Introduction to Reinforcement Learning

Assignment 1

February 12, 2023

Syed Mustafa (sm06554)

Submitted to: Dr. Shahid Shaikh

CS 352 Introduction to Reinforcement Learing

Semester: Spring 2023

Computer Engineering

Batch of 2024

Contents

1 Theory

Question 1

x Points

Exercise 3.1: Devise three example tasks of your own that fit into the reinforcement learning framework, identifying for each its states, actions, and rewards. Make the three examples as different from each other as possible. The framework is abstract and flexible and can be applied in many different ways. Stretch its limits in some way in at least one of your examples.

Answer 1

1. **Traffic Flow Optimization:**
   (a) The environment would be the road structure. For example, it could be an intersection where four roads meet. The environment could be shown as a grid. Where each block of the grid signifies a road. The grid can be coded as a 2-D array
   (b) States: The number of cars on each road could be shown by the value of the 2D array. So, the value would be against the road position on the grid.
   (c) Action: Changing of traffic lights, so that vehicles could move from one block of the grid to the next, and eventually off the grid.
   (d) Rewards: One can model reward as follows. +1 reward for every car that moves from one road to another.
2. **Mobile Robot Path Planning**:
   (a) The environment: The map the robot is currently on. Can be modeled as a graph.
   (b) States: Its position relative to the map.
   (c) Action: Going from one node of the map graph to another.
   (d) Rewards: Measured by the closeness to the actual goal.
3. **Chess Match**
   (a) The environment: The chessboard
   (b) States: The position of the chess pieces w.r.t board
   (c) Action: Changing the position of chess pieces
   (d) Rewards: Winning the game would give you a reward of +1, losing the game would result in a reward of -1, and taking an opponent's chess piece would be a reward of 0.01 * alpha.

Where alpha is a score given to a piece due to its importance in the game, i.e. pawn would be 1 while queen would be 50.

2 Programming

Question 2

x Points

Symmetries in tic-tac-toe

Let the row headings of a tic-tac-toe board be A, B, C, and the column headings be 1, 2, 3, so that the cells may be identified as A1, B2, etc. At any given instant during a game, the position of the board is a state of the game. In Python, the state may be saved as either a nested list (list of lists) or a dictionary such as 'A2': 'x', 'B3':'o'. Two positions of the board are rotationally equivalent–and can be represented by a single state–if one can be obtained from the other by 90∘, 180∘, or 270∘ rotation of the board. Write a program that accepts a dictionary as the position of the board and then determines if the position is a valid state, and: (a) prints an appropriate message, and returns a list containing an empty dictionary, if the input is not valid or is not a valid state; (b) prints an appropriate message, and returns an empty list, if the input is valid but has no rotational equivalents; (c) prints all the rotationally equivalent board positions and returns a list of corresponding dictionaries, if the input is valid.

Answer 2

```
from sympy.solvers.diophantine.diophantine import equivalent
from logging import raiseExceptions
def valid_pose(board):
    '''
    Parameters:
    1) Board := Dictionary ~ contains the state of the board, i.e. {A1: x, A2: y}
    returns:
    Truth value: Boolean variable
    '''
    x_count = y_count = 0
    for elm in board.values():
        if elm == 'x':
            x_count += 1
        elif elm == 'o':
```

```
            y_count += 1
        else:
            print(f"An␣error␣has␣occured")
            raiseExceptions
    truth_value = x_count == y_count or x_count == y_count +1 or x_count == y_count
        ↪ -1
    return truth_value


def rotational_equivalent(board):
    '''
    Parameters:
    1) Board := Dictionary ~ contains the state of the board, i.e. {A1: x, A2: y}
    returns:
    '''
    rotated_board = {}
    for key in board.keys():
        letter, col = tuple(key)
        col = int(col)-1
        lst = ['A', 'B', 'C']
        # letter = A || B || C
        row = ['A', 'B', 'C'].index(letter)
        new_row, new_col = col, 2 - row
        new_key = lst[new_row] + str(new_col + 1)
        rotated_board[new_key] = board[key]
    print(new_key)
    return rotated_board
rotational_equivalent({'A1': 'x', 'B2': 'o', 'C3': 'x'})



def get_equivalent(board):
    '''
    Param: Board
    Return: equivalent Boards
```

```
    Explaination: The board would rotate 3 times by 90 degrees
    '''
    equivalent_boards = [board]
    for i in range(3):
        rot_board = rotational_equivalent(board)
        if not rot_board == board: # For the case like B2
            equivalent_boards.append(rot_board)
    print(f'the rotationally = {equivalent_boards}')
    return equivalent_boards


def tic_tac_symmetry(board):
    if not valid_pose(board):
        print("The input is not a valid state.")
        return [{}]
    equivalent_boards =get_equivalent(board)
    if len(equivalent_boards) == 1:
        print("The input has no rotational equivalents.")
        return []
    for equivalent_board in equivalent_boards:
        print("Equivalent board:", equivalent_board)
    return equivalent_boards



board = { 'B2': 'x'}
equivalent_boards = tic_tac_symmetry(board)
```

