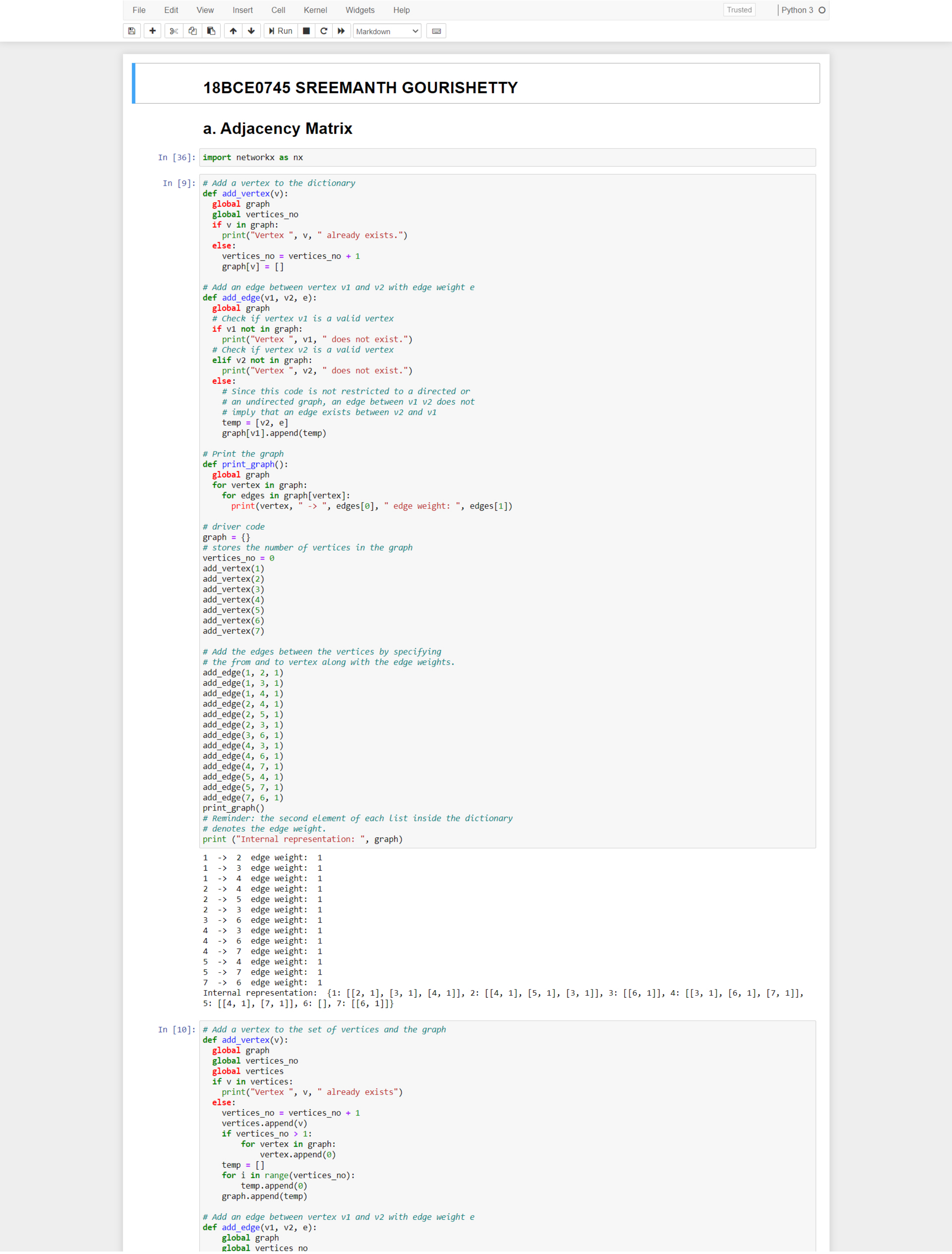


18BCE0745 SREEMANTH GOURISHETTY

a. Adjacency Matrix

In [36]:
```
import networkx as nx
```

In [9]:
```
# Add a vertex to the dictionary
def add_vertex(v):
  global graph
  global vertices_no
  if v in graph:
    print("Vertex ", v, " already exists.")
  else:
    vertices_no = vertices_no + 1
    graph[v] = []

# Add an edge between vertex v1 and v2 with edge weight e
def add_edge(v1, v2, e):
  global graph
  # Check if vertex v1 is a valid vertex
  if v1 not in graph:
    print("Vertex ", v1, " does not exist.")
  # Check if vertex v2 is a valid vertex
  elif v2 not in graph:
    print("Vertex ", v2, " does not exist.")
  else:
    # Since this code is not restricted to a directed or
    # an undirected graph, an edge between v1 v2 does not
    # imply that an edge exists between v2 and v1
    temp = [v2, e]
    graph[v1].append(temp)

# Print the graph
def print_graph():
  global graph
  for vertex in graph:
    for edges in graph[vertex]:
      print(vertex, " -> ", edges[0], " edge weight: ", edges[1])

# driver code
graph = {}
# stores the number of vertices in the graph
vertices_no = 0
add_vertex(1)
add_vertex(2)
add_vertex(3)
add_vertex(4)
add_vertex(5)
add_vertex(6)
add_vertex(7)

# Add the edges between the vertices by specifying
# the from and to vertex along with the edge weights.
add_edge(1, 2, 1)
add_edge(1, 3, 1)
add_edge(1, 4, 1)
add_edge(2, 4, 1)
add_edge(2, 5, 1)
add_edge(2, 3, 1)
add_edge(3, 6, 1)
add_edge(4, 3, 1)
add_edge(4, 6, 1)
add_edge(4, 7, 1)
add_edge(5, 4, 1)
add_edge(5, 7, 1)
add_edge(7, 6, 1)
print_graph()
# Reminder: the second element of each list inside the dictionary
# denotes the edge weight.
print ("Internal representation: ", graph)
```

```
1  ->  2  edge weight:  1
1  ->  3  edge weight:  1
1  ->  4  edge weight:  1
2  ->  4  edge weight:  1
2  ->  5  edge weight:  1
2  ->  3  edge weight:  1
3  ->  6  edge weight:  1
4  ->  3  edge weight:  1
4  ->  6  edge weight:  1
4  ->  7  edge weight:  1
5  ->  4  edge weight:  1
5  ->  7  edge weight:  1
7  ->  6  edge weight:  1
Internal representation:  {1: [[2, 1], [3, 1], [4, 1]], 2: [[4, 1], [5, 1], [3, 1]], 3: [[6, 1]], 4: [[3, 1], [6, 1], [7, 1]],
5: [[4, 1], [7, 1]], 6: [], 7: [[6, 1]]}
```

In [10]:
```
# Add a vertex to the set of vertices and the graph
def add_vertex(v):
  global graph
  global vertices_no
  global vertices
  if v in vertices:
