CS F222

Discrete Structures for Computer Science

Mini Project

|  |  |
| --- | --- |
| Name | ID no. |
| Megha Manoj | 2020A7PS0016U |
| Vansh Jain | 2020A7PS0079U |
| Aamina Taskeen | 2020A7PS0025U |

**Birla Institute of Technology and Science**

**D.I.A.C, Dubai, UAE**

**A Report**

**on**

**Generation of simple graphs**

**Prepared for**

**Dr A Somasundaram**

**Instructor in charge**

**By**

|  |  |
| --- | --- |
| **Name** | **ID no.** |
| **Megha Manoj** | **2020A7PS0016U** |
| **Vansh Jain** | **2020A7PS0079U** |
| **Aamina Taskeen** | **2020A7PS0025U** |

**Approved by**

**Dr. A. Somasundaram**

**Instructor in charge**

**November 2021**

**Abstract**

The main goal of this project is to understand the concept of graphs and use a programming language to implement it with the help of a mathematical background. In this report, we have focused on the types of graphs including digraphs and simple graphs, and generated different graphs using the Python programming language. We have also explained how probability plays an important role in randomly generating these graphs. Python dictionaries and libraries like networkx, matplotlib and string and their packages are used to get the outputs. We have also discussed the various applications of graph theory in real life like how they are used to visually illustrate relationships in the data and how they can be used for networking and navigation, indicating its importance in today’s world.

**Acknowledgement**

I would like to express my gratitude towards Dr. A. Somasundaram, the instructor- in charge of the course on discrete structures, for this opportunity to increase my understanding on the concept of simple graphs, randomness and usage of the Python language to code for programs related to graphs. I would also like to thank Dr. S Baskaran for his guidance and support throughout the project.

**Table of contents**

|  |  |
| --- | --- |
| **Abstract** | **3** |
| **Acknowledgement** | **4** |
| **List of figures** | **7** |
| 1. **Introduction**    1. **Aim**    2. **Graphs**       1. **Digraph**       2. **Simple graph**    3. **probability of an event**       1. **theoretical probability**           1. **equally likely event**       2. **randomness in probability** 2. **Software used**    1. **Packages and functions used**       1. **networkx**          1. **degree(num, w)**          2. **nodes()**          3. **edges(g)**          4. **draw(G, pos= nx.circular\_layout(G), labels, with\_labels, font\_size)**       2. **matplotlib.pyplot**          1. **pyplot.show()**       3. **string**          1. **ascii\_lowercase**       4. **enumerate()**       5. **len()** 3. **Source code**    1. **Performance of programming code**    2. **Displayed output** 4. **Applications of the program**    1. **In computer science**    2. **In navigation**    3. **In electrical engineering**    4. **In linguistics**    5. **In computer network** 5. **Conclusion**    1. **Graphs and probability**    2. **Software**    3. **Applications** 6. **References** | **8**  **8**  **8**  **8**  **9**  **9**  **10**  **10-11**  **11**  **11**  **12**  **12**  **12**  **13**  **13**  **13**  **13**  **14**  **14**  **14**  **14**  **14**  **14**  **14**  **15-16**  **16-17**  **18-27**  **28**  **28**  **28**  **28**  **28**  **29**  **30**  **30**  **30**  **30-31**  **31** |

**List of figures**

|  |  |  |  |
| --- | --- | --- | --- |
| **Serial no.** | **Figure** | **Title** | **Page no.** |
| **1** | **1.2** | **Graph** | **8** |
| **2** | **1.2.1** | **Digraph** | **8** |
| **3** | **1.2.2** | **Simple graph** | **9** |
| **4** | **1.3.1.1** | **Coin toss example** | **11** |
| **5** | **3.2** | **Output code 1** | **18** |
| **6** | **3.2** | **Output graph 1** | **18** |
| **7** | **3.2** | **Output code 2** | **19** |
| **8** | **3.2** | **Output graph 2** | **19** |
| **9** | **3.2** | **Output code 3** | **20** |
| **10** | **3.2** | **Output graph 3** | **20** |
| **11** | **3.2** | **Output code 4** | **21** |
| **12** | **3.2** | **Output graph 4** | **21** |
| **13** | **3.2** | **Output code 5** | **22** |
| **14** | **3.2** | **Output graph 5** | **22** |
| **15** | **3.2** | **Output code 6** | **23** |
| **16** | **3.2** | **Output graph 6** | **23** |
| **17** | **3.2** | **Output code 7** | **24** |
| **18** | **3.2** | **Output graph 7** | **24** |
| **19** | **3.2** | **Output code 8** | **25** |
| **20** | **3.2** | **Output graph 8** | **25** |
| **21** | **3.2** | **Output code 9** | **26** |
| **22** | **3.2** | **Output graph 9** | **26** |
| **23** | **3.2** | **Output code 10** | **27** |
| **24** | **3.2** | **Output graph 10** | **27** |

1. **Introduction**
   1. **Aim-**

Generate at random 10 different simple graphs each with 20 vertices so that each such graph is equally likely to be generated.

* 1. **Graphs-**

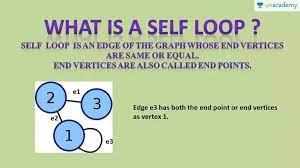
A graph is a pair of sets, **(*V, E*)**, where ***V*** is a set of vertices (nodes) and ***E*** is a set of ordered pairs of vertices.

Figure 1.2. Graph

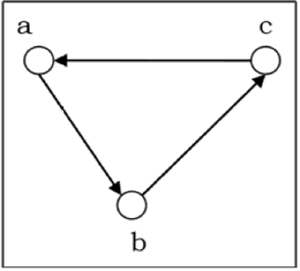
For example,

Let **G (*V, E*)** be a graph.

***V****=* ***V*(G)** = {1,2,3} and

***E****=* ***E*(G)** = {(1, 1), (1, 2), (2, 3)} There are various types of graphs with distinct conditions.

* + 1. **Digraphs**-

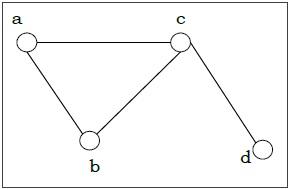
Digraph or directed graph is an ordered pair of sets, ***G* = (*V, A*)**, where ***V*** *i*s a set of vertices and ***A*** is a set of ordered pairs of vertices of ***V.*** A digraph is represented by a diagram composed of points indicating the vertices in ***A***, which are joined by arrows, indicating the direction of edges, from vertex to vertex.

For example-

*Figure 1.2.1. Digraph*

Let graph **G**= (***V, A***), where

***V***= {a, b, c} and ***A***= {(a, b), (b, c), (c, a)} is represented by, figure 1.2.1.

* + 1. **Simple graphs-**

A simple graph is a graph with no loops. A loop is an ordered pair where a vertex is related to itself.

A simple graph’s edges do not have arrows indicating the direction of the edges.

*Figure 1.2.2. Simple graph*

For example,

In figure 1.2.2, ***G*** is a

simple graph with a set of vertices, ***V***= {a, b, c, d} and set of edges ***E***= {(a, b), (a, c), (c, d), (b, c)}.

* 1. **Probability of an event**

The probability of an event refers to the chance of that particular event occurring. The probability of an event is calculated by the number of favorable outcomes divided by the total number of outcomes.

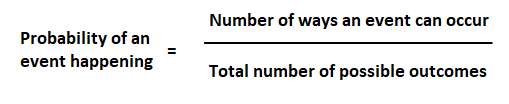


Figure 1.3. Probability of an event equation

The probability of occurrence of an event is between 0 and 1. The type of event plays a role in the probability of the event occurring.

* + 1. **Theoretical probability**

Theoretical probability is a branch of probability that deals with the likelihood of the occurrence of an ideal event.

It deals with the reasoning of how an event would occur on the basis of its outcomes.

For example,

a coin when tossed can land on either of its faces or end up standing on its edge. theoretically, the coin would land on either of its faces. so the probability outcome of which side the coin lands is either of its 2 faces.

* + - 1. **Equally likely event**

A likely event refers to an event that has a higher chance of occurrence than other events under theoretical probability. a likely event has a probability outcome between 0.5 and 1.

An unlikely event is the opposite of a likely event with a probability outcome between 0 and 0.5.

2 or more events are said to be equally likely events when each of the events which has occurred has an equal probability of 0.5 and the event has a fair outcome along with the other ones.

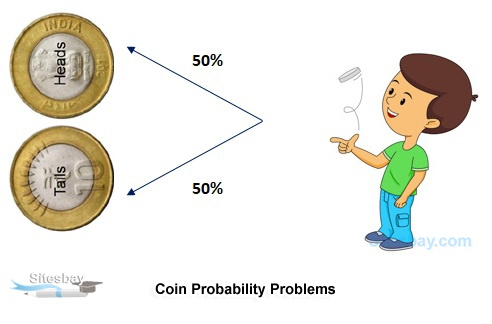


Figure 1.3.1.1. Coin toss example

For example, when a coin is tossed there is an equal probability of it landing on either of its faces.

* + 1. **Randomness in probability**

Randomness in probability refers to the uncertainty and unpredictability of an event. In a finite range of given sets, the probability of occurrence of a random event can be predicted up to some extent by considering the various possible outcomes.

