МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

им. Н.Э. Баумана

Факультет «Информатика и системы управления»

Кафедра «Систем обработки информации и управления»

ОТЧЕТ

**Лабораторная работа №\_\_5\_\_**

по дисциплине«Методы машинного обучения»

Тема: «Обучение на основе временны’х различий»

ИСПОЛНИТЕЛЬ: Лу Сяои

ФИО

группа ИУ5И-22М \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись

"1" Июнь 2023 г.

ПРЕПОДАВАТЕЛЬ: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ФИО

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись

"\_\_"\_\_\_\_\_\_\_\_\_2023 г.

Москва - 2023

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## **описание задания**

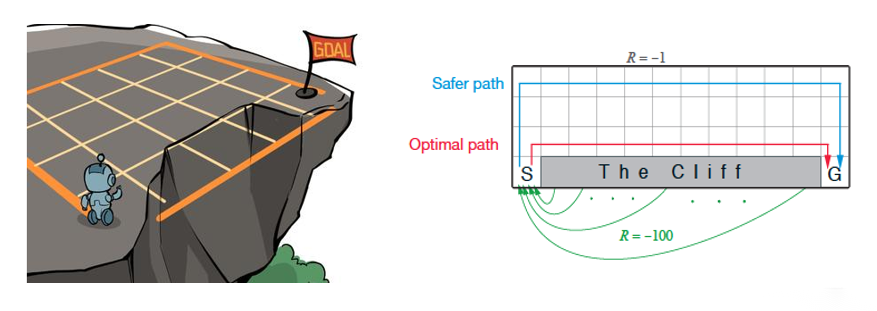
На основе рассмотренного на лекции примера реализуйте следующие алгоритмы:

* SARSA
* Q-обучение
* Двойное Q-обучение

для любой среды обучения с подкреплением (кроме рассмотренной на лекции среды Toy Text / Frozen Lake) из библиотеки [Gym](https://www.gymlibrary.dev/) (или аналогичной библиотеки)

## **текст программы и экранные формы с примерами выполнения программы.**

Я выбрала среду **Cliff Walking**.



Это мир с сеткой 4x12, где каждая сетка представляет собой состояние. Начальной точкой интеллекта является состояние в левом нижнем углу, а целью - состояние в правом нижнем углу. Интеллект может выполнять 4 действия в каждом состоянии: вверх, вниз, влево и вправо. Если интеллект совершает действие и касается пограничной стены, состояние не меняется, в противном случае он переходит в следующее состояние соответственно. В окружении есть участок обрыва, падение в который или достижение целевого состояния завершает действие и возвращает в начальную точку, т.е. падение в обрыв или достижение целевого состояния является конечным состоянием. Награда за каждый шаг равна -1, награда за падение с обрыва равна -100, а награда за достижение конечного состояния равна 0.

import matplotlib.pyplot as plt

import numpy as np

from tqdm import tqdm  # tqdm是显示循环进度条的库

class CliffWalkingEnv:

    def \_\_init\_\_(self, ncol, nrow):

        self.nrow = nrow

        self.ncol = ncol

        self.x = 0  # 记录当前智能体位置的横坐标

        self.y = self.nrow - 1  # 记录当前智能体位置的纵坐标

    def step(self, action):  # 外部调用这个函数来改变当前位置

        # 4种动作, change[0]:上, change[1]:下, change[2]:左, change[3]:右。坐标系原点(0,0)

        # 定义在左上角

        change = [[0, -1], [0, 1], [-1, 0], [1, 0]]

        self.x = min(self.ncol - 1, max(0, self.x + change[action][0]))

        self.y = min(self.nrow - 1, max(0, self.y + change[action][1]))

        next\_state = self.y \* self.ncol + self.x

        reward = -1

        done = False

        if self.y == self.nrow - 1 and self.x > 0:  # 下一个位置在悬崖或者目标

            done = True

            if self.x != self.ncol - 1:

