
بسم الله الرحمن الرحيم

دانشگاه صنعتی اصفها ن
دانشکده برق و کامپیوتر

# گزارش درس یادگیری تقویتی- تکلیف دوم

**دکتر مازیار پالهنگ**

**دانشجو :**

**پردیس مرادبیکی**

**شماره دانشجویی :**

**٤٠٠٠١٥٦٥**

**نیمسال دوم ١٤٠١-١٤٠٠**

فهرست مطالب

# فهرست شکل‌ها

# گزارش تکلیف آشنائی با تفاوت زمانی چندگامی

**تمرین اول :**

**متن سوال یک: برنامه‌ای به زبان پایتون بنویسید و با استفاده از الگوریتم پیشگویی تفاوت زمانی چندگامی را با فرض اینکه احتمال انجام همه‌ی اعمال یکسان است و ضریب تخفیف ۰.۹است مقدار تابع حالت را بدست آورده و نمایش دهید. سعی کنید آزممایش را برای سه مقدار متفاوت برای تعداد گام انجام دهید. (با استفاده از n=۱ حتما انجام دهید.) آیا تفاوتی در مقادیر بدست آمده توسط گامهای متفاوت وجود دارد؟ آیا تغییر مقدار نرخ یادگیری (آلفا) در نتایج به دست آمده اثر دارد؟ چه مقداری به نظرتان مناسبتر بوده است؟**

**شرح برنامه تمرین اول:**

```
#تکلیف اول : الگوریتم پیشگویی تفاوت زمانی

#کتابخانه مورد نیاز
import numpy as np
```

*شکل ۱ - فراخوانی کتابخانه*

**شرح خطوط کد:**

کتابخانه‌ای که جهت اجرای این برنامه فراخوانی می‌شود، به شرح زیر است:

- کتابخانه numpy برای انجام اعمال ریاضی بر روی آرایه ها و ماتریس ها

```
#تعریف همسایه ها در هر حالت
#u ---> بالا
#d ---> پایین
#l ---> چپ
#r ---> راست

neighbors = {0:{'u':8, 'd':0, 'l':0, 'r':1},
             1:{'u':9, 'd':1, 'l':0, 'r':2},
             2:{'u':10, 'd':2, 'l':1, 'r':3},
             3:{'u':11, 'd':3, 'l':2, 'r':4},
             4:{'u':4, 'd':4, 'l':3, 'r':5},
             5:{'u':13, 'd':5, 'l':4, 'r':6},
             6:{'u':14, 'd':6, 'l':5, 'r':7},
             7:{'u':15, 'd':7, 'l':6, 'r':7},


             8:{'u':16, 'd':0, 'l':8, 'r':9},
             9:{'u':17, 'd':1, 'l':8, 'r':10},
             10:{'u':18, 'd':2, 'l':9, 'r':11},
             11:{'u':19, 'd':3, 'l':10, 'r':11},
             12:{'u':12, 'd':12, 'l':12, 'r':12},###دیوار
             13:{'u':21, 'd':5, 'l':13, 'r':14},
             14:{'u':22, 'd':6, 'l':13, 'r':15},
             15:{'u':23, 'd':7, 'l':14, 'r':15},


             16:{'u':24, 'd':8, 'l':16, 'r':17},
             17:{'u':25, 'd':9, 'l':16, 'r':18},
             18:{'u':18, 'd':10, 'l':17, 'r':19},
             19:{'u':27, 'd':11, 'l':18, 'r':20},
```

*شکل۲- ایجاد همسایه های محیط شبکه ای*

```
#ایجاد سیاست تصادفی اولیه


policy  = {0:{'u':0.25, 'd':0.25, 'l':0.25, 'r':0.25},
           1:{'u':0.25, 'd':0.25, 'l':0.25, 'r':0.25},
           2:{'u':0.25, 'd':0.25, 'l':0.25, 'r':0.25},
           3:{'u':0.25, 'd':0.25, 'l':0.25, 'r':0.25},
           4:{'u':0.25, 'd':0.25, 'l':0.25, 'r':0.25},
           5:{'u':0.25, 'd':0.25, 'l':0.25, 'r':0.25},
           6:{'u':0.25, 'd':0.25, 'l':0.25, 'r':0.25},
           7:{'u':0.25, 'd':0.25, 'l':0.25, 'r':0.25},


           8:{'u':0.25, 'd':0.25, 'l':0.25, 'r':0.25},
           9:{'u':0.25, 'd':0.25, 'l':0.25, 'r':0.25},
           10:{'u':0.25, 'd':0.25, 'l':0.25, 'r':0.25},
           11:{'u':0.25, 'd':0.25, 'l':0.25, 'r':0.25},
           12:{'u':0, 'd':0, 'l':0, 'r':0},###دیوار
           13:{'u':0.25, 'd':0.25, 'l':0.25, 'r':0.25},
           14:{'u':0.25, 'd':0.25, 'l':0.25, 'r':0.25},
           15:{'u':0.25, 'd':0.25, 'l':0.25, 'r':0.25},


           16:{'u':0.25, 'd':0.25, 'l':0.25, 'r':0.25},
           17:{'u':0.25, 'd':0.25, 'l':0.25, 'r':0.25},
           18:{'u':0.25, 'd':0.25, 'l':0.25, 'r':0.25},
           19:{'u':0.25, 'd':0.25, 'l':0.25, 'r':0.25},
           20:{'u':0.25, 'd':0.25, 'l':0.25, 'r':0.25},
           21:{'u':0.25, 'd':0.25, 'l':0.25, 'r':0.25},
           22:{'u':0.25, 'd':0.25, 'l':0.25, 'r':0.25},
           23:{'u':0.25, 'd':0.25, 'l':0.25, 'r':0.25},
```