For example,

When a coin is tossed once, it may fall on its head or tail, but the outcome of the next toss cannot be predetermined. The probability of the coin falling on either of its sides is 0.5.

1. **Software used**

The interpreted language of Python has been used for coding and executing the program. As python is a high-level language, the code is compiled faster. The varieties of libraries and packages makes coding easier and much simpler to understand.

* 1. **Packages, libraries and functions used**

For achieving the desired output, the following packages and libraries have been imported into the program using the built-in keyword import:

* + 1. **Networkx**

networkx is a python package used for creation, manipulation and study of structures, graphs and complex networks. It contains a variety of functions which helps to code graphs and read them in different ways.

* + - 1. **nx.gnp\_random\_graph(n,p)**

gnp\_random\_graph() is a function within the networkx package which generates random graphs by taking the parameters,

*n-* number of nodes the graph must have in *int*

*p-* the probability of edge creation in *float*

Additional parameters of this function include-

*s-* an indicator of random generation in *int*

*directed-* a Boolean value which makes the generated

graphs directed if *true.*

* + - 1. **degree(num, w)**

degree() returns the degree of a node, the number of edges adjacent to a particular vertex, as a dictionary or number. the parameters refer to,

*num-* a container of nodes

*w-* edge attribute which holds a numerical value as *string*, otherwise 1 if *None.*

* + - 1. **nodes()**

returns nodes of a graph. It is useful for looking up data and set-like operations through a class NodeView.

* + - 1. **edges(g)**

returns edges of a graph as tuples as well as provides edge-attributes observations and set-like operations through a class EdgeView. Here, *G* is a generated graph, for which all the edges will be returned.

* + - 1. **draw(G, pos= nx.circular\_layout(G), labels, with\_labels, font\_size)**

draw() function is used for displaying the graph with package matplotlib as a simple graph.

the parameters of this function include,

*G-* the graph to be drawn

*pos-* nodes and positions in the form of a dictionary where nodes are keys and positions are the values.

The function *circular\_layout* with the parameter as the required graph, positions the nodes of the graph in a circle.

*labels-* node labels in the form of a dictionary

*with\_labels-* takes *boolean* value. If *True* then draws labels on the nodes

*font\_size-* to set font size of the labels on the nodes

* + 1. **matplotlib.pylot**

matplotlib is a python library used for visualizing and creating and modifying plots. The package pyplot is an interactive interface for plotting.

* + - 1. **pyplot.show()**

show() function displays all the figures which have been created. It waits for the first figure to be closed before displaying the next one.

* + 1. **string**

string is the built-in class for the data type *string*. as some modules which are not built-in within the string class, string class needs to be imported.

* + - 1. **ascii\_lowercase**

It is a predefined string constant which returns lowercase alphabets.

* + 1. **enumerate(g)**

enumerate() is a built-in python function which returns a count and the object within the iterator, *g*, as a tuple.

* + 1. **len()**

returns the length of a string, array, dictionary etc.

1. **Source Code**

To generate 10 different simple graphs which are random, with each graph having 20 vertices such that each graph is equally likely to be generated, a source code has been created in a Python file, pq03\_2020A7PS0079U.py .

"""Q3) Generate random 10 different simple graphs each with 20 vertices so that each such graph is equally likely to be generated."""

import networkx as nx

#library to import graph related functions(to install enter "pip install networkx[default]" )

import matplotlib.pyplot as plt

from string import ascii\_lowercase

labelsdict = {}

#gnp\_random\_graph(n,p), where 'n' refers to number of nodes

#and 'p' corresponds to the case(p=0.5) where all (2^(nC2)) graphs on n vertices are chosen with equal probability.

for i in range (10):

G = nx.gnp\_random\_graph(20,0.5)

for j, node in enumerate(G.nodes()):

#assigns node labels to each node

labelsdict[node] = ascii\_lowercase[j]

print("GRAPH ",i+1,"----------------------------------------------->")

print("\nEdge Pairs:-")

print("G = ",nx.edges(G),)

#prints edge pairs of each node

print("Degree Sequence:-")

#prints Degree Sequence of each node

for k in range(len(labelsdict)):

deg = G.degree[k]

print(labelsdict[k],": ",deg)

print("\n")

nx.draw(G, pos= nx.circular\_layout(G), labels=labelsdict, with\_labels=True, font\_size=14) #draws and displays the graph

plt.show()

* 1. **Performance of programming code**

The packages- *networkx*, *matplotlib.pyplot* and string constant , *ascii\_lowercase*, have been imported into the code. A dictionary *labelsdict* has been created for storing the node labels.

Using a for loop which iterates 10 times, a variable G has been initialized within the *for* loop which creates random graphs, which have 20 vertices, for a probability of 0.5 using *gnp\_random\_graph()*. The function will generate random graphs which have equal probabilities of occurrence from a combination of (2^(nC2)) graphs. With this particular for loop as the main loop, 2 *for* loops has been created within its scope.

nested *for* loop is created which iterates for *enumerate(G.nodes())*. here *G.nodes()* returns the nodes of the graph generated for which enumerate will take up its count. Within this for loop, the dictionary *labelsdict* with key as the node iterated through the for loop assigning labels to each node using *acsii\_lowercase[j]*, where *j* is the count of the node which will be converted and returned as a string constant, thereby giving each node a label.

within the same *for* loop, *edges()* has been used to display the edge pairs of each node of graph G.

for displaying the degree sequence of each node, another *for* loop has been initialized within the main loop, that takes the range of iteration as the length of the dictionary labelsdict and prints the degree sequence of each node.

Finally within the main loop, the function *draw()* has been used for creating and visualizing the graph. The function *show()* from *matplotlib.pyplot* is used at the end for displaying the created graph through a tab.

