PRACTICAL – 7

AIM: Write a python program to implement Q-value iteration algorithm for solving the Markov Decision Process (MDP) problem in the farmer example.

Code:

```
class Q:
  def __init__(self, state, action):
    self.state = state
    self.action = action
    self.q_value = {}
    for s in state:
      for a in action:
        self.q_value[(s, a)] = 0
  def __repr__(self):
    output = "Q-values:\n"
    for s in self.state:
      for a in self.action:
        output += f"Q({s}, {a}) = {self.q_value[(s, a)]:.2f}\n"
    return output.strip()

class FarmerEnvironment:
  def __init__(self, gamma=0.9):
    rich = "rich"
    poor = "poor"
    plant = "plant"
    fallow = "fallow"
    self.state = [rich, poor]
    self.action = [plant, fallow]
    self.T = {
        (rich, plant, rich): 0.1,
        (rich, plant, poor): 0.9,
        (rich, fallow, rich): 0.9,
        (rich, fallow, poor): 0.1,
        (poor, plant, rich): 0.1,
        (poor, plant, poor): 0.9,
        (poor, fallow, rich): 0.9,
        (poor, fallow, poor): 0.1
    }
    self.R = {
        (rich, plant): 100,
        (rich, fallow): 0,
        (poor, plant): 10,
```

```
        (poor, fallow): 0
    }
    self.gamma = gamma
    self.Q = Q(self.state, self.action)
  def q_value_iteration(self, num_iteration=1000, treshhold=1e-6):
    for i in range(num_iteration):
      q_new = self.Q.q_value.copy()
      for s in self.state:
        for a in self.action:
          q_new[(s, a)] = self.R.get((s, a), 0) + self.gamma * sum(
            self.T.get((s, a, s_next), 0) * max(self.Q.q_value[(s_next, a_next)] for a_next in self.action)
            for s_next in self.state
          )
      if max(abs(q_new[(s, a)] - self.Q.q_value[(s, a)]) for s in self.state for a in self.action) < treshhold:
        break
      self.Q.q_value = q_new
  def get_optimal_policy(self):
    policy = {}
    for s in self.state:
      best_action = max(self.action, key=lambda a: self.Q.q_value[(s, a)])
      policy[s] = best_action
    return policy

env = FarmerEnvironment()
env.q_value_iteration()
print(env.Q)
print("\nOptimal policy:")
policy = env.get_optimal_policy()
for s in env.state:
  print(f"State: {s}, Action: {policy[s]}")
```

Output:

```
Q-values:
Q(rich, plant) = 529.07
Q(rich, fallow) = 470.93
Q(poor, plant) = 439.07
Q(poor, fallow) = 470.93

Optimal policy:
State: rich, Action: plant
State: poor, Action: fallow
```