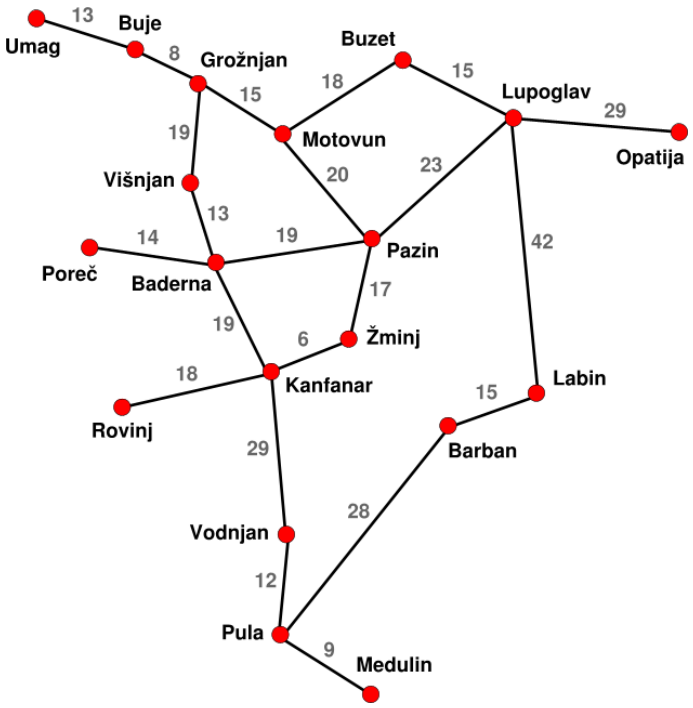
Introduction to Artificial Intelligence – Programming assignment 1

UNIZG FER, academic year 2020/21.

Handed out: March 22, 2021. Due: April 5, 2021 at 23.59.

State space search: from Buzet to the 8-puzzle (24 points)

In the first lab assignment we will analyse state space search problems and the complexity of various blind and heuristic search algorithms. Our first, motivational problem will be finding the shortest path from Pula to Buzet in order to reach the giant truffle omelette.



Trip through Istria

Note: despite the fact that we are using a single example throughout the instructions, your implementation **must** work for **all** files attached with this assignment in reasonable time period (max 2 minutes). This condition does not include checking if heuristic is optimistic for heuristic in "`3x3_misplaced_heuristic.txt`", but it includes all other heuristic checks and state space search problems for all maps (including applying BFS, UCS, and A* to "`3x3_puzzle.txt`"). Together with the attached files, your implementation will also be tested with an *autograder* using examples which are not published, so make sure that you cover all the edge cases that might introduce issues to your implementation.

Together with the lab assignment, you will get access to a sample of test examples, so you can try running them locally.

Please **read, in detail**, the instructions for input and output formatting in the Section "Autograder instructions" as well as the sample outputs in chapter "Appendix: Inputs and expected outputs". In case your submitted lab assignment cannot be compiled or executed with the autograder it will be graded with 0 points **without exception**. Your code may not use **any external libraries**. If you are not sure whether you are allowed to use some library, check whether it is part of the standard library package for that language.

The total number of points in this lab assignment is 24. The number of points that you obtain will be scaled by percentage to 7.5 points.

1. Data loading

In order to analyse the performance of our algorithms on various problems, we will define a standardised input file format for data needed for solving state space search problems. Detailed examples of input file format can be found at the end of the instructions. Each state space search problem is defined with two text files: (1) the state space descriptor and (2) the heuristic descriptor. Each text file can contain comment lines. Such lines always start with the `#` symbol and should be ignored.

The state space descriptor file contains information about the initial state, goal state (or states) and transitions. The first non-comment line of the file contains the initial state, while the second line contains the goal states separated by a single whitespace. The remaining lines of the state space descriptor map out the transition function. Each line is in the following format:

```
state: next_state_1,cost next_state_2,cost
```

An example of the line format for the state space descriptor file:

```
Barban: Pula,28 Labin,15
```

All elements of the line will be separated by a single whitespace. The first element of the line contains the source state followed by a colon, while every subsequent element contains the target state and the transition cost, separated by a comma. The name of each state is a sequence of symbols. The symbols are limited to lower and upper case alphabet letters, numbers and the underscore symbol. The transition costs can be floating point numbers.

The heuristic descriptor file contains the values of the heuristic function for each state. Each line of the file is in the following format:

```
state: heuristic_value
```

An example of the line format for the heuristic descriptor file:

```
Barban: 35
```

State names in the heuristic descriptor will match the ones from the state space descriptor. Values of the heuristic function can be positive floating point numbers. Your first assignment is to implement data loading from the aforementioned format into data structures adequate to be used for state space search algorithms. This part of the programming assignment is not included in the points for this assignment, but is necessary for implementing the remaining tasks.