                reward = -100

        return next\_state, reward, done

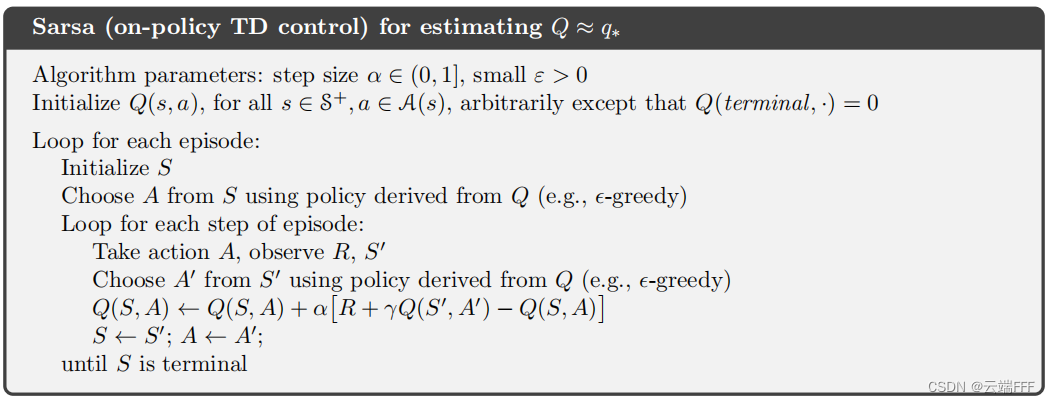
    def reset(self):  # 回归初始状态,坐标轴原点在左上角

        self.x = 0

        self.y = self.nrow - 1

        return self.y \* self.ncol + self.x

### SARSA



class Sarsa:

    """ Sarsa算法 """

    def \_\_init\_\_(self, ncol, nrow, epsilon, alpha, gamma, n\_action=4):

        self.Q\_table = np.zeros([nrow \* ncol, n\_action])  # 初始化Q(s,a)表格

        self.n\_action = n\_action  # 动作个数

        self.alpha = alpha  # 学习率

        self.gamma = gamma  # 折扣因子

        self.epsilon = epsilon  # epsilon-贪婪策略中的参数

    def take\_action(self, state):  # 选取下一步的操作,具体实现为epsilon-贪婪

        if np.random.random() < self.epsilon:

            action = np.random.randint(self.n\_action)

        else:

            action = np.argmax(self.Q\_table[state])

        return action

    def best\_action(self, state):  # 用于打印策略

        Q\_max = np.max(self.Q\_table[state])

        a = [0 for \_ in range(self.n\_action)]

        for i in range(self.n\_action):  # 若两个动作的价值一样,都会记录下来

            if self.Q\_table[state, i] == Q\_max:

                a[i] = 1

        return a

    def update(self, s0, a0, r, s1, a1):

        td\_error = r + self.gamma \* self.Q\_table[s1, a1] - self.Q\_table[s0, a0]

        self.Q\_table[s0, a0] += self.alpha \* td\_error

ncol = 12

nrow = 4

env = CliffWalkingEnv(ncol, nrow)

np.random.seed(0)

epsilon = 0.1

alpha = 0.1

gamma = 0.9

agent = Sarsa(ncol, nrow, epsilon, alpha, gamma)

num\_episodes = 500  # 智能体在环境中运行的序列的数量

return\_list = []  # 记录每一条序列的回报

for i in range(10):  # 显示10个进度条

    # tqdm的进度条功能

    with tqdm(total=int(num\_episodes / 10), desc='Iteration %d' % i) as pbar:

        for i\_episode in range(int(num\_episodes / 10)):  # 每个进度条的序列数

            episode\_return = 0

            state = env.reset()

            action = agent.take\_action(state)

            done = False

            while not done:

                next\_state, reward, done = env.step(action)

                next\_action = agent.take\_action(next\_state)

                episode\_return += reward  # 这里回报的计算不进行折扣因子衰减

                agent.update(state, action, reward, next\_state, next\_action)

                state = next\_state

                action = next\_action

            return\_list.append(episode\_return)

            if (i\_episode + 1) % 10 == 0:  # 每10条序列打印一下这10条序列的平均回报

                pbar.set\_postfix({

                    'episode':

                    '%d' % (num\_episodes / 10 \* i + i\_episode + 1),

                    'return':

                    '%.3f' % np.mean(return\_list[-10:])

                })

            pbar.update(1)

episodes\_list = list(range(len(return\_list)))

