Instructors:
Michael Ritter
Hugo K. Kasuya Rosado

Exercise sheet #2
Combinatorial Optimization
SuSe 2024

1 Executing the Prim's and Kurskal's algorithms

Consider the Graph on Figure 1.

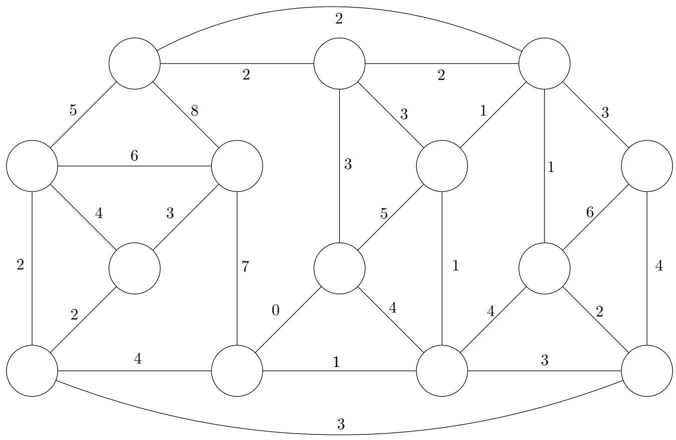


Figure 1: An edge-weighted graph.

a. Write the order of edges included in a solution obtained by an execution of Prim's algorithm.
b. Write the order of edges included in a solution obtained by an execution of Kruskal's algorithm.
c. Did you obtain the same solution in both executions? What is the cost of a minimum spanning tree?

2 Unique MST Solution

Let (G, c) be an instance of the MINIMUM SPANNING TREE PROBLEM where G is connected and $c(e) \neq c(e')$ for any two distinct edges e and e'. Prove that then there is exactly one optimum solution.

3 Union of minimum spanning trees

We want to determine the set of edges e in an undirected graph G with edge weights $c : E(G) \to \mathbb{R}$ for which there exists a minimum weight spanning tree in G containing e (in other words, we are looking for the union of all minimum weight spanning trees in G). Show how this problem can be solved in $O(mn)$ time.

4 Building mazes on random Graphs

Recall that on any tree there is exactly one path from a vertex to another, therefore they are good candidates as a mechanism to build mazes. In this exercise we will make a program to build a maze using kruskal's algorithm for the MINIMUM SPANNING TREE PROBLEM. The idea is to build a grid graph and select edges to obtain a tree, which intuitively should resemble a maze.

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

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

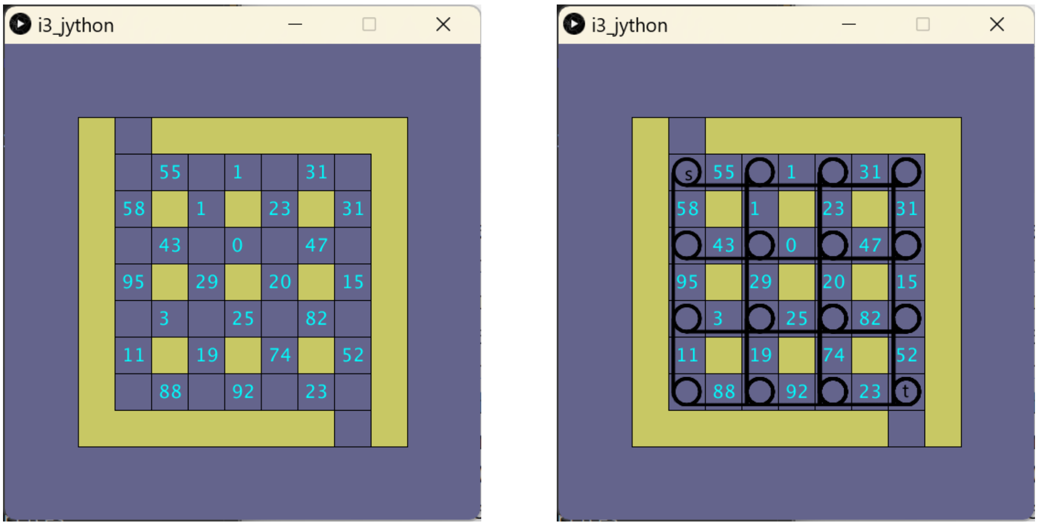
This library contains functions that makes it easy to make or to animate drawings on the screen, and we will be using this library to draw a maze. The first time a python code using the library is executed, a script will be downloaded.
Download from Moodle the following files and place them directory/folder: main.py, random_graph.py, kruskal.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) kruskal.py is the file we will be working on in this exercise where we will be implementing kruskal'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 message should appear on the terminal and a drawing of a trivial 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



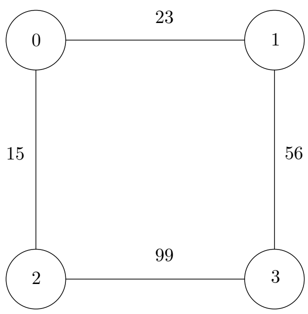
(a) Trivial Maze.

