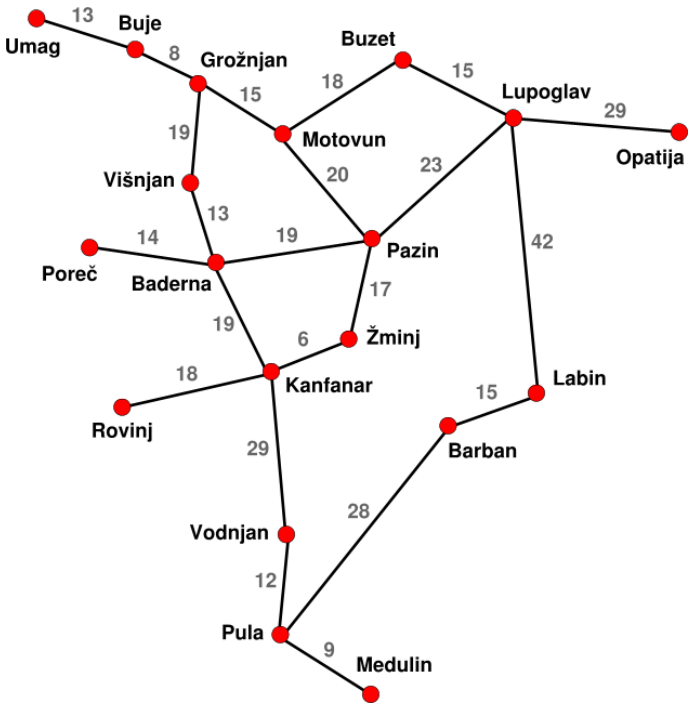
Artificial Intelligence – Programming assignment 1

UNIZG FER, academic year 2019/20.

Handed out: 16. March 2020 Due: 29. March 2020 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

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 as well as the course repository. 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, which 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 **(4 points)**
2. Uniform cost search **(4 points)**
3. The A* algorithm **(4 points)**

Your implementation for each of the aforementioned algorithms should print: (1) the length (and cost) of the found solution, (2) the path of the solution (list of visited states for the solution, from the start state up to the goal state) and (3) the number of visited states (the size of the *closed* list). An example of such output for the A* algorithm for the task of trip through Istria is as follows:

```
States visited = 14
Found path of length 5 with total cost 100:
Pula =>
Barban =>
Labin =>
Lupoglav =>
Buzet
```

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)**

Your implementation of these functions should print **each** case in which either of the optimistic or consistency constraints is not satisfied. As an example, when checking whether a heuristic is optimistic, you need to print each state in which the heuristic function **overestimates** the true cost needed to reach the goal, and for how much is the true cost overestimated. An example of the output for a bad heuristic for the trip through Istria is as follows:

```
Checking if heuristic is optimistic.
  [ERR] h(Lupoglav) > h*: 35.0 > 15
  [ERR] h(Pazin) > h*: 40.0 > 37
Heuristic is not optimistic

Checking if heuristic is consistent.
  [ERR] h(Lupoglav) > h(Buzet) + c: 35.0 > 0.0 + 15
  [ERR] h(Pazin) > h(Motovun) + c: 40.0 > 12.0 + 20
Heuristic is not consistent
```

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 points assignment: the complexity of the 8-puzzle (+6 points)

Note: Your solution for each extra assignment should be submitted to the Ferko system in the same archive as the rest of the lab assignment solution. 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 (2 points)

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 (2 points)

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!

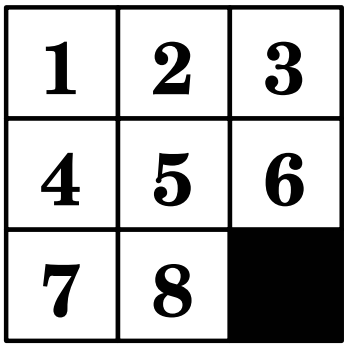
3. Designing heuristic functions (2 points)

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

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.

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.