*شکل۳- ایجاد سیاست تصادفی اولیه برای محیط شبکه ای*

```
reward   = {0:{'u':0, 'd':0, 'l':0, 'r':0},
             1:{'u':0, 'd':0, 'l':0, 'r':0},
             2:{'u':0, 'd':0, 'l':0, 'r':0},
             3:{'u':0, 'd':0, 'l':0, 'r':0},
             4:{'u':0, 'd':0, 'l':0, 'r':0},
             5:{'u':0, 'd':0, 'l':0, 'r':0},
             6:{'u':0, 'd':0, 'l':0, 'r':0},
             7:{'u':0, 'd':0, 'l':0, 'r':0},


             8:{'u':0, 'd':0, 'l':0, 'r':0},
             9:{'u':0, 'd':0, 'l':0, 'r':0},
             10:{'u':0, 'd':0, 'l':0, 'r':0},
             11:{'u':0, 'd':0, 'l':0, 'r':0},
             12:{'u':0, 'd':0, 'l':0, 'r':0},#دیوار
             13:{'u':0, 'd':0, 'l':0, 'r':0},
             14:{'u':0, 'd':0, 'l':0, 'r':0},
             15:{'u':0, 'd':0, 'l':0, 'r':0},


             16:{'u':0, 'd':0, 'l':0, 'r':0},
             17:{'u':0, 'd':0, 'l':0, 'r':0},
             18:{'u':0, 'd':0, 'l':0, 'r':0},
             19:{'u':0, 'd':0, 'l':0, 'r':0},
             20:{'u':0, 'd':0, 'l':0, 'r':0},
             21:{'u':0, 'd':0, 'l':0, 'r':0},
             22:{'u':0, 'd':0, 'l':0, 'r':0},
             23:{'u':0, 'd':0, 'l':0, 'r':0},


             24:{'u':0, 'd':0, 'l':0, 'r':0},
             25:{'u':0, 'd':0, 'l':0, 'r':0},
```

*شکل۴- تعریف پاداش به ازای هر اقدام برای محیط شبکه ای*

```
#تعریف دیوار در محیط شبکه ای

taboo = [12,26,34,42,46,54,62]
```

*شکل ۵- تعریف خانه های دارای دیوار در محیط شبکه ای*

## شرح خطوط کد:

در یادگیری تقویتی[1]، محیط در واقع محلی است که عامل در آن زندگی و تعامل می کند.

محیط باتوجه به صورت سئوال، شبکه‌ای ۸*۸ است که حاصل آن ۶۴ موقعیت[2]یا حالت ممکن برای عامل است. این ۶۴ حالت بخشی از فضای حالت[3] محسوب می‌شوند. عامل[4] تنها موجود در این شبکه است که در حال حرکت به سمت نقطه پایانی[5] است. ابتدا عامل از موقعیت ۲۴ شروع به حرکت میکند.

[1] Reinforcement learning
[2] state
[3] Set of states
[4] agent
[5] goal

عمل معمولا بر پایه محیط است و محیط‌های متفاوت منجر به اعمال متفاوتی می‌شوند. اعمال معمولا تعداد مشخصی دارند. در اینجا ۴ عمل بالا، پایین، چپ و راست را می‌توان در هر حالت انجام داد که به ترتیب اعداد ۰-۳ را به آنها نسبت داده شده است.

در محیط شبکه‌ای مطرح شده در صورت سئوال، موقعیت‌هایی تحت عنوان دیوار مطرح شده که عامل حق ورود به آن‌ها را ندارد و در صورتی که حرکتی برای رفتن به آن موقعیت ها داشته باشد، در همان موقعیت باید بماند.

در این قسمت برای تمام ۶۴ موقعیت که زمین ۸*۸ دارد اگر هدف باشد پاداش خروجی را ۱۰۰ در نظر میگیریم و بقیه پاداش صفر دارند و با هر حرکت عامل در محیط موقعیت عامل با توجه به عمل انجام شده در محیط تغییر پیدا می‌کند.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| ۵۶ | ۵۷ | ۵۸ | ۵۹ | ۶۰ | ۶۱ | ۶۲ | ۶۳ Goal |
| ۴۸ | ۴۹ | ۵۰ | ۵۱ | ۵۲ | ۵۳ | ۵۴ | ۵۵ |
| ۴۰ | ۴۱ | ۴۲ | ۴۳ | ۴۴ | ۴۵ | ۴۶ | ۴۷ |
| ۳۲ | ۳۳ | ۳۴ | ۳۵ | ۳۶ | ۳۷ | ۳۸ | ۳۹ |
| ۲۴ Start | ۲۵ | ۲۶ | ۲۷ | ۲۸ | ۲۹ | ۳۰ | ۳۱ |
| ۱۶ | ۱۷ | ۱۸ | ۱۹ | ۲۰ | ۲۱ | ۲۲ | ۲۳ |
| ۸ | ۹ | ۱۰ | ۱۱ | ۱۲ | ۱۳ | ۱۴ | ۱۵ |
| ۰ | ۱ | ۲ | ۳ | ۴ | ۵ | ۶ | ۷ |

*شکل۶- خروجی محیط شبکه ای ایجاد شده*

## شرح شکل۳:

همانطور که در شکل مشاهده می‌کنید،عامل اگر در حالت ۰ در محیط باشد با انتخاب عمل بالا رفتن موقعیت بعدی عامل در محیط موقعیت ۸ خواهد بود و با احتمال ۰.۲۵ درصد ممکن است عامل عمل بالا رفتن را انتخاب کند و پاداش حرکت عامل در محیط چون به حالت پایانی نمیرسد برابر صفر قرار می‌گیرد. همچنین با انتخاب حرکت به سمت پایین عامل در همان خانه ۰ باقی خواهد ماند.

```
#در این تابع با گرفتن حالت و فعالیت میزان پاداش و حالت بعدی برگردانده میشود

def state_values(s, action, neighbors = neighbors, rewards = reward):

    s_prime = neighbors[s][action]
    reward = rewards[s][action]
    return (s_prime,reward)
```