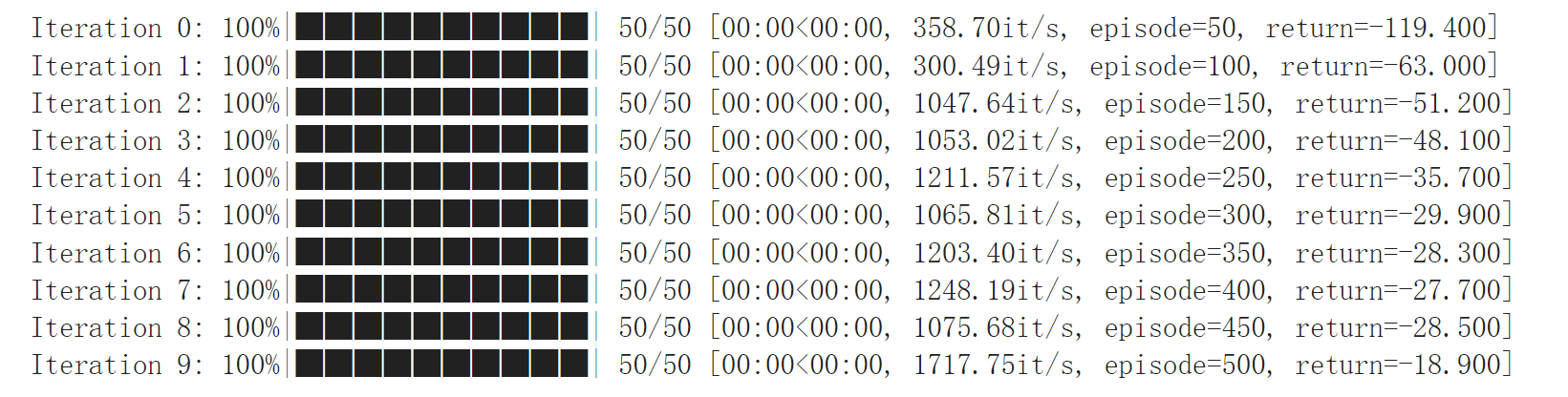
plt.plot(episodes\_list, return\_list)

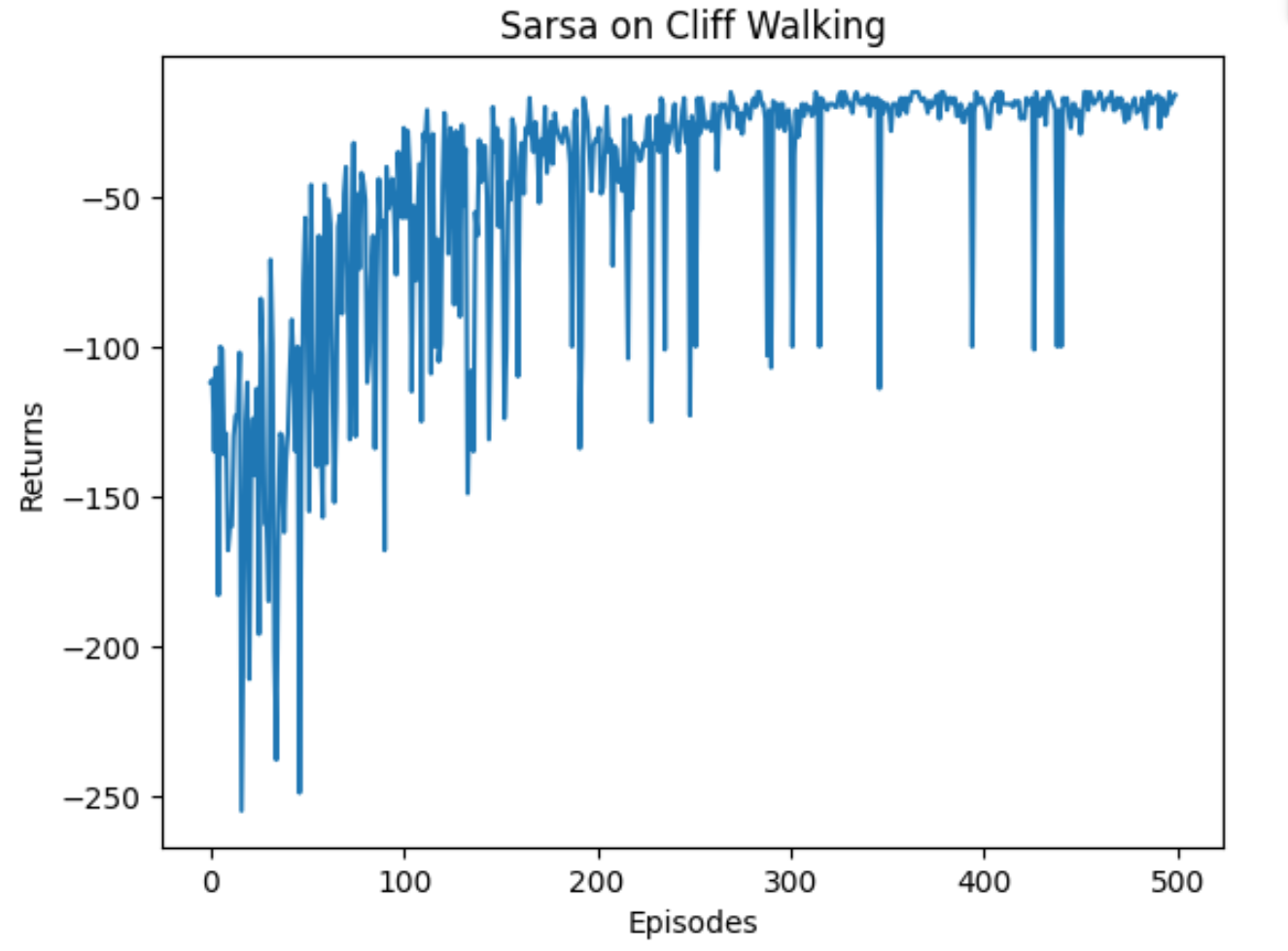
plt.xlabel('Episodes')

