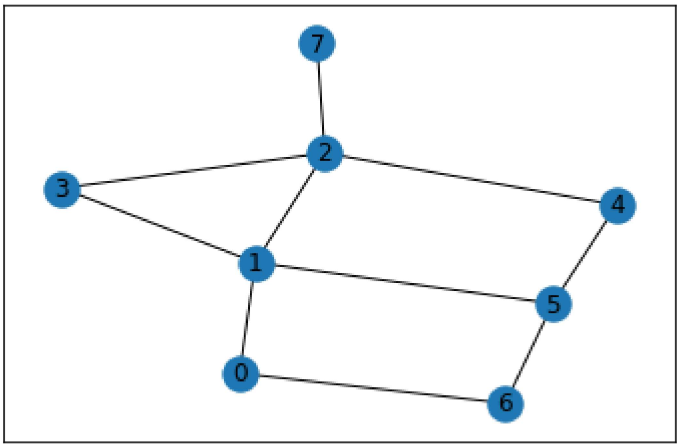
# DL_HW07_곽용하_2014121047

## (a)

In [1]:
```
import numpy as np
import pylab as plt
import networkx as nx
```

In [2]:
```
# map cell to cell, add circular cell to goal point
points_list = [(0,1), (0,6), (1,2), (1,3), (1,5), (2,3), (2,4), (4,5), (5,6), (2,7)]
```

In [3]:
```
goal = 7

G=nx.Graph()
G.add_edges_from(points_list)
pos = nx.spring_layout(G)
nx.draw_networkx_nodes(G,pos)
nx.draw_networkx_edges(G,pos)
nx.draw_networkx_labels(G,pos)
plt.show()
```



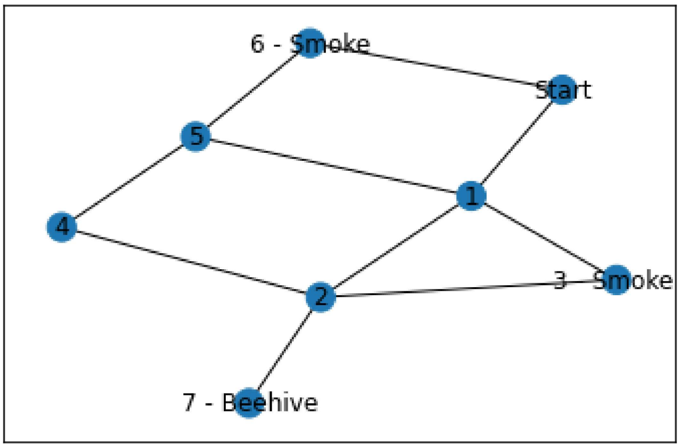

In [ ]:
```
```

## (b)

**(0,1,5,4,2,7) 순으로 로봇이 이동하게 한다는 것은, 로봇이 hive로 가는 비유 상에서, 6과 3에 smoke가 있게 하는 것과 유사합니다. 즉, 이를 그림으로 나타내면 다음과 같습니다.**

In [4]:

```
bees = []
smoke = [3,6]

G=nx.Graph()
G.add_edges_from(points_list)
mapping={0:'Start', 1:'1', 2:'2', 3:'3 - Smoke ', 4:'4', 5:'5', 6:'6 - Smoke', 7:'7 -
H=nx.relabel_nodes(G,mapping)
pos = nx.spring_layout(H)
nx.draw_networkx_nodes(H,pos, node_size=[200,200,200,200,200,200,200,200])
nx.draw_networkx_edges(H,pos)
nx.draw_networkx_labels(H,pos)
plt.show()
```




