

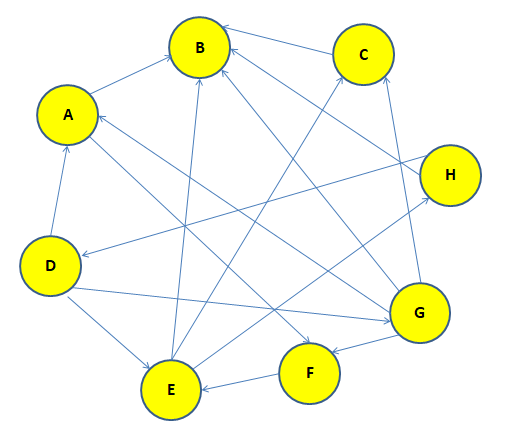
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Programme | : | **B.Tech** | Semester | : | **Win Sem 21-22** |
| Course | : | **Web Mining Lab** | Code | : | **CSE3024** |
| Faculty | : | **Dr.Bhuvaneswari A** | Slot | : | **L7+L8** |
| Date | : | **15-03-2022** | Marks | : | **10 Points** |

**Exercise 9: PAGE RANKING ALGORITHM**

**GOOGLE COLAB LINK:<https://colab.research.google.com/drive/1AVeQFURTkBF-GXp4RrCPVwC7nekFyYys?usp=sharing>**

Implement the basic PageRank algorithm using Python to display the rank of 8 pages.

Verify the consistency of results obtained with Random walk and inbuilt PageRank function.



**CODE:**

import networkx as nx

from matplotlib import pyplot as plt

adjacency\_matrix = [

[0, 1, 0, 0, 0, 1, 0, 0],

[0, 0, 0, 0, 0, 0, 0, 0],

[0, 1, 0, 0, 0, 0, 0, 0],

[1, 0, 0, 0, 1, 0, 1, 0],

[0, 1, 1, 0, 0, 0, 0, 1],

[0, 0, 0, 0, 1, 0, 0, 0],

[1, 1, 1, 0, 0, 1, 0, 0],

[0, 1, 0, 1, 0, 0, 0, 0]

]

num\_vertices = 8

vertices\_list = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H']

graph = nx.DiGraph()

# Load the nodes into the graph

graph.add\_nodes\_from(vertices\_list)

# Add the edges from the adjacency matrix

for i in range(num\_vertices) :

  for j in range(num\_vertices) :

    if adjacency\_matrix[i][j] == 1 :

      graph.add\_edge(vertices\_list[i], vertices\_list[j])

# Draw generated graph

nx.draw\_networkx(graph, pos=nx.circular\_layout(graph), arrows=True, with\_labels= True)

plt.show()

# Compute the page ranks

page\_ranks\_networkx = nx.pagerank(graph, alpha=0.85)

print("The page ranks are :\n")

page\_ranks\_networkx

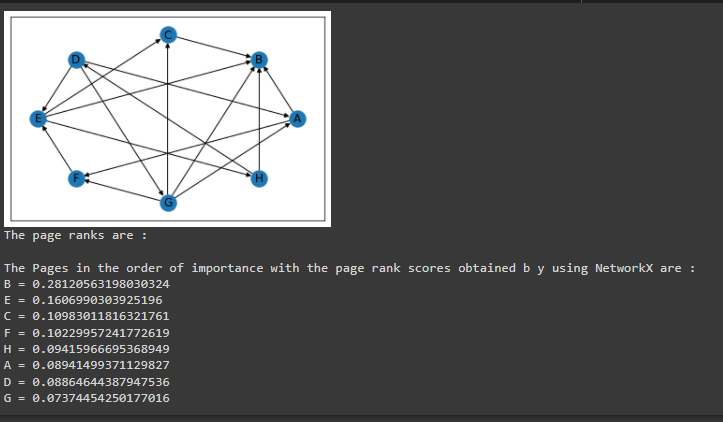
page\_ranks\_networkx = dict(sorted(page\_ranks\_networkx.items(), key=lambda item:item[1], reverse=True))

print("The Pages in the order of importance with the page rank scores obtained b y using NetworkX are : ")

for k, v in page\_ranks\_networkx.items() :

  print(k, "=", v)

**OUTPUT:**

****

**CODE:**

import networkx as nx

import random

import matplotlib.pyplot as plt

import operator

Graph = graph

nx.draw(Graph, with\_labels=True, node\_color='green')

plt.figure(figsize=(15,10))

plt.show()

random\_node = random.choice([i for i in range(Graph.number\_of\_nodes())])

dict\_counter = {}

for i in range(Graph.number\_of\_nodes()):

    dict\_counter[i] = 0

dict\_counter[random\_node] = dict\_counter[random\_node]+1

try:

  for i in range(100):

      list\_for\_nodes = list(Graph.neighbors(random\_node))

      if len(list\_for\_nodes)==0:# if random\_node having no outgoing edges

          random\_node = random.choice([i for i in range(Graph.number\_of\_nodes())])

          dict\_counter[random\_node] = dict\_counter[random\_node]+1

      else:

          random\_node = random.choice(list\_for\_nodes) #choose a node randomly from neighbors

          dict\_counter[random\_node] = dict\_counter[random\_node]+1

except nx.exception.NetworkXError:

  print("")

rank\_node = nx.pagerank(Graph)

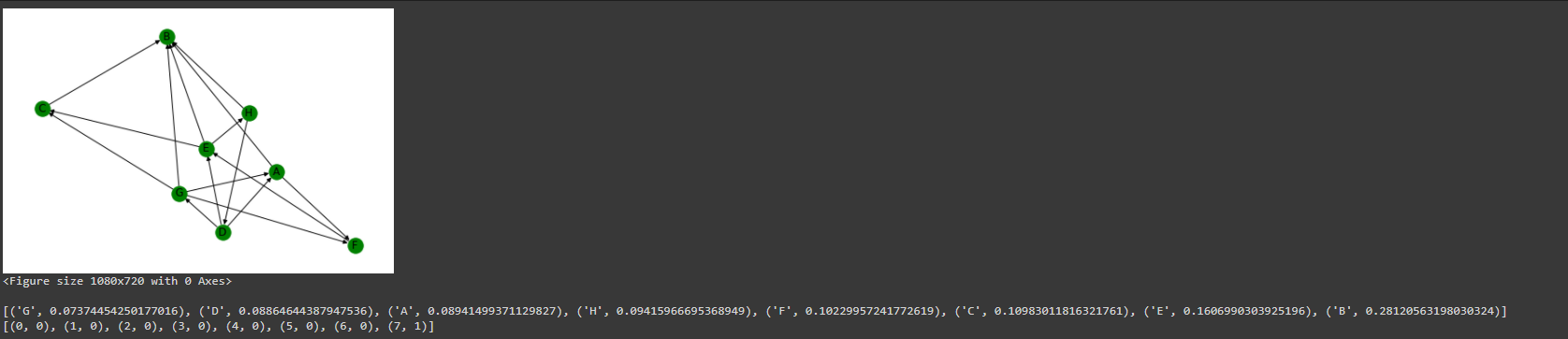
sorted\_rank = sorted(rank\_node.items(), key=operator.itemgetter(1))

sorted\_random\_walk = sorted(dict\_counter.items(), key=operator.itemgetter(1))

print(sorted\_rank)

print(sorted\_random\_walk)

**OUTPUT:**

****