plt.ylabel('Returns')

plt.title('Sarsa on {}'.format('Cliff Walking'))

plt.show()





def print\_agent(agent, env, action\_meaning, disaster=[], end=[]):

    for i in range(env.nrow):

        for j in range(env.ncol):

            if (i \* env.ncol + j) in disaster:

                print('\*\*\*\*', end=' ')

            elif (i \* env.ncol + j) in end:

                print('EEEE', end=' ')

            else:

                a = agent.best\_action(i \* env.ncol + j)

                pi\_str = ''

                for k in range(len(action\_meaning)):

                    pi\_str += action\_meaning[k] if a[k] > 0 else 'o'

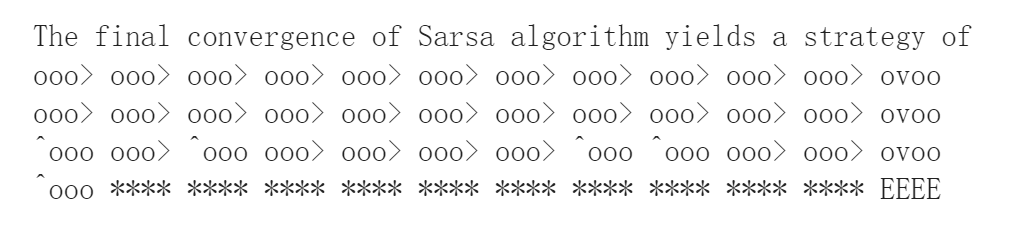
                print(pi\_str, end=' ')

        print()

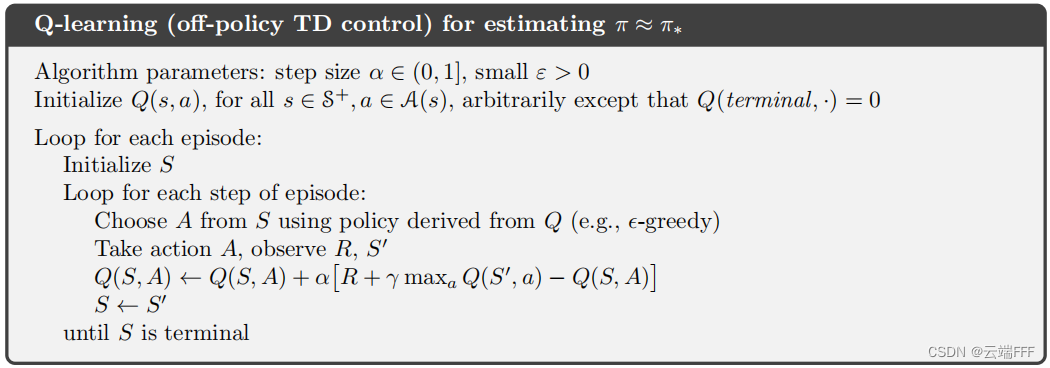
action\_meaning = ['^', 'v', '<', '>']

print('The final convergence of Sarsa algorithm yields a strategy of')

print\_agent(agent, env, action\_meaning, list(range(37, 47)), [47])