2. State space search algorithms (12 points)

Once you have succesfully completed the implementation of data loading, your next task is to implement the basic state space search algorithms. Your task in this part of the programming assignment is to implement:

1. Breadth first search (BFS) **(4 points)**
2. Uniform cost search (UCS) **(4 points)**
3. The A* algorithm (A-STAR) **(4 points)**

In your implementation, for each of the aforementioned algorithms in the first line of the output, which should start with the `#` symbol, you should print the abbreviation of the used algorithm (BFS, UCS, or A-STAR), and in case of A* algorithm the path to the heuristic descriptor file used in the algorithm, separated by a single whitespace from the algorithm abbreviation (e.g., "`# A-STAR istra_heuristic.txt`").

After the line with algorithm abbreviation, if the algorithm has found the solution, for each of the aforementioned algorithms your implementation should output the information about the following **five** elements:

1. whether a solution was found (`yes` or `no`), with prefix "`[FOUND_SOLUTION]:`",
2. the number of visited states (the size of the *closed* set), with prefix "`[STATES_VISITED]:`",
3. the length of the found solution, with prefix "`[PATH_LENGTH]:`",
4. the cost of the found solution (with a single value after decimal point), with prefix "`[TOTAL_COST]:`",
5. the path of the solution (list of visited states for the solution, from the start state up to the goal state separated by " `=>` "), with prefix "`[PATH]:`".

If the solution was not found, it is sufficient to output only the first element ("`[FOUND_SOLUTION]:`") from the list.

An example of such output for the A* algorithm for the problem of the tript through Istria using the heuristic from `istra_heuristic.txt`:

```
# A-STAR istra_heuristic.txt
[FOUND_SOLUTION]: yes
[STATES_VISITED]: 14
[PATH_LENGTH]: 5
[TOTAL_COST]: 100.0
[PATH]: Pula => Barban => Labin => Lupoglav => Buzet
```

It is necessary that your output contains all of the mentioned elements (if the solution was found), including the line with algorithm's abbreviation, and that the values of each element are separated by **a single whitespace** from the element's name (e.g., "`[PATH_LENGTH]:␣5`"). More output examples are given in the chapter "Appendix: Inputs and expected outputs".

NOTE 1: In case of multiple nodes with the same priority in the list *open* in algorithms UCS and A*, the order of opening of those nodes is defined by **alphabetical order** of

the nodes' names. With BFS algorithm, it is necessary to sort the node's neighbours in alphabetical order and insert the nodes in the *open* list in that order.

NOTE 2: Make sure that you maintain the *closed* (or *visited*) set in your implementations for all search algorithms. This is necessary in order for your solutions to match the expected output.

3. Evaluating the heuristic function (12 points)

The quality of the heuristic function significantly affects the performance of the A* algorithm. In situations when the heuristic function is not optimistic or consistent, the algorithm will not necessarily return the optimal solution. We would like to avoid such situations if possible. Your task in this part of the programming assignment is to implement functions that, for a given state space and heuristic function h check:

1. Is the heuristic function optimistic? **(6 points)**
2. Is the heuristic function consistent? **(6 points)**

The output of your implementation for both functions should start with a line indicating which function is being evaluated. For checking whether heuristic function is optimistic, the first line in the output should be in the following format:

```
# HEURISTIC-OPTIMISTIC path_to_the_heuristic_descriptor_file
```

An example of the line format for the heuristic descriptor file `istra_heuristic.txt`:

```
# HEURISTIC-OPTIMISTIC istra_heuristic.txt
```

For checking whether heuristic function is consistent, the output should be of the same format, except that the word "`OPTIMISTIC`" should be replaced with "`CONSISTENT`" (e.g., "`# HEURISTIC-CONSISTENT istra_heuristic.txt`").

Your implementations for heuristic checks should output **each** case for which the condition (optimistic or consistent) is being evaluated, and print whether the condition was satisifed or not for that case. For example, for checking if heuristic is optimistic, beside answering the question whether the heuristic function is optimistic, you should output the true cost needed to reach the goal for each state, and the heuristic value in that state. If the heuristic value is larger than the true cost needed to reach the goal for that state, that is, if the heuristic function **overestimates** the true cost needed to reach the goal, your implementation should output that in this case the condition is not satisfied (with "`[ERR]`" keyword). If the heuristic value is smaller than the true cost needed to reach the goal for that state, that is, if the heuristic function **doesn't overestimate** the true cost needed to reach the goal, your implementation should output that in this case the condition is satisfied (with "`[OK]`" keyword).