شکل۷- ایجاد تابع جهت محاسبه مقدار پاداش در حالت بعدی

```
#تابع انتخاب
#در صورتی که حالت هدف نباشد به صورت رندوم یک عمل را انتخاب میکنیم


def select_a_prime(s_prime, policy, action_index, greedy, goal):

    if(s_prime != goal and greedy):
        probs = list(policy[s_prime].values())
        a_prime_index = nr.choice(action_index, size = 1, p = probs)[0]
        a_prime = list(policy[s_prime].keys())[a_prime_index]
    else:
        a_prime_index = nr.choice(action_index, size = 1)[0]
        a_prime = list(policy[s_prime].keys())[a_prime_index]
    return(a_prime_index, a_prime)
```

شکل۸- تابع انتخاب عمل تصادفی

```
#محاسبه تابع پیشگویی تفاوت زمانی چندگامی

def TD_n(policy, episodes, n, goal, alpha, gamma , epsilon ):

    ##  انتخاب لیست حالت و مقادیر عمل را انتخاب میکنیم
    states = list(policy.keys())
    n_states = len(states)

    ## اقدامات احتمالی و مقادیر عمل را انتخاب میکنیم
    action_index = list(range(len(list(policy[0].keys()))))
    v = [0]*len(list(policy.keys()))
    current_policy = copy.deepcopy(policy)
    s = nr.choice(states, size = 1)[0] ## حال در ابتدا یک حالت اولیه را به صورت تصادفی نمونه میگیریم و بررسی میکنیم که حالت پایانی یا دیوار نباشد
    while(s in taboo+[goal]):
        s = nr.choice(states, size = 1)[0]

    for _ in range(episodes): # به ازای هر اپیزود
        T = float("inf")
        tau = 0
        reward_list = []
        t = 0

        while(tau != T - 1): # اپیزود به پایان می رسد که به حالت دیوار یا هدف رفته باشیم
            if(t < T):

                probs = list(policy[s].values())  ## عمل را با توجه به سیاست خود انتخاب میکنیم
                a = list(policy[s].keys())[nr.choice(action_index, size = 1, p = probs)[0]]
                s_prime, reward = state_values(s, a)  ## حال حالت بعدی را با توجه به عمل داده شده انتخاب میکنیم
                reward_list.append(reward)  # پاداش ها را به لیست اضافه میکنیم
                if(s_prime == goal): T = t + 1  # به حالت هدف رسیدیم

            tau = t - n + 1 ## به روز رسانی میکنیم

            if(tau >= 0): # اگر زمان کافی برای محاسبه بازگشت داریم بازگشت را محاسبه میکنیم
                G = 0.0
                for i in range(tau, min(tau + n - 1, T)):
                    G = G + gamma**(i-tau) * reward_list[i]
                if(tau + n < T): G = G + gamma**n * v[s_prime]  ## در مورد جایی که ما در حالت دیوار یا هدف نیستیم برخورد کنیم
                v[s] = v[s] + alpha * (G - v[s])  ## مقدار تابع مقدار را ایدیت میکنیم

            ## حالت را برای تکرار بعدی تنظیم می کنیم
            if(s_prime != goal):
                s = s_prime
            t = t +1
    return(v)
```

شکل۹- تابع پیشگویی تفاوت زمانی چندگامی

## شرح خطوط کد:

لیستی از تمام موقعیت ها تهیه می‌کنیم و یک پارامتر حاوی اندازه این موقعیت‌ها تعریف می‌کنیم که با توجه به صورت مسئله مقدار آن ۶۴ است. تابع مقدار را به صورت رندوم در تابعی جداگانه انتخاب می‌کنیم و مقادیر عمل را به صورت تصادفی انتخاب می‌کنیم.

در ابتدا یک حالت اولیه را به صورت تصادفی انتخاب می‌کنیم و بررسی میکنیم حالت دیوار یا حالت هدف نباشد. اگر موقعیت حالت ترمینال نباشد تا زمانی که به حالت نهایی یا هدف نرسیدیم فرایند را ادامه می دهیم. حال عمل را با توجه به سیاست خود انتخاب می‌کنیم و با توجه به عمل انتخاب شده براساس سیاست حالت بعدی را انتخاب میکنیم و پاداش را به لیست خود اضافه می‌کنیم تا به حالت هدف برسیم و با توجه به فرمول زیر بازگشت را محاسبه می‌کنیم. محاسبه مقدار بازگشت طبق فرمول معادله ۱ صورت می‌گیرد.