### Q-обучение



class QLearning:

    """ Q-learning算法 """

    def \_\_init\_\_(self, ncol, nrow, epsilon, alpha, gamma, n\_action=4):

        self.Q\_table = np.zeros([nrow \* ncol, n\_action])  # 初始化Q(s,a)表格

        self.n\_action = n\_action  # 动作个数

        self.alpha = alpha  # 学习率

        self.gamma = gamma  # 折扣因子

        self.epsilon = epsilon  # epsilon-贪婪策略中的参数

    def take\_action(self, state):  #选取下一步的操作

        if np.random.random() < self.epsilon:

            action = np.random.randint(self.n\_action)

        else:

            action = np.argmax(self.Q\_table[state])

        return action

    def best\_action(self, state):  # 用于打印策略

        Q\_max = np.max(self.Q\_table[state])

        a = [0 for \_ in range(self.n\_action)]

        for i in range(self.n\_action):

            if self.Q\_table[state, i] == Q\_max:

                a[i] = 1

        return a

    def update(self, s0, a0, r, s1):

        td\_error = r + self.gamma \* self.Q\_table[s1].max(

        ) - self.Q\_table[s0, a0]

        self.Q\_table[s0, a0] += self.alpha \* td\_error

np.random.seed(0)

epsilon = 0.1

alpha = 0.1

gamma = 0.9

agent = QLearning(ncol, nrow, epsilon, alpha, gamma)

num\_episodes = 500  # 智能体在环境中运行的序列的数量

return\_list = []  # 记录每一条序列的回报

for i in range(10):  # 显示10个进度条

    # tqdm的进度条功能

    with tqdm(total=int(num\_episodes / 10), desc='Iteration %d' % i) as pbar:

        for i\_episode in range(int(num\_episodes / 10)):  # 每个进度条的序列数

            episode\_return = 0

            state = env.reset()

            done = False

            while not done:

                action = agent.take\_action(state)

                next\_state, reward, done = env.step(action)

                episode\_return += reward  # 这里回报的计算不进行折扣因子衰减

                agent.update(state, action, reward, next\_state)

                state = next\_state

            return\_list.append(episode\_return)

            if (i\_episode + 1) % 10 == 0:  # 每10条序列打印一下这10条序列的平均回报

                pbar.set\_postfix({

                    'episode':

                    '%d' % (num\_episodes / 10 \* i + i\_episode + 1),

                    'return':

                    '%.3f' % np.mean(return\_list[-10:])

                })

            pbar.update(1)

episodes\_list = list(range(len(return\_list)))

plt.plot(episodes\_list, return\_list)

plt.xlabel('Episodes')

plt.ylabel('Returns')

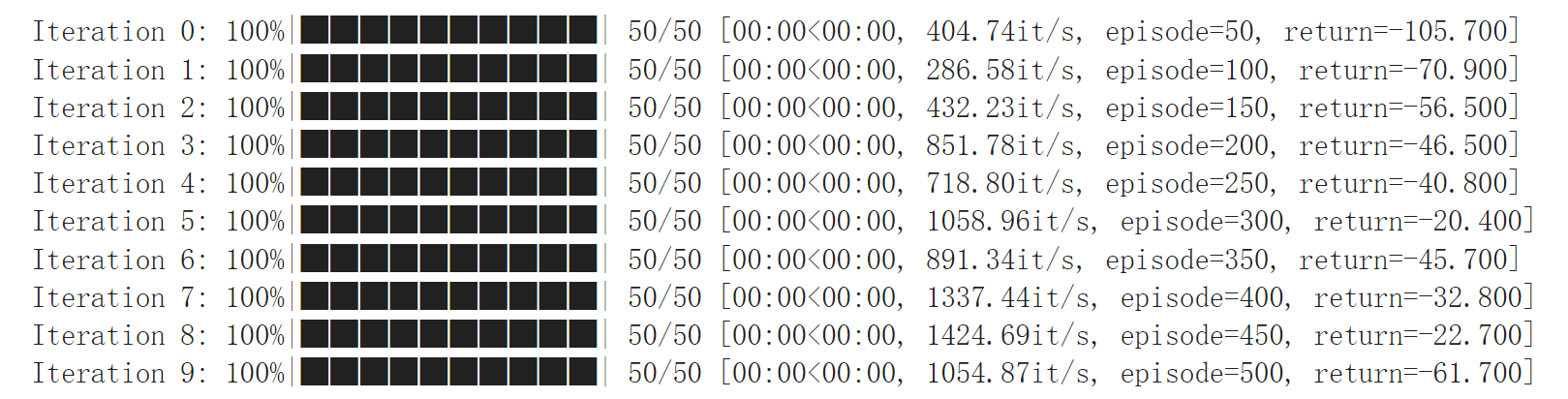
plt.title('Q-learning on {}'.format('Cliff Walking'))

