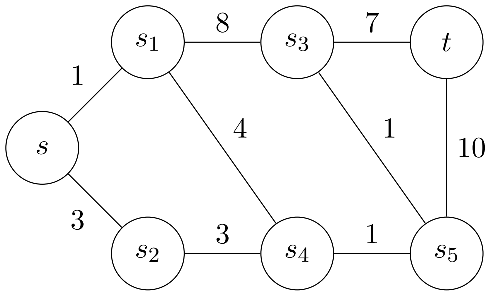
Technische Universität München
Institut für Informatik
Theoretical Computer Science

WS 2017/18

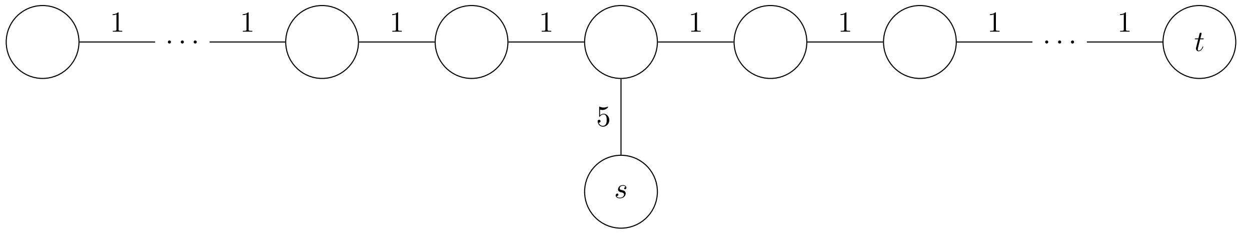
Fundamental Algorithms 12 - Solution Examples

Exercise 1 (Dijkstra)

Apply Dijkstra's algorithm to the following graph to find the shortest path (and its cost) from s to t. Write down all intermediate steps.



Now, instead consider the following graph. The graph is a (slightly simplified) model of a highway which can be accessed from s and leads, amongst other destinations, to t. Think about what happens when you run Dijkstra on this graph and possible ideas how to improve it, given that the nodes represent locations.



Solution:
State of the queue Q:

- $(s_1, 1), (s_2, 3)$
- $(s_2, 3), (s_4, 5), (s_3, 9)$
- $(s_4, 5), (s_3, 9)$
- $(s_5, 6), (s_3, 9)$
- $(s_3, 7), (t, 16)$
- $(t, 14)$

Shortest path: s, s_1, s_4, s_5, s_3, t; Cost: 14.

In the second case, Dijkstra alternates between the left and right direction. In general, there is no better way do deal with this, but, since the graph corresponds to a map and thus we have a notion of distance between nodes, we can use this information to guide our search. The idea is to prefer nodes n which have small $c(n) + d(n, t)$, where c is the cost to reach node n (from Dijkstra) and $d(n, t)$ is the distance between n and t, giving us a lower bound on the actual reach. This idea is known as the A^*-algorithm (A-star).

Exercise 2 (HyperDijsktra)

Think about how to define weighted Hypergraphs and how to tweak Dijkstra's Algorithm to work with this generalization.

Solution:
Given a hypergraph (V, E), where $E \subseteq 2^V \setminus \emptyset$, we define a weight function as $c : E \to \mathbb{Q}$, assigning a cost to each hyperedge. Intutively, we pay cost $c(e)$ whenever we go from any $v_1 \in e$ to any $v_2 \in e$. Dijkstra can trivially be adapted to this notion, since we actually only need to the successors of a node and the respective costs to reach them.

Exercise 3 (Good Flow)

Suppose you are a logistics engineer for a big manufacturing company. This company assembles its product in a big facility in n different manufacturing steps, for example *preprocessing*, *assembling*, and *packaging* $(n = 3)$. The facility is organized as follows. For each processing step i, there are m_i workstations, denoted $w_1^i, \dots, w_{m_i}^i$. Each workstation w_j^i can process c_{ij} items per day. Further, between the workstations of each step, there is a network of conveyor belts, all of which also have a capacity, specified by a weighted graph. This means that the network between step i and step $i + 1$ has the workstations of either steps, i.e. w_j^i and w_j^{i+1}, as nodes and, potentially, some intermediate conveyor junctions.

Think about how you can apply concepts of the lecture to optimize the productivity of the company, i.e. maximize the output of the workstations in the final step.

Solution:
One possible approach to solve this goes as follows. We reduce the problem to an instance of MAXFLOW. Observe that the problem description gives us a weighted graph (with both node and edge weights) by simply concatenating the stages accordingly. We add a super-source s and super-sink t and connect it to the first and last workstations, using edges of infinite capacity. Now, we need to take care of the node weights as follows. We replace each weighted node, i.e. the workstations, with two nodes, which represent the inflow and outflow of this workstation. These two nodes are then connected via a single edge with capacity equal to the one of the workstation. This edge represents the work-capacity of this station. By applying MAXFLOW on this weighted graph between s and t, we get the maximal production capacity and, as a by-product, the bottlenecks of the facility.