Homework 2 - Partial observability

Introduction

In this exercise you will assume the role of the auditor brought after the epidemic to analyze the dynamics of the disease and the adequacy of the government response to it. Some data, however, had gone missing. Your mission is to uncover the true state of some unobserved tiles using logical inference techniques taught in class.

Task

As an input you will be given a series of observed environments, each one partially representing a true environment. The true environment is generated via the system dynamics as presented in exercise 1, in the presence of an independent agent (government) trying to stop the spread of the disease. This agent has the same actions as described in exercise 1, with exactly the same effects.

Observed state

The state of the system is extremely similar to the one described in the first exercise. The true state is unknown to you, but you have access to some observations. The true state evolves according to the dynamics described in exercise 1, with **arbitrary** decisions of the agent responsible for vaccination and quarantine.

Observations are states of the map, where tiles that are marked '?' are unobserved.

Generally, the 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.
- '?' - unknown. These are the tiles you cannot observe, and in the true state can contain either of the five codes above.

Example of an observation:

0	1	2	3	4
U	H	?	H	S
S	U	H	I	H
Q	?	?	H	?
S	S	?	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).

Input

The input includes:

1. Number of teams available to the agent
2. Sequence of states observed
3. Queries

Example:

- “police”: 0
- “medics”: 0
- “observations”: (

H	?
H	H

,

S	?
?	S

)

- “queries”: [((0, 1), 0, “H”), ((1, 0), 1, “S”)]

The meaning of this input is as follows: there are 0 medics and 0 police forces, at turn 0 (before the agent is taking an action) there is a 2x2 board where the status of the tiles (0, 0), (1, 0) and (1, 1) is ”H”, and the status of (0, 1) is unknown. At turn 1, the status of (0, 0) and (1, 1) changes to “S”, and the status of (0, 1) and (1, 0) is now unknown.

Queries are ((0, 1), 0, “H”) and ((1, 0), 1, “S”), i.e. your mission is to determine whether tile (0, 1) was healthy at turn 0, and whether (1, 0) was sick at turn 1.
Note: In contrast to HW1, there will be no ‘Q’ or ‘I’ tiles in the first state in sequence.
Note 2: Assume the agent uses every team at its disposal if it is possible
Note 3: Turn 0 means the initial state (the first one in the sequence given)

Output

Your code should return an answer for every query in the input. The answers are to be a dictionary, where a key should be the query they answer, and the key should be the answer itself. The answers are to be one of the following:

- ‘T’ - if the statement in query is guaranteed to be true
- ‘F’ - if the statement in query is guaranteed to be false
- ‘?’ - if the statement in query can neither be proven true nor proven false

For the example above, you should return the following dictionary:
{
((0, 1), 0, “H”) : ‘F’,
((1, 0), 1, “S”) : ‘F’
}

The dictionary must be returned within the timeout of 300 seconds.

Code handout

Code that you receive has 3 files:

1. ex2.py - the only file that you should modify, implements your solution
2. check.py - the file that includes some wrappers and inputs, the file that you should run
3. utils.py - the file that contains some utility functions. You may use the contents of this file as you see fit

Additional resource: You may download and use the pysat library to solve SAT problems efficiently. If there are other libraries that you’d like to use, send us an email, and we may allow such a resource too.

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 ex2.py as a python file (no zip, rar etc.). We will run check.py with our own inputs, and your ex2.py, comparing the results with the true values. The check is fully automated, so it is important to be careful with the syntax. The grades will be assigned as follows:

- 80% - if your code finishes solving the test inputs. Test inputs will not be significantly more demanding than the ones we publish. The solution must be returned within the timeout.
- 20% - Grading on the relative performance on larger inputs - the submission that solves the most inputs from a set of large inputs gets a 20/20, others are graded proportionally.
- The submission is due on the 28.12, at 23:59
- Submission in pairs/singles only. You may switch partners from HW1 if you wish to do so.
- Write your ID numbers in the appropriate field ('ids' in ex2.py) as strings. If you submit alone, leave only one string.
- The name of the submitted file should be "ex2.py". Do not change it.

Important notes

- We encourage you to double-check the syntax of the output as the check is automated.
- More inputs will be released in the second 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.
- You may import utils.py and use any function/class from there
- We encourage you not to optimize your code prematurely. Simpler solutions tend to work best.