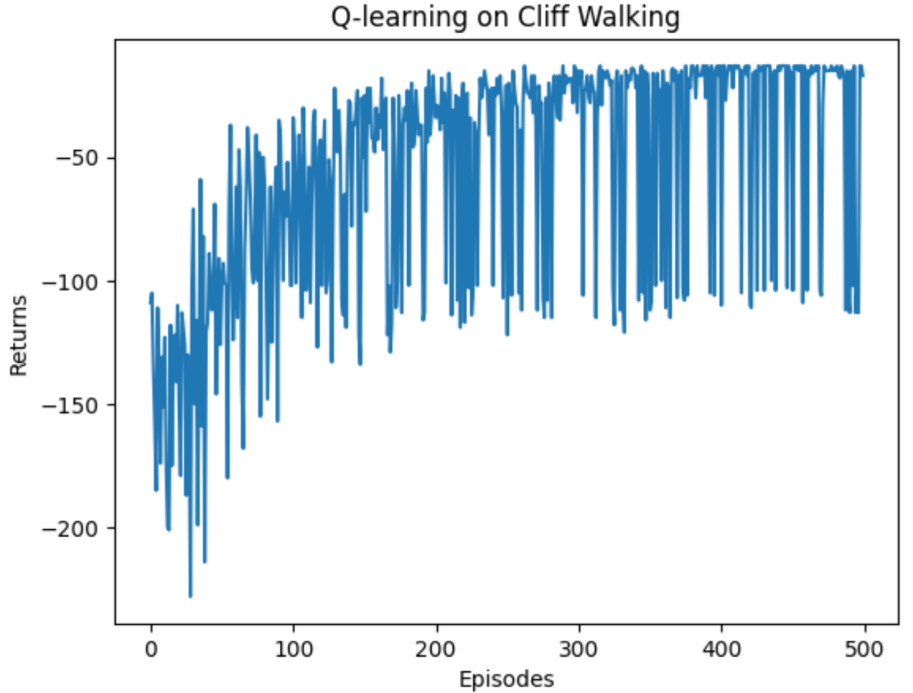
plt.show()

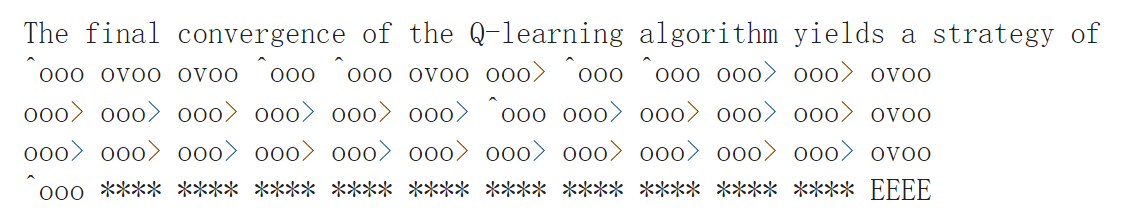
action\_meaning = ['^', 'v', '<', '>']

print('The final convergence of the Q-learning algorithm yields a strategy of')

