Deterministic Search

Introduction

Taking the role of a head of the agency that tries to stop a virus from spreading. There are two main options for stopping the virus: lockdown and vaccination. The 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, need to make use of a deterministic search algorithm. For that, 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 the virus. 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, it is possible to affect the situation. As an input to the problem, will receive an indication how many teams are at the 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:

● 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 having 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: no need 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, choosing which actions to perform. Second, 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

The goal is, 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 the plan is how many turns it takes to arrive to the solution (the less the better).
Other parameters, such as the number of tiles infected do not matter.

Input

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',),
)
}
```

Solution

Functions definitions

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 need to transform it into my representation of the state with a constructor.
Moreover, need 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.

Output

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, number of turns)