A shortened output for such heuristic check for a bad for a bad heuristic for the trip through Istria is given below. The full output, which is **needed** in your implementation, is given in the chapter "Appendix: Inputs and expected outputs".

```
# HEURISTIC-OPTIMISTIC istra_pessimistic_heuristic.txt
[CONDITION]: [OK] h(Opatija) <= h*: 26.0 <= 44.0
...
[CONDITION]: [ERR] h(Pazin) <= h*: 40.0 <= 38.0
```

```
...
[CONCLUSION]: Heuristic is not optimistic.

# HEURISTIC-CONSISTENT istra_pessimistic_heuristic.txt
[CONDITION]: [OK] h(Baderna) <= h(Višnjan) + c: 25.0 <= 20.0 + 13.0
...
[CONDITION]: [ERR] h(Lupoglav) <= h(Buzet) + c: 35.0 <= 0.0 + 15.0
...
[CONCLUSION]: Heuristic is not consistent.
```

Each case that is being checked should be prefixed with "`[CONDITION]:`", after which the result of the check should be printed, with keyword "`[OK]`" if the condition is satisfied in that case, or keyword "`[ERR]`" if the condition was not satisfied. In the last line of the output, your implementation should print the conclusion of the heuristic check, prefixed with "`[CONCLUSION]:`". In case of checking whether heuristic is optimistic, after the conclusion prefix your implementation should output the sentence "`Heuristic is optimistic.`" if the heuristic is optimistc, or the sentence "`Heuristic is not optimistic.`" otherwise. Conclusion sentences for consistency checks are formatted similarly.

In the output for each case, it is necessary that the output for each case is formatted as in the examples given above. Hence, when checking if heuristic is optimistic for a state S, the output for that case (after the keyword "`[OK]`" or "`[ERR]`") should be in the following format:

```
h(S) <= h*: num_1 <= num_2
```

where `num_1` is the heuristic value in the state S (with one value after decimal point) and `num_2` is the true cost needed to reach the goal state from the state S.

When checking if heuristic is consistent for a state S and its successor state T, the output for that case (after the keyword "`[OK]`" or "`[ERR]`") should be in the following format:

```
h(S) <= h(T) + c: num_1 <= num_2 + num_3
```

where `num_1` and `num_2` are the heuristic values in the states S and T, respectively, and `num_3` is the transition cost from the state S to the state T. Heuristic values for states S and T should be printed with one value after the decimal point.

NOTE 3: In the output examples given in the chapter "Appendix: Inputs and expected outputs" states are ordered by alphabetical order of their names. We suggest that you also structure your output lines in the same way, although that is not necessary for autograder checks. It is sufficient that you output all the lines with the heuristic checks for all the states using the required formatting, regardless of the order of the lines.

Determine the algorithmic complexity of your implementation for functions checking whether the heuristic is consistent and optimistic. For simple problems such as the trip through Istria, even naive implementations of heuristic function checks are fast enough. Try to evaluate whether the given heuristic for the 8-puzzle is optimistic and consistent (files `3x3_puzzle.txt`, `3x3_misplaced_heuristic.txt`). Do the functions complete within reasonable time (below 5 minutes)? Can these functions be optimized further? Think of ways you could optimize each of the functions and elaborate your approach when submitting the assignment.

Bonus assignments: the complexity of the 8-puzzle

For those students that want to further expand their knowledge in the state space seach, we suggest solving the following bonus assignments. Although these assignments are given as part of lab assignment, their solutions will **not** be graded for the total number of points in the assignment. If you decide to solve the bonus assignments, you should submit the solutions to these assignments in the same archive as the rest of the lab assignment solution and discuss them with Teaching Assistants during the examination. Solutions which do not require a programming implementation but a written answer also have to be uploaded as either a scanned document, image or a text file.

1. Solvability of the 8-puzzle

When loading the 8-puzzle state space, print out the loaded number of states. Is this the total number of possible configurations for the 8-puzzle? If not, what is the total number of possible configurations?

Is it possible to solve the 8-puzzle starting from any initial configuration? If not, **prove** what is the number of states for which the goal state is unreachable and demonstrate a simple example of a starting position for which the puzzle is unsolvable. Perform the same analysis for the 15-puzzle.

When computing the proof, you are allowed to refer to content on the web, but we strongly advise you invest some time and attempt to reach a solution yourself first.

2. Optimizing checks whether a heuristic is optimistic and consistent

In the scope of the lab assignment you were supposed to answer whether functions checking if a heuristic is optimistic and consistent can be optimized, and suggest a method to optimize one or both of them. Your task in this part of the programming assignment is to implement and evaluate your idea. **Important:** ensure that your solution also works on state spaces which are directed graphs. An example of a directed graph is the state space in the file `ai.txt`.

When evaluating your optimization, (1) compute the complexity of your optimized function and (2) track the time (wall clock time) required to run the function when compared to your original implementation. If your original implementation takes over 5 minutes, you do not need to run it to completion and rather log it as duration of `5+`. If as part of your function you are using a helper function to compute the shortest path between two states, also track (3) the number of calls of that function. Run your evaluation on at least 10 different initial configurations, including the default one and track the average and standard deviation of (2) and (3).