(b) Underlying graph depicted.

Figure 2: Example Maze.

a grid-graph. After the drawing is complete, press enter in the terminal to end the execution of the code.

b. Open kruskal.py. This file contains a function that receives a graph and should return a dictionary of the edges in a minimum spanning tree. The input graph is a grid graph with random edge weights and it is composed by two atributes: graph.vertices is a list of vertices; and graph.edges is a dictionary where the keys are tuples of vertices and the values are their respective weights, e.g., each element is of the form $(v_1, v_2) : w(v_1, v_2)$. See Figure 3. As you may notice, nothing has been implemented in



graph.vertices = [0, 1, 2, 3]

graph.edges = { (0,1) : 23, (0,2) : 15, (2,3) : 99, (1,3) : 56}

sorted_edge_list = [((0,2),15), ((0,1),23), ((1,3),56), ((2,3),99)]

Figure 3: Example of a grid graph with its atributes and a sorted list of the weighted-edges.

this function and the function simply returns the dictionary of edges from the input graph. That is the reason the maze was just a grid in the previous execution.
Recall how Kruskal's algorithm works. The first step of the algorithm is to sort the edges in increasing order of weight. For this, copy the following code in the function and remember to follow identation rules of Python. Identation refers to spaces at the beginning of each line in the code. In Python, identation is used to indicate which codes contained in a block.

```
edge_list = list(graph.edges.items())
sorted_edge_list = sorted(edge_list, key = lambda weighted_edge : weighted_edge[1])
```

This piece of code transforms the edges dictionary into a list, where each element is a tuple of the form $((v_1, v_2), w(w_1, w_2))$, then it obtains another list, which is now sorted. See Figure 3.
Your task in this exercise is to implement the remainder of Kruskal's algorithm - there is no need to optimize time complexity of your implementation. Be aware that the output of the `kruskals_algorithm(\dot)` should be a dictionary where the keys are tuples of vertices and the values are their respective weights, e.g., each element in this dictionary is of the form $(v_1, v_2) : w(v_1, v_2)$.

c. In the file main.py you can change the dimensions of the maze by changing the values of `height`, `length`, and `block_size`. Play with these values to build a maze of a size of your liking. Note by commenting or uncommenting the codes on the lines 27, 28, 29, you can choose if the maze drawn on the screen should be animated or not. See Figure 4

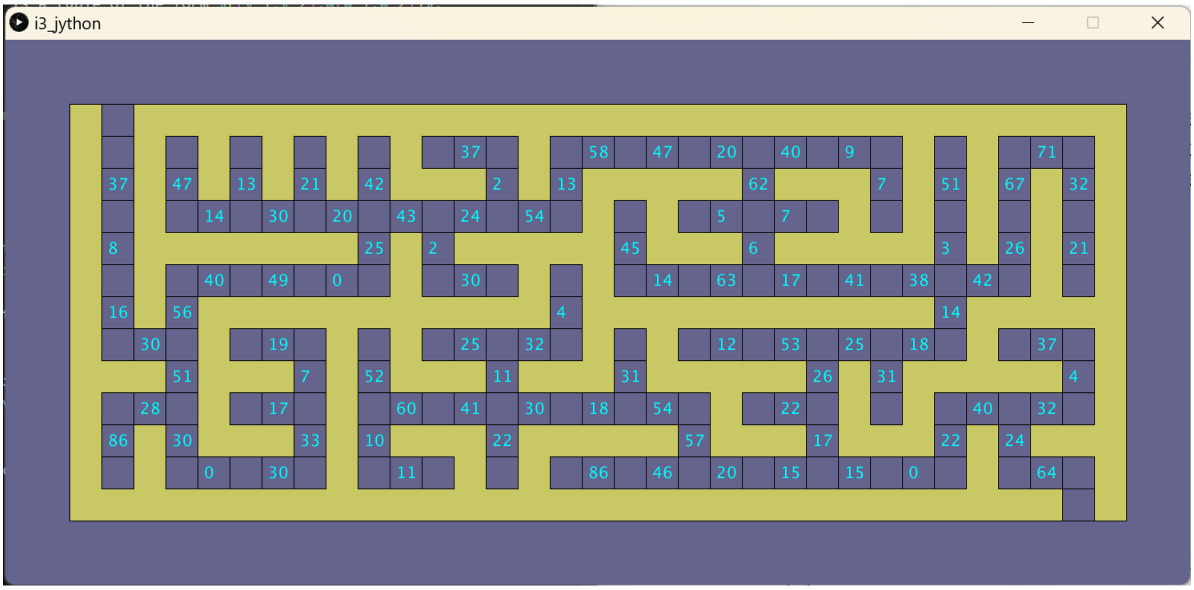


Figure 4: Example of a custom maze.