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^{n} V_{t+n-1}(S_{t+n})$$

(معادله ۱)

بعد از محاسبه بازگشت، طبق فرمول معادله ۲ تابع مقدار را به روز می‌کنیم. اینکار را تا زمانی که به حالت هدف نرسیدیم تکرار می‌کنیم.

$$V_{t+n}(s) \leftarrow V_{t+n-1}(s) + \alpha[G^{(n)}_t - V_{t+n-1}(s)]$$

(معادله ۲)

```
فراخوانی تابع#

V_1 = np.round(np.array(TD_n(policy, episodes = 100000, n = 1, goal = 63, alpha = 0.1, gamma = 0.9, epsilon = 1)).reshape((8,8)), 8)
```

*شکل۱۰-تعیین پارامترها و فراخوانی تابع پیشگویی تفاوت زمانی چندگامی*

## شرح خطوط کد:

در الگوریتم سیاستی را بعنوان سیاست تصادفی با احتمال یکنواخت با توجه به فرمول معادله ۳ تعریف و مقداردهی اولیه می-کنیم.

$$\pi(a|s) = \frac{1}{|A(s)|} \quad s \in S, a \in A(s)$$ (معادله 3)

ماتریس مربوط به سیاست یک ماتریس دو بعدی است. ماتریس ۴*۶۴ که تعداد سطرهای آن تعداد موقعیت‌های موجود در این محیط ( ۶۴) و ستون‌هایش تعداد اعمال (۴) است.

پارامترها در این برنامه به شرح زیر است:

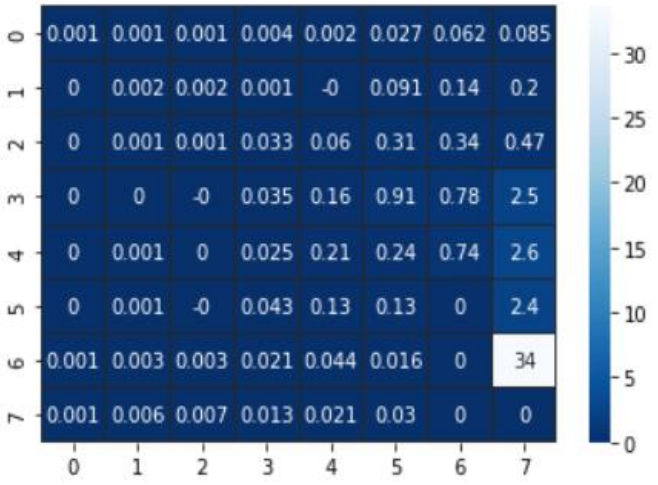
- episode: تعداد تکرار های حلقه برای به روزرسانی پارامترها است.
- n: تعداد گام است که در اینجا از ما خواسته شده برای سه مقدار مختلف گزارش تهیه کنیم.
- alpha : نرخ یادگیری[6] است که سرعت ما در حرکت به سوی هدف اصلی را مشخص می‌کند.
- gamma: ضریب تخفیف[7] است. عددی بین ۰ و ۱ که در این مسئله میزان تاثیر پاداش‌های فعلی را به ازاء رفتن از حالت s به حالت s´ با انتخاب عمل a، تقریبا همانند پاداش‌های قبلی در نظر می‌گیریم(۱=۷).
- epsilon: ضریب انتخاب عمل تصادفی است. با توجه به این که در این مسئله تصادفی یکنواخت اعلام شده است بنابراین در این‌جا مقدار آن برابر یک گذاشته شده است.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| ۰ | ۱ | ۲ | ۳ | ۴ | ۵ | ۶ | ۷ |
| ۸ | ۹ | ۱۰ | ۱۱ | ۱۲ | ۱۳ | ۱۴ | ۱۵ |
| ۱۶ | ۱۷ | ۱۸ | ۱۹ | ۲۰ | ۲۱ | ۲۲ | ۲۳ |
| ۲۴ شروع | ۲۵ | ۲۶ | ۲۷ | ۲۸ | ۲۹ | ۳۰ | ۳۱ |
| ۳۲ | ۳۳ | ۳۴ | ۳۵ | ۳۶ | ۳۷ | ۳۸ | ۳۹ |
| ۴۰ | ۴۱ | ۴۲ | ۴۳ | ۴۴ | ۴۵ | ۴۶ | ۴۷ |
| ۴۸ | ۴۹ | ۵۰ | ۵۱ | ۵۲ | ۵۳ | ۵۴ | ۵۵ |
| ۵۶ | ۷ | ۵۸ | ۵۹ | ۶۰ | ۶۱ | ۶۲ | ۶۳ هدف |

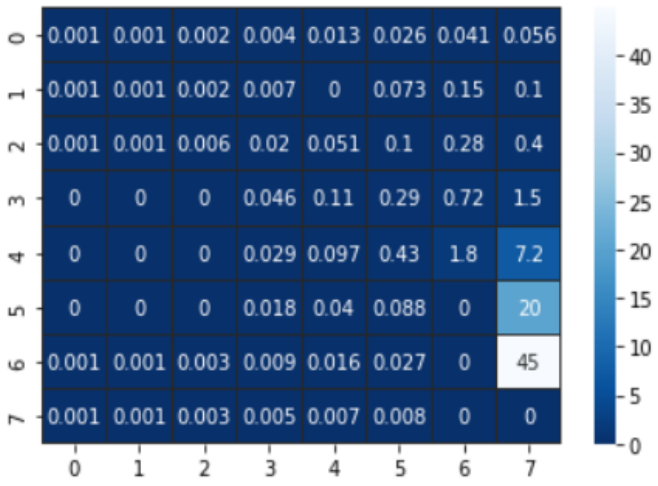
*شکل۱۱- فرم خروجی تابع*

[6] Learning rate
[7] Discount factor

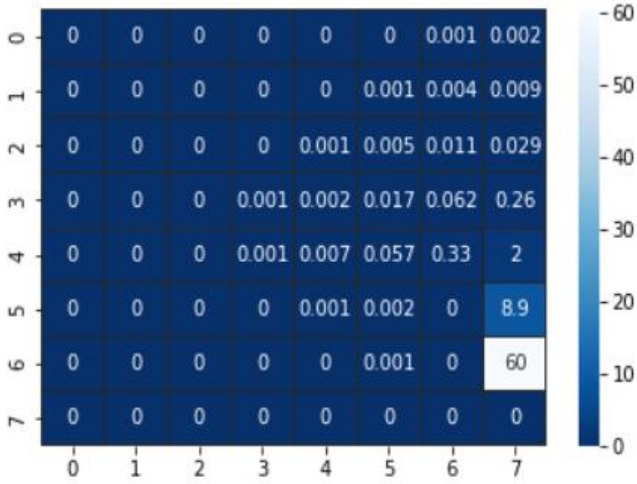
## ۱-۱ خروجی کد تمرین اول:




