exam!):

1. **Definition of the DP table:** What are the dimensions of the dynamic programming table DP[., .]? What is the meaning of each entry (in clearly worded words)?
2. **Calculation of an entry:** Which values of the table are initialized, and how are they initialized? How are entries calculated from other entries? What are the dependencies between entries?
3. **Calculation order:** In what order can you calculate the entries so that these dependencies are fulfilled?
4. **Reading the solution:** How can the solution be read out from the table at the end?

Exercise 11.1 *Bellman-Ford.*

Consider the following graph $G = (V, E)$ with edge weights $w\colon E \to \mathbb{R}$. Execute the Bellman–Ford algorithm on this graph with source node $s = 0$. Compute the values $T(j, \ell)$ for all $j \in V$ and all $\ell \in \{0, 1, \ldots, n-1\}$ as described in class. Use arrows to indicate for every entry $T(j, \ell)$ computed via the recurrence equation which entry in the previous row determined the value of $T(j, \ell)$.



Solution.

$$d[0] = 0, \quad d[1] = 10, \quad d[2] = -2, \quad d[3] = 4, \quad d[4] = 7,$$
$$d[5] = 5, \quad d[6] = 6, \quad d[7] = 2, \quad d[8] = 2, \quad d[9] = 5.$$

$$p[0] = \textbf{null}, \quad p[1] = 9, \quad p[2] = 8, \quad p[3] = 5, \quad p[4] = 3,$$
$$p[5] = 1, \quad p[6] = 0, \quad p[7] = 2, \quad p[8] = 9, \quad p[9] = 0.$$

Exercise 11.2 *Arbitrage* **(1 Point for 1)**.

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 $R \in (\mathbb{Q}^+)^2$. 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.

   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.

   Notice: 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. Show that the Bellman-Ford algorithm can be used to detect negative cycles. Recall that this algorithm computes values $T(j, \ell)$, the minimum weight of an s-j walk with at most ℓ edges. Show that a graph contains a negative cycle reachable from s if and only if there exists a node j such that $T(j, n-1) \neq T(j, n)$, where n is the number of vertices. (In class, we only argued about the "if" direction but not the "only if" direction.)

   Solution.

   $\Leftarrow$ 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 path with exactly n edges that attains $T(j, n)$. A path with n edges visits at least $n+1$ vertices, so we must have visited one vertex at least twice. The path can be written as:

   $$s \to \ldots \to v \to \ldots \to v \to \ldots \to j$$

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

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

   If we remove the cycle from the first time we visit v to the second, we obtain the path $s \to \ldots \to v \to \ldots \to j$. This is a path 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:

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

   $$\Rightarrow w_{v \to v} < 0$$

   So the vertices visisted along the path 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 path from s to v_{i+1} with cost $T(v_i, n-1) + w(v_i, v_{i+1})$.

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

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

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

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,

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

3. Use the previous two parts of this exercise 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 $\Theta(|V||E|) = \Theta(n^3)$.

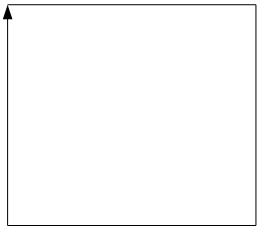
Exercise 11.3 *Side Gig* **(2 Points)**.

To earn cash, you decide to take a second job as driver for a ride hailing service. As a driver, your job is to drive from location to location, picking up passengers and dropping them off.

You decide to use the following rules to model the problem:

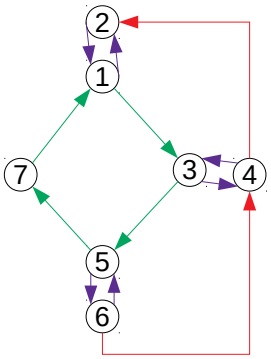
- In order to pick up or drop off a passenger at a location, you must be on the same block and the same side of the street as the location. (Being on the same block as the location means that you and the location are both located on the same street, between two adjacent intersections)
- You drive from intersection to intersection. At an intersection you can:
  - Go straight. This takes 2 minutes.
  - Turn right. This takes 1 minute.
  - Turn left. This takes 3 minutes.
  - Take a U-turn. This reverses your direction, takes you to the opposite side of the street, and takes 4 minutes.
- You drive on the right side of the road.
- There are one-way streets, and you cannot turn or go straight onto a one way street going the wrong way. It takes 0 minutes to get from one side of a one-way street to the other, since you don't have to take a U-turn.

1. Model the problem using a graph and design a solution that allows you to find the shortest path from location to location. Draw the resulting graph for the following street map.



Solution.

We model the problem with a directed graph, where a vertex represents a city block travelling in one direction. Each vertex may have up to four edges, since one may go straight, turn left, turn right, or take a U-turn. The edge connects each vertex to the vertex representing the side of city block that the driver will driving on after turning.



We can use Floyd-Warshall to find the shortest path between any pair of vertices. It takes $\mathcal{O}(n^3)$, where n is the number of streets. Then the driver finds the shortest path between their current location and the location of the next pick up or drop off point.

Since the driver is allowed to pick up or drop the passenger off on either side of the street, they should do two shortest path lookups to the destination: one for either side of the street. Then they should pick the shorter of the two resulting paths. (But we also allow answers that assume the driver must pick up and drop off passengers on the same side of the street as the pick up or drop off location.)

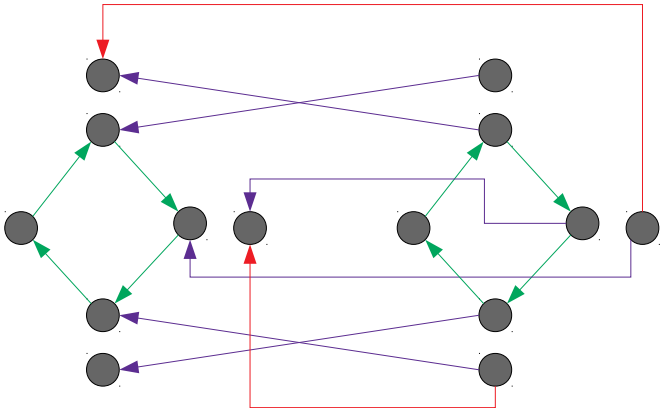
2. When turning left or taking a U-turn, you have to wait at the intersection for a long time, wasting gasoline and emitting greenhouse gases. Therefore, you have decided that you will take at most L such turns during each trip, where $L \geq 1$.

   Model the problem using a graph and design a solution that allows you to find the shortest path from location to location.

   Solution. We can model the problem with a directed graph of $L + 1$ layers, where each layer models the street map. There are two vertices for each city block–one for each direction. Each vertex has up to two connections within a layer, representing travelling straight or turning right. Each vertex in layer k, $k < L$ has one or two connections to vertices in layer $k + 1$, representing taking a U-turn or a left turn. Thus, a vertex in layer $k, k \in 0, \ldots, L$, represents that the car is driving on a certain side of a city block, and has already taken k left and U turns. Once you have taken L left or U turns, it is not possible to take any more, since there are no edges representing left or U-turns originating in layer L.

   Then, you can use an algorithm for shortest paths Bellman-Ford, or Floyd-Warshall.

   As an example, the graph that solves the problem that results from the street map above when $L = 1$ is as follows. Green lines represent right turns, red lines represent left turns, and violet lines represent U-turns.



Submission: On Monday, 10.12.2018, hand in your solution to your TA *before* the exercise class starts.