Instructors:
Michael Ritter
Hugo K. Kasuya Rosado

Exercise sheet #1
Combinatorial Optimization
SuSe 2023

1 Big-O-Notation

For each pair of functions $f, g : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$, show whether $f = O(g)$, $f = \Omega(g)$, $f = \Theta(g)$.

a. $f(n) = n^2 + 10n$ and $g(n) = \frac{1}{3}n^3 + n$
b. $f(n) = \log(n^2 + 10n)$ and $g(n) = \log\left(\frac{1}{3}n^3 + n\right)$
c. $f(n) = 3^n$ and $g(n) = 2^n$
d. $f(n) = \log(3^n)$ and $g(n) = \log(2^n)$

2 Running time

Consider the following pseudo-code.

Algorithm 1: Algorithm

input : A list L of n numbers

```
for j = 2 to n do
    num ← L[j]
    i ← j − 1
    while i > 0 and L[i] > num do
        L[i + 1] ← L[i]
        i ← i − 1
    L[i + 1] ← num
```

a. Suppose that $L = [4, 7, 9, 2, 1]$. What is the list obtained when the algorithm ends?
b. What does the algorithm do? (You don't have to prove your statement)
c. What is the best asymptotical bound you can give to the running time of the algorithm? Is there an instance where the total number of iterations reaches asymptotically this bound? Present your answer in O, Ω, Θ notation and as a function of n.

3 Shortest Path

Consider the Graph on Figure 1. Find a shortest path from vertex s to t. What is its weight?

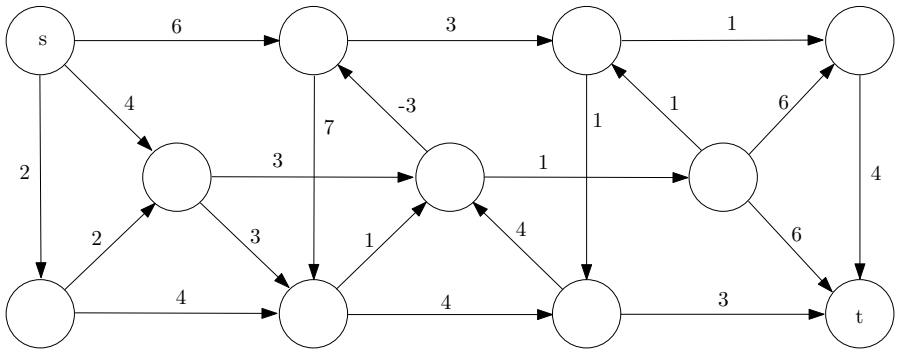


Figure 1: An edge-weighted direct graph.

4 Finding a shortest Path in a Maze

In this exercise we will make a program to find a shortest path in a maze using Dijkstra's algorithm. The input graph is a subgraph of a randomly built grid graph and the program should find a shortest path from a s vertex if there is one, or state that no paths between s and t exists otherwise.

a. With Python 3 installed, run in the terminal the following command to install processing-py.

```
pip install processing-py --upgrade
```

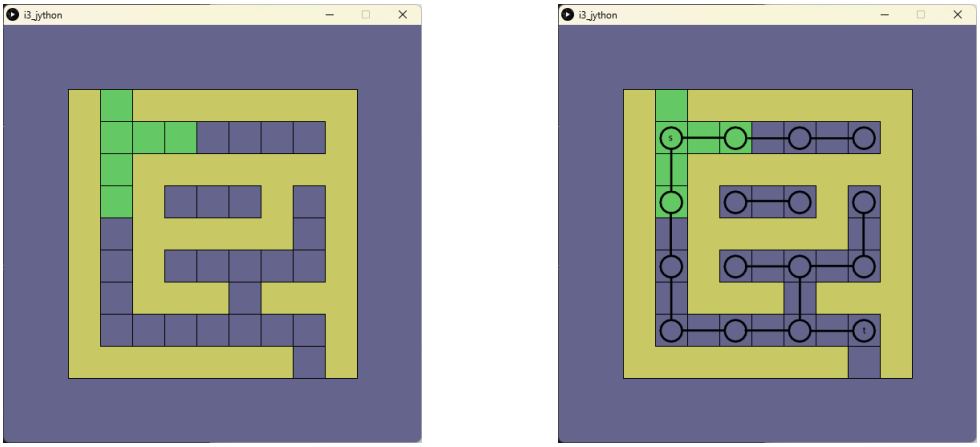
This library contains functions that makes it easy to make or to animate drawings on the screen, and this library will be used to draw a maze. The first time a python code using the library is executed, a script will be downloaded.
Download from Moodle the following files and place them in the same same directory/folder: main.py, random_graph.py, dijkstra.py, and maze.py.

(a) file main.py is the file we will be executing in the terminal and it calls functions and methods defined in the other three files;
(b) random_graph.py contains the code that builds a grid graph with random weights on the edges;
(c) dijkstra.py is the file we will be working on in this exercise where we will be implementing a part of Dijkstra's algorithm;
(d) maze.py contains the code related to building a maze.