(الف)




(ب)



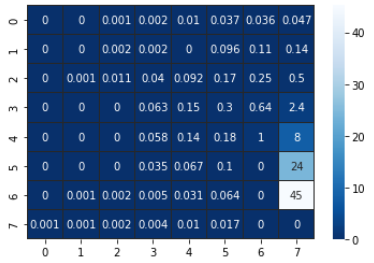

(ج)

*شکل۱۲- مقادیر خروجی الگوریتم پیشگویی تفاوت زمانی چندگامی*

تعداد گام: الف) n=1 ، ب) n=2، ج) n=6

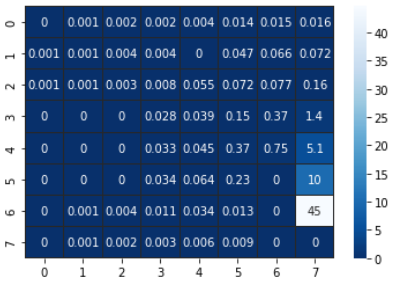
تعداد گام زیاد باعث ندیدن خانه های نزدیک می‌شود در نتیجه صفر می‌شود

```
V_45 = np.round(np.array(TD_n(policy, episodes = 1000, n = 2, goal = 63, alpha = 0.2, gamma = 0.9, epsilon = 1)).reshape((8,8)), 8)
```



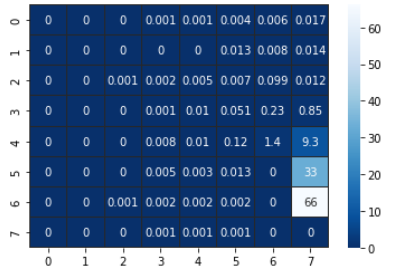

(الف)

```
V_45 = np.round(np.array(TD_n(policy, episodes = 1000, n = 2, goal = 63, alpha = 0.4, gamma = 0.9, epsilon = 1)).reshape((8,8)), 8)
```



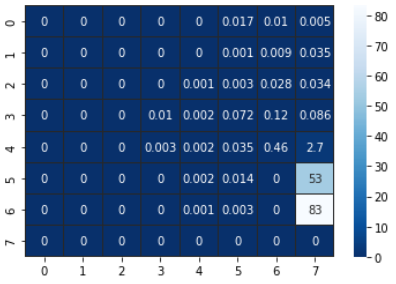

(ب)

```
V_45 = np.round(np.array(TD_n(policy, episodes = 1000, n = 2, goal = 63, alpha = 0.6, gamma = 0.9, epsilon = 1)).reshape((8,8)), 8)
```



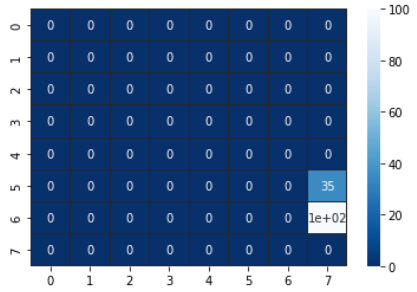

(ج)

```
V_45 = np.round(np.array(TD_n(policy, episodes = 1000, n = 2, goal = 63, alpha = 0.8, gamma = 0.9, epsilon = 1)).reshape((8,8)), 8)
```




(د)

```
V_45 = np.round(np.array(TD_n(policy, episodes = 1000, n = 2, goal = 63, alpha = 1, gamma = 0.9, epsilon = 1)).reshape((8,8)), 8)
```




(ز)

نرخ یادگیری: الف) ۰.۲ ، ب) ۰.۴ ، ج) ۰.۶ د) ۰.۸ ز)۰.۹

مقادیر زیاد برای نرخ یادگیری مناسب نیست.

**نتیجه خروجی کد تمرین اول:** ما باید با تغییر مقدار نرخ یادگیری و مقدار n مقادیر را به تمرین قبلی که با دانش mdp به دست می آمد نزدیک کنیم در اینجا با مقادیر بزرگ n با نرخ یادگیری کمتر یا مقادیر کوچک n با نرخ یادگیری بزرگتر می‌توانیم به مقدار درست نزدیک شویم (در کد به ازای مقادیر مختلف گذاشته شده است). و تفاوت درمقادیر به دست آمده توسط گام‌های مختلف وجود دارد. همچنین تغییر نرخ یادگیری در نتایج به دست آمده اثر دارد. نرخ یادگیری سرعت حرکت به سوی هدف اصلی است نرخ یادگیری و TD باید به طور صحیح انتخاب شود و این پارامتر ها میتوانند تاثیر عمده ای بر عملکرد ما داشته باشد به ویژه پارامتر نرخ یادگیری که برای پایداری فرایند و همنین تعداد مشاهدات مورد نیاز تاثیر می‌گذارد. . نرخ یادگیری نزدیک به صفر به مقادیری که جدیدا به دست آورده اهمیت میدهد و برعکس.

مقداری برای تعداد گام و نرخ یادگیری مناسب است که ما را به مقدار تابع مقدار تمرین قبل نزدیک کند باید با تغییر پارامتر ها به این مقدار برسیم.

