In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
```

3a)

In [2]:

```
# 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]]
```

3b)

In [9]:

```
# 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])/8
    for i in range(8):
        if Atilde[i,k] == 1:
            A[i,k] = norm

# option 2: normalize using numpy.sum() and broadcasting, in a single line
# np.sum(Atilde, axis = 0)
A = Atilde/np.sum(Atilde, axis = 0)

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.        ]]
```

3c) and 3d)

In [10]:

```
# 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]]
```

3e) Do any nodes seem to be more important than other nodes? Explain.

Yes, the 3rd node is more important than other nodes, because it gets higher probability.

4a)

In [17]:

```
# 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],
          [1,1,1,1,1,1,1,1,0]])
# print(Atildehub.shape)

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]]
```

4b)

In [18]:

```
# find weighted adjacency matrix

Ahub = Atildehub/np.sum(Atildehub, axis = 0)

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.   ]]
```

4c) and 4d)

In [25]:

```
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)
print(' ')

bhubr = b0.copy()
for k in range(90):
    bhubr = Ahub @ bhubr

print('90 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]]
90 iterations
bhubr =  [[0.0607036 ]
 [0.09095436]
 [0.0607036 ]
 [0.0607036 ]
 [0.0607036 ]
 [0.0607036 ]
 [0.0607036 ]
 [0.0607036 ]
 [0.48412044]]
```

4e) Are any nodes more important than other nodes? Explain.

Yes, the 9th node is more important than other nodes, because it gets higher probability.

f) Experiment with the number of iterations of the power method that are needed to find an answer that is correct to three decimal places.

After 100 iterations, we can find the answer, because we can compared it with 1000 iteration, and then the vector does not change much (similarly).

Δ Activity 13

1, a)

$$E_r = \sum_{i=r+1}^{n} \sigma_i u_i v_i^T$$

b) $\operatorname{rank}(E_r) = n - r$

c) $$\|E_r\|_{op} = \max_{x \neq 0} \frac{\|E_r x\|_2}{\|x\|_2} = \sigma_{r+1}$$

d) X_r will be a "good" approximation to X

when $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_r > \sigma_{r+1} \dots \approx 0$

X_r takes the most important ~~data with~~ features of data.

Δ Activity 14

item 1) $Be_i = \lambda_i e_i$

$$B = \begin{bmatrix} 3 & -1 \\ -1 & 3 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 3-2 \\ -1+6 \end{bmatrix} = \begin{bmatrix} 1 \\ 5 \end{bmatrix} \times$$

$$\begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \end{bmatrix} = 2\begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 4 \\ -4 \end{bmatrix} = 4\begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

$$\begin{bmatrix} -1 \\ 1 \end{bmatrix} = \begin{bmatrix} -4 \\ 4 \end{bmatrix} = 4\begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 6-1 \\ -1+3 \end{bmatrix} = \begin{bmatrix} 5 \\ 2 \end{bmatrix} \times$$

2) if $e_i = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ $\lambda = 2$

$e_i = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$ $\lambda = 4$

item 2) $\sum_{i=1}^{r} = \sigma_i u_i v_i^T$

↗ change u_i, v_i to get original X

$\rightarrow \sum_{i=1}^{r} = \sigma_i(-u_i)(-v_i^T)$ the same

not unique

both are valid singular vector