* 1. **Displayed outputs**

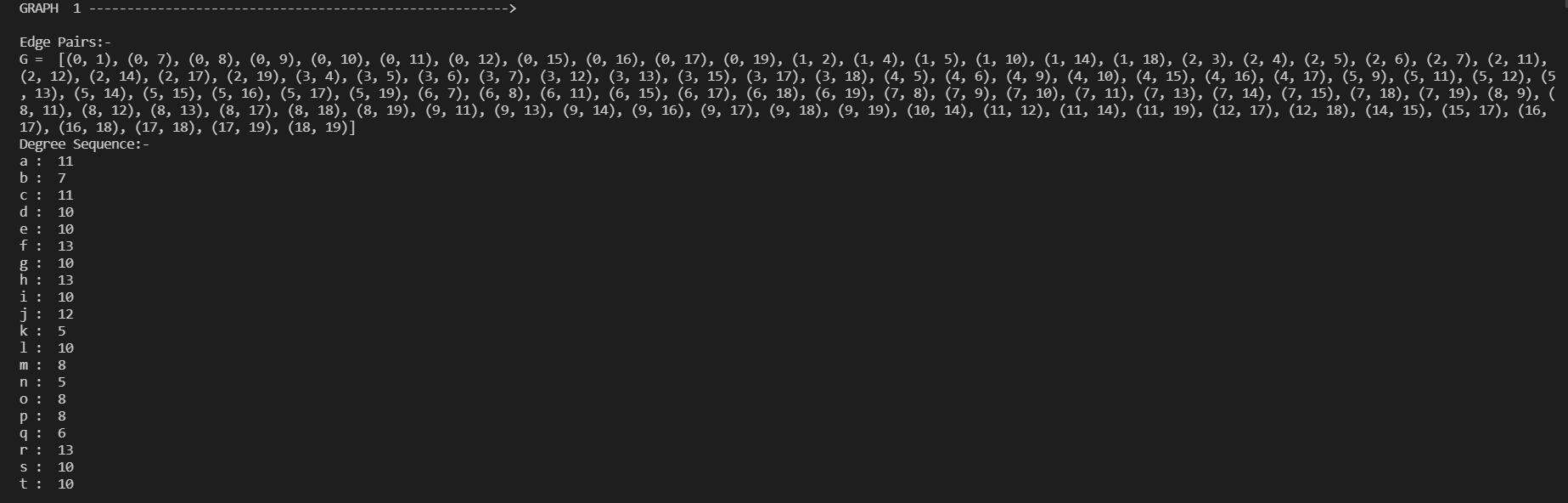
****

Figure 3.2. Output code 1

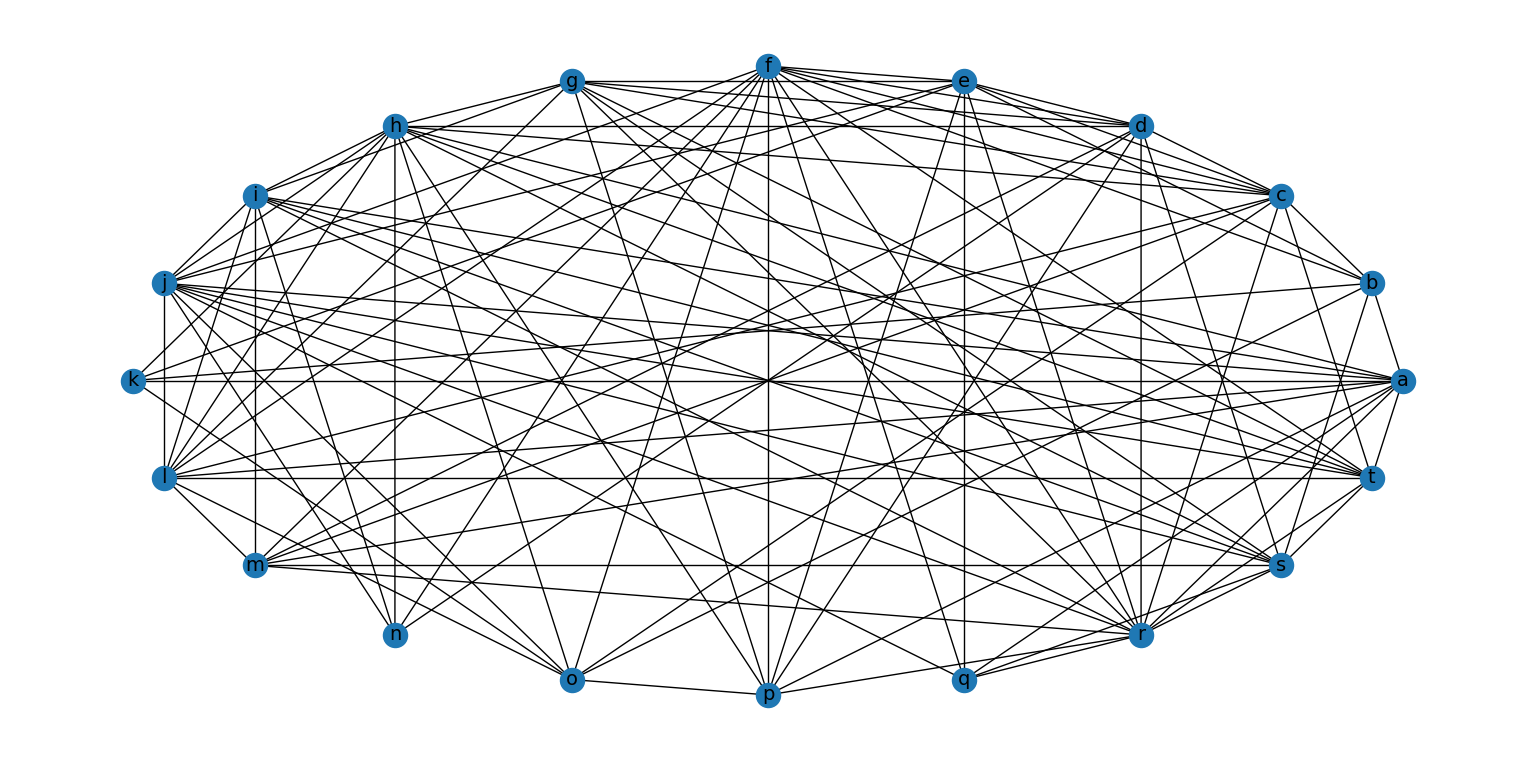
****

Figure 3.2. Output graph 1

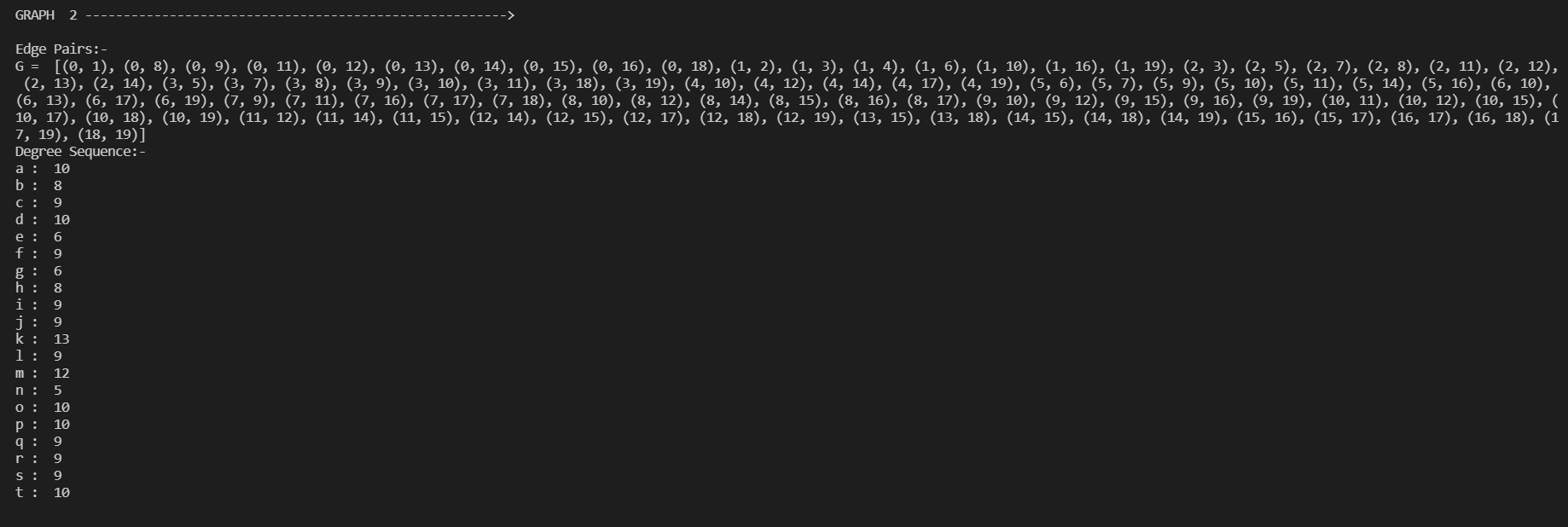
****

Figure 3.2. Output code 2

