Report of Project3

张奕朗 16307130242

Problem 1: Value Iteration

a.

$$
\begin{aligned}
V_{k+1}(s) &= \max_a \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V_k(s')\right] \\
k &= 1: \\
V_1(-1) &= max\{0.8*(20+1*0)+0.2*(-5+1*0), 0.3*(-5+1*0)+0.7*(20+1*0)\} \\
&= 15 \\
V_1(0) &= max\{0.8*(-5+1*0)+0.2*(-5+1*0), 0.3*(-5+1*0)+0.7*(-5+1*0)\} \\
&= -5 \\
V_1(1) &= max\{0.8*(-5+1*0)+0.2*(100+1*0), 0.3*(100+1*0)+0.7*(-5+1*0)\} \\
&= 26.5 \\
k &= 2: \\
V_2(-1) &= max\{0.8*(20+1*0)+0.2*(-5+1*-5), 0.3*(-5+1*-5)+0.7*(20+1*0)\} \\
&= 14 \\
V_2(0) &= max\{0.8*(-5+1*15)+0.2*(-5+1*26.5), 0.3*(-5+1*26.5)+0.7*(-5+1*15)\} \\
&= 13.45 \\
V_2(1) &= max\{0.8*(-5+1*-5)+0.2*(100+1*0), 0.3*(100+1*0)+0.7*(-5+1*-5)\} \\
&= 23
\end{aligned}
$$

	$s=-2$	$s=-1$	$s=0$	$s=1$	$s=2$
$Iter_0$	0	0	0	0	0
$Iter_1$	0	15	-5	26.5	0
$Iter_2$	0	14	13.45	23	0

b.

$$\pi_{opt}(s) = argmax_a \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V_k(s')\right]$$

	$s=-2$	$s=-1$	$s=0$	$s=1$	$s=2$
π_{opt}	0	-1	+1	+1	0

Problem 2: Transforming MDPs

b.

Since the MDP is acyclic, we can implement MDP with the following modifications:

- Organize the directed acyclic graph (DAG) as a tree, where terminal states is the leaves of the tree and starting state is the root of the tree.
- Use a set $S_{visited}$ to note the states who have been updated, i.e. the states whose depth is small than current state $s_{current}$ (the state being updated).
- Each time when we compute the utility for a state, modify the transition model dynamically by setting $T(s,a,s')=0, s' \in S_{visited}$ and scaling $T(s,a,s') \leftarrow \frac{T(s,a,s')}{1-\sum_{s' \in S_{visited}} T(s,a,s')}$ which ensures $\sum_{s' \notin S_{visited}} T(s,a,s') = 1$.
- Update the utility of the states from the leaf nodes to the root node (with a queue). Since the states are organized into a tree, we can update only once to get the utility of all the states.

c.

Since we can rewrite the value iteration as $V(s) \leftarrow \max_a \sum_{s'} \gamma T(s,a,s') \left[\frac{R(s,a)}{\gamma} + V(s')\right]$, we get the new MDP as follows:

- $T'(s,a,s') = \gamma T(s,a,s')$ for all $s \in States$
- $T'(s,a,o) = 1-\gamma$
- $Reward'(s,a,s') = \frac{Reward(s,a,s')}{\gamma}$ for all $s' \in States$
- $Reward'(s,a,o) = 0$

Problem 4: Learning to Play Blackjack

b.

This is the number of differences in one simulation on `QLearningAlgorithm` and `ValueIteration`:

	SmallMDP	**LargeMDP**
differences	2	897

The main reason accounting for the difference is the `explorationProb` in `QLearningAlgorithm`, which enables Q-learning to perform random exploration with some probabilities.

- In the case of `SmallMDP`, the number of states is small so `QLearningAlgorithm` could explore almost all the states, leaving only 2 differences in policies.
- While for `LargeMDP`, the number of states grows exponentially with the scale of card deck, so `QLearningAlgorithm` could only explore some of the states and and shows a significant discrepency compared with `ValueIteration` who always converges the primal policy in offline learning.
- Except for the scale of MDP problem, the features used in `QLearningAlgorithm` also affect its performance. In problem **4.c**, number of differences are reduced to 598 in `LargeMDP` with our domain

knowledge.

In offline learning, `ValueIteration` have a better performance than `QLearningAlgorithm` since the former computes the value of all the possible states while the latter seldom explores the states with low utility.

d.

Results of simulation with 10 trials for each of two algorithm:

Trial	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**mean**
Reward for VI	6	6	6	6	6	7	6	6	10	6	6.5
Reward for QL	11	12	11	12	7	12	12	12	11	12	11.2

The policy learned by `ValueIteration` shows a worse performance than that of `QLearningAlgorithm` for the following reasons:

- `ValueIteration` is an offline learning algorithm. Its policy which was learned from `originMDP` is not applicable for `newThresholdMDP`, since the condition is different from `originMDP` (here they differs in `threshold`).
- `QLearningAlgorithm` is an online learning algorithm which don't need pre-training. It learns from samples and adjusts its Q-values and corresponding optimal policy dynamically to fit the environment.

To sum up, the environment may change and become unknown. Offline learning like value iteration which performs a fixed policy learned from previous knowledge about the environment would have a terrible performance in a mutative environment, while online learning like Q-learning could modify its policy with the changing of environment though observed samples.