You are required to have all of these statistics ready before the submission of the lab assignment!

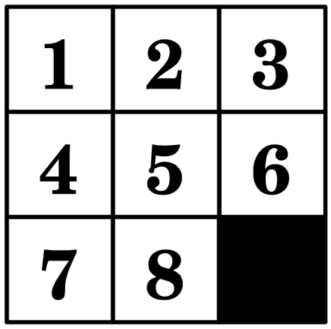
3. Designing heuristic functions

The value of the default heuristic function for the 8-puzzle is the number of misplaced puzzle elements (compared to the goal state), where we differ from the example heuristic from the lectures by **including the empty element** in the misplaced count. This heuristic is neither optimistic nor consistent. Can you think of a simple example showing that this heuristic isn't optimistic? Which simple change should be done to the heuristic to make it both optimistic and consistent?

Try to come up with at least two new good heuristic functions. Measure the number of visited states for the A* algorithm for each heuristic on at least 10 different initial configurations, including the default one and store the number of visited states for each configuration.

The state of the 8-puzzle ;

Since the state space of the 8-puzzle is too large to manually compute the heuristic, you should instead implement a function to do so. To do this, you need to understand how to transform the name of a state to a matrix form appropriate for computing heuristic values. Each state in the 8-puzzle state space is of the following form: `123_456_78x`. The rows of the puzzle are separated by the underscore (`_`) symbol, and the image which corresponds to the state is:



The state "123_456_78x" of the 8-puzzle

Autograder instructions

Uploaded archive structure

The archive that you will upload to Moodle **has to** be named `JMBAG.zip`, while the structure of the unpacked archive **has to** be as in the following example (the following example is for solutions written in Python, while examples for other languages are given in subsequent sections):

```
|JMBAG.zip
|-- lab1py
|----solution.py [!]
|----...
```

Uploaded archives that are not structured in the given format will **not be graded**. Your code must be able to execute with the following arguments from command line:

- `--alg`: abbreviation for state space search algorithm (values: `bfs`, `ucs`, or `astar`),
- `--ss`: path to state space descriptor file,
- `--h`: path to heuristic descriptor file,
- `--check-optimistic`: flag for checking if given heuristic is optimistic,

- `--check-consistent`: flag for checking if given heuristic is consistent.

When running your code on test examples in the autograder, for testing BFS and UCS algorithms, your code will be run with the first two arguments (`--alg` i `--ss`), while in case of A* algorithm, argument with path to heuristic descriptor file will also be given (`--h`). In the tests for checking if heuristic is optimistic and consistent, arguments `--ss`, `--h`, and `--check-optimistic` will be passed for checking if heuristic is optimistic, while arguments `--ss`, `--h`, and `--check-consistent` will be used for checking if heuristic is consistent. In the output examples given in the chapter "Appendix: Inputs and expected outputs" examples for running appropriate tests for Python solutions are given. For Java and C++ solutions, the arguments will be used in the same way. **NOTE**: The order of arguments passed to your solution does not necessarily have to be in the same order as they are listed in the above given list. Make sure that your solution accepts the arguments regardless of their order.

Your code will be executed on linux. This does not affect your code in any way except if you hardcode the paths to files (which in any way, **you should not do**). Your code should **not use any external libraries**. Use the `UTF-8` encoding for all your source code files.

An example of running your code for A* algorithm on state space `istra.txt` using heuristic descriptor file `istra_heuristic.txt` (for Python):

```
>>> python solution.py --alg astar --ss istra.txt --h istra_heuristic.txt
```

Instructions for Python

The entry point for your code **must** be in the `solution.py` file. You can structure the rest of your code using additional files and folders, or you can leave all of it inside `solution.py` file. Your code will always be executed from the folder of your assignment (`lab1py`).

The directory structure and execution example can be seen at the end of the previous chapter. Your code will be executed with python version `3.7.4`.

Instructions for Java

Along with the lab assignment, we will publish a project template which should be imported in your IDE. The structure inside your archive `JMBAG.zip` is defined in the template and has to be as in the following example:

```
|JMBAG.zip
|--lab1java
|----src
|------main.java.ui
|--------Solution.java [!]
|--------...
|----target
|----pom.xml
```

The entry point for your code **must** be in the file `Solution.java`. You can structure the rest of the code using additional files and folders, or you can leave all of it inside the `Solution.java` file. Your code will be compiled using Maven.

