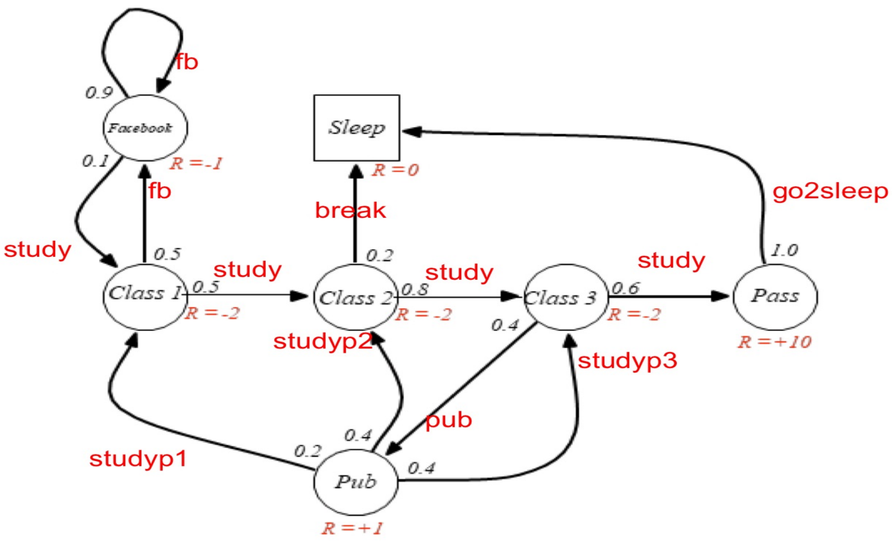
강화학습 HW 1

2022314416

이기창



1. Evaluate value of each State(i.e. V(s) given below.

Value func $V(S) = \mathbb{E}[G_t \mid S_t = s]$, return $G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$

$= R_{t+1} + \gamma V(S_{t+1})$

$$V(s) = R_s + \gamma \sum_{s' \in S} P_{ss'} V(s')$$

$$\underbrace{\begin{bmatrix} V(\text{Sleep}) \\ V(\text{Pass}) \\ V(\text{fb}) \\ V(\text{Pub}) \\ V(C1) \\ V(C2) \\ V(C3) \end{bmatrix}}_{\mathbb{V}} = \underbrace{\begin{bmatrix} 0 \\ 10 \\ -1 \\ 1 \\ -2 \\ -2 \\ -2 \end{bmatrix}}_{\mathbb{R}} + \gamma \underbrace{\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.9 & 0 & 0.1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.2 & 0.4 & 0.4 \\ 0 & 0 & 0.5 & 0 & 0 & 0.5 & 0 \\ 0.2 & 0 & 0 & 0 & 0 & 0 & 0.8 \\ 0 & 0.6 & 0 & 0.4 & 0 & 0 & 0 \end{bmatrix}}_{\mathbb{P}} \underbrace{\begin{bmatrix} V(\text{Sleep}) \\ V(\text{Pass}) \\ V(\text{fb}) \\ V(\text{Pub}) \\ V(C1) \\ V(C2) \\ V(C3) \end{bmatrix}}_{\mathbb{V}}$$

```
GAMMA: 0
   sleep: 0.00
    pass: 10.00
facebook: -1.00
     pub: 1.00
 class 1: -2.00
 class 2: -2.00
 class 3: -2.00
```

$= \gamma = 0$

```
GAMMA: 0.9
   sleep: 0.00
    pass: 10.00
facebook: -7.64
     pub: 1.91
 class 1: -5.01
 class 2: 0.94
 class 3: 4.09
```

$= \gamma = 0.9$

```
GAMMA: 1
   sleep: 0.00
    pass: 10.00
facebook: -22.54
     pub: 0.80
 class 1: -12.54
 class 2: 1.46
 class 3: 4.32
```