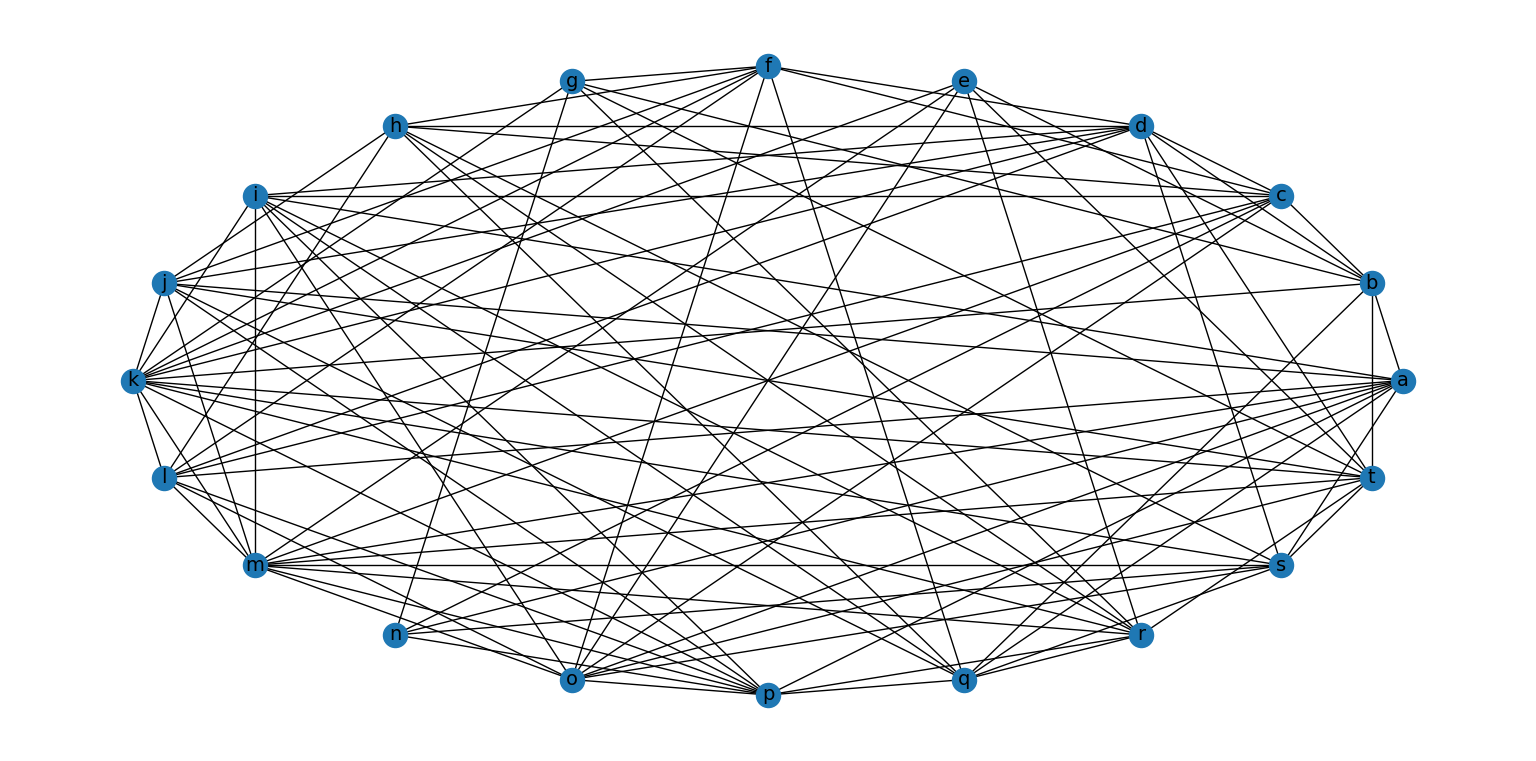
****

Figure 3.2. Output graph 2

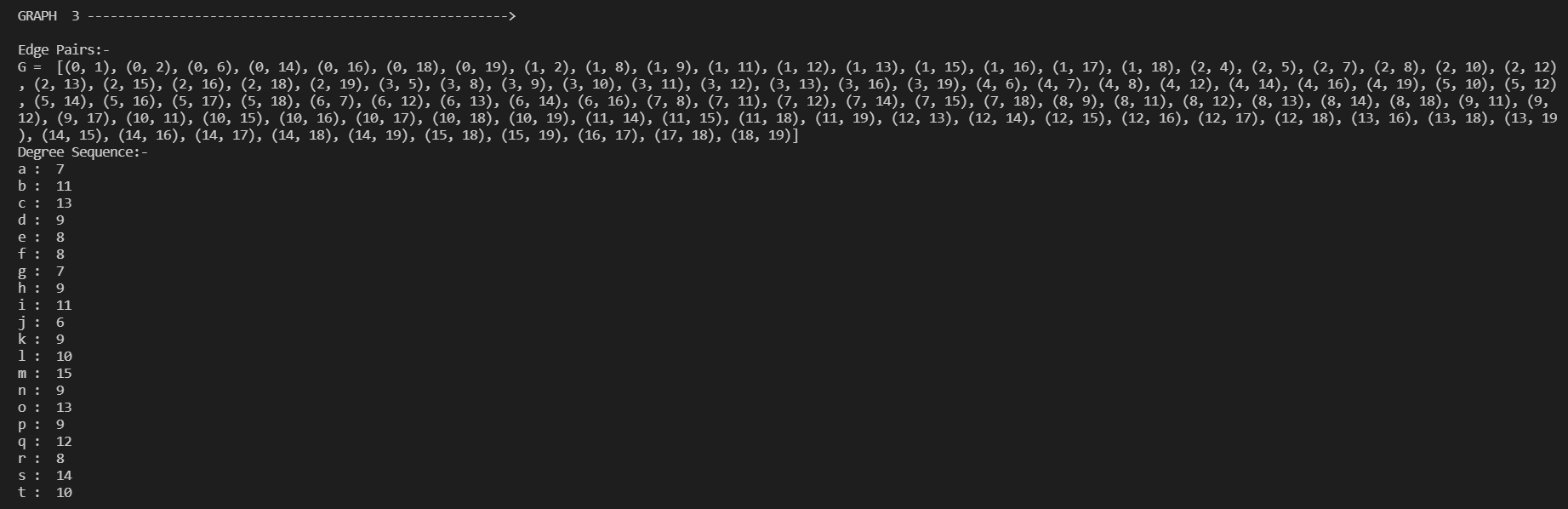
****

Figure 3.2. Output code 3

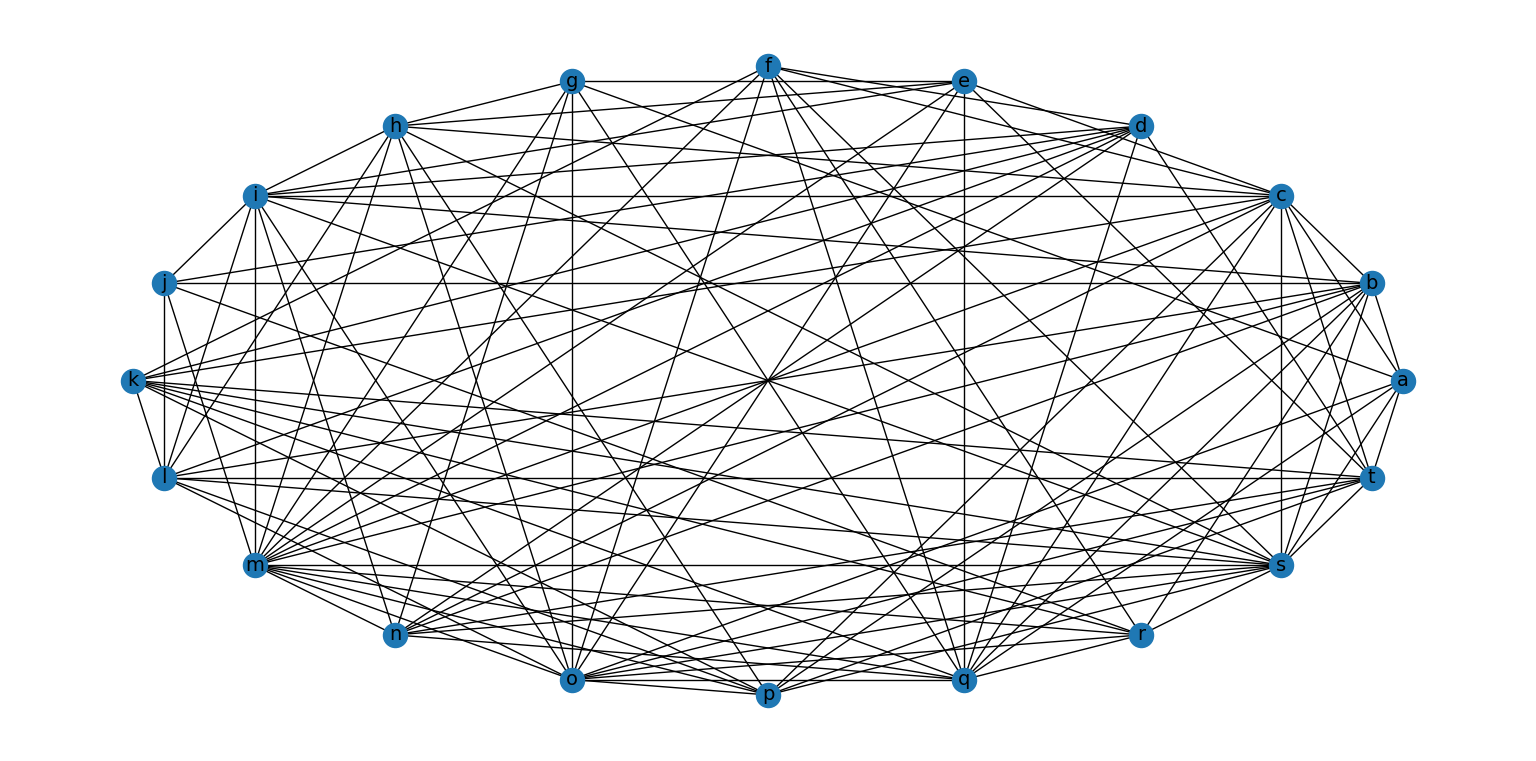
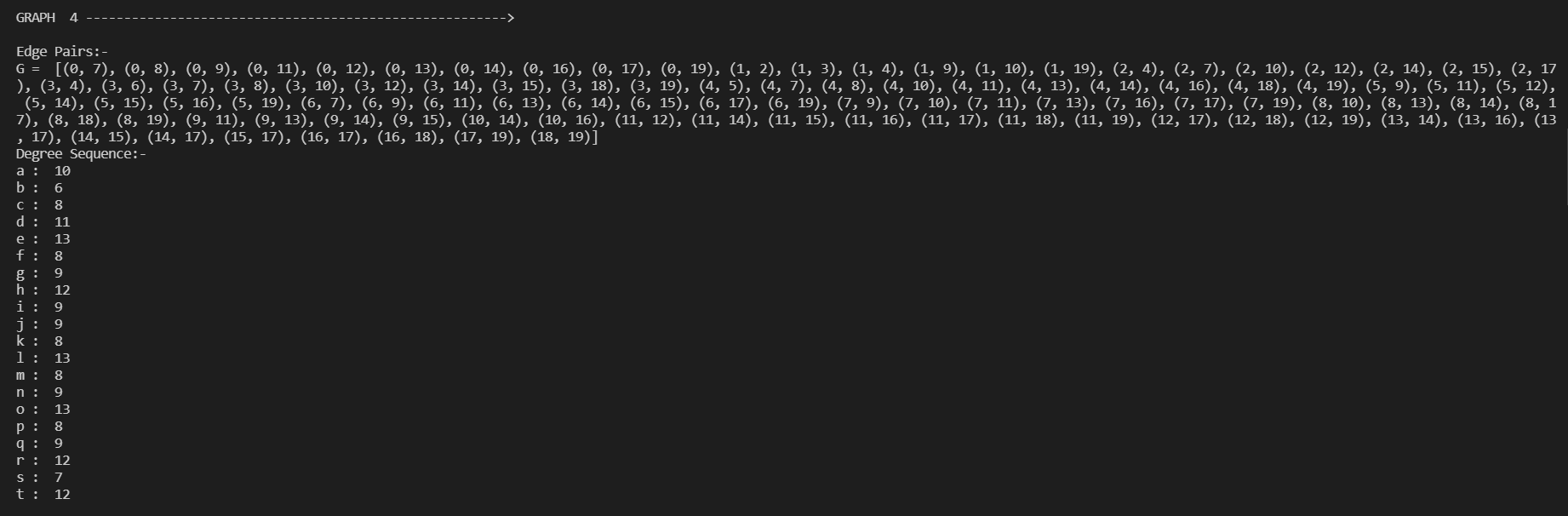
****

