Assignment 3

Introduction

This assignment includes two parts, programming assignment and writing assignment. The deadline of this assignment is **13 Apr (11:59 PM), 2024**.

Part 1: Programming assignment

Download the code for this assignment here and then unzip the archive.

As in assignments 1 and 2, this assignment uses python 3. Do not use python 2. You can work on the assignment using your favorite python editor. We recommend VSCode.

Post any questions or issues with this assignment to our discussion forum. Alternatively you may also contact your TA directly.

This assignment uses autograding. We rely on the random number generator generating the same random sequence. Same as in Assignment 2. This time, we will be using the random.choices() function as follows.

```
import random
from collections import Counter
seed = 2
if seed!=-1:
    random.seed(seed, version=1)
n = 0.1 #noise
a = 'N' #intended action
d = {'N':['N', 'E', 'W'], 'E':['E', 'S', 'N'], 'S':['S', 'W', 'E'], 'W':['W', 'N', 'S']}
l = []
for _ in range(100000):
    l += random.choices(population=d[a], weights=[1 - n*2, n, n])[0]
print(Counter(l).keys()) # equals to list(set(words))
print(Counter(l).values()) # counts the elements' frequency
print(l[:5])
```

This will give the following output.

```
dict_keys(['W', 'N', 'E'])
dict_values([10025, 80059, 9916])
['W', 'W', 'N', 'N', 'E']
```

Note that we obtain ~80% intendent actions and 10% unintended actions here. Make sure that you understand the output and that you can reproduce it on your machine before proceeding. Note that we use anaconda python 3.9 to obtain the above result.

Problem 1: An MDP Episode (25 points)

In this part of the assignment, we are going to play an episode in an MDP by following a given policy. Consider the first test case of problem 1 (available in the file `test_cases/p1/1.prob`).

```
seed: 2
noise: 0.1
livingReward: -0.05
grid:
    _    _    _    1
    _    #    _   -1
    S    _    _    _
policy:
    E    E    E exit
    N    #    N exit
    N    W    N    W
```

The first part of this file specifies an MDP. `S` is the start state with four available actions (`N`, `E`, `S`, `W`), `_` is an ordinary state with the same four available actions and `1`,`-1` are states where the only available action is `exit` and the reward are `1` and `-1` respectively. The reward for action in other states is `-0.05`. `#` is a wall.
Actions are not deterministic in this environment. In this case with `noise = 0.1`, we are successfully acting 80% of the time and 20% of the time we will act perpendicular to the intended direction with equal probability, i.e. 10%, for each unintended direction. If the agent attempts to move into a wall, the agent will stay in the same position. Note that this MDP is identical to the example that we covered extensively in our class.