$= \gamma = 1$

```python
import numpy as np

GAMMA = 1
states = {0:'sleep',
          1:'pass',
          2:'facebook',
          3:'pub',
          4:'class 1',
          5:'class 2',
          6:'class 3'}

with open('./rewards.txt', 'r') as f: # R
    reward = f.readlines()

reward = np.array(reward, dtype=np.float64)

with open('./policy.txt', 'r') as f: # P
    lines = f.readlines()

probs = []

for line in lines:
    tmp = line.strip().split(' ')
    probs.append(np.array(tmp, dtype=np.float64))

probs = np.array(probs)
I = np.eye(7)

inv_mat = np.linalg.inv(I-GAMMA*probs) # (I-gamma*P)
values = np.matmul(inv_mat,reward)     # (I-gamma*P)^-1 x R

for idx, val in enumerate(values):
    print(f'{states[idx]:>8}: {val:<3.2f}')
```

rewards.txt

```
0
10
-1
1
-2
-2
-2
```

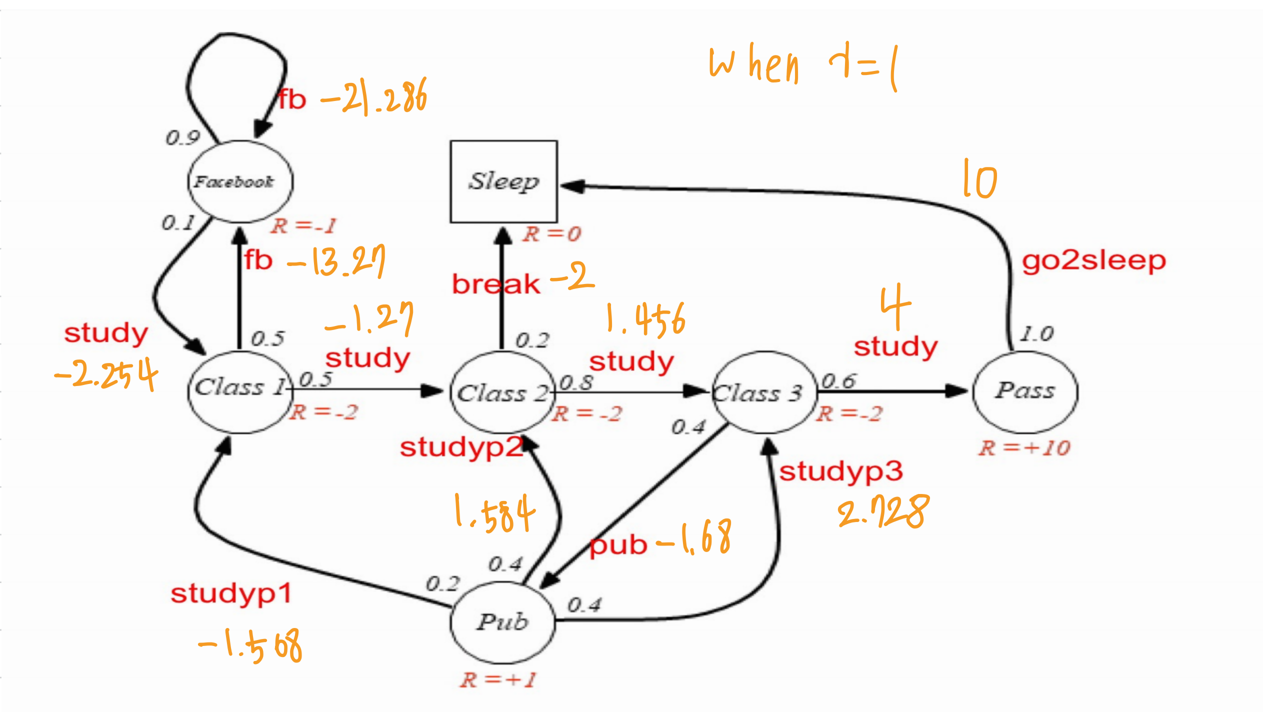
policy.txt

```
0 0 0 0 0 0 0
1 0 0 0 0 0 0
0 0 0.9 0 0.1 0 0
0 0 0 0 0.2 0.4 0.4
0 0 0.5 0 0 0.5 0
0.2 0 0 0 0 0 0.8
0 0.6 0 0.4 0 0 0
```