**Reward Matrix 도출 과정은 아래와 같습니다.**

In [5]:

```
MATRIX_SIZE = 8

# create matrix
# create matrix x*y
R = np.matrix(np.ones(shape=(MATRIX_SIZE, MATRIX_SIZE)))
R *= -1
print(R)
```

```
[[-1. -1. -1. -1. -1. -1. -1. -1.]
 [-1. -1. -1. -1. -1. -1. -1. -1.]
 [-1. -1. -1. -1. -1. -1. -1. -1.]
 [-1. -1. -1. -1. -1. -1. -1. -1.]
 [-1. -1. -1. -1. -1. -1. -1. -1.]
 [-1. -1. -1. -1. -1. -1. -1. -1.]
 [-1. -1. -1. -1. -1. -1. -1. -1.]
 [-1. -1. -1. -1. -1. -1. -1. -1.]]
```

In [6]:
```python
# assign zeros to paths and 1000 to goal-reaching point
for point in points_list:
    print(point)
    if point[1] == goal:
        R[point] = 100
    else:
        R[point] = 0

    if point[0] == goal:
        R[point[::-1]] = 100
    else:
        R[point[::-1]]= 0
```

```
(0, 1)
(0, 6)
(1, 2)
(1, 3)
(1, 5)
(2, 3)
(2, 4)
(4, 5)
(5, 6)
(2, 7)
```

**다만, 아래와 같이 [1 -> 2] 이동에 대하여는 음의 reward를 부여하도록 하겠습니다. 원하는 경로에 대해 직접적으로 높은 reward를 부여하더라도 - 예를 들어 1->5 에 대한 높은 reward를 부여하는 등 - 회전율이 떨어짐에 따라 총보수 측면에서는 오히려 부정적인 영향을 줄 수 있습니다. 즉, 후술할 environment 설정을 통해 로봇이 강화 학습을 하더라도 회전율이 높은 path(여기서는 0,1,2,7)를 선택함으로써 얻는 총보수를 능가하기 어려울 수 있습니다.**

**따라서 저는 어떤 제약도 없을 경우 max sigma_reward이자 최단 경로가 되는데 중요한 역할을 하는 [1->2] 부분에 대하여 음의 reward를 부여하도록 하겠습니다.**

In [7]:
```python
R[1,2]= -1

R[goal,goal]= 100

R
```

Out[7]:
```
matrix([[ -1.,   0.,  -1.,  -1.,  -1.,  -1.,   0.,  -1.],
        [  0.,  -1.,  -1.,   0.,  -1.,   0.,  -1.,  -1.],
        [ -1.,   0.,  -1.,   0.,   0.,  -1.,  -1., 100.],
        [ -1.,   0.,   0.,  -1.,  -1.,  -1.,  -1.,  -1.],
        [ -1.,  -1.,   0.,  -1.,  -1.,   0.,  -1.,  -1.],
        [ -1.,   0.,  -1.,  -1.,   0.,  -1.,   0.,  -1.],
        [  0.,  -1.,  -1.,  -1.,  -1.,   0.,  -1.,  -1.],
        [ -1.,  -1.,   0.,  -1.,  -1.,  -1.,  -1., 100.]])
```