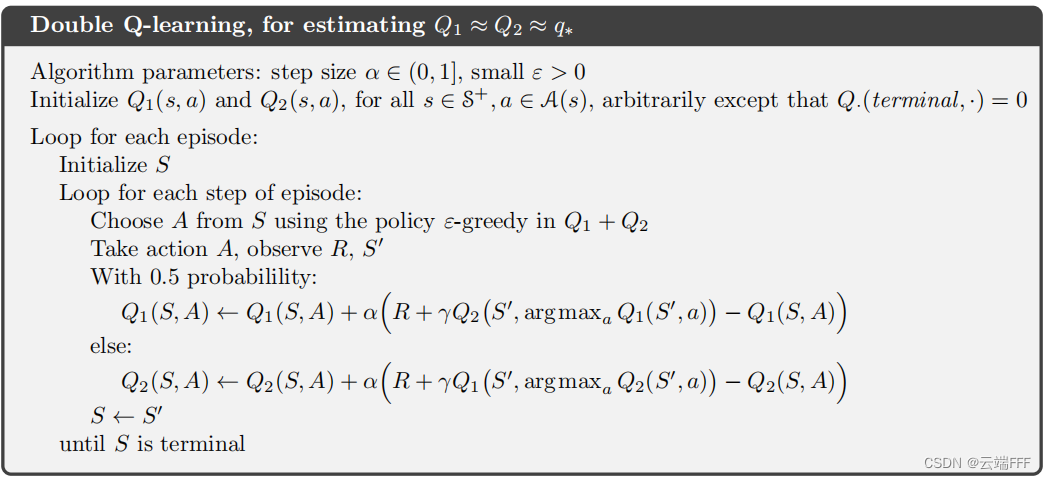
print\_agent(agent, env, action\_meaning, list(range(37, 47)), [47])







### Двойное Q-обучение



import gym

import numpy as np

import matplotlib.pyplot as plt

from tqdm import tqdm

class DoubleQLearning:

    def \_\_init\_\_(self, env, alpha=0.1, gamma=0.9, epsilon=0.1):

        self.env = env

        self.alpha = alpha

        self.gamma = gamma

        self.epsilon = epsilon

        self.Q1 = np.zeros((env.observation\_space.n, env.action\_space.n))

        self.Q2 = np.zeros((env.observation\_space.n, env.action\_space.n))

    def take\_action(self, state):

        if np.random.uniform() < self.epsilon:

            action = self.env.action\_space.sample()

        else:

            action = self.best\_action(state)

        return action

    def update(self, state, action, reward, next\_state):

        if np.random.uniform() < 0.5:

            Q1, Q2 = self.Q1, self.Q2

        else:

            Q1, Q2 = self.Q2, self.Q1

        a\_max = np.argmax(Q1[next\_state])

        td\_target = reward + self.gamma \* Q2[next\_state][a\_max]

        td\_error = td\_target - Q1[state][action]

        Q1[state][action] += self.alpha \* td\_error

    def best\_action(self, state):

        return np.argmax(self.Q1[state])

    def train(self, num\_episodes=1000):

        return\_list = []  # 记录每一条序列的回报

        for i in range(10):  # 显示10个进度条

            # tqdm的进度条功能

            with tqdm(total=int(num\_episodes / 10), desc='Iteration %d' % i) as pbar:

                for i\_episode in range(int(num\_episodes / 10)):  # 每个进度条的序列数

                    episode\_return = 0

                    state = self.env.reset()

                    done = False

                    while not done:

                        action = self.take\_action(state)

                        next\_state, reward, done, info = self.env.step(action)

                        episode\_return += reward  # 这里回报的计算不进行折扣因子衰减

                        self.update(state, action, reward, next\_state)

                        state = next\_state

                    return\_list.append(episode\_return)

                    if (i\_episode + 1) % 10 == 0:  # 每10条序列打印一下这10条序列的平均回报

                        pbar.set\_postfix({

                            'episode':

                            '%d' % (num\_episodes / 10 \* i + i\_episode + 1),

                            'return':

                            '%.3f' % np.mean(return\_list[-10:])

                        })

                    pbar.update(1)

        return return\_list

    def test(self):

        state = self.env.reset()

        done = False

        steps = 0

        while not done:

            action = self.best\_action(state)

            state, reward, done, info = self.env.step(action)

            steps += 1

        return steps

env = gym.make('CliffWalking-v0')

agent = DoubleQLearning(env)

return\_list = agent.train(num\_episodes=500)

episodes\_list = list(range(len(return\_list)))

plt.plot(episodes\_list, return\_list)

plt.xlabel('Episodes')

plt.ylabel('Returns')

plt.title('Double Q-learning on {}'.format('Cliff Walking'))

plt.show()