An example of running your code for A* algorithm on state space `istra.txt` using heuristic descriptor file `istra_heuristic.txt` (from the `lab1java` folder):

```
>>> mvn compile
>>> java -cp target/classes ui.Solution --alg astar --ss istra.txt --h
    istra_heuristic.txt
```

Info regarding the Maven and Java versions:

```
>>> mvn -version
Apache Maven 3.6.3
Maven home: /opt/maven
Java version: 15.0.2, vendor: Oracle Corporation, runtime: /opt/jdk-15.0.2
Default locale: en_US, platform encoding: UTF-8
OS name: "linux", version: "4.15.0-139-generic", arch: "amd64", family: "unix"

>>> java -version
openjdk version "15.0.2" 2021-01-19
OpenJDK Runtime Environment (build 15.0.2+7-27)
OpenJDK 64-Bit Server VM (build 15.0.2+7-27, mixed mode, sharing)
```

Inportant: please check that your implementation can be compiled with the predefined `pom.xml` file.

Instructions for C++

The structure inside your archive `JMBAG.zip` has to be as in the following example:

```
|JMBAG.zip
|--lab3cpp
|----solution.cpp [!]
|----...
```

If your submitted archive does not contain a `Makefile`, we will use the Makefile template available along with the assignment. **If** you submit a `Makefile` in the archive (which we don't suggest, unless you really know what you're doing), we expect it to work.

An example of running your code for A* algorithm on state space `istra.txt` using heuristic descriptor file `istra_heuristic.txt` (from the `lab1cpp` folder):

```
>>> make
>>> ./solution --alg astar --ss istra.txt --h istra_heuristic.txt
```

Info regarding the gcc version:

```
>>> gcc --version
gcc (Ubuntu 9.3.0-11ubuntu0~18.04.1) 9.3.0
```

Appendix: Inputs and expected outputs

In this section we will go over a number of examples of inputs and their corresponding outputs which can be used to validate whether your solution is correct. Depending on how your code handles ties when sorting priorities, the number of visited states can very.

Note: Some lines are folded so that their content can fit in the document. In your output, each element of the output should be printed in one line.

0.0.1 1. Passing the AI course

As an additional check for the validity of your implementation, in the file `ai.txt` we have provided a state space for a (simplified) route through passing (or failing) this course. The state space is a directed graph with two goal states and is of reasonable size to proof check the execution by hand.

Breadth first search:

```
# BFS
[FOUND_SOLUTION]: yes
[STATES_VISITED]: 6
[PATH_LENGTH]: 3
[TOTAL_COST]: 21.0
[PATH]: enroll_artificial_intelligence => fail_lab => fail_course
```

Uniform cost search:

```
# UCS
[FOUND_SOLUTION]: yes
[STATES_VISITED]: 7
[PATH_LENGTH]: 4
[TOTAL_COST]: 17.0
[PATH]: enroll_artificial_intelligence => complete_lab => pass_continuous =>
    pass_course
```

A* algorithm + heuristic `ai_fail.txt`:

```
# A-STAR ai_fail.txt
[FOUND_SOLUTION]: yes
[STATES_VISITED]: 6
[PATH_LENGTH]: 3
[TOTAL_COST]: 21.0
[PATH]: enroll_artificial_intelligence => fail_lab => fail_course
```

Checking if heuristic is optimistic for the `ai_fail.txt` heuristic:

```
# HEURISTIC-OPTIMISTIC ai_fail.txt
[CONDITION]: [OK] h(complete_lab) <= h*: 10.0 <= 13.0
[CONDITION]: [OK] h(enroll_artificial_intelligence) <= h*: 17.0 <= 17.0
[CONDITION]: [OK] h(fail_continuous) <= h*: 6.0 <= 17.0
[CONDITION]: [OK] h(fail_course) <= h*: 0.0 <= 0.0
[CONDITION]: [OK] h(fail_exam) <= h*: 5.0 <= 20.0
[CONDITION]: [OK] h(fail_lab) <= h*: 1.0 <= 17.0
[CONDITION]: [ERR] h(pass_continuous) <= h*: 20.0 <= 1.0
[CONDITION]: [OK] h(pass_course) <= h*: 0.0 <= 0.0
[CONDITION]: [OK] h(pass_exam) <= h*: 1.0 <= 1.0
[CONCLUSION]: Heuristic is not optimistic.
```