## (c)

**reward matrix 외에도 environment setting을 반영하여 학습시킵니다.**

In [12]:

```python
Q = np.matrix(np.zeros([MATRIX_SIZE,MATRIX_SIZE]))
enviro_bees = np.matrix(np.zeros([MATRIX_SIZE,MATRIX_SIZE]))
enviro_smoke = np.matrix(np.zeros([MATRIX_SIZE,MATRIX_SIZE]))

initial_state = 1
gamma = 0.7

def available_actions(state):
    current_state_row = R[state,]
    av_act = np.where(current_state_row >= 0)[1]
    return av_act

def sample_next_action(available_actions_range):
    next_action = int(np.random.choice(available_act,1))
    return next_action
available_act = available_actions(initial_state)
action = sample_next_action(available_act)

def collect_environmental_data(action):
    found = []
    if action in bees:
        found.append('b')

    if action in smoke:
        found.append('s')
    return (found)



def update(current_state, action, gamma):
    max_index = np.where(Q[action,] == np.max(Q[action,]))[1]

    if max_index.shape[0] > 1:
        max_index = int(np.random.choice(max_index, size = 1))
    else:
        max_index = int(max_index)
    max_value = Q[action, max_index]

    Q[current_state, action] = R[current_state, action] + gamma * max_value
    print('max_value', R[current_state, action] + gamma * max_value)

    environment = collect_environmental_data(action)
    if 'b' in environment:
        enviro_bees[current_state, action] += 1

    if 's' in environment:
        enviro_smoke[current_state, action] += 1

    if (np.max(Q) > 0):
        return(np.sum(Q/np.max(Q)*100))
    else:
        return (0)

update(initial_state,action,gamma)

scores = []
for i in range(700):
    current_state = np.random.randint(0, int(Q.shape[0]))
    available_act = available_actions(current_state)
    action = sample_next_action(available_act)
    score = update(current_state,action,gamma)

# print environmental matrices
print('Smoke Found')
```

```python
print(enviro_smoke)
```

```
max_value 233.32807415393208
max_value 114.3307563354267
max_value 80.02965190775247
max_value 163.32965190775244
max_value 114.3307563354267
max_value 80.02807415393211
max_value 163.32965190775244

max_value 114.3307563354267
max_value 80.03152943479868
max_value 233.32807415393208
max_value 163.32965190775244
max_value 233.32807415393208
max_value 233.32807415393208
max_value 80.03152943479868
max_value 333.325820219903
max_value 114.3307563354267
max_value 333.3280741539321
max_value 233.32965190775246
max_value 233.32965190775246
max_value 163.3307563354267
```

**enviroment 설정을 matrix에 반영할 때, bees에 가면 양의 값을, smoke에 가면 음의 값이 부여될 수 있도록 해야 합니다.**

In [13]:

```python
Q = np.matrix(np.zeros([MATRIX_SIZE,MATRIX_SIZE]))

# subtract bees with smoke, this gives smoke a negative effect
enviro_matrix = - enviro_smoke
```

In [14]:

```python
enviro_matrix
```

Out[14]:

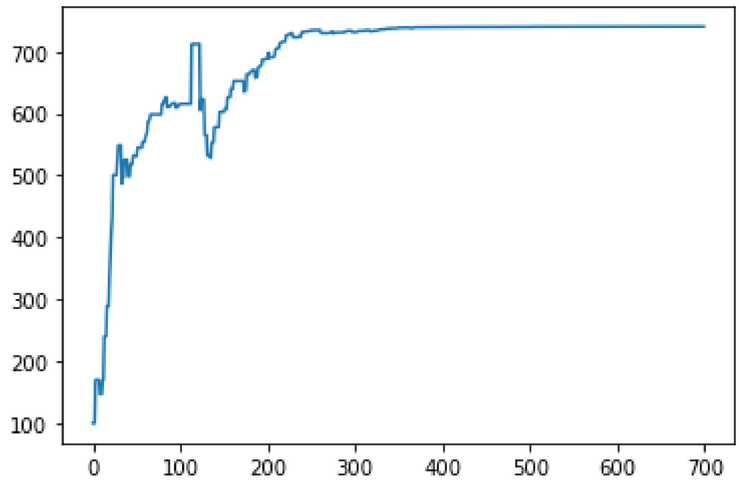
```
matrix([[ -0.,  -0.,  -0.,  -0.,  -0.,  -0., -36.,  -0.],
        [ -0.,  -0.,  -0., -33.,  -0.,  -0.,  -0.,  -0.],
        [ -0.,  -0.,  -0., -23.,  -0.,  -0.,  -0.,  -0.],
        [ -0.,  -0.,  -0.,  -0.,  -0.,  -0.,  -0.,  -0.],
        [ -0.,  -0.,  -0.,  -0.,  -0.,  -0.,  -0.,  -0.],
        [ -0.,  -0.,  -0.,  -0.,  -0.,  -0., -29.,  -0.],
        [ -0.,  -0.,  -0.,  -0.,  -0.,  -0.,  -0.,  -0.],
        [ -0.,  -0.,  -0.,  -0.,  -0.,  -0.,  -0.,  -0.]])
```