2. Obtain action values, q(s,a) for each arrow in the given example.

$$q_\pi(s,a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a]$$

$$q_\pi(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \sum_{a' \in A} \pi(a'|s') q_\pi(s',a')$$



$$
\left.
\begin{aligned}
q_\pi(\text{Sleep}, \cdot) &= 0 && = 0 \\
q_\pi(\text{Pass}, \text{go2sleep}) &= 10 + \gamma \cdot 1.0 \cdot 0 && = 10 \\
q_\pi(\text{Class3}, \text{Study}) &= -2 + \gamma \cdot 0.6 \cdot 10 && = 4 \\
q_\pi(\text{Class3}, \text{Pub}) &= -2 + \gamma \cdot 0.4 \cdot V(\text{Pub}) && = -1.68 \\
q_\pi(\text{Class2}, \text{study}) &= -2 + \gamma \cdot 0.8 \cdot V(\text{class3}) && = 1.456 \\
q_\pi(\text{Class2}, \text{break}) &= -2 + \gamma \cdot 0.2 \cdot V(\text{Sleep}) && = -2 \\
q_\pi(\text{Class1}, \text{study}) &= -2 + \gamma \cdot 0.5 \cdot V(\text{Class2}) && = -1.27 \\
q_\pi(\text{Class1}, \text{fb}) &= -2 + \gamma \cdot 0.5 \cdot V(\text{fb}) && = -13.27 \\
q_\pi(\text{fb}, \text{fb}) &= -1 + \gamma \cdot 0.9 \cdot V(\text{fb}) && = -21.286 \\
q_\pi(\text{fb}, \text{study}) &= -1 + \gamma \cdot 0.1 \cdot V(\text{class1}) && = -2.254 \\
q_\pi(\text{Pub}, \text{studyP1}) &= 1 + \gamma \cdot 0.2 \cdot V(\text{Class1}) && = -1.508 \\
q_\pi(\text{Pub}, \text{studyP2}) &= 1 + \gamma \cdot 0.4 \cdot V(\text{Class2}) && = 1.584 \\
q_\pi(\text{Pub}, \text{StudyP3}) &= 1 + \gamma \cdot 0.4 \cdot V(\text{Class3}) && = 2.728
\end{aligned}
\right\} \gamma = 1
$$

```python
# action values
action_value = np.zeros(probs.shape)
for idx, state in enumerate(states.keys()):
    for i in range(len(list(states.keys()))):
        if probs[idx][i] != 0:
            action_value[idx][i] = reward[idx] + GAMMA*probs[idx][i]*values[i]
for idx, vec in enumerate(action_value):
    print(f'{states[idx]:>10}:  ', end='')
    for element in vec:
        print(f'{element:^6.2f} ', end='')
    print()
```

```
GAMMA: 1
   sleep: 0.00
    pass: 10.00
facebook: -22.54
     pub: 0.80
 class 1: -12.54
 class 2: 1.46
 class 3: 4.32
   sleep:   0.00   0.00   0.00   0.00   0.00   0.00   0.00
    pass:  10.00   0.00   0.00   0.00   0.00   0.00   0.00
facebook:   0.00   0.00 -21.29   0.00  -2.25   0.00   0.00
     pub:   0.00   0.00   0.00   0.00  -1.51   1.58   2.73
 class 1:   0.00   0.00 -13.27   0.00   0.00  -1.27   0.00
 class 2:  -2.00   0.00   0.00   0.00   0.00   0.00   1.46
 class 3:   0.00   4.00   0.00  -1.68   0.00   0.00   0.00
```

studyP2

go2sleep

fb

study

break

||

$\gamma = 1$

Pub

studyP1

studyP3

```
GAMMA: 0.9
   sleep: 0.00
    pass: 10.00
facebook: -7.64
     pub: 1.91
 class 1: -5.01
 class 2: 0.94
 class 3: 4.09
   sleep:   0.00   0.00   0.00   0.00   0.00   0.00   0.00
    pass:  10.00   0.00   0.00   0.00   0.00   0.00   0.00
facebook:   0.00   0.00  -7.19   0.00  -1.45   0.00   0.00
     pub:   0.00   0.00   0.00   0.00   0.10   1.34   2.47
 class 1:   0.00   0.00  -5.44   0.00   0.00  -1.58   0.00
 class 2:  -2.00   0.00   0.00   0.00   0.00   0.00   0.94
 class 3:   0.00   3.40   0.00  -1.31   0.00   0.00   0.00
```

||

$\gamma = 0.9$

3. How many iterations do we require to obtain the final value both for V(s) and q(s,a).

```python
state_values     = np.zeros(values.shape)
i = 0
while True:
    print(f'Iteration {i:>02}: ', end='')
    print_val(state_values)
    new_state_values = reward + np.matmul((GAMMA*probs), state_values)
    i += 1
    if np.power(np.power(state_values-new_state_values, 2), 0.5).sum() < 1e-3:
        state_values = new_state_values
        break
    state_values = new_state_values

print(f'Iteration {i:>02}: ', end='')
print_val(state_values)
print()
```

