Albert-Ludwigs-Universität
Institut für Informatik
Prof. Dr. F. Kuhn
P. Bamberger
P. Schneider

Algorithmen und Datenstrukturen
Sommersemester 2020
Musterlösung Übungsblatt 10

Abgabe: Mittwoch, 22.07.2020, 16:00 Uhr.

Aufgabe 1: Dijkstra's Algorithmus *(10 Punkte)*

Gegeben sei ein Labyrinth dessen Kantenmenge eine Teilmenge eines $n \times n$ Gitters ist. D.h., jeder Gitterknoten hat maximal vier Kanten, jeweils höchstens zwei in horizontaler und in vertikaler Richtung. Eine horizontale Kante hat Kantengewicht 2, eine vertikale Kante hat Kantengewicht 1.
Wir nummerieren die n^2 Gitterknoten zeilenweise durch. Das Labyrinth sei gegeben durch eine Adjazenzliste A. Eintrag $A[i]$ enthält Tupel der Form $(j, w(i,j))$, wobei j ein horizontal oder vertikal benachbarter Knoten und $w(i,j) \in \{1,2\}$ das Gewicht der Kante $\{i,j\}$ darstellt.

(a) Implementieren Sie einen Algorithmus der für eine solche Adjazenzliste und zwei Gitterknoten $s, t \in \{0, \dots, n^2-1\}$ den kürzesten Pfad von s zu t im Labyrinth als Folge besuchter Gitterknoten in $\mathcal{O}(n^2 \log n)$ Zeit ausgibt. Sie dürfen die Vorlage `Maze.py` sowie zuvor bereits verwendete Hilfsdatenstrukturen benutzen. Erklären Sie in Ihren `erfahrungen.txt` kurz warum Ihr Algorithmus die geforderte Laufzeit einhält.

(b) Wenden Sie Ihren Algorithmus auf die Adjazenzliste die als Textdatei in `maze.txt` gegeben ist und $s = 0$, $t = 899$ an. Eine entsprechende Funktion um `maze.txt` einzulesen steht in `Maze.py` zur Verfügung. Wenden Sie die Funktion `visualize_path` auf Ihren Ergebnispfad an, schreiben Sie die Ausgabe in eine Datei `solution_path.txt` und geben Sie diese Datei mit ab.

Musterlösung

(a) Siehe `Maze.py` in unserer Musterlösung. Dijkstra mit einem Min-Heap als Priority-Queue hat eine Laufzeit von $\mathcal{O}(n' + m \log n')$ wobei n' der Anzahl der Knoten im Labyrinth entspricht. Da die Knoten im Labyrinth einem $n \times n$ Gitter entsprechen ist $n' \in \mathcal{O}(n^2)$. Außerdem hat jeder Knoten maximal 4 inzidente Kanten im Gitter, und damit gilt $m \leq 4n' \in \mathcal{O}(n') = \mathcal{O}(n^2)$. Damit hat Dijkstra im Labyrinth die Laufzeit $\mathcal{O}(n' + m \log n') = \mathcal{O}(n^2 + n^2 \log n^2) = \mathcal{O}(n^2 \log n)$.

(b) Siehe Abbildung 1 oder `maze_viz.txt`.

Aufgabe 2: Währungsarbitrage *(10 Punkte)*

Gegeben seien n Währungen $w_1, \dots, w_n$. Die Umrechungskurse der Währungen seien in einer $n \times n$-Matrix A mit Einträgen a_{ij} $(i, j \in \{1, \dots, n\})$ gegeben. Eintrag a_{ij} ist der Umrechnungskurs von w_i nach w_j, d.h. für eine Einheit von w_i bekommt man a_{ij} Einheiten von w_j.
Gegeben eine Währung w_{i_0} möchte man herausfinden, ob es eine Folge $i_0, i_1, \dots, i_k$ gibt, sodass man Gewinn macht, wenn man eine Einheit von w_{i_0} zu w_{i_1} tauscht, danach zu w_{i_2} etc. bis zu w_{i_k} und schließlich wieder zurück zu w_{i_0}.

