دانشگاه اصفهان

دانشکده مهندسی کامپیوتر

پروژه :

cliff walking ، پیاده سازی الگوریتم MDP

اعضای گروه :

حمید مهران فر (۴۰۰۳۶۱۳۰۵۸)

رادمهر آقاخانی (۴۰۰۳۶۶۳۰۰۲)

علی کثیری (۴۰۰۳۶۱۳۰۵۲)

استاد : حسین کارشناس

گروه ۱۰

پاییز ۱۴۰۲

ماژول cliff_walking :

در این ماژول مقادیر مربوط به محیط بازی (مثل تعداد صخره ها ، کنش های ممکن ، احتمال انتخاب کنش ها و نحوه قرار گیری انان) قرار دارد .

همچنین در حین ساخت محیط بازی ، دیکشنری P مقدار دهی می شود . به این صورت که کلید های ان ، state های موجود در محیط است . ارزش این کلید ها ، یک دیکشنری دیگر از کنش های ممکن است . کلید های ان دیکشنری ، کنش های ممکن ، و ارزش ان کنش ها ، یک list است که دارای ویژگی های next_state ، reward و احتمال رفتن به ان state است .

همچنین این ماژول یک تابع step دارد که مشخص می کند هر کنش با یک action به کدام خانه برود که با احتمال یک سوم به ان خانه یا خانه های اطراف ان می رود .

ماژول main :

ابتدا یک سری ثابت تعریف می کنیم . ثابت ها عبارت اند از :

V : ارزش خانه ها ، discount_factory : همان گاما موجود در فرمول است که برابر با ۰.۹ است .

States_num : تعداد کل state ها ، action_num : تعداد کل action ها ،

start_state : state شروع ، end_state : state پایان ، policy : ارایه ای به اندازه ی تمام state ها که مشخص می کند بهترین action برای هر خانه کدام است .

```
V = np.zeros(shape=48)
discount_factory = 0.999
states_num = 48
action_num = 4
start_state = 36
end_state = 47
policy = np.zeros(shape=states_num)
```

تابع reward :

next_state و reward را میگیرد و براساس شرط خاصی ، reward را تغییر می دهد . اگر next_state در خانه ی اخر باشد ، reward برابر با ۴۰۰۰ می شود . اگر هم نبود ، reward تغییری نمی کند .(۱- و ۱۰۰- است)

```
def getReward(next_state , r):
    if next_state == end_state:
        return 4000
    return r
```

تابع check_cliff :

بررسی می کند که خانه ی مربوطه ، صخره نباشد . درواقع این کار برای این انجام می شود که ارزش خانه های صخره ثابت بماند . همچنین با استفاده از لیست cliff_position در کلاس CliffWalking بررسی انجام می شود .

```
def check_cliff(state):
    positions = env.cliff_positions
    row = state // 12
    column = state % 12
    value = (row, column)

    if value in positions:
        return True
    return False
```

تابع setCliff :

کار این تابع مقدار دهی اولیه ی صخره هاست . به این صورت که در خانه های صخره ها مقدار ۱۰۰- قرار می دهد .

تابع optimal_policy :

در این تابع با چندین اجرا ، بهترین سیاست برای عامل مشخص می شود . به این صورت که ابتدا به صورت رندوم ، همه ی سیاست ها و ارزش ها با صفر مقدار دهی می شوند . سپس مقادیر هدف و صخره ها مقدار دهی می شوند (با ۴۰۰۰ و ۱۰۰- مقداردهی می شود) با هر بار اجرا ، مقادیر ارایه های V و policy اپدیت می شود . برای هر state ، بررسی می شود که با هر action با چه احتمال و reward و به چه state می رود (همه این موارد در دیکشنری P ذخیره شده است) . سپس با استفاده از فرمول :

$$Q(s,a) \leftarrow T(s'|s,a)*(R(s,a,s')+\gamma V(s))$$

```
q_values = np.zeros(shape=action_num)
for q in range(action_num):
    for a in range(action_num):
        if abs(q - a) != 2:
            P_value = P[s][a][0]
            next_state = P_value[1]
            reward = P_value[2]

            q_values[q] += (1/3) * (getReward(next_state, reward) + discount_factory * V_old[next_state])
```

هر بار مقدار Q (داخل ارایه ی q_values ذخیره می شود) حساب شده و در نهایت Q با بیشترین مقدار داخل V قرار داده می شود . همچین اندیس بیشترین مقدار Q هم به عنوان سیاست انتخاب می شود .

این تغییر ارزش تا جایی پیش می رود که مقدار ارزش ها ، قبل و بعد از تغییر ثابت بماند .

پس از محاسبه سیاست ، ابتدا مقادیر ان چاپ می شود و سپس مقادیر ان به عامل داده می شود . عامل هم با احتمال یک سوم به ان خانه می رود .

منابع :

https://www.youtube.com/watch?v=l87rgLg90HI

https://www.geeksforgeeks.org/markov-decision-process/

https://www.youtube.com/watch?v=BAetsPIojg4

https://stackoverflow.com/questions/37370015/what-is-the-difference-between-value-iteration-and-policy-iteration