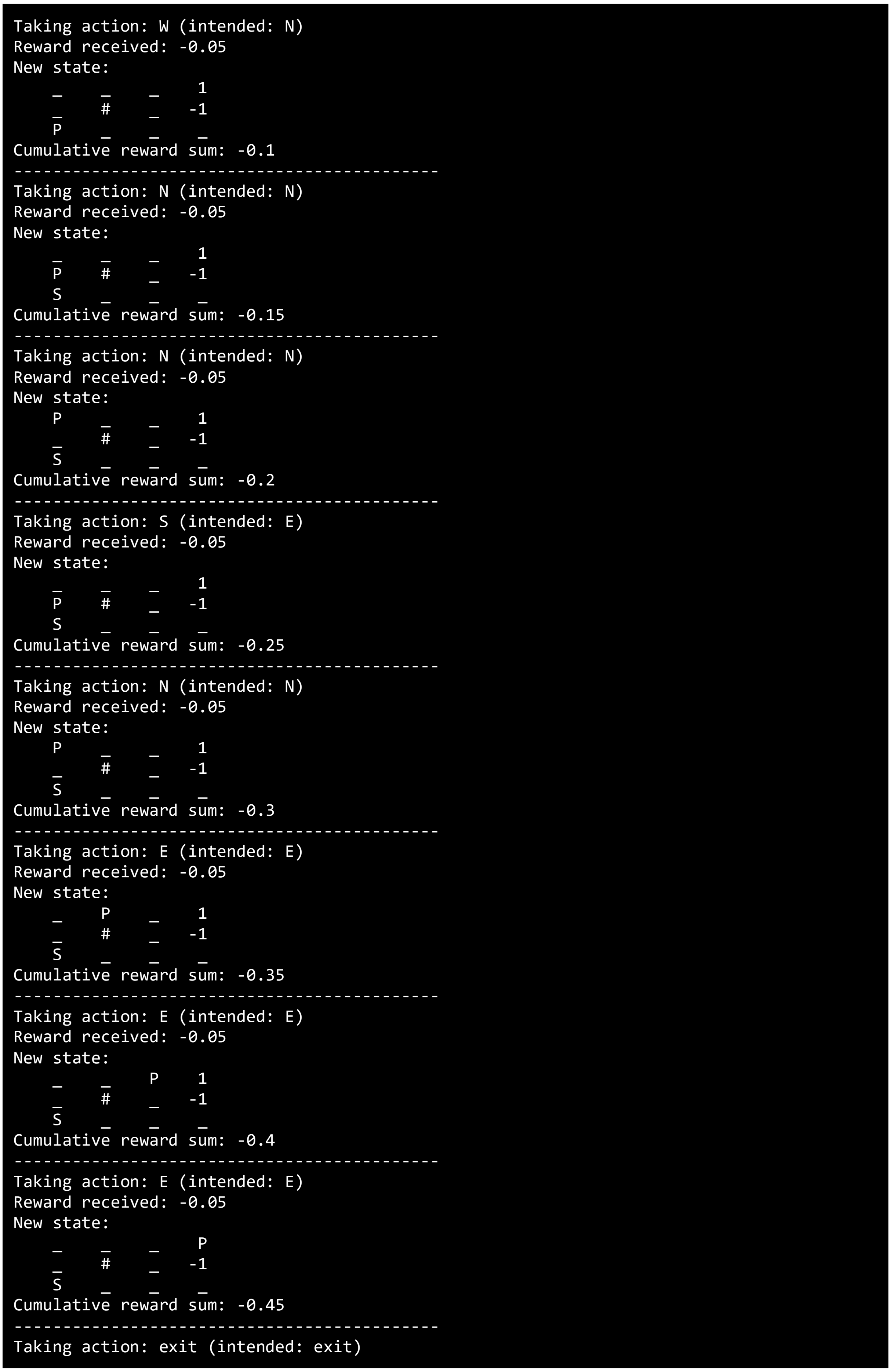
The second part of this file specifies the policy to be executed.

As usual, your first task is to implement the parsing of this grid MDP in the function `read_grid_mdp_problem_p1(file_path)` of the file `parse.py`. You may use any appropriate data structure.

Next, you should implement running the episode in the function `play_episode(problem)` in the file `p1.py`.

