PJ3: Black Jack

杜闻博　18307110359

1 Problem 1: Value Iteration

a. Give the value of $V_{opt}(s)$ for each state s after 0, 1, and 2 iterations

iter. \ state	-2	-1	0	1	2
0	0	0	0	0	0
1	0	15	-5	26.5	0
2	0	14	13.45	23	0

Steps:

First, in this case, we have:

$$V_{k+1} = \max_a \sum_{s'} P(s'|s,a) \cdot [R(s,a,s') + \gamma V_k(s')]$$

1. In iteration 0, we initialize all the states values as 0.

2. In iteration 1, we set two terminal states $V_1(-2) = V_1(2) = 0$, and calculate other states as follows:

$$V_1(-1) = max \begin{cases} 0.8 \times (20 + 1 \times 0) + 0.2 \times (-5 + 1 \times 0) & (a = -1) \\ 0.7 \times (20 + 1 \times 0) + 0.3 \times (-5 + 1 \times 0) & (a = +1) \end{cases}$$
$$= 15$$
$$V_1(0) = max \begin{cases} 0.8 \times (-5 + 1 \times 0) + 0.2 \times (-5 + 1 \times 0) & (a = -1) \\ 0.7 \times (-5 + 1 \times 0) + 0.3 \times (-5 + 1 \times 0) & (a = +1) \end{cases}$$
$$= -5$$
$$V_1(1) = max \begin{cases} 0.8 \times (-5 + 1 \times 0) + 0.2 \times (100 + 1 \times 0) & (a = -1) \\ 0.7 \times (-5 + 1 \times 0) + 0.3 \times (100 + 1 \times 0) & (a = +1) \end{cases}$$
$$= 26.5$$

3. In iteration 2, we set two terminal states $V_1(-2) = V_1(2) = 0$, and calculate other

states as follows:

$$V_2(-1) = max \begin{cases} 0.8 \times (20 + 1 \times 0) + 0.2 \times (-5 + 1 \times (-5)) & (a = -1) \\ 0.7 \times (20 + 1 \times 0) + 0.3 \times (-5 + 1 \times (-5)) & (a = +1) \end{cases}$$
$$= 14$$
$$V_2(0) = max \begin{cases} 0.8 \times (-5 + 1 \times 15) + 0.2 \times (-5 + 1 \times 26.5) & (a = -1) \\ 0.7 \times (-5 + 1 \times 15) + 0.3 \times (-5 + 1 \times 26.5) & (a = +1) \end{cases}$$
$$= 13.45$$
$$V_2(1) = max \begin{cases} 0.8 \times (-5 + 1 \times (-5)) + 0.2 \times (100 + 1 \times 0) & (a = -1) \\ 0.7 \times (-5 + 1 \times (-5)) + 0.3 \times (100 + 1 \times 0) & (a = +1) \end{cases}$$
$$= 23$$

b. Corresponding optimal policy π_{opt} based on $V_{opt}(s)$

state	-1	0	1
policy	-1	+1	+1

Derive:

Because we have:

$$\pi(s) = \underset{a}{argmax} \sum_{s'} P(s'|s,a) \cdot [R(s,a,s') + \gamma V_k(s')]$$

So from the calculate in question (a), we have $\pi(-1) = -1,\ \pi(0) = +1,\ \pi(1) = +1$ based on $V_{opt}(s)$ after the second iteration.

2 Problem 2: Transforming MDPs

a. It isn't always the case that $V_1(s_{start}) \geq V_2(s_{start})$, and a counterexample is provided in the *CounterexampleMDP* in *submission.py*.

b.Because in topological order, the state appears later will only influenced by states appears before, so compute V_{opt} for each node with only a single pass over all the $(s,\ a,\ s\ )$ triples if we do it in the inverse topological order, so the algorithm can be explained in two steps as follows:

1. Firstly, we get the **Topological Sort** for all states, and obtain their topological order.

2. Then, we can compute V_{opt} iteratively for each node in their inverse topological order.

c. Firstly, we derive from the original MDP:

$$V_{k+1} = \underset{a}{max} \sum_{s'} T(s,a,s') \cdot [R(s,a,s') + \gamma V_k(s')]$$
$$= \underset{a}{max} \left\{ (1-\gamma)\cdot(-V_k(o) + 1 \times V_k(o)) + \sum_{s'(s' \neq o)} \gamma T(s,a,s') \cdot \left[\frac{1}{\gamma} R(s,a,s') + 1 \times V_k(s') \right] \right\}$$

In this form, the total transition probability will be $(1-\gamma)+\gamma\cdot\left(\sum_{s'(s'\neq o)} T(s,a,s')\right)$. Hence we already know that $\sum_{s'(s'\neq o)} T(s,a,s') = 1$, the total transition probability in new MDP will be $(1-\gamma)+\gamma = 1$.

At the same time, we can obtain the result as follows:

$$T'(s,a,s') = \begin{cases} 1-\gamma & if\ s' = o; \\ \gamma \cdot T(s,a,s') & otherwise. \end{cases}$$

$$R'(s,a,s') = \begin{cases} -V_k(s')\ (=0) & if\ s' = o; \\ \frac{1}{\gamma} \cdot R(s,a,s') & otherwise. \end{cases}$$

We set state o as a terminal state, then $-V_k(s') = 0$ will always hold, such that the optimal values $V_{opt}(s)$ for all $s \in States$ are equal under the original MDP and the new MDP.

3 Problem 4: Learning to Play Blackjack

b. Comparing Q-learning policy and value iteration policy, I calculate the different ratio as follows:

```
----- START PART 4b-helper: Helper function to run Q-learning simulations for question 4b.
ValueIteration: 5 iterations
different-ratio between q learning and value iteration:
0.25925925925925924
ValueIteration: 15 iterations
different-ratio between q learning and value iteration:
0.331511839708561
----- END PART 4b-helper [took 0:00:05.253943 (max allowed 60 seconds), 0/0 points]
```

$$different\ ratio = \frac{different\ action\ states}{total\ states} = \begin{cases} 0.2593 & smallMDP; \\ 0.3315 & largeMDP. \end{cases}$$

The result shows that there exists some difference between Q-learning policy and value iteration policy, and for largeMDP, it needs more steps to converge. The reason for that difference might be the explorationProb is a little bit small.

d. The result is showed as follows:

```
----- START PART 4d-helper: Helper function to compare rewards when simulating RL over two diffe
ValueIteration: 5 iterations
fixed_RL reward : 6.83542
q_learning reward : 9.66402
----- END PART 4d-helper [took 0:00:05.818502 (max allowed 60 seconds), 0/0 points]
```

We can find that the expected reward of fixedRL is less than Q-learning Reward. Because the fixedRL use the same strategy learned from original MDPs to solve Modified MDPs, but in fact their optimal strategy is not the same. However, the Q-learning updates its strategy in each simulation, so it has a higher reward. From this we know that the Q-learning is more robust than value iteration.