Checking if heuristic is consistent for the `ai_fail.txt` heuristic:

```
# HEURISTIC-CONSISTENT ai_fail.txt
[CONDITION]: [ERR] h(complete_lab) <= h(fail_continuous) + c: 10.0 <= 6.0 + 1.0
[CONDITION]: [OK] h(complete_lab) <= h(pass_continuous) + c: 10.0 <= 20.0 + 12.0
[CONDITION]: [ERR] h(enroll_artificial_intelligence) <= h(complete_lab) + c:
    17.0 <= 10.0 + 4.0
```

```
[CONDITION]: [ERR] h(enroll_artificial_intelligence) <= h(fail_lab) + c: 17.0 <=
    1.0 + 1.0
[CONDITION]: [OK] h(fail_continuous) <= h(fail_exam) + c: 6.0 <= 5.0 + 1.0
[CONDITION]: [OK] h(fail_continuous) <= h(pass_exam) + c: 6.0 <= 1.0 + 16.0
[CONDITION]: [OK] h(fail_exam) <= h(fail_course) + c: 5.0 <= 0.0 + 20.0
[CONDITION]: [OK] h(fail_lab) <= h(complete_lab) + c: 1.0 <= 10.0 + 4.0
[CONDITION]: [OK] h(fail_lab) <= h(fail_course) + c: 1.0 <= 0.0 + 20.0
[CONDITION]: [OK] h(fail_lab) <= h(fail_lab) + c: 1.0 <= 1.0 + 1.0
[CONDITION]: [ERR] h(pass_continuous) <= h(pass_course) + c: 20.0 <= 0.0 + 1.0
[CONDITION]: [OK] h(pass_exam) <= h(pass_course) + c: 1.0 <= 0.0 + 1.0
[CONCLUSION]: Heuristic is not consistent.
```

A* algorithm + heuristic `ai_pass.txt`:

```
# A-STAR ai_pass.txt
[FOUND_SOLUTION]: yes
[STATES_VISITED]: 4
[PATH_LENGTH]: 4
[TOTAL_COST]: 17.0
[PATH]: enroll_artificial_intelligence => complete_lab => pass_continuous =>
    pass_course
```

Checking if heuristic is optimistic for the `ai_pass.txt` heuristic:

```
# HEURISTIC-OPTIMISTIC ai_pass.txt
[CONDITION]: [OK] h(complete_lab) <= h*: 13.0 <= 13.0
[CONDITION]: [OK] h(enroll_artificial_intelligence) <= h*: 17.0 <= 17.0
[CONDITION]: [OK] h(fail_continuous) <= h*: 17.0 <= 17.0
[CONDITION]: [OK] h(fail_course) <= h*: 0.0 <= 0.0
[CONDITION]: [OK] h(fail_exam) <= h*: 20.0 <= 20.0
[CONDITION]: [OK] h(fail_lab) <= h*: 17.0 <= 17.0
[CONDITION]: [OK] h(pass_continuous) <= h*: 1.0 <= 1.0
[CONDITION]: [OK] h(pass_course) <= h*: 0.0 <= 0.0
[CONDITION]: [OK] h(pass_exam) <= h*: 1.0 <= 1.0
[CONCLUSION]: Heuristic is optimistic.
```

Checking if heuristic is consistent for the `ai_pass.txt` heuristic:

```
# HEURISTIC-CONSISTENT ai_pass.txt
[CONDITION]: [OK] h(complete_lab) <= h(fail_continuous) + c: 13.0 <= 17.0 + 1.0
[CONDITION]: [OK] h(complete_lab) <= h(pass_continuous) + c: 13.0 <= 1.0 + 12.0
[CONDITION]: [OK] h(enroll_artificial_intelligence) <= h(complete_lab) + c: 17.0
    <= 13.0 + 4.0
[CONDITION]: [OK] h(enroll_artificial_intelligence) <= h(fail_lab) + c: 17.0 <=
    17.0 + 1.0
[CONDITION]: [OK] h(fail_continuous) <= h(fail_exam) + c: 17.0 <= 20.0 + 1.0
[CONDITION]: [OK] h(fail_continuous) <= h(pass_exam) + c: 17.0 <= 1.0 + 16.0
[CONDITION]: [OK] h(fail_exam) <= h(fail_course) + c: 20.0 <= 0.0 + 20.0
[CONDITION]: [OK] h(fail_lab) <= h(complete_lab) + c: 17.0 <= 13.0 + 4.0
[CONDITION]: [OK] h(fail_lab) <= h(fail_course) + c: 17.0 <= 0.0 + 20.0
[CONDITION]: [OK] h(fail_lab) <= h(fail_lab) + c: 17.0 <= 17.0 + 1.0
[CONDITION]: [OK] h(pass_continuous) <= h(pass_course) + c: 1.0 <= 0.0 + 1.0
[CONDITION]: [OK] h(pass_exam) <= h(pass_course) + c: 1.0 <= 0.0 + 1.0
[CONCLUSION]: Heuristic is consistent.
```

2. Trip to Buzet

In the following outputs we will use the state space descriptor `istra.txt`.