Figure 3.2. Output graph 3

****Figure 3.2. Output code 4

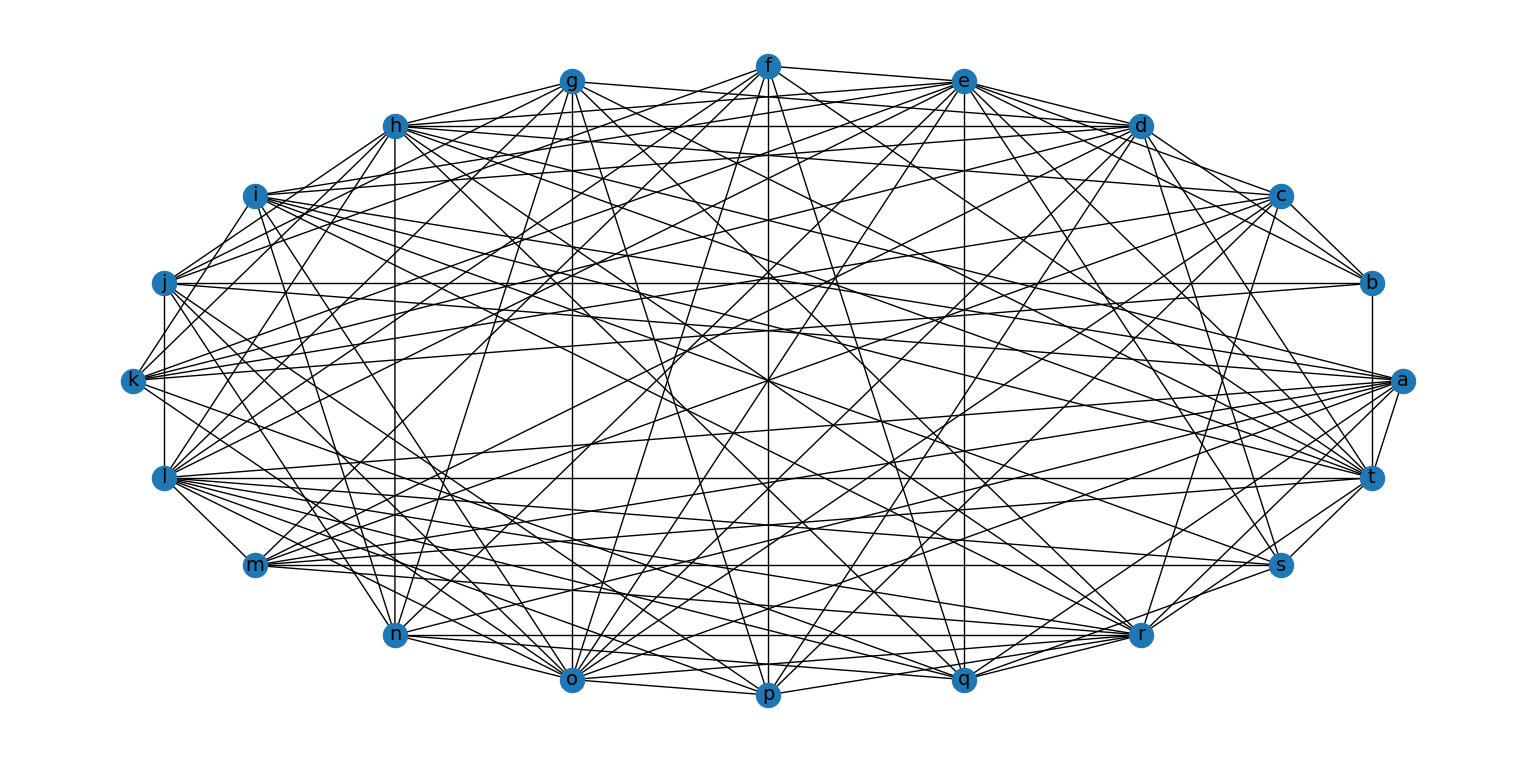
****

Figure 3.2. Output graph 4

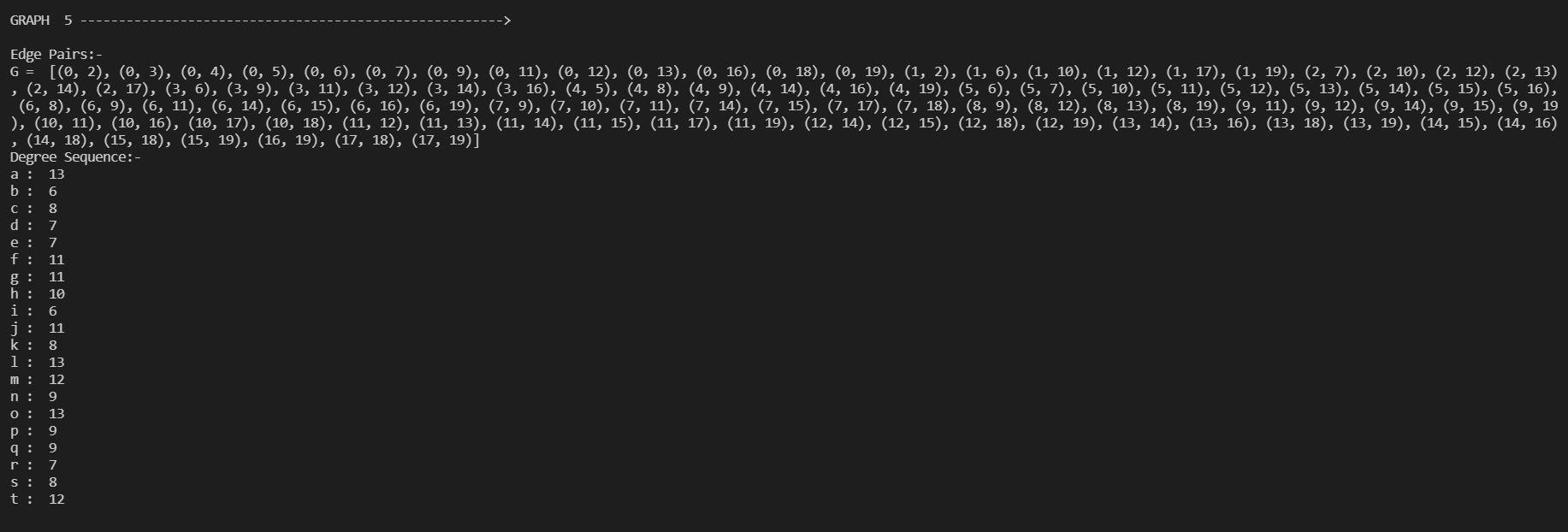
****

Figure 3.2. Output code 5

