



Departement of Computer Science

6. December 2021

Markus Püschel, David Steurer

Gleb Novikov, Tommaso d'Orsi, Ulysse Schaller, Rajai Nasser

Algorithms & Data Structures

Exercise sheet 11

HS 21

Exercise Class (Room & TA): _____

Submitted by: _____

Peer Feedback by: _____

Points: _____

Submission: On Monday, 13. December 2021, 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 (2 points).*

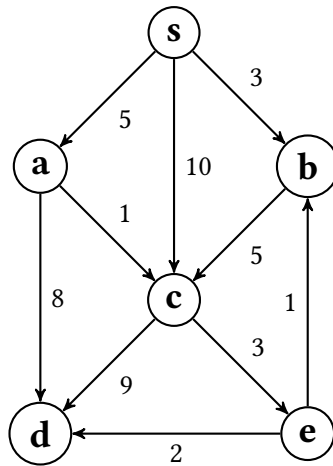
Dijkstra's algorithm allows to find shortest paths in a directed graph when all edge costs are nonnegative. Here is a pseudo-code for that algorithm:

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

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 this exercise.

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 $v \in V \setminus S$, as you saw in the lecture on December 2. In the other exercises/sheets and in the exam you should use the running time of the efficient version of the algorithm (and not the running time of 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 .
- b) Change the weight of the edge (a, c) 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?
- c) Now, additionally change the weight of the edge (e, b) from 1 to -6 (so edges (a, c) and (e, b) now have negative weights). 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.

Exercise 11.2 Arbitrage.

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%.

- a) You are given n currencies $\{1, \dots, n\}$ and an $(n \times n)$ exchange rate matrix R with positive rational number entries. For two currencies $i, j \in \{1, \dots, n\}$ 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 $W_1, \dots, W_k \in \{1, \dots, n\}$ 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.

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

- b) The Bellman-Ford algorithm can be used to find out whether a graph contains negative cycles. After ℓ iterations of the Bellman-Ford loop $d[v]$ is equal to $d(s, v)^{\leq \ell}$, i.e., the minimum weight of a s - v walk with at most ℓ edges. 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.

Use the previous part 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)?

Exercise 11.3 *Single-source shortest allowable paths (1 point).*

Consider a weighted directed graph $G = (V, E)$ that is given in the adjacency-list representation. The graph G has n vertices and m edges. The weights $w : E \rightarrow \mathbb{R}$ of the edges are not necessarily positive, but the graph does not contain any negative weight cycle. Every vertex is colored either red or green, so that the vertex set is partitioned as $V = V_{red} \cup V_{green}$. A path is said to be *allowable* if it contains at most one red vertex.

We are given a source $s \in V$ and would like to find the weights of the *shortest allowable paths* from s to all vertices in V . That is, we are interested in $(\delta_a(s, v))_{v \in V}$, where $\delta_a(s, v)$ is the weight of the shortest allowable path from s to v . If v is not reachable from s by an allowable path, then $\delta_a(s, v) = \infty$.

Describe an algorithm that can efficiently compute $(\delta_a(s, v))_{v \in V}$. In order to get full points, the runtime of your algorithm should be $O(mn)$.

Hint: Construct a weighted directed graph G' with $2n$ vertices and apply an algorithm that you learned in class on it.

Exercise 11.4 *Traveling in Examistan (This exercise is from the August 2020 exam).*

Assume that there are n towns T_1, \dots, T_n in the country Examistan. For each pair of distinct towns T_i and T_j , there is exactly one road from T_i to T_j . All of the roads in Examistan are one-way. This implies that there is always a road from T_i to T_j and another road from T_j to T_i . Each road has a nonnegative integer cost that you need to pay to use this road.

For simplicity you can assume that each town T_i is represented by its index i .

- a) Model the n towns, the roads and their costs as a directed weighted graph: give a precise description of the vertices, edges and the weights of the edges of the graph $G = (V, E, w)$ involved (if possible, in words and not formal). What are $|V|$ and $|E|$ in terms of n ?

In the following subtask b), you can assume that the directed graph in a) is represented by a data structure that allows you to traverse the direct successors and direct predecessors of a vertex u in time $\mathcal{O}(\deg_+(u))$ and $\mathcal{O}(\deg_-(u))$ respectively, where $\deg_-(u)$ is the in-degree of vertex u and $\deg_+(u)$ is the out-degree of vertex u .

- b) Due to the epidemiological situation in Examistan, the authorities decided to reduce the number of trips between different towns. Now the only way to get from one town to another is to use the roads. Moreover, if you want to travel from town T_i to the other town T_j , you must visit a test center during your trip (in T_i or T_j or elsewhere with a detour). Since test centers are expensive, there are only $k < n$ of them, and they are located only in the first k towns T_1, \dots, T_k (i.e., one test center in each of these towns). To compensate for this, the authorities make the roads between all different T_i and T_j among T_1, \dots, T_k free of charge (i.e. their cost is now 0).

Provide an as efficient as possible algorithm that takes as input a graph G from task a) and a number k , and outputs a table C such that $C[i][j]$ is the minimal total cost of roads that one should use to get from T_i to T_j with visiting a test center.

What is the running time of your algorithm in concise Θ -notation in terms of n and k ? Justify your answer.