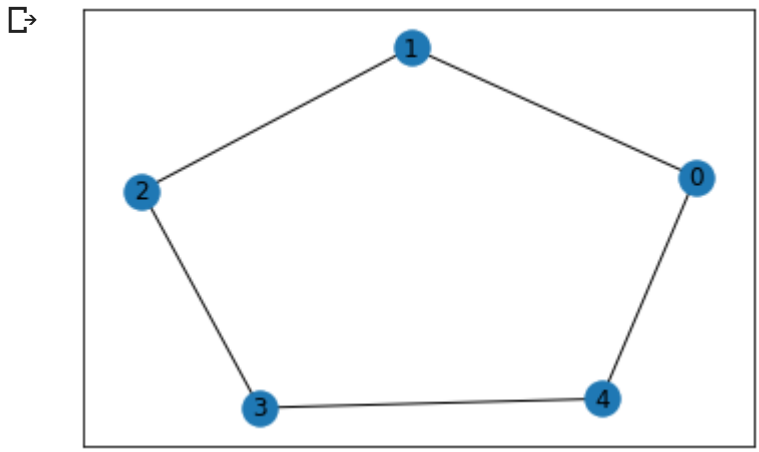
```
import matplotlib.pyplot as plt
import networkx as nx
import numpy as np
import random
%matplotlib inline


# edges = [(0,1), (1,0), (0,2), (2,0), (1,3), (3,1), (2,3), (3,1)] # *DYNAMIC
edge = [(2,3), (3,2), (2,1), (1,2), (1,5), (5,1), (5,4), (4,5), (4,3), (3,4)] # *DY
edges = []
for e in edge:
  edges.append((e[0]-1, e[1]-1))

G = nx.Graph()
G.add_edges_from(edges)
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()
```




```
R = np.matrix(np.zeros(shape=(5,5))) # *DYNAMIC
# print(G[4])
for key in G[4]: # *DYNAMIC
  # print(key)
  R[key,4] = 100 # NEED TO CALCULATE REWARD FOR EVERY LINK DYNAMICALLY, VIA INPUTS

# R[1,2] = 90
# R[2,1] = 81
# R[2,3] = 85
# R[3,2] = 65
# R[3,4] = 100
# R[4,3] = 78

# R[1,0] = 16
# R[0,1] = 29
# R[0,4] = 100
# R[4,0] = 28.8
```

```
print(R)
```

```
[[  0.    29.     0.     0.   100.  ]
 [ 16.     0.    90.     0.     0.  ]
 [  0.     0.81   0.    85.     0.  ]
 [  0.     0.    65.     0.   100.  ]
 [ 28.8    0.     0.    78.     0.  ]]
```

```
Q = np.matrix(np.zeros(shape=(5,5))) # NEED TO MAKE DYNAMIC
Q-=100
for node in G.nodes:
  for x in G[node]:
    Q[node-1, x-1] = 0
    Q[x-1, node-1] = 0
```

```
import pandas as pd
pd.DataFrame(R)
# pd.DataFrame(Q)
```

	0	1	2	3	4
0	0.0	29.00	0.0	0.0	100.0
1	16.0	0.00	90.0	0.0	0.0
2	0.0	0.81	0.0	85.0	0.0
3	0.0	0.00	65.0	0.0	100.0
4	28.8	0.00	0.0	78.0	0.0

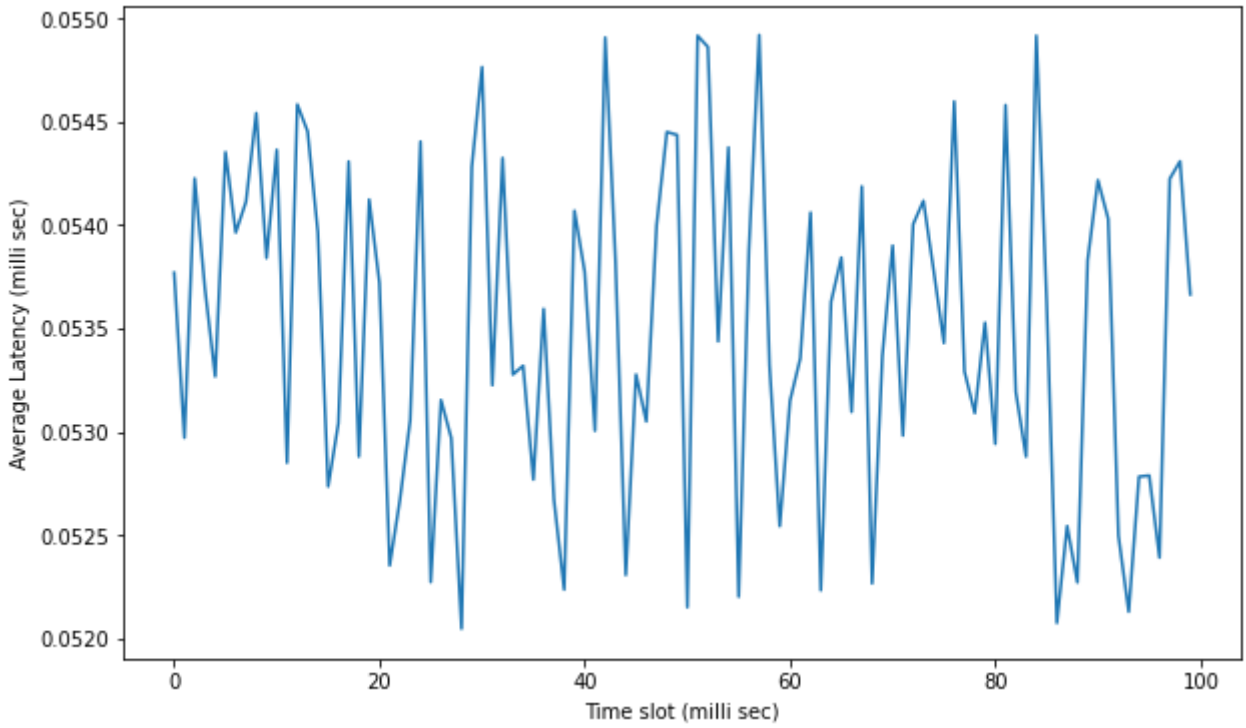
```
xpoints = np.arange(start=1, stop=101)
# print(xpoints)
# ypoints = np.array([])
# ypoints = np.array([90,81,85,65,100,78,29,100,28])
fig = plt.figure(figsize = (10,6))
testList = [90,81,85,65,100,78,29,100,28]
sum = 0
for i in testList:
  sum+=i
print(sum/len(testList))
ypoints = np.random.uniform(0.055,0.052,100)