Below is the expected output. Note that we always use exactly 5 characters for the output of a single grid and that the last line does not contain a new line.

```
Start state:
    _    _    _    1
    _    #    _   -1
    P    _    _    _
Cumulative reward sum: 0.0
--------------------------------------------
Taking action: W (intended: N)
Reward received: -0.05
New state:
    _    _    _    1
    _    #    _   -1
    P    _    _    _
Cumulative reward sum: -0.05
--------------------------------------------
```


```
Taking action: W (intended: N)
Reward received: -0.05
New state:
    _    _    _    1
    _    #    _   -1
    P    _    _    _
Cumulative reward sum: -0.1
--------------------------------------------
Taking action: N (intended: N)
Reward received: -0.05
New state:
    _    _    _    1
    P    #    _   -1
    S    _    _    _
Cumulative reward sum: -0.15
--------------------------------------------
Taking action: N (intended: N)
Reward received: -0.05
New state:
    P    _    _    1
    _    #    _   -1
    S    _    _    _
Cumulative reward sum: -0.2
--------------------------------------------
Taking action: S (intended: E)
Reward received: -0.05
New state:
    _    _    _    1
    P    #    _   -1
    S    _    _    _
Cumulative reward sum: -0.25
--------------------------------------------
Taking action: N (intended: N)
Reward received: -0.05
New state:
    P    _    _    1
    _    #    _   -1
    S    _    _    _
Cumulative reward sum: -0.3
--------------------------------------------
Taking action: E (intended: E)
Reward received: -0.05
New state:
    _    P    _    1
    _    #    _   -1
    S    _    _    _
Cumulative reward sum: -0.35
--------------------------------------------
Taking action: E (intended: E)
Reward received: -0.05
New state:
    _    _    P    1
    _    #    _   -1
    S    _    _    _
Cumulative reward sum: -0.4
--------------------------------------------
Taking action: E (intended: E)
Reward received: -0.05
New state:
    _    _    _    P
    _    #    _   -1
    S    _    _    _
Cumulative reward sum: -0.45
--------------------------------------------
Taking action: exit (intended: exit)
```

```
Reward received: 1.0
New state:
    _    _    _    1
    _    #    _   -1
    S    _    _    _
Cumulative reward sum: 0.55
```

As you can see, in this question we don’t use any discount factor. We will introduce that in the next question. You can also try some of the other test cases such as `test_cases/p1/8.prob`.

```
seed: 42
noise: 0.2
livingReward: -1
grid:
    #   10    #
 -100    _ -100
 -100    _ -100
 -100    _ -100
 -100    _ -100
 -100    S -100
    #    1    #
policy:
    # exit    #
 exit    N exit
 exit    N exit
 exit    N exit
 exit    N exit
 exit    N exit
    #    1    #
```

With a correct implementation, you should be able to pass all test cases.

```
(base) AICourse@AICourse a3 % python p1.py -8
Grading Problem 1 :
----------> Test case 1 PASSED <----------
----------> Test case 2 PASSED <----------
----------> Test case 3 PASSED <----------
----------> Test case 4 PASSED <----------
----------> Test case 5 PASSED <----------
----------> Test case 6 PASSED <----------
----------> Test case 7 PASSED <----------
----------> Test case 8 PASSED <----------
```

Problem 2: Policy Evaluation (25 points)

In problem 2 you will implement policy evaluation as follows

$$V^{\pi}(s) = \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V^{\pi}(s')]$$

This time we have discounting and we also introduce a new variable for the number of iterations. Here is the first test case.

```
discount: 0.9
noise: 0.1
livingReward: 0
iterations: 10
grid:
  -10  100  -10
  -10    _  -10
  -10    _  -10
  -10    S  -10
policy:
  exit exit exit
  exit    N exit
  exit    N exit
  exit    N exit
```

Note that there is no randomness involved this time and that we use discounting.
As usual, your first task is to implement the parsing of this grid MDP in the function `read_grid_mdp_problem_p2(file_path)` of the file `parse.py`. You may use any appropriate data structure.

Next you implement value iteration for policy evaluation as discussed in class. Your `policy_evaluation(problem)` function in `p2.py` should return the evolution of values as follows.

```
V^pi_k=0
|   0.00||   0.00||   0.00|
|   0.00||   0.00||   0.00|
|   0.00||   0.00||   0.00|
|   0.00||   0.00||   0.00|
V^pi_k=1
| -10.00|| 100.00|| -10.00|
| -10.00||   0.00|| -10.00|
| -10.00||   0.00|| -10.00|
| -10.00||   0.00|| -10.00|
V^pi_k=2
| -10.00|| 100.00|| -10.00|
| -10.00||  70.20|| -10.00|
| -10.00||  -1.80|| -10.00|
| -10.00||  -1.80|| -10.00|
V^pi_k=3
| -10.00|| 100.00|| -10.00|
| -10.00||  70.20|| -10.00|
| -10.00||  48.74|| -10.00|
| -10.00||  -3.10|| -10.00|
V^pi_k=4
| -10.00|| 100.00|| -10.00|
| -10.00||  70.20|| -10.00|
| -10.00||  48.74|| -10.00|
| -10.00||  33.30|| -10.00|
V^pi_k=5
| -10.00|| 100.00|| -10.00|
| -10.00||  70.20|| -10.00|
| -10.00||  48.74|| -10.00|
| -10.00||  33.30|| -10.00|
V^pi_k=6
| -10.00|| 100.00|| -10.00|
| -10.00||  70.20|| -10.00|
| -10.00||  48.74|| -10.00|
| -10.00||  33.30|| -10.00|
V^pi_k=7
| -10.00|| 100.00|| -10.00|
| -10.00||  70.20|| -10.00|
| -10.00||  48.74|| -10.00|
| -10.00||  33.30|| -10.00|
V^pi_k=8
| -10.00|| 100.00|| -10.00|
| -10.00||  70.20|| -10.00|
| -10.00||  48.74|| -10.00|
| -10.00||  33.30|| -10.00|
V^pi_k=9
| -10.00|| 100.00|| -10.00|
| -10.00||  70.20|| -10.00|
| -10.00||  48.74|| -10.00|
| -10.00||  33.30|| -10.00|
```