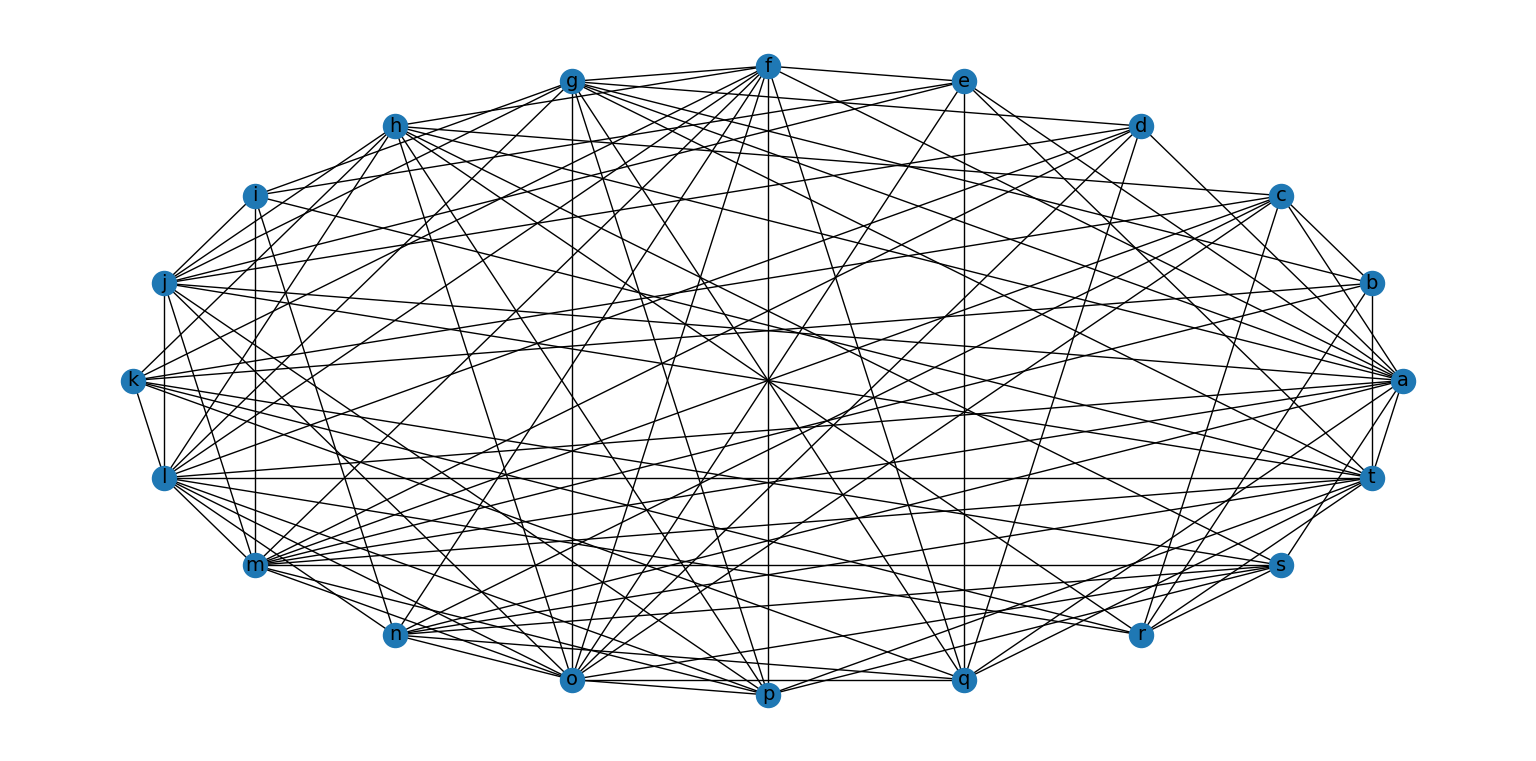
****

Figure 3.2. Output graph 5

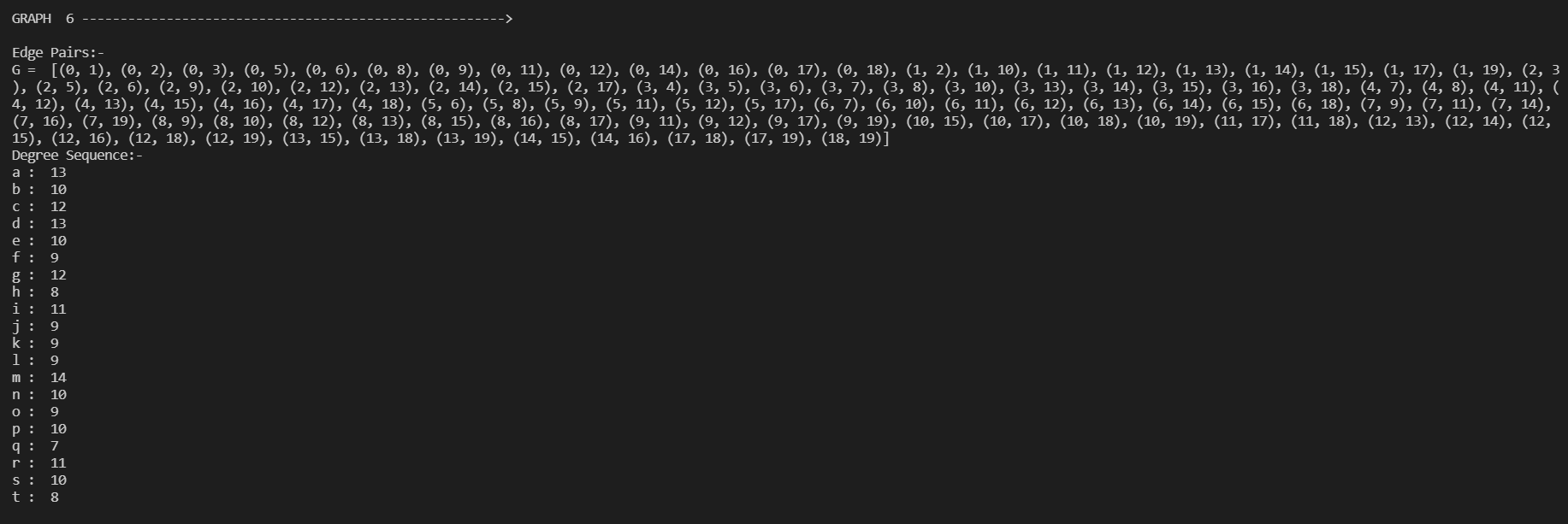
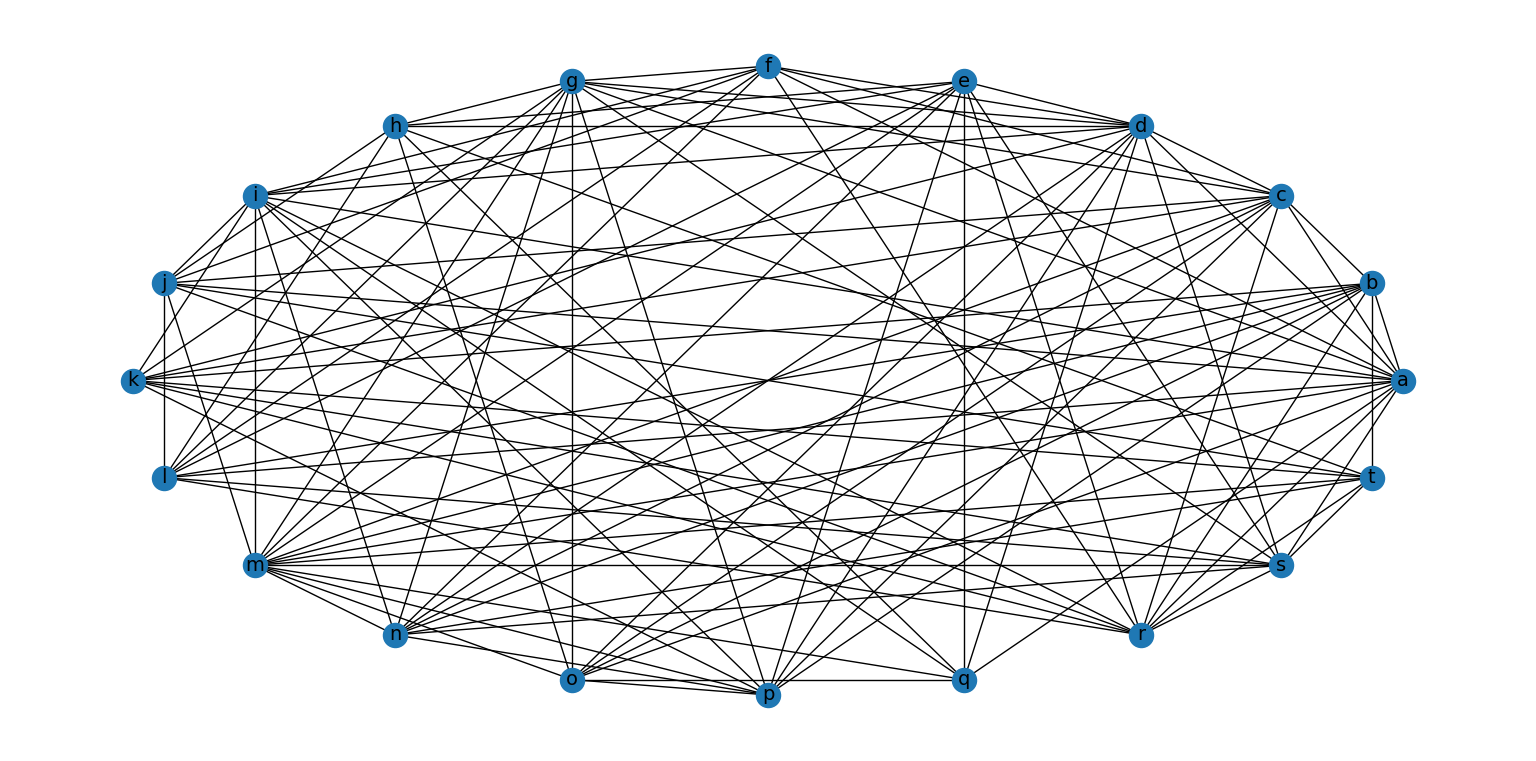
****Figure 3.2. Output code 6 ****