## ۲- تمرین دوم:

**متن سوال دوم: با استفاده از الگوریتم سارسای چندگامی سیاست بهینه ای که با استفاده از ین الگوریتم بدست می آید را بدست آورده و نمایش دهید. آزمایش را برای سه مقدار متفاوت برای تعداد گام انجام دهید(با n=۱ حتمًا انجام دهید). آیا تفاوتی در مقادیر بدست آمده توسط گامهای متفاوت وجود دارد؟ آیا تغییر مقدار نرخ یادگیری در نتایج بدست آمده اثر دارد؟ چه مقداری به نظرتان مناسبتر بوده است؟تعداد**

**وقایع چه مقدار مناسب بوده است؟ در تعیین مقدار اپسیلون هنگام استفاده از روش اپسیلون-حریصانه دقت نمائید و توضیح دهید که چگونه آن را تعیین کرده اید.**

## شرح برنامه تمرین دوم:

```
#تکلیف دوم : الگوریتم سارسای چند گامی

#کتابخانه مورد نیاز
import numpy as np
```

*شکل۱۳- فراخوانی کتابخانه*

## شرح خطوط کد:

کتابخانه‌ی مورد نیاز جهت اجرای این برنامه که فراخوانی می‌شود، به شرح زیر است:

- کتابخانه numpy برای انجام اعمال ریاضی بر روی آرایه ها و ماتریس ها

```
# الگوریتم سارسای چند گامی

def SARSA_n(policy, episodes, n, goal, alpha = 0.2, gamma = 0.9, epsilon = 0.1):
    states = list(policy.keys())   ##  انتخاب لیست حالت و مقادیر عمل را انتخاب میکنیم
    n_states = len(states)
    action_index = list(range(len(list(policy[0].keys()))))      ## اقدامات احتمالی و مقادیر عمل را انتخاب میکنیم
    Q = np.zeros((len(action_index),len(states)))
    current_policy = copy.deepcopy(policy)
    for _ in range(episodes): # برای هر اپیزود یا واقعه
        s = nr.choice(states, size = 1)[0] ##یک حالت را به صورت تصادفی انتخاب میکنیم و بررسی میکنیم که حالت پایانی یا دیوار نباشد
        while(s in taboo+[goal]):
            s = nr.choice(states, size = 1)[0]
        a_index, a = select_a_prime(s, current_policy, action_index, True, goal) ##با توجه به سیاست یک عمل را انتخاب میکنیم
        t = 0   ## شروع time step
        T = float("inf")
        tau = 0
        reward_list = []
        while(tau != T - 1): # تا رسیدن به حالت پایانی ادامه میدهیم
            if(t < T):
                s_prime, reward = state_values(s, a) ##محاسبه حالت بعدی و پاداش آن با توجه به مقدار عمل
                reward_list.append(reward)  # اضافه کردن پاداش به لیست خود
                if(s_prime == goal): T = t + 1  # اگر به حالت هدف رسیده باشیم
                else:
                    a_prime_index, a_prime = select_a_prime(s_prime, current_policy, action_index, True, goal) ##اقدام بعدی را با استفاده از سیاست انتخاب و ذخیره کنید
            tau = t - n + 1 ## به روزرسانی time step
            if(tau >= 0): # اگر زمان کافی بود مقدار بازگشت را محاسبه میکنیم
                G = 0.0
                for i in range(tau, min(tau + n, T)):
                    G = G + gamma**(i-tau) * reward_list[i]
                if(tau + n < T): G = G + gamma**n * Q[a_prime_index,s_prime]
                Q[a_index,s] = Q[a_index,s] + alpha * (G - Q[a_index,s])  #را اپدیت میکنیمQ مقدار
            if(s_prime != goal):   ##اقدام و حالت را برای تکرار بعدی تنظیم کنید
                s = s_prime
                a = a_prime
                a_index = a_prime_index

            t = t + 1
    return(Q)
```

*شکل۱۴- ایجاد تابع سارسای چندگامی*

## شرح خطوط کد:

لیستی از تمام موقعیت ها تهیه می‌کنیم و یک پارامتر حاوی اندازه این موقعیت‌ها تعریف می‌کنیم که با توجه به صورت مسئله مقدار آن ۶۴ است. تابع مقدار را به صورت رندوم در تابعی جداگانه انتخاب می‌کنیم و مقادیر عمل را به صورت تصادفی انتخاب می‌کنیم.

در ابتدا یک حالت اولیه را به صورت تصادفی انتخاب می‌کنیم و بررسی می‌کنیم حالت دیوار یا حالت هدف نباشد. اگر موقعیت حالت ترمینال نباشد تا زمانی که به حالت نهایی یا هدف نرسیدیم فرایند را ادامه می دهیم.  حال عمل را با توجه به سیاست خود انتخاب می‌کنیم و با توجه به عمل انتخاب شده براساس سیاست حالت بعدی را انتخاب می‌کنیم و پاداش را به لیست خود اضافه می‌کنیم تا به حالت هدف برسیم و با توجه به فرمول زیر بازگشت را محاسبه می‌کنیم.  محاسبه مقدار بازگشت طبق فرمول معادله ۱ صورت می‌گیرد.

$$G_t^{(n)} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n})$$

(معادله ۴)

```
Q = SARSA_n(policy, episodes = 1000, n = 1, goal = 63, alpha = 0.2, gamma = 0.9 ,epsilon = 0)
```

*شکل۱۵-تعیین پارامترها و فراخوانی تابع سارسای چندگامی*

پارامترها در این برنامه به شرح زیر است:

- episode: تعداد تکرار های حلقه برای به روزرسانی پارامترها است.
- n: تعداد گام است که در اینجا از ما خواسته شده برای سه مقدار مختلف گزارش تهیه کنیم.
- alpha : نرخ یادگیری[8] است که سرعت ما در حرکت به سوی هدف اصلی را مشخص می‌کند.
- gamma: ضریب تخفیف[9] است. عددی بین ۰ و ۱ که در این مسئله میزان تاثیر پاداش‌های فعلی را به ازاء رفتن از حالت s به حالت s′ با انتخاب عمل a، تقریبا همانند پاداش‌های قبلی در نظر می‌گیریم(۱=γ).
- epsilon: ضریب انتخاب عمل تصادفی است. با توجه به این که در این مسئله تصادفی یکنواخت اعلام شده است بنابراین در این‌جا مقدار آن برابر یک گذاشته شده است.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| ۰ | ۱ | ۲ | ۳ | ۴ | ۵ | ۶ | ۷ |
| ۸ | ۹ | ۱۰ | ۱۱ | ۱۲ | ۱۳ | ۱۴ | ۱۵ |
| ۱۶ | ۱۷ | ۱۸ | ۱۹ | ۲۰ | ۲۱ | ۲۲ | ۲۳ |
| ۲۴ شروع | ۲۵ | ۲۶ | ۲۷ | ۲۸ | ۲۹ | ۳۰ | ۳۱ |
| ۳۲ | ۳۳ | ۳۴ | ۳۵ | ۳۶ | ۳۷ | ۳۸ | ۳۹ |
| ۴۰ | ۴۱ | ۴۲ | ۴۳ | ۴۴ | ۴۵ | ۴۶ | ۴۷ |
| ۴۸ | ۴۹ | ۵۰ | ۵۱ | ۵۲ | ۵۳ | ۵۴ | ۵۵ |
| ۵۶ | ۷ | ۵۸ | ۵۹ | ۶۰ | ۶۱ | ۶۲ | ۶۳ هدف |

*شکل۱۶-فرم خروجی تابع*

[8] Learning rate
[9] Discount factor

## ۲-۱ خروجی کد تمرین دوم:

همانطور که درشکل میبینید

```
Q = SARSA_n(policy, episodes = 1000, n = 1, goal = 63, alpha = 0.2, gamma = 0.9 ,epsilon = 0)

optimal_policy=
 [['>' '>' '>' '>' '>' 'v' 'v' 'v']
 ['>' '>' '>' 'v' '-' 'v' 'v' 'v']
 ['>' '>' '>' '>' '>' '>' 'v' 'v']
 ['^' '^' '-' '>' '>' '>' '>' 'v']
 ['v' '^' '-' '>' '>' '>' '>' 'v']
 ['v' 'v' '-' '>' '^' '^' '-' 'v']
 ['>' '>' '>' '^' '^' '^' '-' 'v']
 ['>' '>' '>' '^' '^' '^' '-' 'v']]
```

(الف)

```
Q = SARSA_n(policy, episodes = 1000, n = 2, goal = 63, alpha = 0.2, gamma = 0.9 ,epsilon = 0)

optimal_policy=
 [['>' '>' '>' '>' '>' 'v' 'v' 'v']
 ['>' '>' 'v' 'v' '-' '>' 'v' 'v']
 ['>' '>' '>' '>' '>' 'v' 'v' 'v']
 ['>' '^' '-' '>' '>' '>' 'v' 'v']
 ['>' '^' '-' '>' '>' '>' '>' 'v']
 ['v' 'v' '-' '>' '>' '^' '-' 'v']
 ['>' '>' '>' '>' '^' '^' '-' 'v']
 ['>' '>' '>' '>' '>' '^' '-' 'v']]
```

(ب)

```
Q = SARSA_n(policy, episodes = 1000, n = 6, goal = 63, alpha = 0.2, gamma = 0.9 ,epsilon = 0)

optimal_policy=
 [['>' '>' '>' '>' '>' 'v' 'v' 'v']
 ['>' '>' '>' 'v' '-' '>' 'v' 'v']
 ['>' '>' '>' '>' '>' '>' 'v' 'v']
 ['>' '^' '-' '>' '>' '>' '>' 'v']
 ['>' '^' '-' '>' '>' '>' '>' 'v']
 ['>' 'v' '-' '>' '^' '^' '-' 'v']
 ['>' '>' '>' '^' '^' '^' '-' 'v']
 ['>' '>' '>' '^' '^' '^' '-' 'v']]
```

(ج)

*شکل۱۷- خروجی کد تمرین دوم، تعداد گام: الف) n=1، ب) n=2، ج) n=6*

**نتیجه خروجی کد تمرین دوم:** بهمانطور که در شکل میبینید برای تعداد گام های بالا حالت هدف همیت پیدا می‌کند و همه برای رسیدن به حالت پایانی مسیر را نشان می‌دهند و در گام های بالا همه فقط یک سمت را نشان می-دهند.

```
Q = SARSA_n(policy, episodes = 1000, n = 1, goal = 63, alpha = 0.4, gamma = 0.9 ,epsilon = 0)

optimal_policy=
 [['>' '>' '>' '>' '>' 'v' 'v' 'v']
 ['>' '>' '>' 'v' '-' 'v' 'v' 'v']
 ['>' '>' '>' '>' '>' '>' '>' 'v']
 ['<' '^' '-' '>' '>' '>' 'v' 'v']
 ['^' '^' '-' '>' '>' '>' '>' 'v']
 ['v' 'v' '-' '>' '>' '^' '-' 'v']
 ['>' '>' '>' '>' '^' '^' '-' 'v']
 ['>' '>' '>' '^' '>' '^' '-' 'v']]
```