This example should look familiar. We have covered it in chapter 2 of our lecture slides.

Hint: The output of an individual floating point value `v` was done as follows

```
return_value += '|{:7.2f}|'.format(v)
```

Finally, check the correctness of your implementation via

```
(base) AICourse@AICourse a3 % python p2.py -7
Grading Problem 2 :
----------> Test case 1 PASSED <----------
----------> Test case 2 PASSED <----------
----------> Test case 3 PASSED <----------
----------> Test case 4 PASSED <----------
----------> Test case 5 PASSED <----------
----------> Test case 6 PASSED <----------
----------> Test case 7 PASSED <----------
```

Problem 3: Value Iteration (20 points)

Now it is time to complement value iteration.

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_k(s')]$$

This time, the provided problems do not include policies:

```
discount: 1
noise: 0.1
livingReward: -0.1
iterations: 10
grid:
    _    _    _    1
    _    #    _   -1
    S    _    _    _
```

As usual, load the problem definition in the function `read_grid_mdp_problem_p3(file_path)` of the file `parse.py`.

Next implement value_iteration(problem) in p3.py such that it returns the following string for the first test case. Note that this is still a non-deterministic environment (i.e., the taken action can be different from the intended one) as before with the same stochastic motion.

```
V_k=0
|   0.00||   0.00||   0.00||   0.00|
|   0.00|| ##### ||   0.00||   0.00|
|   0.00||   0.00||   0.00||   0.00|
V_k=1e
| N || # || N || x |
| N || N || N || N |
V_k=2
|  -0.20||  -0.20||   0.68||   1.00|
|  -0.20|| ##### ||  -0.20||  -1.00|
|  -0.20||  -0.20||  -0.20||  -0.20|
pi_k=2
| N || N || E || x |
| N || # || W || x |
| N || N || N || S |
V_k=3
|  -0.30||   0.40||   0.75||   1.00|
|  -0.30|| ##### ||   0.32||  -1.00|
|  -0.30||  -0.30||  -0.30||  -0.30|
pi_k=3
| N || E || E || x |
| N || # || N || x |
| N || N || N || S |
V_k=4
|   0.16||   0.58||   0.81||   1.00|
|  -0.40|| ##### ||   0.43||  -1.00|
|  -0.40||  -0.40||   0.10||  -0.40|
pi_k=4
| E || E || E || x |
| N || # || N || x |
| N || N || N || S |
V_k=5
|   0.34||   0.66||   0.82||   1.00|
|  -0.05|| ##### ||   0.49||  -1.00|
|  -0.50||  -0.10||   0.16||  -0.16|
pi_k=5
| E || E || E || x |
| N || # || N || x |
| N || E || N || W |
V_k=6
|   0.46||   0.69||   0.83||   1.00|
|   0.16|| ##### ||   0.51||  -1.00|
|  -0.20||   0.01||   0.26||  -0.08|
pi_k=6
| E || E || E || x |
| N || # || N || x |
| N || E || N || W |
V_k=7
|   0.52||   0.70||   0.83||   1.00|
|   0.30|| ##### ||   0.52||  -1.00|
|   0.01||   0.11||   0.30||   0.00|
pi_k=7
| E || E || E || x |
| N || # || N || x |
| N || E || N || W |
V_k=8
|   0.54||   0.71||   0.83||   1.00|
|   0.37|| ##### ||   0.52||  -1.00|
|   0.15||   0.16||   0.32||   0.04|
pi_k=8
| E || E || E || x |
| N || # || N || x |
| N || E || N || W |
V_k=9
|   0.56||   0.71||   0.84||   1.00|
|   0.41|| ##### ||   0.52||  -1.00|
|   0.23||   0.19||   0.34||   0.06|
pi_k=9
| E || E || E || x |
| N || # || N || x |
| N || E || N || W |
```

