In [3]:
```
import numpy as np
import matplotlib.pyplot as plt
```

1a)

In [8]:
```
# 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],
         [0,1,0,1,0,0,0,0],
         [0,0,1,0,1,0,0,0],
         [0,0,0,1,0,1,0,0],
         [0,0,0,0,1,0,1,0],
         [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]
 [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 0]
 [0 0 0 0 0 1 0 1]
 [1 0 0 0 0 0 1 0]]
```

1b)

In [12]:
```
# 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 [14]:
```
# Power method

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

b1 = 0.125*np.ones((8,1))
print('b1 = ', b1)
print(' ')

b = b0.copy()
for k in range(1000):
    b = 0.125*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]
 [0.125]
 [0.125]
 [0.125]
 [0.125]
 [0.125]
 [0.125]
 [0.125]]

1000 iterations
b =  [[0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]]
```

1e) Explination goes here.

In [ ]:
```
# yes, since the eigenvector corresponds to each node reflects the importance
```

2a)

In [18]:
```
# Hub topology

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

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

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

2b)

In [19]:
```
# find weighted adjacency matrix
A = np.zeros((9,9), dtype=float)
for k in range(9):
    norm = np.sum(Atildehub[:,k])
    A[:,k] = Atildehub[:,k]/norm
Ahub = A

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

```
Weighted adjacency matrix
[[0.         0.33333333 0.         0.         0.
  0.         0.33333333 0.125     ]
 [0.25       0.         0.33333333 0.         0.
  0.         0.         0.125     ]
 [0.25       0.33333333 0.          0.33333333 0.25
  0.         0.         0.125     ]
 [0.         0.         0.33333333 0.         0.25
  0.         0.         0.125     ]
 [0.         0.         0.         0.33333333 0.
  0.         0.         0.125     ]
 [0.         0.         0.         0.         0.25
  0.         0.33333333 0.125     ]
 [0.         0.         0.         0.         0.
  0.33333333 0.         0.125     ]
 [0.25       0.         0.         0.         0.
  0.33333333 0.         0.125     ]
 [0.25       0.33333333 0.33333333 0.33333333 0.25
  0.33333333 0.33333333 0.          ]]
```

2c) and 2d)

In [20]:
```
b0 = (1/9)*np.ones((9,1))
print('b0 = ', b0)
print(' ')

bhub1 = (1/9)*np.ones((9,1))
print('bhub1 = ', bhub1)
print(' ')

bhub = b0.copy()
for k in range(1000):
    bhub = (1/9)*bhub

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

bhubr = b0.copy()
for k in range(100):
    bhubr = (1/9)*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 =  [[0.11111111]
 [0.11111111]
 [0.11111111]
 [0.11111111]
 [0.11111111]
 [0.11111111]
 [0.11111111]
 [0.11111111]
 [0.11111111]]

1000 iterations
bhub =  [[0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]]

100 iterations
bhubr =  [[4.18317994e-97]
 [4.18317994e-97]
 [4.18317994e-97]
 [4.18317994e-97]
 [4.18317994e-97]
 [4.18317994e-97]
 [4.18317994e-97]
 [4.18317994e-97]
 [4.18317994e-97]]
```

Complete 2e and 2f below.

In [ ]:
```
# e) yes, since the eigenvector corresponds to each node reflects the importance
```