(الف)

```
Q = SARSA_n(policy, episodes = 1000, n = 1, goal = 63, alpha = 0.9, gamma = 0.9 ,epsilon = 0)

optimal_policy=
 [['>' '>' '>' 'v' 'v' '>' 'v' '<']
 ['<' '>' '>' '>' '-' '<' 'v' '<']
 ['>' '^' '>' 'v' '^' '<' '<' 'v']
 ['>' '^' '-' '>' 'v' '<' '>' 'v']
 ['<' '^' '-' '>' '^' '>' '>' '^']
 ['<' '<' '-' '>' '>' '^' '-' '>']
 ['v' 'v' '>' '^' '^' '^' '-' 'v']
 ['v' '<' '^' 'v' 'v' '>' '-' 'v']]
```

(ب)

*شکل۱۸ - خروجی کد تمرین*

نرخ یادگیری الف) ۰.۴ ، ب) ۰.۹

تغییر مقدار نرخ یادگیری در نتایج بسیار اهمیت داشت همانطور که در شکل۱۸ میبینید نرخ یادگیری بسیار بزرگ موجب اشتباه در مقادیر میشود و گاهی حتی به سمت دیوار ها اشاره می‌کند. مقدار نرخ یادگیری کمتر مناسب تر بود.

```
Q = SARSA_n(policy, episodes = 1000, n = 1, goal = 63, alpha = 0.4, gamma = 0.9 ,epsilon = 0)

optimal_policy=
 [['>' '>' '>' '>' '>' 'v' 'v' 'v']
 ['>' '>' '>' 'v' '-' 'v' 'v' 'v']
 ['>' '>' '>' '>' '>' '>' '>' 'v']
 ['<' '^' '-' '>' '>' '>' 'v' 'v']
 ['^' '^' '-' '>' '>' '>' '>' 'v']
 ['v' 'v' '-' '>' '>' '^' '-' 'v']
 ['>' '>' '>' '>' '^' '^' '-' 'v']
 ['>' '>' '>' '^' '>' '^' '-' 'v']]
```

(الف)

```
Q = SARSA_n(policy, episodes = 100, n = 1, goal = 63, alpha = 0.2, gamma = 0.9 ,epsilon = 0)

optimal_policy=
 [['>' '>' '>' '>' '>' '>' 'v' 'v']
 ['>' '>' '>' 'v' '-' 'v' 'v' 'v']
 ['>' '>' '>' '>' '>' '>' 'v' 'v']
 ['^' '^' '-' '>' '>' '>' 'v' 'v']
 ['>' '^' '-' '>' '>' '>' '>' 'v']
 ['v' 'v' '-' '>' '^' '^' '-' 'v']
 ['>' '>' '>' '>' '^' '^' '-' 'v']
 ['>' '>' '>' '^' '^' '^' '-' 'v']]
```

(ب)

```
Q = SARSA_n(policy, episodes = 10, n = 1, goal = 63, alpha = 0.2, gamma = 0.9 ,epsilon = 0)

optimal_policy=
 [['>' '>' 'v' 'v' '>' 'v' 'v' 'v']
 ['>' 'v' 'v' 'v' '-' 'v' 'v' 'v']
 ['>' '>' '>' '>' 'v' '>' '>' 'v']
 ['>' '^' '-' '>' '>' 'v' 'v' 'v']
 ['^' '^' '-' '>' '>' '>' '>' 'v']
 ['^' '^' '-' '>' '>' '^' '-' 'v']
 ['^' 'v' '>' '^' '^' '^' '-' 'v']
 ['>' '>' '^' '^' '>' '^' '-' 'v']]
```

(ج)

```
Q = SARSA_n(policy, episodes = 1, n = 1, goal = 63, alpha = 0.2, gamma = 0.9 ,epsilon = 0)

optimal_policy=
 [['v' 'v' 'v' 'v' 'v' 'v' 'v' 'v']
 ['v' 'v' 'v' 'v' '-' 'v' 'v' 'v']
 ['v' 'v' 'v' 'v' 'v' 'v' 'v' 'v']
 ['v' 'v' '-' 'v' 'v' 'v' 'v' 'v']
 ['v' 'v' '-' 'v' 'v' 'v' 'v' 'v']
 ['v' 'v' '-' 'v' 'v' 'v' '-' 'v']
 ['v' 'v' 'v' 'v' 'v' 'v' '-' 'v']
 ['v' 'v' 'v' 'v' 'v' 'v' '-' 'v']]
```

(د)

*شکل۱۹-خروجی کد تمرین دوم: تعداد واقعه الف) ۱۰۰۰ ب) ۱۰۰ ج) ۱۰ د) ۱*

تعداد واقعه هر چه بیشتر بهتر است تا کمتر باشد.

مقدار پارامتر اپسیلون نشان دهنده درصد حریصانه عمل کردن تابع است. هر چه اپسیلون بزرگتر باشد یعنی همه با یک درصد انتخاب شوند ولی اگر اپسیلون کوچک باشد هر چه کمتر باشد انتخاب اعمال به صورت حریصانه است. برای یافتن سیاست بهینه مقدار پارامتر اپسیلون را صفر میگذاریم تا همه اعمال را حریصانه، و سیاست بهینه را پیدا کند. پارامتر اپسیلون هر چه کمتر باشد از قدرت کاوش الگوریتم کاسته میشود و دیرتر مسیرهای نزدیک به هدف را پیدا می‌کند و درنتیجه واقعه ها طولانی تر، زمان اجرا شدن و رسیدن به نتایج طولانی می‌شود.

برای مقدار تعداد گام و نرخ یادگیری هر دو سوال مثل هم بودند. و تقریبا هر دو همان جواب بهینه را دادند.

Average RMS error over 19 states and first 10 episodes