→ V(S)를 모두 0으로 두고

iterative하게 V(S) 계산,

```
Iteration 00:  0.00  0.00  0.00  0.00  0.00  0.00  0.00
Iteration 01:  0.00 10.00 -1.00  1.00 -2.00 -2.00 -2.00
Iteration 02:  0.00 10.00 -2.10 -1.00 -3.50 -3.60  4.40
Iteration 03:  0.00 10.00 -3.24  0.62 -4.85  1.52  3.60
Iteration 04:  0.00 10.00 -4.40  2.08 -2.86  0.88  4.25
Iteration 05:  0.00 10.00 -5.25  2.48 -3.76  1.40  4.83
Iteration 06:  0.00 10.00 -6.10  2.74 -3.92  1.86  4.99
Iteration 07:  0.00 10.00 -6.88  2.96 -4.12  1.99  5.10
Iteration 08:  0.00 10.00 -7.60  3.01 -4.44  2.08  5.18
Iteration 09:  0.00 10.00 -8.29  3.02 -4.76  2.15  5.20
Iteration 10:  0.00 10.00 -8.94  2.99 -5.07  2.16  5.21
Iteration 11:  0.00 10.00 -9.55  2.93 -5.39  2.16  5.20
Iteration 12:  0.00 10.00 -10.13  2.87 -5.69  2.16  5.17
Iteration 13:  0.00 10.00 -10.69  2.79 -5.99  2.14  5.15
Iteration 14:  0.00 10.00 -11.22  2.72 -6.28  2.12  5.12
Iteration 15:  0.00 10.00 -11.72  2.64 -6.55  2.09  5.09
Iteration 16:  0.00 10.00 -12.21  2.56 -6.82  2.07  5.06
Iteration 17:  0.00 10.00 -12.67  2.49 -7.07  2.04  5.02
Iteration 18:  0.00 10.00 -13.11  2.41 -7.31  2.02  4.99
Iteration 19:  0.00 10.00 -13.53  2.34 -7.54  2.00  4.97
Iteration 20:  0.00 10.00 -13.93  2.28 -7.77  1.97  4.94
```

⋮

```
Iteration 146:  0.00 10.00 -22.52  0.81 -12.53  1.46  4.32
Iteration 147:  0.00 10.00 -22.52  0.81 -12.53  1.46  4.32
Iteration 148:  0.00 10.00 -22.52  0.81 -12.53  1.46  4.32
Iteration 149:  0.00 10.00 -22.52  0.81 -12.53  1.46  4.32
Iteration 150:  0.00 10.00 -22.52  0.81 -12.53  1.46  4.32
Iteration 151:  0.00 10.00 -22.52  0.81 -12.53  1.46  4.32
Iteration 152:  0.00 10.00 -22.52  0.81 -12.53  1.46  4.32
Iteration 153:  0.00 10.00 -22.52  0.81 -12.53  1.46  4.32
Iteration 154:  0.00 10.00 -22.52  0.81 -12.53  1.46  4.32
Iteration 155:  0.00 10.00 -22.52  0.81 -12.53  1.46  4.32
Iteration 156:  0.00 10.00 -22.53  0.81 -12.53  1.46  4.32
Iteration 157:  0.00 10.00 -22.53  0.81 -12.53  1.46  4.32
Iteration 158:  0.00 10.00 -22.53  0.81 -12.53  1.46  4.32
Iteration 159:  0.00 10.00 -22.53  0.81 -12.53  1.46  4.32
Iteration 160:  0.00 10.00 -22.53  0.80 -12.54  1.46  4.32
Iteration 161:  0.00 10.00 -22.53  0.80 -12.54  1.46  4.32
Iteration 162:  0.00 10.00 -22.53  0.80 -12.54  1.46  4.32
Iteration 163:  0.00 10.00 -22.53  0.80 -12.54  1.46  4.32
Iteration 164:  0.00 10.00 -22.53  0.80 -12.54  1.46  4.32
Iteration 165:  0.00 10.00 -22.53  0.80 -12.54  1.46  4.32
Iteration 166:  0.00 10.00 -22.53  0.80 -12.54  1.46  4.32
```

→ 166 번째 iteration에서

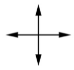
직전 iter과 비교하여

1e-3 이하의 차이를 보임

Write algorithm code for a synchronous Value iteration agent.
Value iteration computes k-step estimates of the state values, V(k) for k=0, 1, 5, 10 and ∞.

