```
In [2]: import numpy as np
        import matplotlib.pyplot as plt
```

1a)

```
In [3]: # Circle topology
        # Unweighted adjacency matrix

        # Option 1: Manually enter the entries
        Atilde = np.array(
                  [[0,1,0,0,0,0,0,1],
                   [1,0,1,0,0,0,0,0],
                   [1,1,0,1,1,0,0,0],
                   [0,0,1,0,1,0,0,0],
                   [0,0,0,1,0,1,0,0],
                   [0,0,0,0,1,0,1,0],
                   [0,0,0,0,0,1,0,1],
                   [1,0,0,0,0,0,1,0]])

        # Option 2: or you can exploit the patterns
        # Atilde = np.zeros((8,8))
        # for i in range(8): #
        #     Atilde[i,(i+1)%8] = 1
        #     Atilde[i,(i-1)%8] = 1
        # Atilde[2,0] = 1
        # Atilde[2,4] = 1

        print('Unweighted adjacency matrix')
        print(Atilde)
        print(' ')
```

```
Unweighted adjacency matrix
[[0 1 0 0 0 0 0 1]
 [1 0 1 0 0 0 0 0]
 [1 1 0 1 1 0 0 0]
 [0 0 1 0 1 0 0 0]
 [0 0 0 1 0 1 0 0]
 [0 0 0 0 1 0 1 0]
 [0 0 0 0 0 1 0 1]
 [1 0 0 0 0 0 1 0]]
```

1b)

```
In [18]: # Find weighted adjacency matrix
         # option 1: normalize columns with a for loop
         A = np.zeros((8,8), dtype=float)
         for k in range(8):
             norm = np.sum(Atilde[:,k])
             A[:,k] = np.round(Atilde[:,k]/norm, 2)

         # option 2: normalize using numpy.sum() and broadcasting, in a single line
         # A = ???
```

```
print('Weighted adjacency matrix')
print(A)
```

```
Weighted adjacency matrix
[[0.   0.5  0.   0.   0.   0.   0.   0.5 ]
 [0.33 0.   0.5  0.   0.   0.   0.   0.  ]
 [0.33 0.5  0.   0.5  0.33 0.   0.   0.  ]
 [0.   0.   0.5  0.   0.33 0.   0.   0.  ]
 [0.   0.   0.   0.5  0.   0.5  0.   0.  ]
 [0.   0.   0.   0.   0.33 0.   0.5  0.  ]
 [0.   0.   0.   0.   0.   0.5  0.   0.5 ]
 [0.33 0.   0.   0.   0.   0.   0.5  0.  ]]
```

1c) and 1d)

In [20]:
```
# Power method

b0 = 0.125*np.ones((8,1))
print('b0 = ', b0)
print(' ')

b1 = A@b0
print('b1 = ', b1)
print(' ')

b = b0.copy()
for k in range(1000):
    b = A@b

print('1000 iterations')
print('b = ',b)
```

```
b0 =  [[0.125]
 [0.125]
 [0.125]
 [0.125]
 [0.125]
 [0.125]
 [0.125]
 [0.125]]

b1 =  [[0.125  ]
 [0.10375]
 [0.2075 ]
 [0.10375]
 [0.125  ]
 [0.10375]
 [0.125  ]
 [0.10375]]

1000 iterations
b =  [[0.01142111]
 [0.01519881]
 [0.02278941]
 [0.01519881]
 [0.01142111]
 [0.00759061]
 [0.0076082 ]
 [0.00759061]]
```

1e) Nodes 2, 3 and 4 seem to be more important because they have the highest probability of being visited from the current state.

2a)

In [21]:
```python
# Hub topology

Atildehub = np.array(
         [[0,0,0,0,0,0,0,0,1],
          [1,0,0,0,0,0,0,0,1],
          [0,0,0,0,0,0,0,0,1],
          [0,0,0,0,0,0,0,0,1],
          [0,0,0,0,0,0,0,0,1],
          [0,0,0,0,0,0,0,0,1],
          [0,0,0,0,0,0,0,0,1],
          [0,0,0,0,0,0,0,0,1],
          [1,1,1,1,1,1,1,1,1]])

print('Unweighted adjacency matrix')
print(Atildehub)
print(' ')
```

```
Unweighted adjacency matrix
[[0 0 0 0 0 0 0 0 1]
 [1 0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 0 1]
 [1 1 1 1 1 1 1 1 1]]
```

2b)

In [22]:
```
# find weighted adjacency matrix

Ahub = np.zeros((9,9), dtype=float)
for k in range(9):
    norm = np.sum(Atildehub[:,k])
    Ahub[:,k] = np.round(Atildehub[:,k]/norm, 2)

print('Weighted adjacency matrix')
print(Ahub)
```

```
Weighted adjacency matrix
[[0.   0.   0.   0.   0.   0.   0.   0.   0.11]
 [0.5  0.   0.   0.   0.   0.   0.   0.   0.11]
 [0.   0.   0.   0.   0.   0.   0.   0.   0.11]
 [0.   0.   0.   0.   0.   0.   0.   0.   0.11]
 [0.   0.   0.   0.   0.   0.   0.   0.   0.11]
 [0.   0.   0.   0.   0.   0.   0.   0.   0.11]
 [0.   0.   0.   0.   0.   0.   0.   0.   0.11]
 [0.   0.   0.   0.   0.   0.   0.   0.   0.11]
 [0.5  1.   1.   1.   1.   1.   1.   1.   0.11]]
```

2c) and 2d)

In [23]:
```
b0 = (1/9)*np.ones((9,1))
print('b0 = ', b0)
print(' ')

bhub1 = Ahub@b0
print('bhub1 = ', bhub1)
print(' ')

bhub = b0.copy()
for k in range(1000):
    bhub = Ahub@bhub

print('1000 iterations')
print('bhub = ', bhub)
print(' ')

bhubr = b0.copy()
for k in range(100):
    bhubr = Ahub@bhubr

print('100 iterations')
print('bhubr = ',bhubr)
```

```
b0 =  [[0.11111111]
 [0.11111111]
 [0.11111111]
 [0.11111111]
 [0.11111111]
 [0.11111111]
 [0.11111111]
 [0.11111111]
 [0.11111111]]

bhub1 =  [[0.01222222]
 [0.06777778]
 [0.01222222]
 [0.01222222]
 [0.01222222]
 [0.01222222]
 [0.01222222]
 [0.01222222]
 [0.84555556]]

1000 iterations
bhub =  [[0.00032546]
 [0.00048904]
 [0.00032546]
 [0.00032546]
 [0.00032546]
 [0.00032546]
 [0.00032546]
 [0.00032546]
 [0.00294352]]

100 iterations
bhubr =  [[0.03406798]
 [0.05119022]
 [0.03406798]
 [0.03406798]
 [0.03406798]
 [0.03406798]
 [0.03406798]
 [0.03406798]
 [0.30811254]]
```

2e) and 2f)

2e) Nodes 9 is most important because it consistently has the highest probability of being visited from the current state compared to all other nodes which are all roughly the same probably (which is lower).

2f) below

```
In [32]: bhubr = b0.copy()
         for k in range(15):
             bhubr = Ahub@bhubr

         print('bhubr = ',bhubr)
```

```
bhubr =  [[0.05021348]
 [0.07839725]
 [0.05021348]
 [0.05021348]
 [0.05021348]
 [0.05021348]
 [0.05021348]
 [0.05021348]
 [0.49769375]]
```

```
In [ ]:
```