In [15]:

```python
# Get available actions in the current state
available_act = available_actions(initial_state)

# Sample next action to be performed
action = sample_next_action(available_act)

# This function updates the Q matrix according to the path selected and the Q
# learning algorithm
def update(current_state, action, gamma):

    max_index = np.where(Q[action,] == np.max(Q[action,]))[1]

    if max_index.shape[0] > 1:
        max_index = int(np.random.choice(max_index, size = 1))
    else:
        max_index = int(max_index)
    max_value = Q[action, max_index]

    Q[current_state, action] = R[current_state, action] + gamma * max_value
    print('max_value', R[current_state, action] + gamma * max_value)

    environment = collect_environmental_data(action)
    if 'b' in environment:
        enviro_matrix[current_state, action] += 1
    if 's' in environment:
        enviro_matrix[current_state, action] -= 1

    return(np.sum(Q/np.max(Q)*100))

update(initial_state,action,gamma)

enviro_matrix_snap = enviro_matrix.copy()

def available_actions_with_enviro_help(state):
    current_state_row = R[state,]
    av_act = np.where(current_state_row >= 0)[1]
    # if there are multiple routes, dis-favor anything negative
    env_pos_row = enviro_matrix_snap[state,av_act]
    if (np.sum(env_pos_row < 0)):
        # can we remove the negative directions from av_act?
        temp_av_act = av_act[np.array(env_pos_row)[0]>=0]
        if len(temp_av_act) > 0:
            print('going from:',av_act)
            print('to:',temp_av_act)
            av_act = temp_av_act
    return av_act

# Training
scores = []
for i in range(700):
    current_state = np.random.randint(0, int(Q.shape[0]))
    available_act = available_actions_with_enviro_help(current_state)
    action = sample_next_action(available_act)
    score = update(current_state,action,gamma)
    scores.append(score)
    print ('Score:', str(score))


plt.plot(scores)
plt.show()
```

```
max_value 56.02114680712804
Score: 740.5424128512761
```

```
max_value 80.03020972446863
Score: 740.5424128512761
```




**Q matrix**

In [16]:

```
Q
```

Out[16]:

```
matrix([[  0.        ,  80.03020972,   0.        ,   0.        ,
           0.        ,   0.        ,   0.        ,   0.        ],
        [ 56.02114681,   0.        ,   0.        ,   0.        ,
           0.        , 114.32887103,   0.        ,   0.        ],
        [  0.        ,  80.0242266 ,   0.        ,   0.        ,
         163.33114681,   0.        ,   0.        , 333.32887103],
        [  0.        ,  80.03020972, 233.33020972,   0.        ,
           0.        ,   0.        ,   0.        ,   0.        ],
        [  0.        ,   0.        , 233.33020972,   0.        ,
           0.        , 114.32887103,   0.        ,   0.        ],
        [  0.        ,  80.03020972,   0.        ,   0.        ,
         163.32695862,   0.        ,   0.        ,   0.        ],
        [ 56.02114681,   0.        ,   0.        ,   0.        ,
           0.        , 114.32887103,   0.        ,   0.        ],
        [  0.        ,   0.        , 233.33020972,   0.        ,
           0.        ,   0.        ,   0.        , 333.33020972]])
```

**The optimal path from the trained Q matrix**

In [17]:

