```python
'''
Name: Chongyi Xu
Student ID: 1531273
Course: Math 381
Title: Python Scripts and Outputs for HW 2
Instructor: Dr. Matthew Conroy
Due Date: 10/13/2017
'''


'''
Using networkx package credit to
Aric A. Hagberg, Daniel A. Schult and Pieter J. Swart,
"Exploring network structure, dynamics, and function using
NetworkX", in Proceedings of the 7th Python in Science
Conference (SciPy2008), Gäel Varoquaux, Travis Vaught,
and Jarrod Millman (Eds), (Pasadena, CA USA), pp. 11--15,
Aug 2008
'''
import networkx as g

''' Helper Methods'''

def isConnect(graph):
    '''Determine if a graph is connected.

    Parameters
    ----------
    graph: the graph with edges and vertices

    Retures
    ---------
    Boolean: True if connected, otherwise False
    '''
    # For every two vertices
    for v in graph.nodes():
        for u in graph.nodes():
            if v != u: # Two vertices should not be identical
                if not g.has_path(graph, v, u):
                    # There is no path between v and u => the graph is not connected
                    return False

def findEdges(V):
    '''Find all edges in the vertices list using the rule:
    Define a graph H with V as its vertex set and edge set E
    defined by (v1, v2) ⍰ E iff v1 6= v2 and v1 divides v2
    or v2 divides v1. So (2, 6) is an edge in H; (3, 4) is not.

    Parameters
    ---------
    V: array of vertices

    Returns
    ---------
    list: An array list of edges found using the rules
    '''
    Edge = []
    for i in range(2, 23):
        for j in range(2, 23):
            if i != j and i % j == 0 :
                Edge.append([i, j])

    return Edge

from collections import defaultdict
```

```
def findDegree(Edges):
    '''Find degress of each vertices with given
    aray of edges, print out the degress of each
    vertices and the a degree sequence in
    decreasing order.

    Parameters
    ----------
    Edges: array of edges
    '''
    degree = defaultdict(int)
    for i in range(2, 23):
        degree[str(i)] = 0
    for edge in Edges:
        degree[str(edge[0])] += 1
        degree[str(edge[1])] += 1
    print(degree)

    degreeSeq = []
    for vertex in degree:
        degreeSeq.append(degree[vertex])
    degreeSeq.sort(reverse=True)
    print(degreeSeq)

def distanceIfConnect(graph):
    '''Find if the graph is connected and find the furthest
    vertices apart from each other with its path by applying
    dijkstra alogrithm.

    Parameters
    ----------
    graph: the given graph with vertices and edge.

    Returns
    ---------
    list: A list of elements including
            Boolean: True if the graph is connected,
            Array: the path of the vertices that are furthest apart
            Integer: the largest length of the path (furthest distance)
    '''
    furthest = 0
    answer = [True, [], furthest]
    # For every two vertices
    for v in graph.nodes():
        for u in graph.nodes():
            if v != u: # Two vertices should not be identical
                if not g.has_path(graph, v, u):
                    # There is no path between v and u => the graph is not connected
                    answer[0] = False
                else:
                    # Find the shortest path using dijkstra alogrithm
                    path = g.dijkstra_path(graph, v, u)
                    if len(path) > answer[2]: # Update the furthest path
                        answer[2] = len(path)
                        answer[1] = path
    # The number of edges between should be 1 less than the length of array
    answer[2] = answer[2] - 1
    return answer


'''-------------------------------------------
Homework Sections
----------------------------------------------'''
# 1
matrix = [[0, 0, 1, 0, 0, 0, 0, 0, 1],
          [0, 0, 0, 1, 1, 0, 0, 0, 0],
          [1, 0, 0, 0, 0, 0, 1, 0, 0],
          [0, 1, 0, 0, 1, 0, 1, 0, 0],
          [0, 1, 0, 1, 0, 1, 0, 0, 0],
          [0, 0, 0, 0, 1, 0, 0, 1, 1],
          [0, 0, 1, 1, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 1, 0, 0, 0],
          [1, 0, 0, 0, 0, 1, 0, 0, 0]]
graph1 = g.Graph()

for i in range(len(matrix)):
    graph1.add_node(str(i))
    for j in range(len(matrix[i])):
        if matrix[i][j] != 0:
            graph1.add_edge(i, j)
print(isConnect(graph1))

''' output ------------------
isConnected =
False
---------------------------'''

# 2
V = []
# Adding vertices to array V
for i in range(2, 23):
        V.append(i)
print(V)
Edges = findEdges(V)
print(Edges)

''' output ------------------
V =
[2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22]

Edges =
[[4, 2], [6, 2], [6, 3], [8, 2], [8, 4], [9, 3], [10, 2], [10, 5], [12, 2],
[12, 3], [12, 4], [12, 6], [14, 2], [14, 7], [15, 3], [15, 5], [16, 2],
[16, 4], [16, 8], [18, 2], [18, 3], [18, 6], [18, 9], [20, 2], [20, 4],
[20, 5], [20, 10], [21, 3], [21, 7], [22, 2], [22, 11]]
--------------------------'''

# 2 part(a)
findDegree(Edges)

''' output ------------------
degree =
defaultdict(<class 'int'>, {'2': 10, '3': 6, '4': 5, '5': 3,
'6': 4, '7': 2, '8': 3, '9': 2, '10': 3, '11': 1, '12': 4,
'13': 0, '14': 2, '15': 2, '16': 3, '17': 0, '18': 4, '19': 0,
'20': 4, '21': 2, '22': 2})

degreeSeq =
[10, 6, 5, 4, 4, 4, 4, 3, 3, 3, 3, 2, 2, 2, 2, 2, 2, 1, 0, 0, 0]
--------------------------'''

# 2 part(b)
graph = g.Graph()

# Initialize the graph with empty vertices
for i in range(2, 23):
    graph.add_node(str(i))

# Add found edges to the graph
for edge in Edges:
    graph.add_edge(str(edge[0]), str(edge[1]))

print(distanceIfConnect(graph))
''' output ----------------------------------
distanceIfConnect(graph) =
[False, ['15', '3', '6', '2', '22', '11'], 6]

----Conclusion---
By using dijkstra algorithm, it can be conclude that the graph H'
is not connected, and the largest distance between vertices found
in the graph is from '11' to '15', which are 5 edges apart from
each other.
----------------------------------------------------------------'''
```