    print("Vertex ", v, " already exists")
  else:
    vertices_no = vertices_no + 1
    vertices.append(v)
    if vertices_no > 1:
        for vertex in graph:
            vertex.append(0)
    temp = []
    for i in range(vertices_no):
        temp.append(0)
    graph.append(temp)

# Add an edge between vertex v1 and v2 with edge weight e
def add_edge(v1, v2, e):
    global graph
    global vertices_no
    global vertices
    # Check if vertex v1 is a valid vertex
    if v1 not in vertices:
        print("Vertex ", v1, " does not exist.")
    # Check if vertex v1 is a valid vertex
    elif v2 not in vertices:
        print("Vertex ", v2, " does not exist.")
    # Since this code is not restricted to a directed or
    # an undirected graph, an edge between v1 v2 does not
    # imply that an edge exists between v2 and v1
    else:
        index1 = vertices.index(v1)
        index2 = vertices.index(v2)
        graph[index1][index2] = e

# Print the graph
def print_graph():
  global graph
  global vertices_no
  for i in range(vertices_no):
    for j in range(vertices_no):
      if graph[i][j] != 0:
        print(vertices[i], " -> ", vertices[j], \
        " edge weight: ", graph[i][j])

# Driver code
# stores the vertices in the graph
vertices = []
# stores the number of vertices in the graph
vertices_no = 0
graph = []
# Add vertices to the graph
add_vertex(1)
add_vertex(2)
add_vertex(3)
add_vertex(4)
add_vertex(5)
add_vertex(6)
add_vertex(7)
# Add the edges between the vertices by specifying
# the from and to vertex along with the edge weights.
add_edge(1, 2, 1)
add_edge(1, 3, 1)
add_edge(1, 4, 1)
add_edge(2, 4, 1)
add_edge(2, 5, 1)
add_edge(2, 3, 1)
add_edge(3, 6, 1)
add_edge(4, 3, 1)
add_edge(4, 6, 1)
add_edge(4, 7, 1)
add_edge(5, 4, 1)
add_edge(5, 7, 1)
add_edge(7, 6, 1)
print_graph()
print("Internal representation: ", graph)
```

```
1  ->  2  edge weight:  1
1  ->  3  edge weight:  1
1  ->  4  edge weight:  1
2  ->  3  edge weight:  1
2  ->  4  edge weight:  1
2  ->  5  edge weight:  1
3  ->  6  edge weight:  1
4  ->  3  edge weight:  1
4  ->  6  edge weight:  1
4  ->  7  edge weight:  1
5  ->  4  edge weight:  1
5  ->  7  edge weight:  1
7  ->  6  edge weight:  1
Internal representation:  [[0, 1, 1, 1, 0, 0, 0], [0, 0, 1, 1, 1, 0, 0], [0, 0, 0, 0, 0, 1, 0], [0, 0, 1, 0, 0, 1, 1], [0, 0,
0, 1, 0, 0, 1], [0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 1, 0]]
```