Figure 3.2. Output graph 6

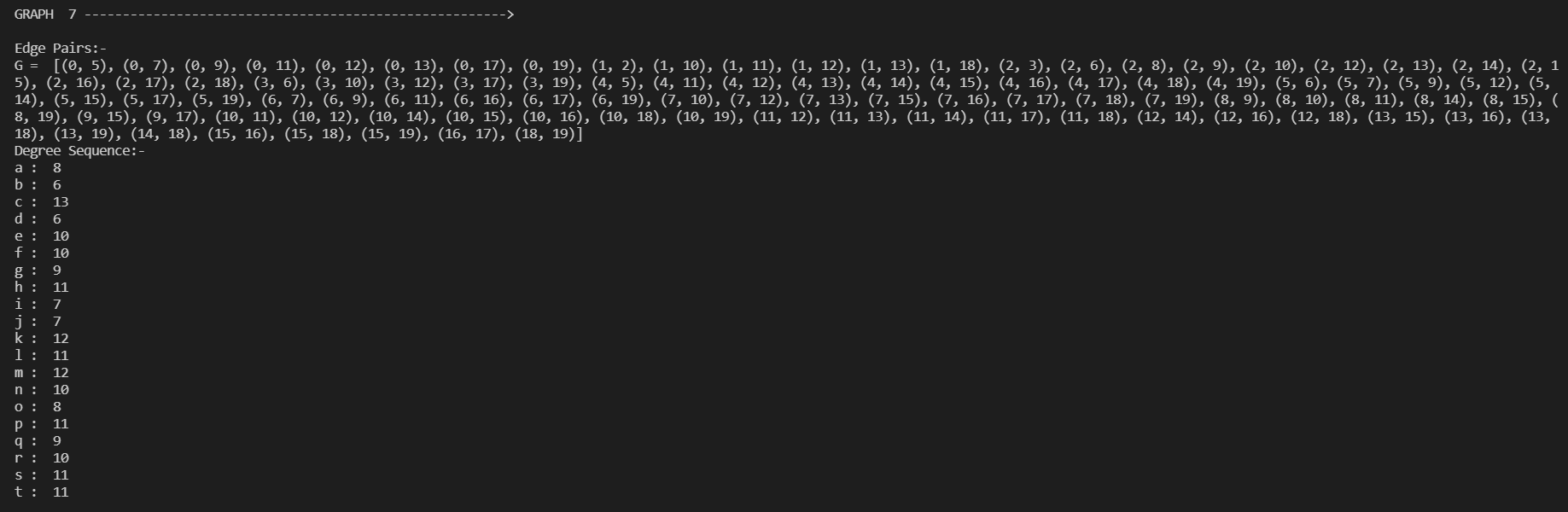
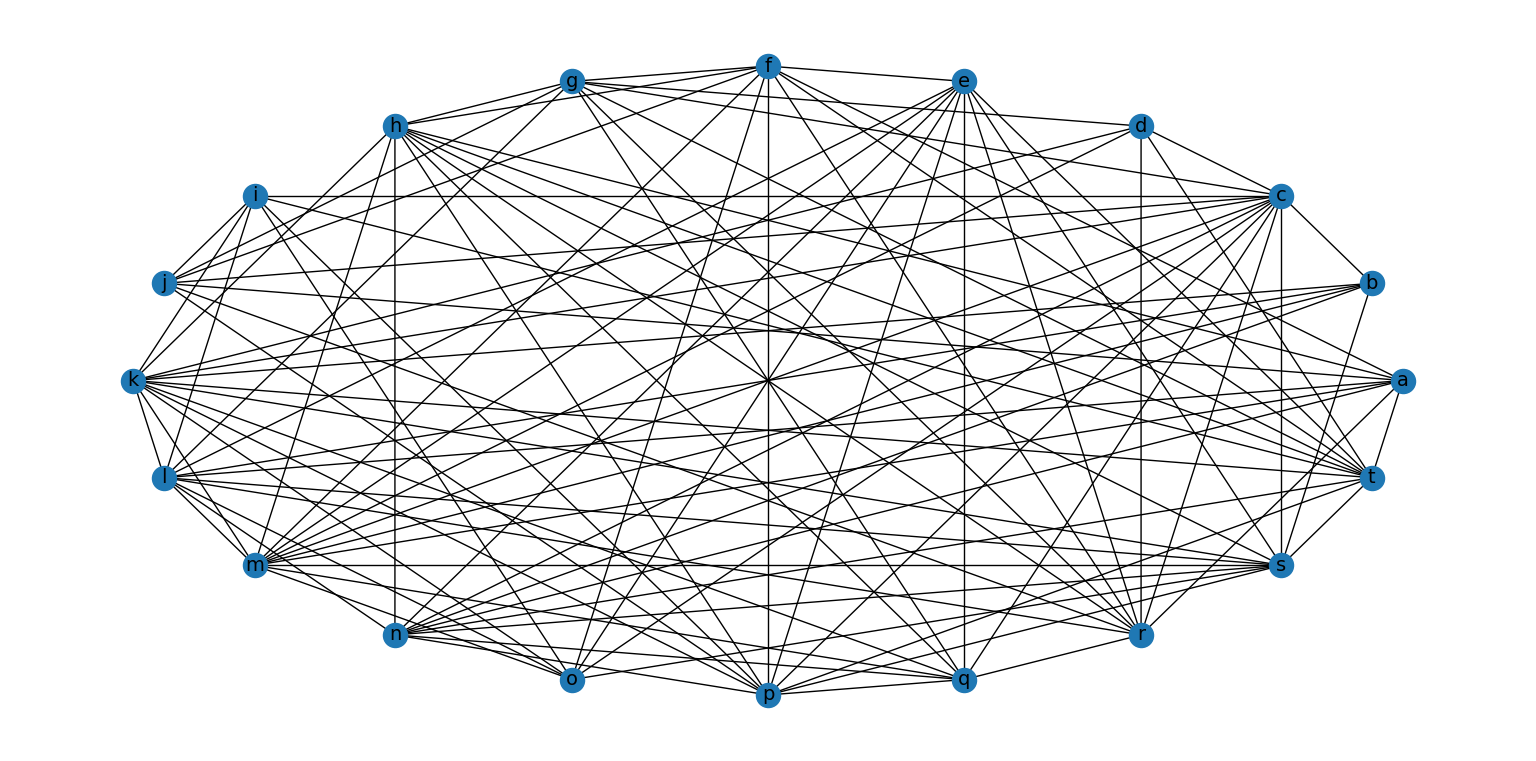
****1Figure 3.2. Output code 7****

Figure 3.2. Output graph 7

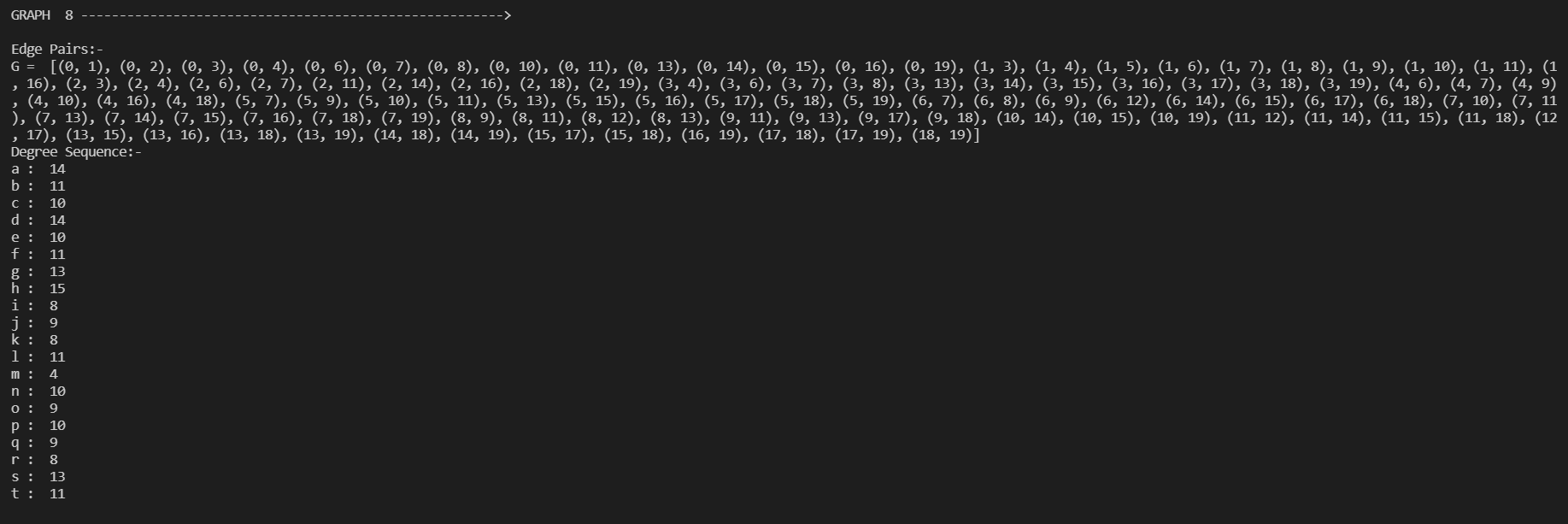
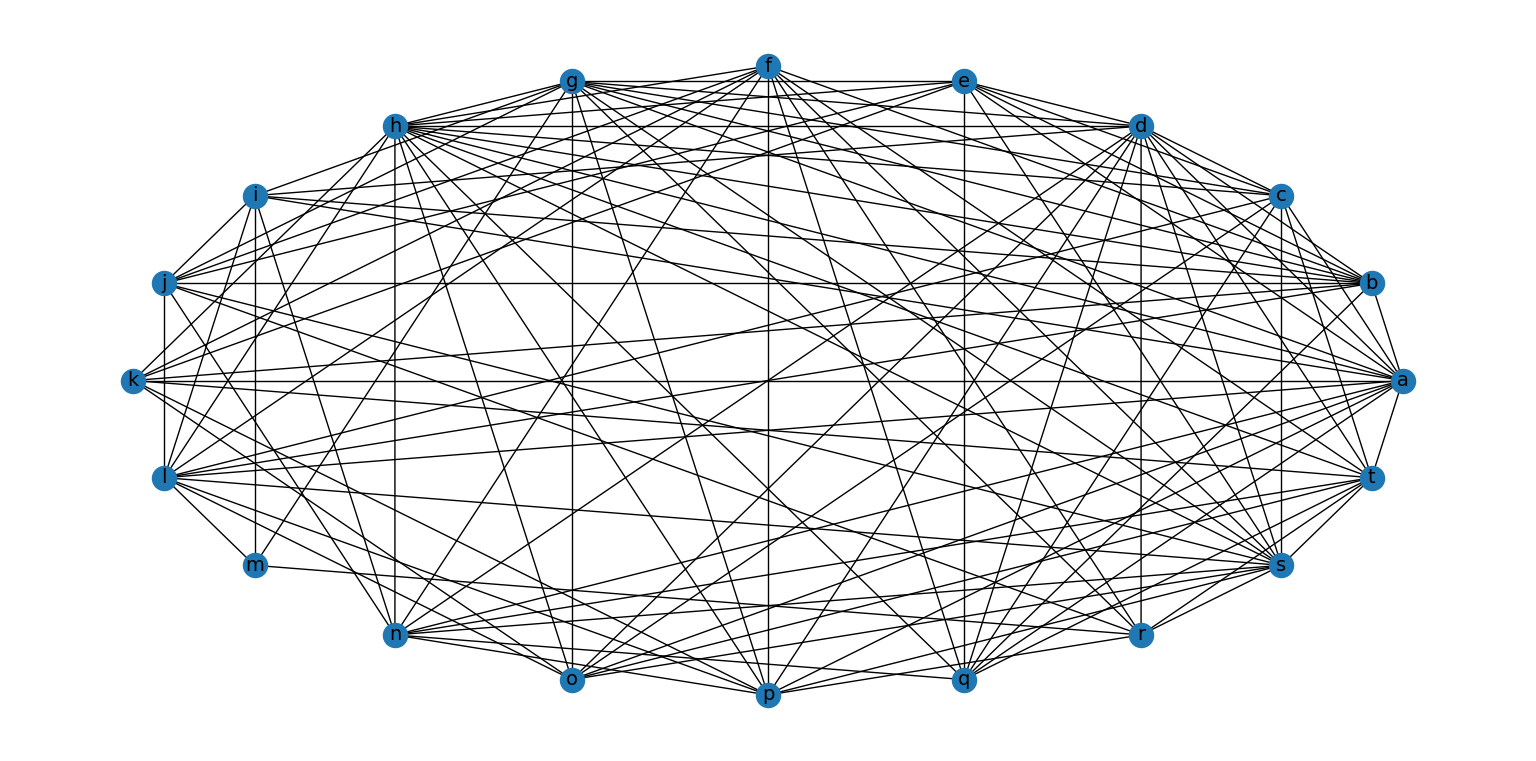
****Figure 3.2. Output code 8****