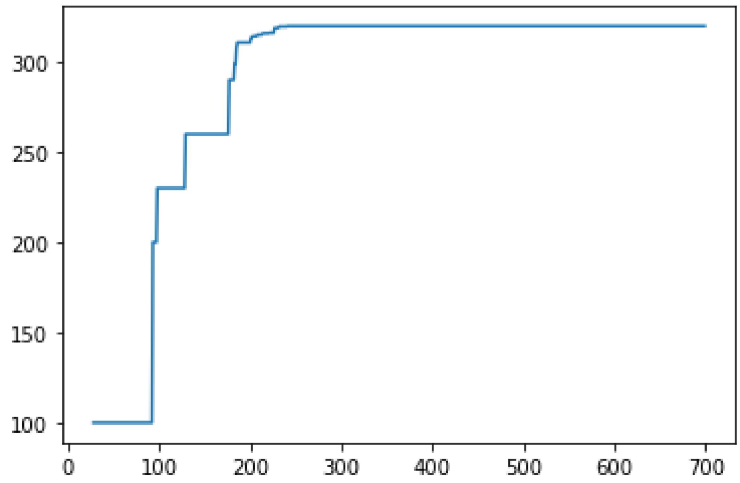
```
current_state = 0
steps = [current_state]
while current_state != 7:
    next_step_index = np.where(Q[current_state,] == np.max(Q[current_state,]))[1]

    if next_step_index.shape[0] > 1:
        next_step_index = int(np.random.choice(next_step_index, size = 1))
    else:
        next_step_index = int(next_step_index)

    steps.append(next_step_index)
    current_state = next_step_index

print("최적 경로는?")
print(steps)
```

```
최적 경로는?
[0, 1, 5, 4, 2, 7]
```

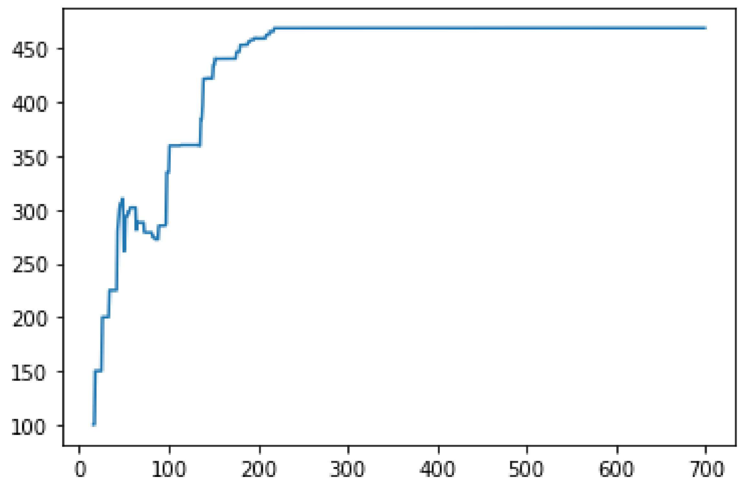
예상한 바와 같이 움직입니다.

## (d) ; gamma 값만 변경하고 나머지는 모두 동일하여, 그래프만 첨부하였습니다.

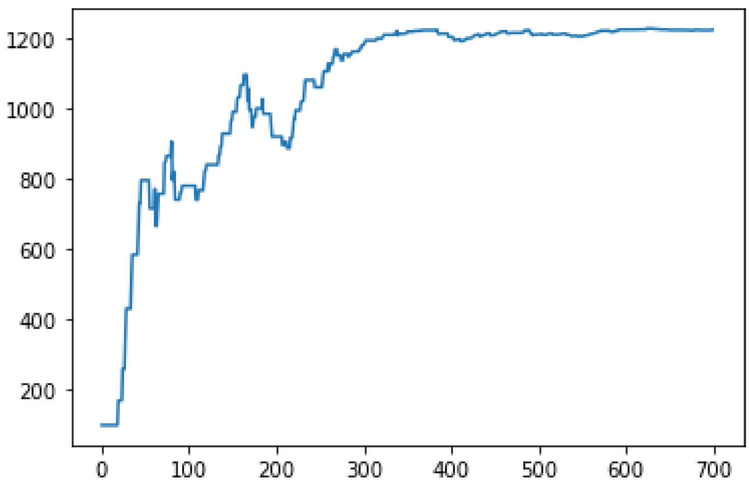
## case1) gamma = 0.3




**case2) gamma = 0.5**




**case3) gamma = 0.9**




**How does the result change?**

최적 경로는 [0,1,5,4,2,7]로 모두 동일한 결과에 도달합니다. 그러나 감마(학습률)가 작을수록 수렴 속도가 빨랐습니다. 특히 감마 값이 클 경우 수렴 과정에 있어 변동성(등락)이 커졌습니다. 다만 절대적인 스코어 값은 감마가 커질수록 그 값도 커졌습니다.