e. Page rank of all the seven nodes after each iteration

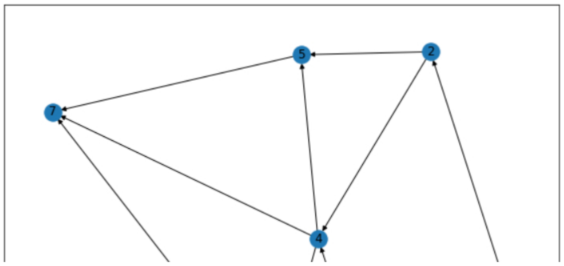
In [11]:

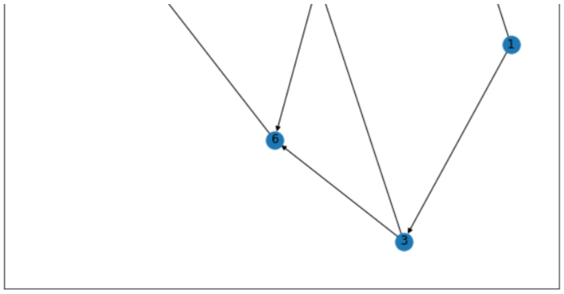
```
import matplotlib.pyplot as plt
import networkx as nx

#G = nx.Graph()
G = nx.DiGraph(Directed=True)

G.add_edge("1", "2")
G.add_edge("1", "3")
G.add_edge("2", "4")
G.add_edge("2", "5")
G.add_edge("3", "4")
G.add_edge("3", "6")
G.add_edge("4", "5")
G.add_edge("4", "6")
G.add_edge("4", "7")
G.add_edge("5", "7")
G.add_edge("6", "7")
```

In [12]:

```
pos=nx.spring_layout(G)
#pos=nx.planar_layout(G)
plt.figure(figsize =(10, 10))
nx.draw_networkx(G,pos)
labels = nx.get_edge_attributes(G,'weight')
nx.draw_networkx_edge_labels(G,pos,edge_labels=labels)
```

Out[12]: {}





```
In [13]: pr = nx.pagerank(G, alpha=0.85)
         print("Node: PageRank")
         pr

         Node: PageRank

Out[13]: {'1': 0.06270921614934422,
          '2': 0.08936045577355974,
          '3': 0.08936045577355974,
          '4': 0.13866468030257129,
          '5': 0.13997501666256587,
          '6': 0.13997501666256587,
          '7': 0.33995515867583315}
```

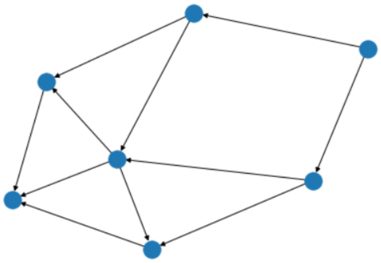
b. Handling the nodes with no outgoing links

```
In [34]: for i in graph:
             s=0;
             for j in i:
                 s+=j
             if(s==0):
                 graph.remove(i)
```

```
In [35]: graph

Out[35]: [[0, 1, 1, 1, 0, 0, 0],
          [0, 0, 1, 1, 1, 0, 0],
          [0, 0, 0, 0, 0, 1, 0],
          [0, 0, 1, 0, 0, 1, 1],
          [0, 0, 0, 1, 0, 0, 1],
          [0, 0, 0, 0, 0, 1, 0]]
```

c. Stochastic matrix formation

```
In [20]: stochasticMatrix = nx.stochastic_graph(G)
         nx.draw(stochasticMatrix)
```



```
In [39]: nx.find_cycle(G, source=None)

         ---------------------------------------------------------------------------
         NetworkXNoCycle                           Traceback (most recent call last)
         <ipython-input-39-3b0108b71d6d> in <module>
         ----> 1 nx.find_cycle(G, source=None)

         C:\Users\Sreemanth\anaconda3\lib\site-packages\networkx\algorithms\cycles.py in find_cycle(G, source, orientation)
             489     else:
             490         assert(len(cycle) == 0)
         --> 491         raise nx.exception.NetworkXNoCycle('No cycle found.')
             492 
             493     # We now have a list of edges which ends on a cycle.

         NetworkXNoCycle: No cycle found.
```

```
In [19]: 

Out[19]: <networkx.classes.digraph.DiGraph at 0x272f5e256c8>
```

```
In [ ]: 
```