Output

```
# Test Cases
board = { 'B2': 'x'}
equivalent_boards = tic_tac_symmetry(board)
board = {'A1': 'x', 'B2': 'o', 'C3': 'x'}
equivalent_boards = tic_tac_symmetry(board)
board = {'A1': '0', 'C2': 'o', 'C3': '0'}
equivalent_boards = tic_tac_symmetry(board)
board = {'A1': 'x', 'B2': 'o', 'C3': '0'}
equivalent_boards = tic_tac_symmetry(board)
board = {'A1': 'x', 'C2': 'o', 'A3': 'x'}
equivalent_boards = tic_tac_symmetry(board)
```

```
The input has no rotational equivalents.
Equivalent board: {'A1': 'x', 'B2': 'o', 'C3': 'x'}
Equivalent board: {'A3': 'x', 'B2': 'o', 'C1': 'x'}
Equivalent board: {'A3': 'x', 'B2': 'o', 'C1': 'x'}
Equivalent board: {'A3': 'x', 'B2': 'o', 'C1': 'x'}
An error has occured
An error has occured
Equivalent board: {'A1': '0', 'C2': 'o', 'C3': '0'}
Equivalent board: {'A3': '0', 'B1': 'o', 'C1': '0'}
Equivalent board: {'A3': '0', 'B1': 'o', 'C1': '0'}
Equivalent board: {'A3': '0', 'B1': 'o', 'C1': '0'}
An error has occured
Equivalent board: {'A1': 'x', 'B2': 'o', 'C3': '0'}
Equivalent board: {'A3': 'x', 'B2': 'o', 'C1': '0'}
Equivalent board: {'A3': 'x', 'B2': 'o', 'C1': '0'}
Equivalent board: {'A3': 'x', 'B2': 'o', 'C1': '0'}
Equivalent board: {'A1': 'x', 'C2': 'o', 'A3': 'x'}
Equivalent board: {'A3': 'x', 'B1': 'o', 'C3': 'x'}
Equivalent board: {'A3': 'x', 'B1': 'o', 'C3': 'x'}
Equivalent board: {'A3': 'x', 'B1': 'o', 'C3': 'x'}
```

Figure 1: Output for question 2. With Multiple different test cases

Question 3

x Points

Shortest Path Problem
In this problem we will implement the Shortest Path Algorithm that we discussed in the class. You have to write Python program that implements the algorithm discussed in the class as follows:

(i) The input to your program is a symmetric matrix (list of lists) representing a weighted undirected graph.

(ii) Your code should return two dictionaries: the first corresponding to the value function and the second corresponding to the optimal policy or policies.

(iii) You code should cater to the case of non-unique optimal policies (see example below).

You may use the `math` library but no other library such as `NumPy` or `SciPy`.

Example:

Answer 3

```
import math
from sympy.tensor.indexed import IndexException
def get_remaining( value_dic):
    key_tup = list(value_dic.keys())
    optimal = list(value_dic.values())
    remaining = []
    try:
        index = [ind for ind, ele in enumerate(optimal) if ele == math.inf]
    except:
        index = -1
    if not index == -1:
        for i in index:
            # print('this is us', key_tup[i])
            remaining.append(key_tup[i])
    return remaining

print(get_remaining( {'s':"no", 'a':math.inf,'ka':'just␣lidding','k':math.inf,'v':
    ↪ math.inf,'6':69}))
```

```
def min_index(lst):
    return [ind for ind, ele in enumerate(lst) if ele == min(lst)], min(lst)
a = [2,5,6,8,5,-1,4,-1,-1,-1,-69]
def sort_dict(myDict):
    myKeys = list(myDict.keys())
    myKeys.sort()
    return {i: myDict[i] for i in myKeys}


def get_value_function(matrix):
    j = 00
    value = {}
    policy = {}
    num_nodes = len(matrix)
    for i in range(len(matrix[-1])):
        value[i] = matrix[-1][i]
        if value[i] != math.inf:
            policy[i] = num_nodes -1
    while True:
        remaining_state = get_remaining(value)
        print(f"The remaining states are = {remaining_state}")
        for state in remaining_state:
            possible = {}
            ans = []
            for node in range(num_nodes):
                try:
                    possible[node].append((value[node]+matrix[node][state]))
                except:
                    possible[node] = (value[node]+matrix[node][state])
            print(f'The possible paths are = {possible} \n and the min values are = {
                ↪ min_index(list(possible.values()))}')
            possible_keys = list(possible.keys());
            possible_values = list(possible.values());
            min_lst, min_value = min_index(possible_values)
            value[state] = min_value
```

```
            if len(min_lst) > 1:
                policy[state] = min_lst
            else:
                policy[state] = min_lst[0]
        if len(remaining_state) == 0:
            print(f'All␣states␣now␣have␣the␣optimal␣value')
            break
    policy = sort_dict(policy)
    value = sort_dict(value)
    return value, policy
x =[[0, 2, 3, math.inf],
[2, 0, math.inf, 3],
[3, math.inf, 0, 2],
[math.inf, 3, 2, 0]]

test_case1 = [[0, 2, 4, 2, math.inf, math.inf, math.inf, math.inf],
[2, 0, math.inf, math.inf, 7, 4, 6, math.inf],
[4, math.inf, 0, math.inf, 3, 2, 1, math.inf],
[2, math.inf, math.inf, 0, 4, 1, 5, math.inf],
[math.inf, 7, 3, 4, 0, math.inf, math.inf, 7],
[math.inf, 4, 2, 1, math.inf, 0, math.inf, 9],
[math.inf, 6, 1, 5, math.inf, math.inf, 0, 6],
[math.inf, math.inf, math.inf, math.inf, 7, 9, 6, 0]]




print(get_value_function(test_case1))
```

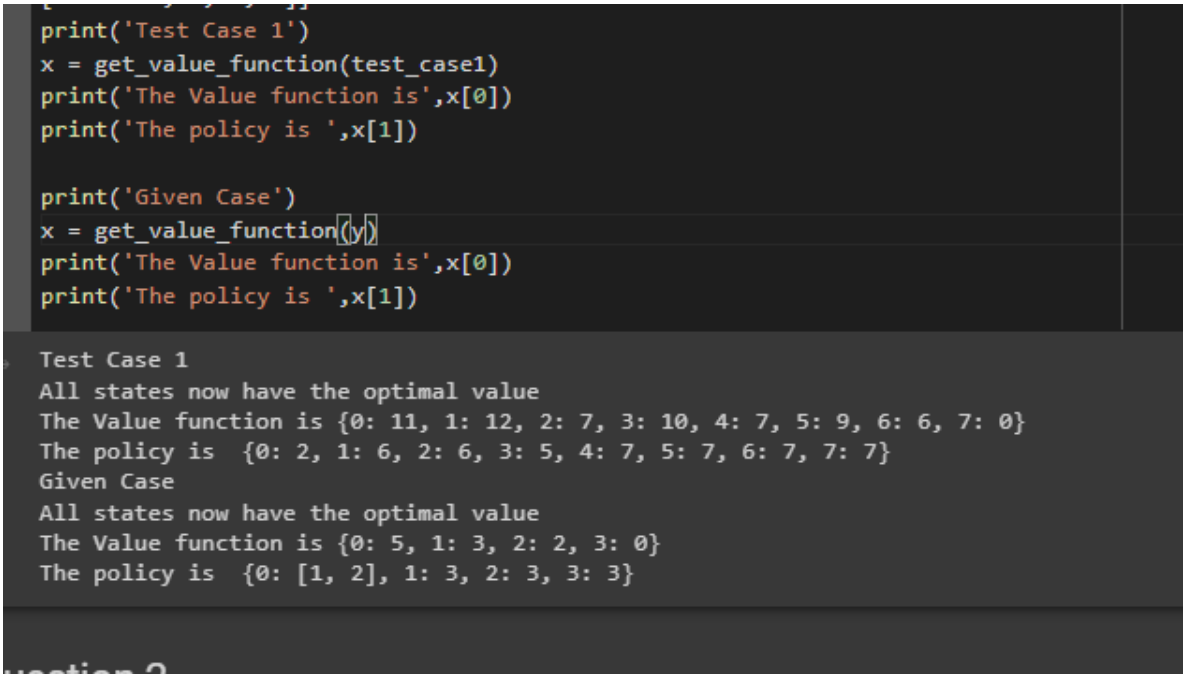

Figure 2: Output for question 3. With two different test cases. Test case 1 is the one we did in class

Note: The nodes of the graphs are such that 0 corresponds to the start node and last node (len(graph matrix)- 1) corresponds to the terminal node Please refer to the collab notebook link below **Link:**

https://colab.research.google.com/drive/1Cwv7O2bG3zc5kAfTQV6bNJDGucJXhyGi?usp=sharing