In [1]:
```python
import numpy as np
import matplotlib.pyplot as plt
```

1a)

In [2]:
```python
# 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(' ')
print(round(1/3,2))
```

```
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]]
 
0.33
```

1b)

In [3]:
```python
# 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] = (Atilde[:,k])/norm

# 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       ]
 [0.33333333 0.         0.5        0.         0.         0.
  0.         0.        ]
 [0.33333333 0.5        0.         0.5        0.33333333 0.
  0.         0.        ]
 [0.         0.         0.5        0.         0.33333333 0.
  0.         0.        ]
 [0.         0.         0.         0.5        0.         0.5
  0.         0.        ]
 [0.         0.         0.         0.         0.33333333 0.
  0.5        0.        ]
 [0.         0.         0.         0.         0.         0.5
  0.         0.5       ]
 [0.33333333 0.         0.         0.         0.         0.
  0.5        0.        ]]
```

1c) and 1d)

In [4]:
```python
# 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     ]
 [0.10416667]
 [0.20833333]
 [0.10416667]
 [0.125     ]
 [0.10416667]
 [0.125     ]
 [0.10416667]]

1000 iterations
b =  [[0.11538462]
 [0.15384615]
 [0.23076923]
 [0.15384615]
 [0.11538462]
 [0.07692308]
 [0.07692308]
 [0.07692308]]
```

1e) Explination goes here.

The third node has more incoming nodes than any other, so itseems to be the most important. Note that the PageRank vector shows the thirdnode to be the most regularly visited node.

2a)

In [5]:
```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,

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 0]]
```

2b)

In [6]:

```
# find weighted adjacency matrix
Ahub = np.zeros((9,9), dtype=float)
for k in range(9):
    norm = np.sum(Atildehub[:,k])
    Ahub[:,k] = (Atildehub[:,k])/norm


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

```
Weighted adjacency matrix
[[0.    0.    0.    0.    0.    0.    0.    0.    0.125]
 [0.5   0.    0.    0.    0.    0.    0.    0.    0.125]
 [0.    0.    0.    0.    0.    0.    0.    0.    0.125]
 [0.    0.    0.    0.    0.    0.    0.    0.    0.125]
 [0.    0.    0.    0.    0.    0.    0.    0.    0.125]
 [0.    0.    0.    0.    0.    0.    0.    0.    0.125]
 [0.    0.    0.    0.    0.    0.    0.    0.    0.125]
 [0.    0.    0.    0.    0.    0.    0.    0.    0.125]
 [0.5   1.    1.    1.    1.    1.    1.    1.    0.   ]]
```

2c) and 2d)

In [7]:

```
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 =  [[0.01388889]
 [0.06944444]
 [0.01388889]
 [0.01388889]
 [0.01388889]
 [0.01388889]
 [0.01388889]
 [0.01388889]
 [0.83333333]]

1000 iterations
bhub =  [[0.06060606]
 [0.09090909]
 [0.06060606]
 [0.06060606]
 [0.06060606]
 [0.06060606]
 [0.06060606]
 [0.06060606]
 [0.48484848]]

100 iterations
bhubr =  [[0.06065482]
 [0.09093172]
 [0.06065482]
 [0.06065482]
 [0.06065482]
 [0.06065482]
 [0.06065482]
 [0.06065482]
 [0.48448454]]
```

Complete 2e and 2f below.

2e) Yes, some nodes are more important than other nodes. Node 9 is most important in 100 and 1000 iterations.

In [8]:
```python
# 2f

bhube = b0.copy()

for k in range(85):
    bhube = Ahub@bhube

print('2f:')
print('85 iterations')
print('bhube = ', bhube)
```

```
2f:
85 iterations
bhube =  [[0.0604681 ]
 [0.09084507]
 [0.0604681 ]
 [0.0604681 ]
 [0.0604681 ]
 [0.0604681 ]
 [0.0604681 ]
 [0.0604681 ]
 [0.4858782 ]]
```

3

The signs of the singular vectors are not unique. If you change the sign of u1 to -u1, you can also change the sign of v1 and the product u1@v1.T (mailto:u1@v1.T) is unchanged. Thus you get the same results.

In [ ]:
```python

```