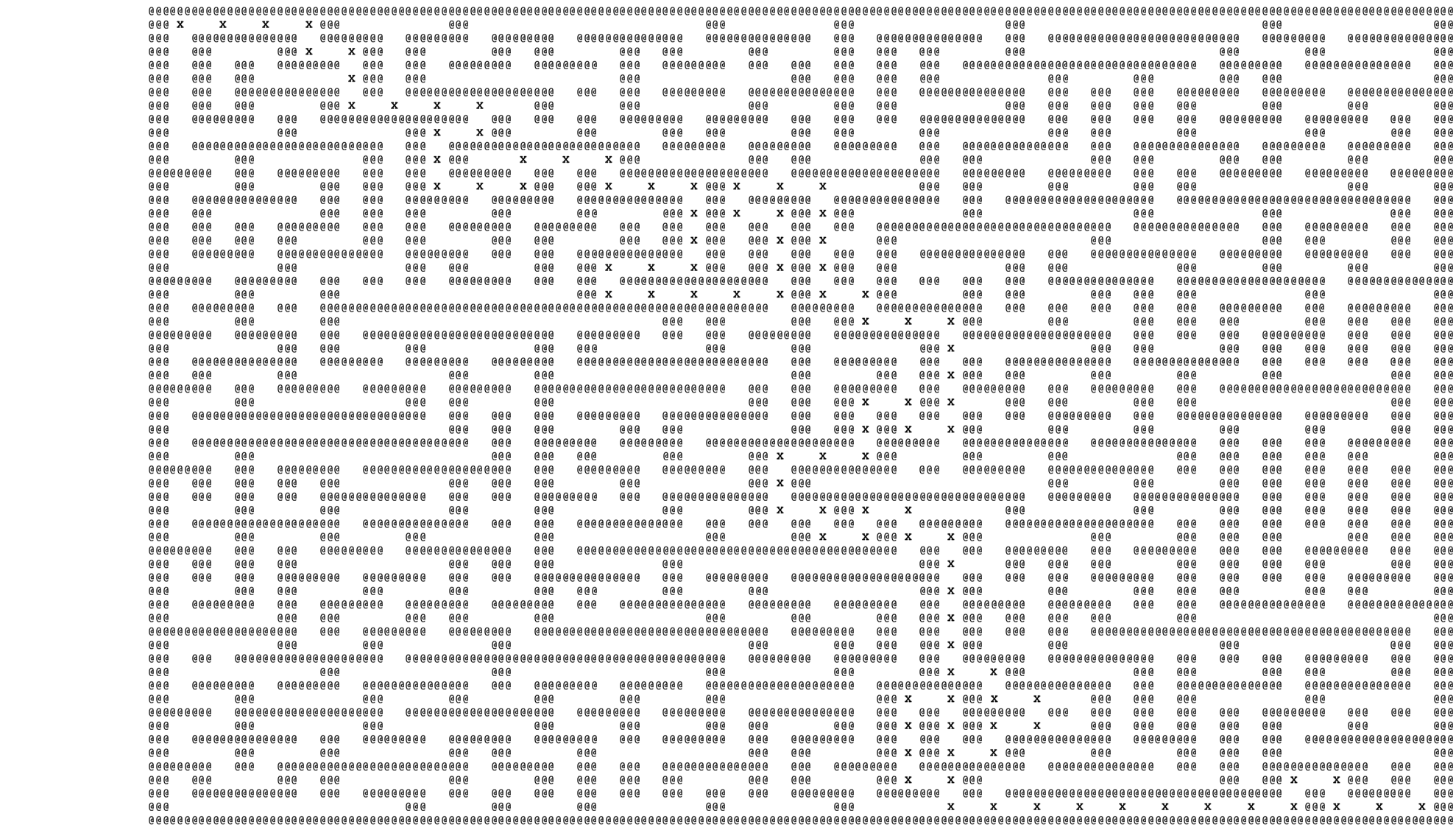

Figure 1: Abbildung des Labyrinths mit Lösungspfad.

(a) Formulieren Sie die Fragestellung als Graphproblem. Definieren Sie dazu einen geeigneten Graphen sowie eine Bedingung, welche der Graph genau dann erfüllt, wenn es eine Folge von Währungen wie oben beschrieben gibt. *(4 Punkte)*

(b) Geben Sie einen Algorithmus an, welcher in $\mathcal{O}(n^3)$ Zeitschritten entscheidet, ob es eine Folge von Währungen wie oben beschrieben gibt. Begründen Sie Laufzeit und Korrektheit. *(6 Punkte)*

Hinweis: Es gilt $\log(a \cdot b) = \log a + \log b$

Musterlösung

(a) Wir definieren einen gewichteten Graphen $G = (V, E, w)$ mit $V = \{1, \ldots, n\}$, $E = V^2$ (d.h. der Graph ist gerichtet und vollständig) und $w(i, j) = a_{ij}$ (d.h. A entspricht der Adjazenzmatrix). Eine Folge von Währungen wie beschrieben gibt es genau dann, wenn es einen Kreis $(i_0, i_1, \ldots, i_k, i_0)$ gibt, so dass

$$\prod_{j=0}^{k-1} w(i_j, i_{j+1}) \cdot w(i_k, i_0) > 1 \ . \tag{1}$$

(b) Wie ersetzen in der Adjazenzmatrix a_{ij} durch $-\log a_{ij}$, d.h. wir definieren einen Graphen $G = (V, E, w')$ mit V und E wie oben und $w'(i, j) = -\log w(i, j)$. Wir wenden Bellman-Ford mit Startpunkt i_0 an. Dieser prüft, ob es einen negativen Kreis gibt, d.h. Knoten $i_0, \ldots, i_k$ mit

$$\begin{aligned}
& \sum_{j=0}^{k-1} w'(i_j, i_{j+1}) + w'(i_k, i_0) < 0 \\
\iff \quad & \sum_{j=0}^{k-1} -\log w(i_j, i_{j+1}) - \log w(i_k, i_0) < 0 \\
\iff \quad & \sum_{j=0}^{k-1} \log w(i_j, i_{j+1}) + \log w(i_k, i_0) > 0 \\
\iff \quad & \log \left(\prod_{j=0}^{k-1} w(i_j, i_{j+1}) \cdot w(i_k, i_0) \right) > 0 \\
\iff \quad & \prod_{j=0}^{k-1} w(i_j, i_{j+1}) \cdot w(i_k, i_0) > 1 \ .
\end{aligned}$$

Das heißt der Algorithmus prüft, ob Bedingung (1) aus Teil (a) erfüllt ist. Die Laufzeit von Bellman-Ford ist $\mathcal{O}(|V| \cdot |E|)$. Mit $|V| = n$ und $|E| = n^2$ erhalten wir also die Laufzeit $\mathcal{O}(n^3)$.