Breadth first search:

```
# BFS
[FOUND_SOLUTION]: yes
[STATES_VISITED]: 11
[PATH_LENGTH]: 5
[TOTAL_COST]: 100.0
[PATH]: Pula => Barban => Labin => Lupoglav => Buzet
```

Uniform cost search:

```
# UCS
[FOUND_SOLUTION]: yes
[STATES_VISITED]: 17
[PATH_LENGTH]: 5
[TOTAL_COST]: 100.0
[PATH]: Pula => Barban => Labin => Lupoglav => Buzet
```

A* algorithm + heuristic `istra_heuristic.txt`:

```
# A-STAR istra_heuristic.txt
[FOUND_SOLUTION]: yes
[STATES_VISITED]: 14
[PATH_LENGTH]: 5
[TOTAL_COST]: 100.0
[PATH]: Pula => Barban => Labin => Lupoglav => Buzet
```

A* algorithm + heuristic `istra_pessimistic_heuristic.txt`:

```
# A-STAR istra_pessimistic_heuristic.txt
[FOUND_SOLUTION]: yes
[STATES_VISITED]: 13
[PATH_LENGTH]: 7
[TOTAL_COST]: 102.0
[PATH]: Pula => Vodnjan => Kanfanar => Žminj => Pazin => Motovun => Buzet
```

Checking if heuristic is optimistic for the `istra_pessimistic_heuristic.txt` heuristic:

```
# HEURISTIC-OPTIMISTIC istra_pessimistic_heuristic.txt
[CONDITION]: [OK] h(Baderna) <= h*: 25.0 <= 57.0
[CONDITION]: [OK] h(Barban) <= h*: 35.0 <= 72.0
[CONDITION]: [OK] h(Buje) <= h*: 21.0 <= 41.0
[CONDITION]: [OK] h(Buzet) <= h*: 0.0 <= 0.0
[CONDITION]: [OK] h(Grožnjan) <= h*: 17.0 <= 33.0
[CONDITION]: [OK] h(Kanfanar) <= h*: 30.0 <= 61.0
[CONDITION]: [OK] h(Labin) <= h*: 35.0 <= 57.0
[CONDITION]: [ERR] h(Lupoglav) <= h*: 35.0 <= 15.0
[CONDITION]: [OK] h(Medulin) <= h*: 61.0 <= 109.0
[CONDITION]: [OK] h(Motovun) <= h*: 12.0 <= 18.0
[CONDITION]: [OK] h(Opatija) <= h*: 26.0 <= 44.0
[CONDITION]: [ERR] h(Pazin) <= h*: 40.0 <= 38.0
```

```
[CONDITION]: [OK] h(Poreč) <= h*: 32.0 <= 71.0
[CONDITION]: [OK] h(Pula) <= h*: 57.0 <= 100.0
[CONDITION]: [OK] h(Rovinj) <= h*: 40.0 <= 79.0
[CONDITION]: [OK] h(Umag) <= h*: 31.0 <= 54.0
[CONDITION]: [OK] h(Višnjan) <= h*: 20.0 <= 52.0
[CONDITION]: [OK] h(Vodnjan) <= h*: 47.0 <= 90.0
[CONDITION]: [OK] h(Žminj) <= h*: 27.0 <= 55.0
[CONCLUSION]: Heuristic is not optimistic.
```