1	2	3	+1 Goal with (award) 4
5	6	7	-1 Goal with (penalty) 8
9	10	11	12
13	14	15	16

- Initial policy : random

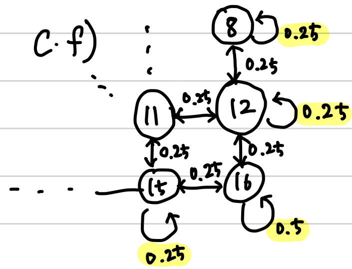

actions

- Tabular MDP with 16 states
- Action : agent allow to move 4 normal direction
- Discount = none (gamma=1.0)
- reward = - 0.1 on all transition
- Two terminal states : one +1 award, the other -1 penalty

총 16개의 state 중 6,7,10,11은 상하좌우 모두 이동이 가능함
1,13,16은 2개의 방향으로만 이동가능
2,3,5,9,12,14,15는 3개의 방향으로 이동가능
4,8은 terminal state로 이동 X

→ 이 2경우에 대해 움직일 수 없는 방향으로의 이동은 제자리로 돌아오도록함

c.f)



<Code>

```
import numpy as np

def print_val(val):
    for v in val:
        print(f'{v:>5.2f}', end=' ')
    print()

#코너에 있는 state들은 self-edge가 있는 case로 볼 수 있음
with open('./probs.csv', 'r', encoding='utf-8') as f:
    lines = f.readlines()

probs = np.ndarray((16, 16))

for idx, line in enumerate(lines):
    for jdx, p in enumerate(line.strip().split(',')):
        probs[idx][jdx] = float(p)

reward = -0.1*np.ones(16)
reward[3] += 1
reward[7] += -1

GAMMA = 1.0

I = np.eye(16)

inv_mat = np.linalg.inv(I-GAMMA*probs)
values = np.matmul(inv_mat, reward)
for idx, val in enumerate(values):
    print(f'{val:<3.2f}')
print(values.reshape(4, 4))

i = 0
state_values = np.zeros(reward.shape)
state_values[3] += 1
state_values[7] += -1
while True:
    if i==0 or i == 1 or i == 5 or i == 10:
        print(f'Iteration {i:>02}: ', end='')
        print_val(state_values)
    new_state_values = reward + np.matmul((GAMMA*probs), state_values)
    i += 1
    if np.abs(state_values-new_state_values).sum() < 1e-3:
        state_values = new_state_values
        break
    state_values = new_state_values


print(f'Iteration {i:>02}: ', end='')
print_val(state_values)
print()

print(state_values.reshape(4, 4))
```

→ 이동 -0.1+1 (award)
→ 이동 -0.1-1 (penalty)

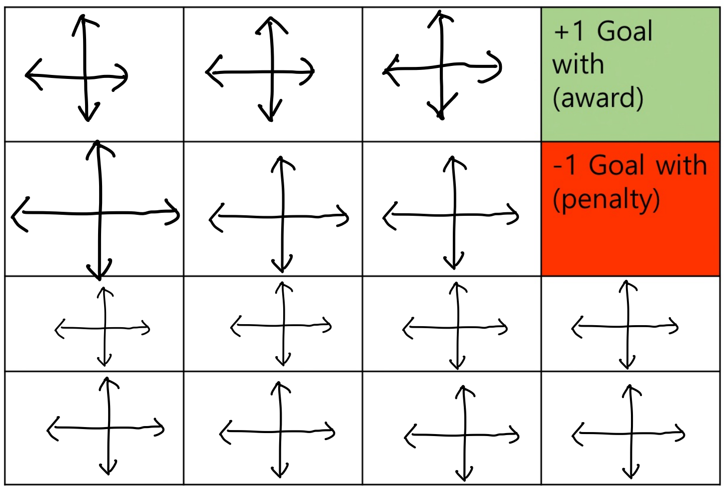
direct solution

iterative solution

Reward 값이 모두 0.1로 동일하므로 action에 대한 value의 max 값이 차이 나지 않음. (Terminal state에서는 다르게 설정했으나 전반적인 수렴의 경향에 큰 영향을 주지 않을 것으로 보여 생략함.)

