Exercise 1: Dijkstra’s Algorithm (10 Points)

Consider a maze which is given as a subgraph of an \( n \times n \) grid, i.e., each node in the grid has at most four incident edges; at most two in horizontal and at most two in vertical direction. We assume that walking through the maze in horizontal direction takes longer than in vertical direction, so we assign each horizontal edge weight 2 and each vertical edge weight 1.

We number the \( n^2 \) nodes line by line. The maze is given by an adjacency list \( A \). Entry \( A[i] \) contains tuples of the form \( (j, w(i, j)) \), where \( j \) is a neighbor of node \( i \) and \( w(i, j) \in \{1, 2\} \) the weight of edge \( \{i, j\} \).

(a) Implement an algorithm that computes for such an adjacency list and two grid nodes \( s, t \in \{0, \ldots, n^2 - 1\} \) the shortest path from \( s \) to \( t \) as a sequence of visited grid nodes in time \( O(n^2 \log n) \).

You may use the template Maze.py as well as any data structures used on former exercise sheets.\(^1\) Shortly explain the runtime of your algorithm in erfahrungen.txt.

(b) Run your algorithm on the maze given in maze.txt for \( s = 0 \) and \( t = 899 \). In Maze.py you can find a function to convert the data from maze.txt into an adjacency list. Use the function visualize_path on your result and store the output into a file solution.txt.

Sample Solution

(a) Cf. Maze.py. Dijkstra using a Min-Heap as Priority-Queue has runtime \( O(n' + m \log n') \) where \( n' \) is the number of nodes in the maze, i.e., \( n' = n^2 \). Each node has at most four neighbors, so we have \( m \leq 4n' = O(n^2) \). Thus using Dijkstra in the maze takes \( O(n' + m \log n') = O(n'^2 + n^2 \log n^2) = O(n^2 \log n) \).

(b) Cf. figure 1 or maze_viz.txt.

Exercise 2: Currency Exchange (10 Points)

Consider \( n \) currencies \( w_1, \ldots, w_n \). The exchange rates are given in an \( n \times n \)-matrix \( A \) with entries \( a_{ij} \) \((i, j \in \{1, \ldots, n\}) \). Entry \( a_{ij} \) is the exchange rate from \( w_i \) to \( w_j \), i.e., for one unit of \( w_i \) one gets \( a_{ij} \) units of \( w_j \).

Given a currency \( w_{i_0} \), we want to find out whether there is a sequence \( i_0, i_1, \ldots, i_k \) such that we make profit if we exchange one unit of \( w_{i_0} \) to \( w_{i_1} \), then to \( w_{i_2} \) etc. until \( w_{i_k} \) and then back to \( w_{i_0} \).

\(^1\)E.g., heapq and networkx.utils.union_find. In heapq the function heappush corresponds to the insert operation and heappop to the delete-min operation from the lecture. You can use heappush and heappop on Python-lists (more details here).
Figure 1: Maze with solution path.
(a) Translate this problem to a graph problem. That is, define a graph and a property which the graph fulfills if and only if there is a sequence of currencies as described above. (4 Points)

(b) Give an algorithm that decides in $O(n^3)$ time steps whether there is a sequence of currencies as described above. Explain the correctness and runtime. (6 Points)

Hint: It is $a \cdot b > 1 \iff -\log a - \log b < 0$

Sample Solution

(a) We define a weighted graph $G = (V, E, w)$ with $V = \{1, \ldots, n\}$, $E = V^2$ (i.e., the graph is directed and complete) and $w(i, j) = a_{ij}$ (i.e., $A$ is the adjacency matrix). A sequence of currencies as described exists if and only if there is a cycle $(i_0, i_1, \ldots, i_k, i_0)$ such that

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

(b) In the adjacency matrix, we replace $a_{ij}$ by $-\log a_{ij}$. That is, we define a graph $G' = (V, E, w')$ with $V$ and $E$ as before and $w'(i, j) = -\log w(i, j)$. We run Bellman-Ford on $G'$ with source $i_0$. This algorithm checks if $G'$ contains a negative cycle, i.e., nodes $i_0, \ldots, i_k$ with

$$\sum_{j=0}^{k-1} w'(i_j, i_{j+1}) + w'(i_k, i_0) < 0$$

$$\iff \sum_{j=0}^{k-1} -\log w(i_j, i_{j+1}) - \log w(i_k, i_0) < 0$$

$$\iff \sum_{j=0}^{k-1} \log w(i_j, i_{j+1}) + \log w(i_k, i_0) > 0$$

$$\iff \log \left( \prod_{j=0}^{k-1} w(i_j, i_{j+1}) \cdot w(i_k, i_0) \right) > 0$$

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

So the algorithm checks property (1) from part (a). The runtime of Bellman-Ford is $O(|V| \cdot |E|)$. With $|V| = n$ and $|E| = n^2$ we obtain a runtime of $O(n^3)$. 