```

```
plt.plot(ypoints)
plt.xlabel("Time slot (milli sec)")
plt.ylabel("Average Latency (milli sec)")

print(ypoints.mean())
```

```
plt.show()
# print(R.flatten())
```

```
72.88888888888889
0.053506297371126346
```




```
'''
Takes a node and returns the next node
a) On the basis of the highest Q value for moving from s to s'
b) Randomly choosing the next node from the available connected nodes
'''
def nextNode(start, exp_rate): # exp_rate -> exploration rate (higher exp_rate = mo
  random_value = random.uniform(0,1)

  if random_value < exp_rate: # if random_value is lower than exp_rate, choose the
    sample = G[start]
  else:
    sample = np.where(Q[start,] == np.max(Q[start,]))[1] # finds index of the highe
  next_node = int(np.random.choice(sample,1))
  return next_node


'''
Updating the Q values for the action taken
'''
def updateQ(node1, node2, alpha, gamma): # alpha: learning rate, gamma: discount fa
  max_index = np.where(Q[node2,]== np.max(Q[node2,]))[1] # finds index of the highe
  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[node2, max_index]
  Q[node1, node2] = int(Q[node1, node2] + alpha*(R[node1,node2] + gamma * max_value
```

```
def learn(exp_rate, alpha, gamma):
  for i in range(50000): # UPDATE THE SIZE OF THE WALK HERE
    start = np.random.randint(0,5) # UPPER LIMIT -> DYNAMIC
    next_node = nextNode(start, exp_rate)
    updateQ(start, next_node, alpha, gamma)

learn(0.9, 0.81, 0.96) # INCLUDE IT IN A FUNCTION


def shortest_path(begin, end):
  path = [begin+1]
  next_node = np.argmax(Q[begin,])
  path.append(next_node+1)
  while next_node != end:
    next_node = np.argmax(Q[next_node,])
    path.append(next_node+1)

  # return [1,2,3,5]
  return path


shortest_path(1,4)
```

```
[2, 3, 4, 5]
```