Homework 1 - Deterministic Search

Introduction

In this exercise you take the role of a head of the agency that tries to stop a virus from spreading. You have two main options for stopping the virus: lockdown and vaccination. Your goal is to combine these two methods in a simulated environment to clear a certain area from the virus completely.

To apply them in the most efficient manner you must make use of the search algorithms shown in class. For that, you will need to model the problem precisely

Environment

The environment is a rectangular grid - given as the tuple of tuples (exact representation can be found in the “Input” section). Each point on a grid represents an area and population in it. Codes for the state of population in area are as follows:

- ‘U’ - unpopulated. Unpopulated areas can’t be affected by infection, and can’t spread it
- ‘H’ - healthy. Healthy population can be infected, and does not spread the virus
- ‘S’ - sick. Sick populations spread disease
- ‘I’ - immune. Populations behave as healthy, but can’t be affected by illness. A population can become immune through vaccination
- ‘Q’ - quarantined. Quarantined populations are those that already have been affected by the disease, but can’t spread it further

Example of the environment:

0	1	2	3	4
U	H	H	H	S
S	U	H	I	H
Q	U	S	H	S
S	S	I	U	H
H	S	U	U	H

The top left corner tile is numbered (0,0), where the first number is the row number, i.e. the tile that has ‘Q’ in it is numbered (2, 0).

Actions

Fortunately you will be able to affect the situation. As an input to the problem you will receive an indication how many teams are at your disposal. There are two types of teams: police and medical. Police teams can enforce quarantine, while medical teams can vaccinate the population. Every turn each team will execute at most one action. The conditions for actions are as follows:

- For political reasons, police teams can only quarantine populations that are already sick
- Only healthy populations can be vaccinated by medical teams
- Unpopulated areas and populations that are already immune can be neither quarantined nor vaccinated

Other than that, there are no other preconditions for actions.

Format of actions should be tuple of pairs (tuples) where the first item in each pair is the action and the second item is the coordinate where it should apply. For example this is a valid action on the environment above, assuming you have at least 1 medical and 1 police team:

((“vaccinate”, (0, 1)), (“quarantine”, (1, 0)))

Performing this action will turn the tile (0, 1) to be ‘I’, and tile ‘(1, 0)’ to ‘Q’

Note: you don’t have to use all the teams at every given turn.

Dynamics of the system

The system changes depending on your actions every turn. Turn consists of the following stages:

First, you choose which actions to perform.
Second, your chosen actions take effect:

- “vaccinate” turns a tile ‘H’ into ‘I’ permanently
- “quarantine” turns a tile ‘S’ into ‘Q’ temporarily (for 2 turns)

After that, the infection spreads, i.e. all tiles that are ‘H’ and have a neighbor that is ‘S’ become ‘S’. Neighbor is a tile directly above, below, to the right, or to the left, **not** on a diagonal.

After that, the sickness expires, i.e. each tile that was ‘S’ for 3 turns becomes ‘H’ again (note that the recovery from infection provides no lasting immunity, so a recovered tile can be infected again!)

Finally, the quarantine expires, i.e. a tile that was ‘Q’ for 2 turns becomes ‘H’.

Note: every tile that has ‘Q’ or ‘S’ in the initial state is assumed to be in the beginning of the quarantine/sickness, and will last for 2/3 turns respectively.

Goals

Your goal is the complete eradication of the infection, i.e. a situation where at the end of the turn there is no tile that has ‘S’ in it. The area suffers immensely while the infection persists, so the quality metric of your plan is how many turns it takes to arrive to the solution (the less the better).
Other parameters, such as the amount of tiles infected do not matter.

Input and the task

As an input you will get a dictionary that describes the environment and the amount of teams that are available to you. For example:

```
{
    "police": 1,
    "medics": 1,
    "map": (
        ('U', 'H', 'S', 'H',),
        ('H', 'U', 'H', 'S',),
        ('U', 'H', 'H', 'U',),
    )
}
```

This input is given to the constructor of the class MedicalProblem as the variable “initial”. The variable “initial” in the constructor is then used to create the root node of the search, so you will have to transform it into your representation of the state somewhere before the `search.Problem.__init__(self, initial)` line.

Moreover, you have to implement the following functions in the MedicalProblem class:
`def actions(self, state)` - the function that returns all available actions from a given state.
`def result(self, state, action)` - the function that returns next state, given a previous state and an action.
`def goal_test(self, state)` - returns “True” if a given state is a goal, “False” otherwise.
`def h(self, node)` - returns a heuristic estimate of a given node.

Note: you are free to choose your own representation of the state. Only one restriction applies - the state should be *Hashable*. We **strongly suggest** not to create a new class for storing the state, and stick to the built-in data types.

Having implemented all the functions above, you may launch the GBFS search (already implemented in the code) by running check.py. Your code is expected to finish the task in 60 seconds.

Output

You may encounter one of the following outputs:

- A bug - self explanatory
- (-2, -2, None) - timeout (it took more than 60 seconds of real time for the algorithm to finish)
- A solution of the form (list of actions taken, run time, amount of turns)

Code handout

Code that you receive has 4 files:

1. ex1.py - the only file that you should modify, implements the specific problem
2. check.py - the file that includes some wrappers and inputs, the file that you should run
3. search.py - a file that has implementations of different search algorithms (including GBFS, A* and many more)
4. utils.py - the file that contains some utility functions. You may use the contents of this file as you see fit

Note: we do not provide any means to check whether the solution your code provided is correct, so it is your responsibility to validate your solutions.

Submission and grading

You are to submit **only** the file named ex1.py as a python file (no zip, rar etc.). We will run check.py with our own inputs, and your ex1.py, using GBFS as the search algorithm of choice. The check is fully automated, so it is important to be careful with names of functions and classes. The grades will be assigned as follows:

- 80% - if your code finishes solving the test inputs (where maps will not be larger than 8x8) within the timeout, with no regard to the cost of the solution
- 20% - Grading on the relative performance of the algorithm, which is judged by length of the solution only (as long it finishes within the timeout).
- The submission is due on the 29.11, at 23:59
- Submission in pairs/singles only.
- Write your ID numbers in the appropriate field (‘ids’ in ex1.py) as strings. If you submit alone, leave only one string.
- The name of the submitted file should be “ex1.py”. Do not change it.

Important notes

- Note that you can access the state of the node by using node.state (for calculating h for example)
- We encourage you to start from implementing a working system without a heuristic, and add the heuristic later.
- We encourage you to double-check the syntax of the actions as the check is automated.
- More inputs will be released a week after release of the exercise
- You may use any package that appears in the standard library and the Anaconda package list, however the exercise is built in a way that most packages will be useless.
- We encourage you not to optimize your code prematurely. Simpler solutions tend to work best.