Grundlagen der Künstlichen Intelligenz

Programming Exercise 1: Search — 23th November 2018

Markus Koschi, Behtarin Ferdousi, and Yang Liu (last updated 23rd November 2018)

The programming exercise on search is divided into two subproblem. If you pass both Problem 1.1 and 1.2, you have passed the programming exercise 1 on search.

Problem 1.1: Gathering allies while avoiding obstacles

A robot, who is hell-bent on conquering the universe, needs to travel to different planets in order to gather allies. Help it to navigate through asteroid belts (the obstacles) before its fuel runs out.

We illustrate the problem with a matrix, where $\boxed{*}$ represents obstacles in the astroid belt, $\boxed{-}$ valid positions for a path, $\boxed{X}$ the position of the ally (hence the goal destination), and $\boxed{R}$ the starting position.

Going different directions costs different amount of fuel:

To move left/right:	5 units
To move up/down:	6 units
To move diagonally:	10 units

Calculate the minimum units of fuel (in integer) the robot needs to reach the ally. If not possible to reach the ally, print "No path found!".

Example
Input: a (4×5) matrix

$$\begin{array}{ccccc} - & - & - & - & - \\ - & - & - & * & - \\ - & - & - & * & - \\ R & - & * & * & X \end{array}$$

Output:
52

Optimal path:

Move	New Matrix	Cost
-	$\begin{matrix} \_ & \_ & \_ & \_ & \_ \\ \_ & \_ & \_ & * & \_ \\ \_ & \_ & \_ & * & \_ \\ R & \_ & * & * & X \end{matrix}$	-
Diagonal	$\begin{matrix} \_ & \_ & \_ & \_ & \_ \\ \_ & \_ & \_ & * & \_ \\ \_ & R & \_ & * & \_ \\ \_ & \_ & * & * & X \end{matrix}$	10
Diagonal	$\begin{matrix} \_ & \_ & \_ & \_ & \_ \\ \_ & \_ & R & * & \_ \\ \_ & \_ & \_ & * & \_ \\ \_ & \_ & * & * & X \end{matrix}$	10
Diagonal	$\begin{matrix} \_ & \_ & \_ & R & \_ \\ \_ & \_ & \_ & * & \_ \\ \_ & \_ & \_ & * & \_ \\ \_ & \_ & * & * & X \end{matrix}$	10
Diagonal	$\begin{matrix} \_ & \_ & \_ & \_ & \_ \\ \_ & \_ & \_ & * & R \\ \_ & \_ & \_ & * & \_ \\ \_ & \_ & * & * & X \end{matrix}$	10
Down	$\begin{matrix} \_ & \_ & \_ & \_ & \_ \\ \_ & \_ & \_ & * & \_ \\ \_ & \_ & \_ & * & R \\ \_ & \_ & * & * & X \end{matrix}$	6
Down	$\begin{matrix} \_ & \_ & \_ & \_ & \_ \\ \_ & \_ & \_ & * & \_ \\ \_ & \_ & \_ & * & \_ \\ \_ & \_ & * & * & RX \end{matrix}$	6
	Total Cost:	52

This can be modeled as a search problem, where each node represents a possible path that the robot can take. You should implement a search algorithm to find the optimal path (i.e. minimum fuel used) from the start position to the goal destination. Note that some problems might have more than one possible paths that give the minimum cost or even no solution. If you print the minimum cost (integer) for all possible test cases and "No path found!" otherwise, you pass task 1.

Problem 1.2: Conquering space

Now that you have helped the robot to gather enough allies, it is time to battle. We use a matrix again to model this problem, where $\boxed{X}$ represents the ally's spaceship and $\boxed{\cdot}$ (a dot) the enemy. The robot's army conquers all enemies which are fully surrounded by allies (i.e. on all four sides) and turns them into allies. Thus, all fully surrounded $\boxed{\cdot}$ are replaced by $\boxed{X}$.

Example
Input: a (5×5) matrix

$$\begin{matrix} X & X & X & X & X \\ . & . & X & . & X \\ X & . & X & X & X \\ X & X & . & . & X \\ X & X & X & X & X \end{matrix}$$

Output:

X	X	X	X	X
.	.	X	$\boldsymbol{X}$	X
X	.	X	X	X
X	X	$\boldsymbol{X}$	$\boldsymbol{X}$	X
X	X	X	X	X

Explanation: The enemy ships which are surrounded from left, right, up, and down will be conquered. However, enemies on the border are not fully surrounded and can escape the siege.

This can also be modeled as a search problem. For this task there is only one valid **output matrix**. Please write a function or use existing packages to output the resulting matrix to a file named `output_for_task2.txt`. If the output matrix is correct for different test cases, you pass task 2.

Submission information

You may implement this in Java, Python, or MATLAB (only in the version specified below). Your submission should consist of the following files at least:

1. `readme.txt` – details on the version of Java/Python/Matlab you used as well as a description of your solution, and any packages you used
2. `Solution1.java / Solution1.py / Solution1.m` – see below for details
3. `Solution2.java / Solution2.py / Solution2.m` – details below hold analogously
4. `requirements.txt` – if you use python modules; see below for details

These files should be directly in the root directory of your submission folder (not in sub-directories or compressed files). Do not use absolute paths, but paths relative to the working directory.

Your code shall read `input_for_task1.txt` and `input_for_task2.txt` as input for task 1 and 2, respectively. See an example of each file on Moodle. The output matrix for task 2 must be the written to a file named `output_for_task2.txt`, which is saved in the root directory. An example is provided in Moodle.

Grading

To summarize, your solution counts as a pass only if:

1. In **task 1**, your program prints the minimum cost (if exists) for every test cases (even with other test cases that are not given to you).
2. In **task 2**, your program gives the correct output matrix for every test cases (even with other test cases that are not given to you).
3. You have not copied your solution (use of existing packages is allowed as long as cited in the readme). We will use a plagiarism detection tool and any copied code will annul all bonus exercises from both the copier and the copied person!

Submission will close on **Friday, 21.12.2018 at 23:59**. Your solution will be marked using a shell script. You can either pass (if all requirements listed above are met) or fail (unfortunately, we cannot manually check for minor errors). Thus, it is very important to follow the instructions exactly!

We offer a preliminary check of your solution: All solutions uploaded until Thursday, 13.12.2018 at 23:59 will be checked for formatting errors, i.e. whether your code runs on our machine, reads an input file, and prints a formally correct output (e.g. correct file name). An automated feedback will be submitted via Moodle until Monday, 17.12.2018 so that you can (re)submit a formally correct solution until the main submission deadline.

Java 9

If you implement this in Java 9, we will compile and execute:

```
>> javac Solution1.java
>> java Solution1 input_for_task1.txt
```

Python 3.6

If you implement this in Python 3.6, we will execute:

```
>> python Solution1.py input_for_task1.txt
```

You may use python modules/packages. If they are not distributed with Python by default, we need to install them automatically in a virtualenv. Thus, you have to list all of them in separate lines in a txt file named `requirements.txt`[1], which contains e.g.

```
numpy
queue
pythonds
```

MATLAB 2018b

If you implement this in MATLAB 2018b, we will execute in the MATLAB command line:

```
>> Solution1('input_for_task1.txt')
```

[1] `https://pip.pypa.io/en/latest/user_guide/#requirements-files`