Introduction to Artificial Intelligence

Exercise 1: Search

Valeriya Khan

Summer 2025

# 1 Task details

## Variant 1

Using the template from the files folder (**lab1_v1.py**), write a program that solves a maze using two search algorithms: **breadth-first search (BFS)** and **depth-first search (DFS)**.

You are given a **maze which is a 2D grid** with empty spaces, walls, a start, and an end position. The objective is to return **number of steps** from start to end position and **make a step-by-step visualization of search** (all cells visited during the search, not only the ones that are in the final path).

Come up **with different test cases and corner cases**. I will run the code against my test cases. The score for coding depends on the passing all test cases, so make sure it works properly. Leave all test cases in the submitted code.

In the **report**:

- discuss the differences in the obtained results between BFS and DFS
- discuss test cases (a maze) that shows the strengths and weaknesses (if any) of each of the algorithm
- make conclusions from the discussion above

Regarding **visualization**, it can be done in the console, and the interface may be as simple as possible. Example solution: https://angeluriot.com/maze_solver/.

## Variant 2

Using the template from the files folder (**lab1_v2.py**), write a program that solves a maze using **greedy best-first search** algorithm. The maze is a 2D grid with empty spaces, walls, a start, and an end position.

You are given a **maze which is a 2D grid** with empty spaces, walls, a start, and an end position. The objective is to return **number of steps** from start to end position and **make a step-by-step**

**visualization of search** (all cells visited during the search, not only the ones that are in the final path). Implement at least **two heuristics h(n)**.

Come up **with different test cases and corner cases**. I will run the code against my test cases. The score for coding depends on the passing all test cases, so make sure it works properly. Leave all test cases in the submitted code.

In the **report**:

- discuss the differences of results obtained by different heuristics
- discuss test cases (a maze) that shows the strengths and weaknesses (if any) of the algorithm
- make conclusions from the discussion above

Regarding **visualization**, it can be done in the console, and the interface may be as simple as possible. Example solution: https://angeluriot.com/maze_solver/.

## Variant 3

Using the template from the files folder (**lab1_v3.py**), write a program that solves a maze using **A* algorithm**.

You are given a **maze which is a 2D grid** with empty spaces, walls, a start, and an end position. The objective is to return **number of steps** from start to end position and **make a step-by-step visualization of search** (all cells visited during the search, not only the ones that are in the final path). Implement at least **two heuristics h(n)**.

Come up **with different test cases and corner cases**. I will run the code against my test cases. The score for coding depends on the passing all test cases, so make sure it works properly. Leave all test cases in the submitted code.

In the **report**:

- discuss the differences of results obtained by different heuristics
- discuss test cases (a maze) that shows the strengths and weaknesses (if any) of the algorithm
- make conclusions from the discussion above

Regarding **visualization**, it can be done in the console, and the interface may be as simple as possible. Example solution: https://angeluriot.com/maze_solver/.

## Variant 4

Using the template from the files folder (**lab1_v4.py**), write a program that minimizes a function using **Newton's method** algorithm.

You are given a **function** definition in the code template (don't change it). The objective is to return the found **solution x*** and the **number of iterations** and make **visualization** of the algorithm for **x in range [-5,5]** and **y in range [-5,5]**.

Come up with different **initial point** and **step size pairs** to obtain as many different results in the given coordinate ranges as possible. Use visualizations for this. Leave all examples in the submitted code.

In the **report**:

- discuss the impact of different initial vectors
- discuss the impact of different step size parameters
- make conclusions from the discussion above

Regarding **visualization**, it can be as simple as putting points for iterations (even not all) on the plot of objective function.

## Variant 5

Using the template from the files folder (**lab1_v5.py**), write a program that minimizes a function using **straight gradient descent** algorithm.

You are given a **function** definition in the code template (don't change it). The objective is to return the found **solution x*** and the **number of iterations** and make **visualization** of the algorithm for **x in range [-5,5]** and **y in range [-5,5]**.

Come up with different **initial point** and **learning rate pairs** to obtain as many different results in the given coordinate ranges as possible. Use visualizations for this. Leave all examples in the submitted code.

In the **report**:

- discuss the impact of different initial vectors
- discuss the impact of different learning rates
- make conclusions from the discussion above

Regarding **visualization**, it can be as simple as putting points for iterations (even not all) on the plot of objective function.

# 2 Technical details

- The submission should include the **python file** and the **report in the format of .pdf**
- Please ensure that your code adheres to basic standards of lean coding in accordance with PEP8. Additionally, it should contain comments on the crucial parts to help with **readability and understanding**.
- The **inputs to the code** should be provided inside the python code or from the file. In case of providing inputs from the file, the python code should include reading from the file part and provide instructions in the comments. In addition, the file with the inputs should be included to the submission.
- **All the test cases and examples used should be in the submission**. The code without proper test cases may get less points.
- **Templates** contain crucial information for some tasks (e.g. function definition), so, start from them. However, they can be changed and adapted as long as objectives of the task are reached.

# 3 Submission guidelines

You should submit all the files via Teams not later than:

- **24.03.2025** until 23:59 for Wednesday group
- **26.03.2025** until 23:59 for Friday group
- **29.03.2025** until 23:59 for Monday group

The on-line assessment will take place during your labs in two weeks. In case of questions, please contact me via Teams.

Submit to: valeriya.khan.dokt@pw.edu.pl to email or in teams chat
**Names of all files** should be:
lab1_cg{class_group_number}_g{group_number}_v{variant_number}_{surname1}_{surname2}.(py,pdf,zip, etc)
E.g. lab1_cg101_g1_v1_Smith_Jankowski.py

# 4 Assessment criteria

You can get **[0, 5] points** for the lab. The following criteria will be used to evaluate your work:

- Proper **code implementation** of the algorithm: **1 point**
- Final **report** including clear explanation of the programmed solutions and discussion of the results: **2 points**
- Explaining the solution and answering questions during the **online assessment**: **2 points**