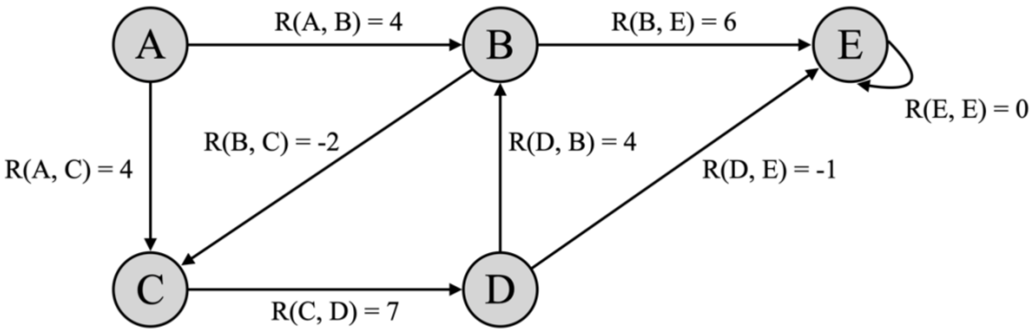
Once you are done. Check if you can pass all test cases as follows.

```
(base) AICourse@AICourse a3 % python p3.py -4
Grading Problem 3 :
----------> Test case 1 PASSED <----------
----------> Test case 2 PASSED <----------
----------> Test case 3 PASSED <----------
----------> Test case 4 PASSED <----------
```

Part 2: Writing assignment

Answer the questions below and save your answers as "report.pdf".

1. **(15 points)** Consider the following Markov Decision Process. Unlike most MDP models where actions have many potential outcomes of varying probability, assume that the transitions are *deterministic*, i.e. occur with $T(s, a, s') = 100\%$ probability as shown in the graph below. The reward for each transition is displayed adjacent to each edge:



(1) Compute the following six quantities: $V_2(A)$, $V^*(A)$, $V_2(B)$, $V^*(B)$, $V_5(A)$, $V_5(B)$, assuming a discount factor of 1. Please also provide the computation process. **(6 points)**

(2) Re-compute the two quantities: $V^*(A)$, $V^*(B)$, assuming a discount factor of 0.9. Please also provide the computation process. **(4 points)**

(3) Consider using feature-based Q-learning to learn what the optimal policy should be. Suppose we use the feature $f_1(s, a)$ equal to the smallest number of edges from s to E, and feature $f_2(s, a)$ equal to the smallest number of edges from a's target to E. For example $f_1(A, \rightarrow B)$ is 2, and $f_2(C, \rightarrow D)$ is 1. Let the current weight w for these features be $[3, 2]$. What is the current prediction for $Q(C, \rightarrow D)$ given these weights for these two features? **(2 points)**

(4) Suppose we update w based on a single observation of a transition from C to D, assuming learning rate $\alpha = 0.5$. What is w after updating based on this observation? **(3 points)**

