Albert-Ludwigs-Universität
Institut für Informatik
Prof. Dr. F. Kuhn
M. Fuchs, G. Schmid

Algorithmen und Datenstrukturen
Sommersemester 2026
Musterlösung Übungsblatt 10

Abgabe: Dienstag, 7. Juli, 2026, 10:00 Uhr

Aufgabe 1: Dijkstra *(14 Punkte)*

In dieser Aufgabe geht es darum, den Dijkstra-Algorithmus aus der Vorlesung zu implementieren. Im Folgenden ist eine Graustufenkarte wie die folgende gegeben:

Diese besteht aus einem Gitter unterschiedlicher Grauschattierungen. Wir wollen diese Karte nun als Graph auffassen. Hier repräsentiert jedes Quadrat einen Knoten und ist mit jedem seiner (bis zu 4) Nachbarn durch eine gerichtete Kante verbunden. Das Kantengewicht hängt vom Grauton des Nachbarn ab, je heller der Grauton, desto weniger Gewicht hat die Kante zu diesem Nachbarn (und ist der Nachbar ein schwarzer Knoten, ist er nicht erreichbar, hat also Gewicht ∞). In `dijkstra_greyscale_map_large.txt` ist dieser Graph als Textfile gegeben, dieser kann mittels `graph.py` in einen Graphen/eine Adjazenzliste umgewandelt werden[1]. Der Graph in der Textdatei besteht aus der ersten Zeile, welche die Größe des Graphen angibt, jede weitere Zeile zeigt die x- und y-Koordinate eines Knotens zusammen mit seinem Grauwert (nur wichtig für das Plotten, nicht für Dijkstra selbst) und den entsprechenden Gewichten für die Konstruktion der Kanten. Implementieren Sie Dijkstra aus der Vorlesung in die Vorlage `shortest_path.py`. Was ist der kürzeste Pfad zwischen dem Knoten links oben $(0, 0)$ und rechts unten $(59, 32)$? Schreiben Sie Ihre Lösung in die PDF. Beachten Sie außerdem folgende Punkte:

- Um einigermaßen vernünftig testen zu können, ist die Datei `dijkstra_greyscale_map_test.txt` beigelegt. Die Tests in der Vorlage `shortest_path` sollten hierfür durchlaufen. Weiter unten ist auch eine Visualisierung dieser Test-Karte mit dem kürzesten Weg zu finden.

[1] Wie man den Graph einlesen kann können Sie in der `main` von `gridToImg.py` sehen.

- Für die Prioritätswarteschlange nutzen wir `PriorityQueue` von Python. Diese unterstützt offiziell nicht die `decrease-key`-Operation wie in der Vorlesung. Das ist aber kein Problem, denn anstatt die Distanz zu einem Knoten zu updaten, können Sie auch einfach den Knoten zusammen mit der neuen Distanz nochmal in die `PriorityQueue` hinzufügen.
- Falls Sie Ihre Lösung plotten wollen (sinnvoll zum testen), können Sie auch dazu die Funktion aus der Vorlage `gridToImg.py` verwenden.