Checking if heuristic is consistent for the `istra_pessimistic_heuristic.txt` heuristic:

```
# HEURISTIC-CONSISTENT istra_pessimistic_heuristic.txt
[CONDITION]: [OK] h(Baderna) <= h(Kanfanar) + c: 25.0 <= 30.0 + 19.0
[CONDITION]: [OK] h(Baderna) <= h(Pazin) + c: 25.0 <= 40.0 + 19.0
[CONDITION]: [OK] h(Baderna) <= h(Poreč) + c: 25.0 <= 32.0 + 14.0
[CONDITION]: [OK] h(Baderna) <= h(Višnjan) + c: 25.0 <= 20.0 + 13.0
[CONDITION]: [OK] h(Barban) <= h(Labin) + c: 35.0 <= 35.0 + 15.0
[CONDITION]: [OK] h(Barban) <= h(Pula) + c: 35.0 <= 57.0 + 28.0
[CONDITION]: [OK] h(Buje) <= h(Grožnjan) + c: 21.0 <= 17.0 + 8.0
[CONDITION]: [OK] h(Buje) <= h(Umag) + c: 21.0 <= 31.0 + 13.0
[CONDITION]: [OK] h(Buzet) <= h(Lupoglav) + c: 0.0 <= 35.0 + 15.0
[CONDITION]: [OK] h(Buzet) <= h(Motovun) + c: 0.0 <= 12.0 + 18.0
[CONDITION]: [OK] h(Grožnjan) <= h(Buje) + c: 17.0 <= 21.0 + 8.0
[CONDITION]: [OK] h(Grožnjan) <= h(Motovun) + c: 17.0 <= 12.0 + 15.0
[CONDITION]: [OK] h(Grožnjan) <= h(Višnjan) + c: 17.0 <= 20.0 + 19.0
[CONDITION]: [OK] h(Kanfanar) <= h(Baderna) + c: 30.0 <= 25.0 + 19.0
[CONDITION]: [OK] h(Kanfanar) <= h(Rovinj) + c: 30.0 <= 40.0 + 18.0
[CONDITION]: [OK] h(Kanfanar) <= h(Vodnjan) + c: 30.0 <= 47.0 + 29.0
[CONDITION]: [OK] h(Kanfanar) <= h(Žminj) + c: 30.0 <= 27.0 + 6.0
[CONDITION]: [OK] h(Labin) <= h(Barban) + c: 35.0 <= 35.0 + 15.0
[CONDITION]: [OK] h(Labin) <= h(Lupoglav) + c: 35.0 <= 35.0 + 42.0
[CONDITION]: [ERR] h(Lupoglav) <= h(Buzet) + c: 35.0 <= 0.0 + 15.0
[CONDITION]: [OK] h(Lupoglav) <= h(Labin) + c: 35.0 <= 35.0 + 42.0
[CONDITION]: [OK] h(Lupoglav) <= h(Opatija) + c: 35.0 <= 26.0 + 29.0
[CONDITION]: [OK] h(Lupoglav) <= h(Pazin) + c: 35.0 <= 40.0 + 23.0
[CONDITION]: [OK] h(Medulin) <= h(Pula) + c: 61.0 <= 57.0 + 9.0
[CONDITION]: [OK] h(Motovun) <= h(Buzet) + c: 12.0 <= 0.0 + 18.0
[CONDITION]: [OK] h(Motovun) <= h(Grožnjan) + c: 12.0 <= 17.0 + 15.0
[CONDITION]: [OK] h(Motovun) <= h(Pazin) + c: 12.0 <= 40.0 + 20.0
[CONDITION]: [OK] h(Opatija) <= h(Lupoglav) + c: 26.0 <= 35.0 + 29.0
[CONDITION]: [OK] h(Pazin) <= h(Baderna) + c: 40.0 <= 25.0 + 19.0
[CONDITION]: [OK] h(Pazin) <= h(Lupoglav) + c: 40.0 <= 35.0 + 23.0
[CONDITION]: [ERR] h(Pazin) <= h(Motovun) + c: 40.0 <= 12.0 + 20.0
[CONDITION]: [OK] h(Pazin) <= h(Žminj) + c: 40.0 <= 27.0 + 17.0
[CONDITION]: [OK] h(Poreč) <= h(Baderna) + c: 32.0 <= 25.0 + 14.0
[CONDITION]: [OK] h(Pula) <= h(Barban) + c: 57.0 <= 35.0 + 28.0
[CONDITION]: [OK] h(Pula) <= h(Medulin) + c: 57.0 <= 61.0 + 9.0
[CONDITION]: [OK] h(Pula) <= h(Vodnjan) + c: 57.0 <= 47.0 + 12.0
[CONDITION]: [OK] h(Rovinj) <= h(Kanfanar) + c: 40.0 <= 30.0 + 18.0
[CONDITION]: [OK] h(Umag) <= h(Buje) + c: 31.0 <= 21.0 + 13.0
[CONDITION]: [OK] h(Višnjan) <= h(Baderna) + c: 20.0 <= 25.0 + 13.0
[CONDITION]: [OK] h(Višnjan) <= h(Grožnjan) + c: 20.0 <= 17.0 + 19.0
[CONDITION]: [OK] h(Vodnjan) <= h(Kanfanar) + c: 47.0 <= 30.0 + 29.0
[CONDITION]: [OK] h(Vodnjan) <= h(Pula) + c: 47.0 <= 57.0 + 12.0
```

```
[CONDITION]: [OK] h(Žminj) <= h(Kanfanar) + c: 27.0 <= 30.0 + 6.0
[CONDITION]: [OK] h(Žminj) <= h(Pazin) + c: 27.0 <= 40.0 + 17.0
[CONCLUSION]: Heuristic is not consistent.
```

3. 8-puzzle

In the following outputs we will use the state space descriptor `3x3_puzzle.txt`.

A* algorithm + heuristic `3x3_misplaced_heuristic.txt`:

```
# A-STAR 3x3_misplaced_heuristic.txt
[FOUND_SOLUTION]: yes
[STATES_VISITED]: 95544
[PATH_LENGTH]: 31
[TOTAL_COST]: 30.0
[PATH]: 876_543_21x => 876_54x_213 => 876_5x4_213 => 876_x54_213 => 876_254_x13
    => 876_254_1x3 => 876_2x4_153 => 876_24x_153 => 876_243_15x => 876_243_1x5
    => 876_2x3_145 => 8x6_273_145 => x86_273_145 => 286_x73_145 => 286_173_x45
    => 286_173_4x5 => 286_1x3_475 => 2x6_183_475 => 26x_183_475 => 263_18x_475
    => 263_1x8_475 => 2x3_168_475 => x23_168_475 => 123_x68_475 => 123_468_x75
    => 123_468_7x5 => 123_468_75x => 123_46x_758 => 123_4x6_758 => 123_456_7x8
    => 123_456_78x
```