Technische Universität München
Fakultät für Informatik
Lehrstuhl für Effiziente Algorithmen
Prof. Dr. Ernst W. Mayr
Chris Pinkau, Stefan Toman, Jeremias Weihmann

Sommersemster 2013
Aufgabenblatt 3
29.04.2013

Praktikum Diskrete Optimierung

Due date: Monday, 6th May 2012, 14:00

Aufgabe 1 (Shortest paths: the Dijkstra algorithm `dijkstra`)

Given is a directed graph $G = (V, E)$ with positive edge weights. Every node is reachable by all other nodes in G. Implement and animate the Dijkstra algorithm, such that the shortest paths from a node of your choice to all other nodes are computed and displayed. Each node's label should be the length of the computed path to this node.

While the algorithm is running, at every time, the alredy processed nodes and the nodes in the Priority Queue should be marked in terms of colors. The already found shortest paths should be identifiable as well. For every node in the Priority Queue, you should additionally mark the edge over which the currently shortest path from an already processed node to this node is passing.

Aufgabe 2 (Shortest paths: the Bellman-Ford algorithm `bellman`)

Given is a directed graph $G = (V, E)$ with edge weights $c : E \to \mathbb{R}$, which can be negative. Implement the Bellman-Ford algorithm, which computes the shortest paths from a node of your choice to all reachable nodes in time $O(|V| \cdot |E|)$. If the queue isn't empty after $|V|$ phases, the algorithm signals that the graph contains a negative cycle, and computes and marks such a cycle efficiently in time $O(|V|)$.

Animate the algorithm such that the user is able to follow the course of events and such that at every time the provisionally shortest paths, that are already computed by the algorithm, as well as the nodes in the queue are displayed. Moreover, at every time, the animation should display the phase in which the algorithm is. After the algorithm terminated, the complete shortest-paths-tree should be displayed.

Hints

You may test the algorithm in exercise 1 with the graphs `pos1.gw` and `pos2.gw`. As input for exercise 2 you may use the graphs `neg1.gw` to `neg5.gw`. Graphs `neg3` to `neg5` contain negative cycles. The edge weights are stored as strings in the user-label and should be read into an `edge_array<double>`.

Note that the processed graphs are directed. Thus, your programs should contain `gw.set_directed(true)` and *not* `g.make_undirected()`.