Data loading:

```
Start state: enroll_artificial_intelligence
End state(s): ['pass_course', 'fail_course']
State space size: 9
Total transitions: 12
```

Breadth first search:

```
Running bfs:
States visited = 4
Found path of length 3:
enroll_artificial_intelligence =>
fail_lab =>
fail_course
```

Uniform cost search:

```
Running ucs:
States visited = 7
Found path of length 4 with total cost 17.0:
enroll_artificial_intelligence =>
complete_lab =>
pass_continuous =>
pass_course
```

A* algorithm + heuristic `ai_fail.txt`:

```
Running astar:
States visited = 6
Found path of length 3 with total cost 21.0:
enroll_artificial_intelligence =>
fail_lab =>
fail_course
```

Heuristic checks for the `ai_fail.txt` heuristic:

```
Checking heuristic
Checking if heuristic is optimistic.
  [ERR] h(pass_continuous) > h*: 20.0 > 1.0
Heuristic is not optimistic
Checking if heuristic is consistent.
  [ERR] h(complete_lab) > h(fail_continuous) + c: 10.0 > 6.0 + 1.0
  [ERR] h(enroll_artificial_intelligence) > h(fail_lab) + c: 17.0 > 1.0 + 1.0
  [ERR] h(enroll_artificial_intelligence) > h(complete_lab) + c: 17.0 > 10.0 + 4.0
  [ERR] h(pass_continuous) > h(pass_course) + c: 20.0 > 0.0 + 1.0
Heuristic is not consistent
```

A* algorithm + heuristic `ai_pass.txt`:

```
Running astar:
States visited = 4
Found path of length 4 with total cost 17.0:
enroll_artificial_intelligence =>
complete_lab =>
pass_continuous =>
pass_course
```

Heuristic checks for the `ai_pass.txt` heuristic:

```
Checking heuristic
Checking if heuristic is optimistic.
Heuristic is optimistic
Checking if heuristic is consistent.
Heuristic is consistent
```

2. Trip to Buzet

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

Data loading:

```
Start state: Pula
End state(s): ['Buzet']
State space size: 19
Total transitions: 44
```

Breadth first search:

```
Running bfs:
States visited = 15
Found path of length 5:
Pula =>
Barban =>
Labin =>
Lupoglav =>
Buzet
```

Uniform cost search:

```
Running ucs:
States visited = 17
Found path of length 5 with total cost 100:
Pula =>
Barban =>
Labin =>
Lupoglav =>
Buzet
```

A* algorithm + heuristic `istra_heuristic.txt`:

```
Running astar:
States visited = 14
Found path of length 5 with total cost 100:
Pula =>
Barban =>
Labin =>
Lupoglav =>
Buzet
```

A* algorithm + heuristic `istra_pessimistic_heuristic.txt`:

```
Running astar:
States visited = 13
Found path of length 7 with total cost 102:
Pula =>
Vodnjan =>
Kanfanar =>
Žminj =>
Pazin =>
Motovun =>
Buzet
```

Heuristic checks for the `istra_heuristic.txt` heuristic:

```
Checking if heuristic is optimistic.
Heuristic is optimistic
Checking if heuristic is consistent.
Heuristic is consistent
```

Heuristic checks for the `istra_pessimistic_heuristic.txt` heuristic:

```
Checking if heuristic is optimistic.
  [ERR] h(Lupoglav) > h*: 35.0 > 15
  [ERR] h(Pazin) > h*: 40.0 > 38
Heuristic is not optimistic
Checking if heuristic is consistent.
  [ERR] h(Lupoglav) > h(Buzet) + c: 35.0 > 0.0 + 15
  [ERR] h(Pazin) > h(Motovun) + c: 40.0 > 12.0 + 20
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`:

```
Running astar:
States visited = 95544
Found path of length 31 with total cost 30:
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
```

Heuristic checks for the `3x3_misplaced_heuristic.txt` heuristic:

```
Checking if heuristic is consistent.
  [ERR] 5040 errors, omitting output.
Heuristic is not consistent
Checking if heuristic is optimistic.
  [ERR] 78 errors, omitting output.
Heuristic is not optimistic
```