To run the code, simply open the terminal, and navigate to the folder containing the files, then run the following command to start python

```
python3 main.py
```

Some status messages should appear in the terminal and a drawing of a grid maze should appear on the screen, see Figure 2. In our maze, a square represent either an edge or a vertex of subgraph of a



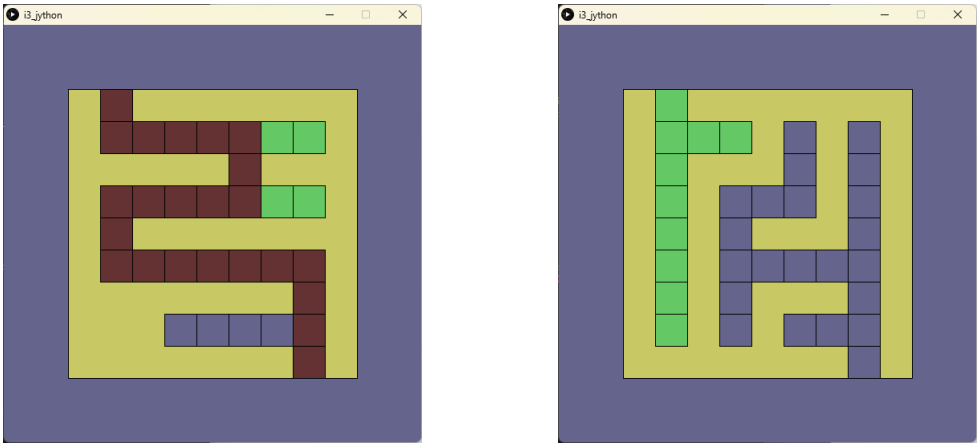
(a) Unexplored Maze. (b) Underlying graph depicted.

Figure 2: Example Maze.

grid-graph. In special, green squares represent places that have been visited by the algorithm, while the blue squares are unexplored locations. After the drawing is complete, press enter on the terminal to end the execution of the code.

b. Open dijkstra.py. This file contains functions for an implementation of Dijkstra's algorithm.
Recall how Dijkstra's algorithm works. Starting from the vertex s, Dijkstra's algorithm iteratively visits vertices and each vertex v is assigned a distance value $dist$ that measures the current known distance from v to s. Initially, the function is defined as $dist(s) = 0$ and $dist(v) = \infty$ for all other vertices, then, it iteratively updates the distance value of each unvisited vertex v to be $min_{\text{visited } u}\{dist(u) + w(u, v)\}$ and it visits an unvisited vertex with minimum distance value. If t is becomes visited, the algorithm returns a shortest-path from s to t, and the distance is $dist(t)$. Otherwise, if at some iteration no new vertices can be visited, then there is no path from s to t.
In this exercise your task is to code the function `minDistance(dist, V, visited)`. The variable `dist` is a list representing the distance values, and `dist[v]` returns the value of the distance of `v`; `v` is a list of vertices; and `visited` if a list that such that `visited[v]` returns `True` if `v` has been visited and `False` otherwise. This function should return the index of an unvisited vertex with minimum distance value. If no such vertex exists, then this function should return the value `-1`.

After implementing this function and running the code, some red squares denoting a shortest-path should appear on the screen if the maze is solvable. See Figure 3.



(a) Maze with solution.

(b) Maze with no solution.

Figure 3: Example of execution after implementation of `minDistance(dist, V, visited)`.

c. In the file main.py you can change the dimensions of the maze by changing the values of `height`, `length`, and `block_size`, and you can also change the probability `p` for which each edge is sampled. Play with these values to build a maze of your liking. See Figure 4

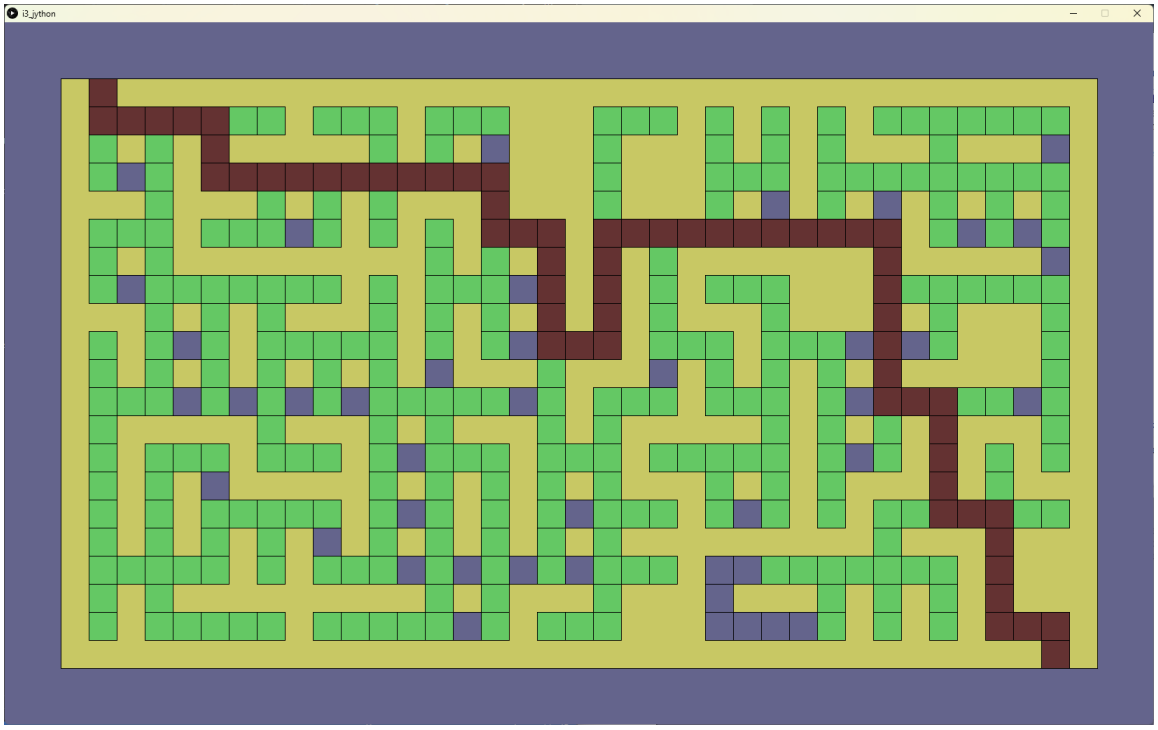


Figure 4: Example of a custom maze.