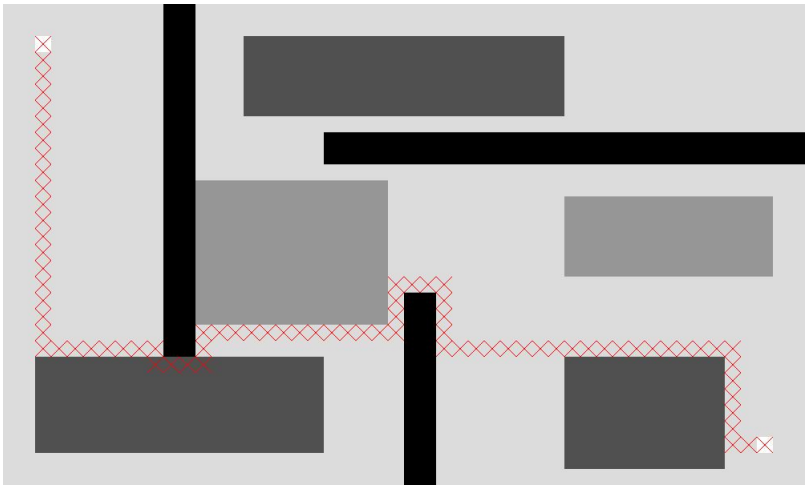
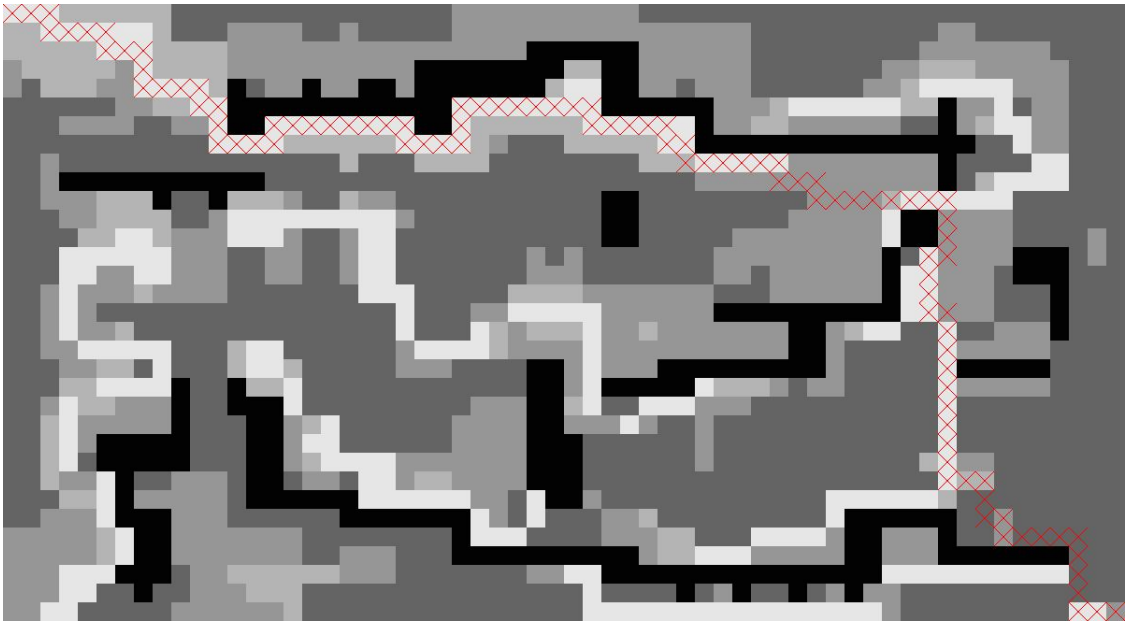


Figure 1: Ein kürzester Weg auf dem *Testgraphen*

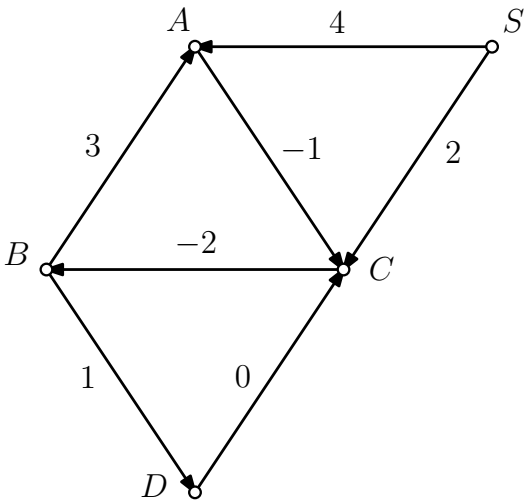
Musterlösung

Die Distanz ist 217 und der Weg sieht wie folgt aus:



Aufgabe 2: Bellman-Ford *(6 Punkte)*

Gegeben sei der folgende gerichtete Graph mit $n = 5$ Knoten als Schaubild.



(a) Hat dieser Graph einen negativen Zyklus? Falls ja, markieren Sie die Kanten von einem negativen Zyklus (Umkreisung oder Färbung der Kanten). *(1 Punkt)*

(b) Führen Sie auf diesem Graphen den *Bellman-Ford* Algorithmus mit Startknoten S aus und tragen Sie *nach jeder* Iteration der äußeren Schleife die bisher berechneten Distanzen von S *in eine Tabelle* (siehe Muster weiter unten) ein: *(5 Punkte)*

Wichtig: Iterieren Sie in der inneren Schleife des Bellman-Ford Algorithmus die Kanten aufsteigend nach Größe der Gewichte, d.h., stets in der Reihenfolge $-2, -1, 0, 1, 2, 3, 4$ *!*

	$\delta(S,S)$	$\delta(S,A)$	$\delta(S,B)$	$\delta(S,C)$	$\delta(S,D)$
Initial	0	∞	∞	∞	∞
$i=1$					
$i=2$					
$i=3$					
$i=4$					

Musterlösung

(a) Die Kanten $(C,B),(B,D),(D,C)$ bilden einen negativen Zyklus.

(b) Lösungstabelle:

	$\delta(S,S)$	$\delta(S,A)$	$\delta(S,B)$	$\delta(S,C)$	$\delta(S,D)$
Initial	0	∞	∞	∞	∞
$i=1$	0	4	∞	2	∞
$i=2$	0	3	0	2	1
$i=3$	0	3	0	1	1
$i=4$	0	2	-1	1	0