V k=0

0	0	0	+1
0	0	0	-1
0	0	0	0
0	0	0	0



V k=1

−0.1	−0.1	0.15	+1 0.9
−0.1	−0.1	−0.35	-1 −1.1
−0.1	−0.1	−0.1	−0.55
−0.1	−0.1	−0.1	−0.1

↓→	→	→	+1 Goal with (award)
↑↓→	←↑↓	↑	-1 Goal with (penalty)
↑↓→	←→↑↓	←↓	←
↑→	←↑→	←↑→	←

V k=5

−0.47	−0.4	−0.11	+1 0.9
−0.51	−0.53	−0.64	-1 −1.1
−0.52	−0.56	−0.65	−0.84
−0.51	−0.53	−0.6	−0.68

→	→	→	+1 Goal with (award)
↑	↑	↑	-1 Goal with (penalty)
↑↓	←	←	←
↑	←	←	←

V k=10

−0.92	−0.78	−0.35	+1 Goal with 0.9 (award)
−0.98	−0.96	−0.93	-1 Goal with −1.1 (penalty)
−1.03	−1.05	−1.1	−1.19
−1.05	−1.08	−1.14	−1.20

→	→	→	+1 Goal with (award)
↑	↑	↑	-1 Goal with (penalty)
↑	↑	↑	←
↑	←↑	←	←

V k= ∞

−3.66	−3.05	−1.67	+1 Goal with 0.9 (award)
−3.87	−3.41	−2.47	-1 Goal with −1.1 (penalty)
−4.15	−3.84	−3.31	−2.75
−4.33	−4.11	−3.76	−3.45

→	→	→	+1 Goal with (award)
→	→	↑	-1 Goal with (penalty)
→	→	↑	←
→	→	↑	↑

```
Iteration 00:
[[ 0.  0.  0.  1.]
 [ 0.  0.  0. -1.]
 [ 0.  0.  0.  0.]
 [ 0.  0.  0.  0.]]

Iteration 01:
[[-0.1  -0.1   0.15  0.9 ]
 [-0.1  -0.1  -0.35 -1.1 ]
 [-0.1  -0.1  -0.1  -0.35]
 [-0.1  -0.1  -0.1  -0.1 ]]

Iteration 05:
[[-0.47011719 -0.40117187 -0.10859375  0.9       ]
 [-0.5140625  -0.53027344 -0.64394531 -1.1       ]
 [-0.515625   -0.56367187 -0.65449219 -0.84082031]
 [-0.50976562 -0.53320312 -0.60273438 -0.68105469]]

Iteration 10:
[[-0.9153616  -0.77954102 -0.34628258  0.9       ]
 [-0.98342991 -0.95948486 -0.92506542 -1.1       ]
 [-1.03296738 -1.05198612 -1.1014637  -1.18898697]
 [-1.04966202 -1.08099174 -1.13628254 -1.19582729]]

Iteration 224:
[[-3.65751058 -3.04538685 -1.67207178  0.9       ]
 [-3.86995178 -3.4068396  -2.47098078 -1.1       ]
 [-4.1458376  -3.84131965 -3.30518746 -2.75307969]
 [-4.32659754 -4.1077307  -3.75561688 -3.45422419]]
```

계산결과,,

probs

0.5	0.25	0	0	0.25	0	0	0	0	0	0	0	0	0	0	0
0.25	0.25	0.25	0	0	0.25	0	0	0	0	0	0	0	0	0	0
0	0.25	0.25	0.25	0	0	0.25	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0.25	0	0	0	0.25	0.25	0	0	0.25	0	0	0	0	0	0	0
0	0.25	0	0	0.25	0	0.25	0	0	0.25	0	0	0	0	0	0
0	0	0.25	0	0	0.25	0	0.25	0	0	0.25	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0.25	0	0	0	0.25	0.25	0	0	0.25	0	0	0
0	0	0	0	0	0.25	0	0	0.25	0	0.25	0	0	0.25	0	0
0	0	0	0	0	0	0.25	0	0	0.25	0	0.25	0	0	0.25	0
0	0	0	0	0	0	0	0.25	0	0	0.25	0.25	0	0	0	0.25
0	0	0	0	0	0	0	0	0.25	0	0	0	0.5	0.25	0	0
0	0	0	0	0	0	0	0	0	0.25	0	0	0.25	0.25	0.25	0
0	0	0	0	0	0	0	0	0	0	0.25	0	0	0.25	0.25	0.25
0	0	0	0	0	0	0	0	0	0	0	0.25	0	0	0.25	0.5

transition probability 정의