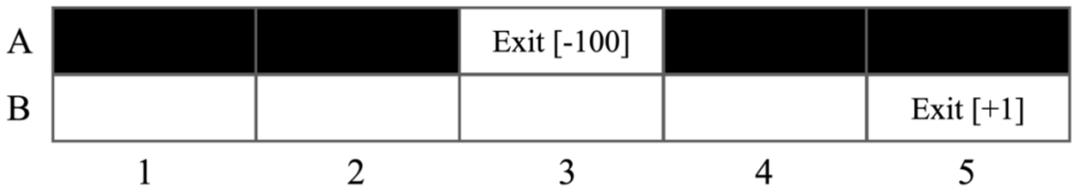
2. **(8 points)** Suppose we have a robot that has 5 possible actions: {↑, ↓, ←, →, Exit}. If the robot picks a direction, it has an 60% chance of going in that direction, a 20% chance of going 90 degrees counter-clockwise of the direction it picked (e.g. picked →, but actually goes ↑), and a 20% chance of going 90 degrees clockwise of the direction it picks.

   If the robot hits a wall, it does not move this timestep, but time still elapses. For example, if the robot picks ↓ in state B1 and is 'successful' (60% chance), it will hit the wall and not move this timestep. As another example, if it picks → in state B1, but gets the 20% chance of going down, it will hit the wall and not move this timestep.

   In states B1, B2, B3, and B4, the set of possible actions is {↑, ↓, ←, →}. In states A3 and B5, the only possible action is {Exit}. Exit is always successful. The blackened areas (A1, A2, A4, and A5) represent walls.

   All transition rewards are 0, except R(A3, Exit) = -100, R(B5, Exit) = 1.

(1) In the boxes below, draw an optimal policy for states B1, B2, B3, and B4, if there is no discounting, i.e. $\gamma = 1$. Use the symbols {↑, ↓, ←, →}. **(4 points)**



(2) Give the expected utility for states B1, B2, B3, and B4 under the policy you gave above. **(2 points)**

(3) What is $V^*(B4)$, the expected utility of state B4, if we have a discount $0 < \gamma < 1$ and use the optimal policy? Give your answer in terms of γ. **(2 points)**

3. **(7 points)** Consider a simple MDP with two states, S_1 and S_2, two actions, A and B, a discount factor γ of $1/2$, reward function R given by: $R(s, a, S_1) = 1$, $R(s, a, S_2) = -1$, and a transition function specified by the following table.

s	a	s'	$T(s, a, s')$
S_1	A	S_1	$1/2$
S_1	A	S_2	$1/2$
S_1	B	S_1	$2/3$
S_1	B	S_2	$1/3$
S_2	A	S_1	$1/2$
S_2	A	S_2	$1/2$
S_2	B	S_1	$1/3$
S_2	B	S_2	$2/3$

(1) Perform a single iteration of value iteration, filling in the resultant Q-values and state values in the following tables. Use the specified initial value function V_0, rather than starting from all zero state values. Only compute the entries not labeled "skip". **(3 points)**

s	a	$Q_1(s, a)$
S_1	A	1.25
S_1	B	1.5
S_2	A	skip
S_2	B	skip

s	$V_0(s)$	$V_1(s)$
S_1	2	1.5
S_2	3	skip

(2) Suppose that Q-learning with a learning rate α of 1/2 is being run, and the following episode is observed.

s_1	a_1	r_1	s_2	a_2	r_2	s_3
S_1	A	1	S_1	A	-1	S_2

Using the initial Q-values Q_0, fill in the following table to indicate the resultant progression of Q-values. **(4 points)**

s	a	$Q_0(s, a)$	$Q_1(s, a)$	$Q_2(s, a)$
S_1	A	$-1/2$	1/4	-1/8
S_1	B	0	0	
S_2	A	-1	-1	
S_2	B	1	1	

Submission

To submit your assignment to Moodle, `*.zip` the following files ONLY:

- `p1.py`
- `p2.py`
- `p3.py`
- `parse.py`
- `report.pdf`

Do not zip any other files. Use the `*.zip` file format. Name the file UID.zip, where UID is your university number, like 3030661123.zip. Make sure that you have submitted the correct files and named all files correctly. We will deduct up to 5% for files with incorrectly file names.