Figure 3.2. Output graph 8

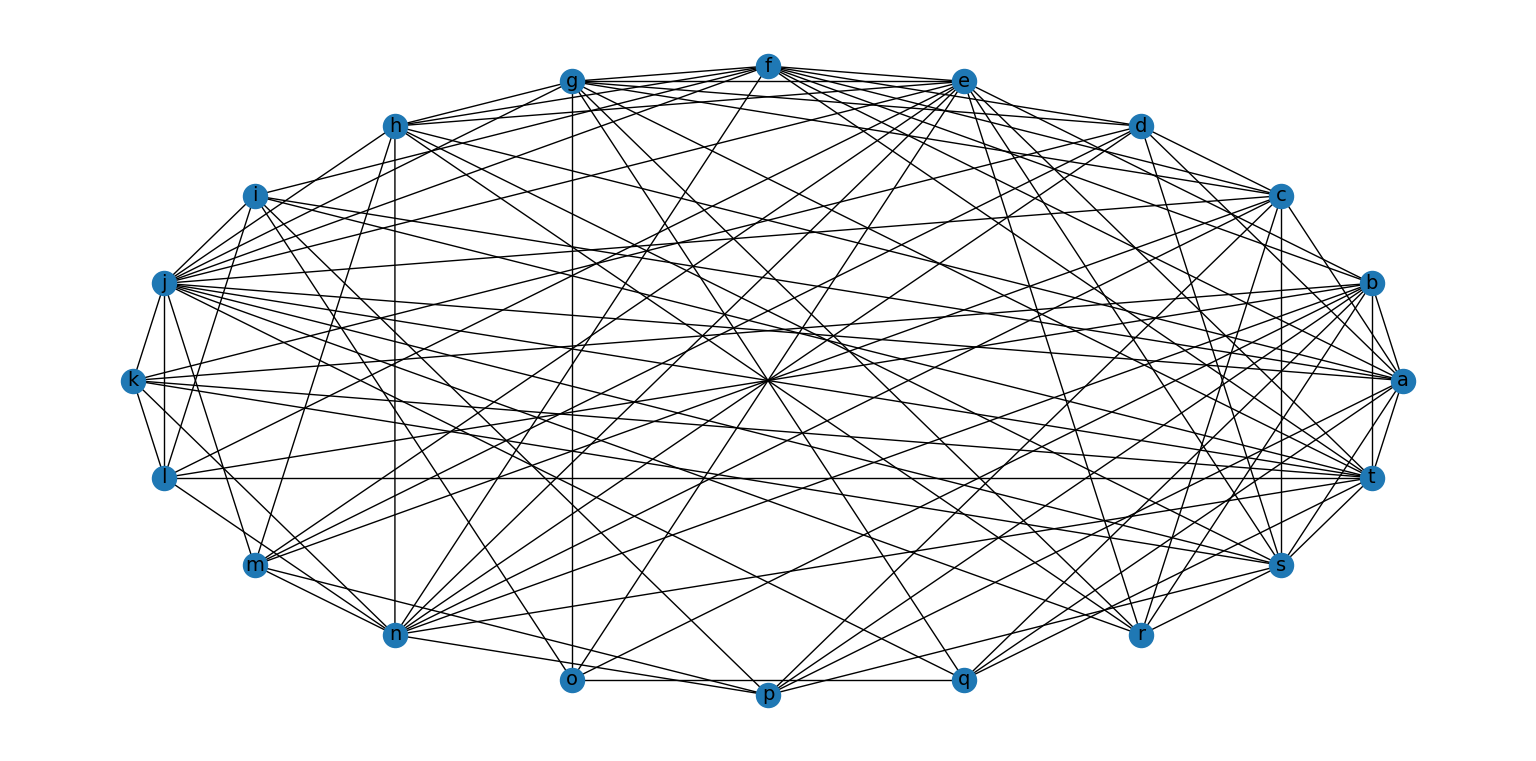
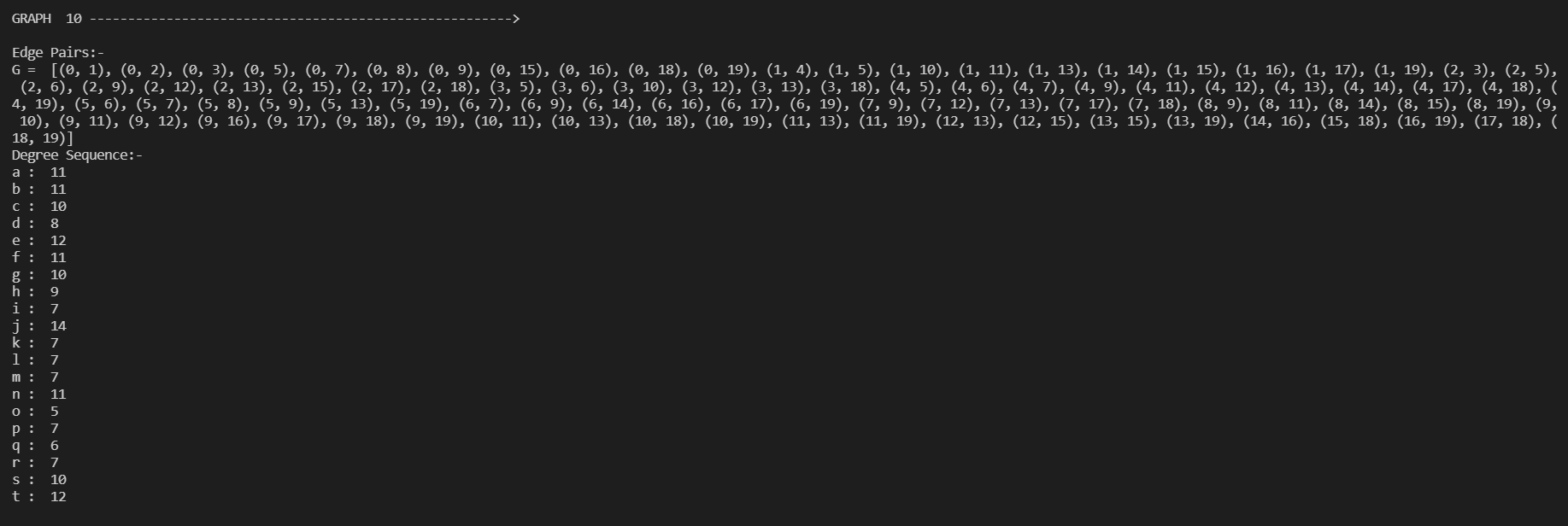
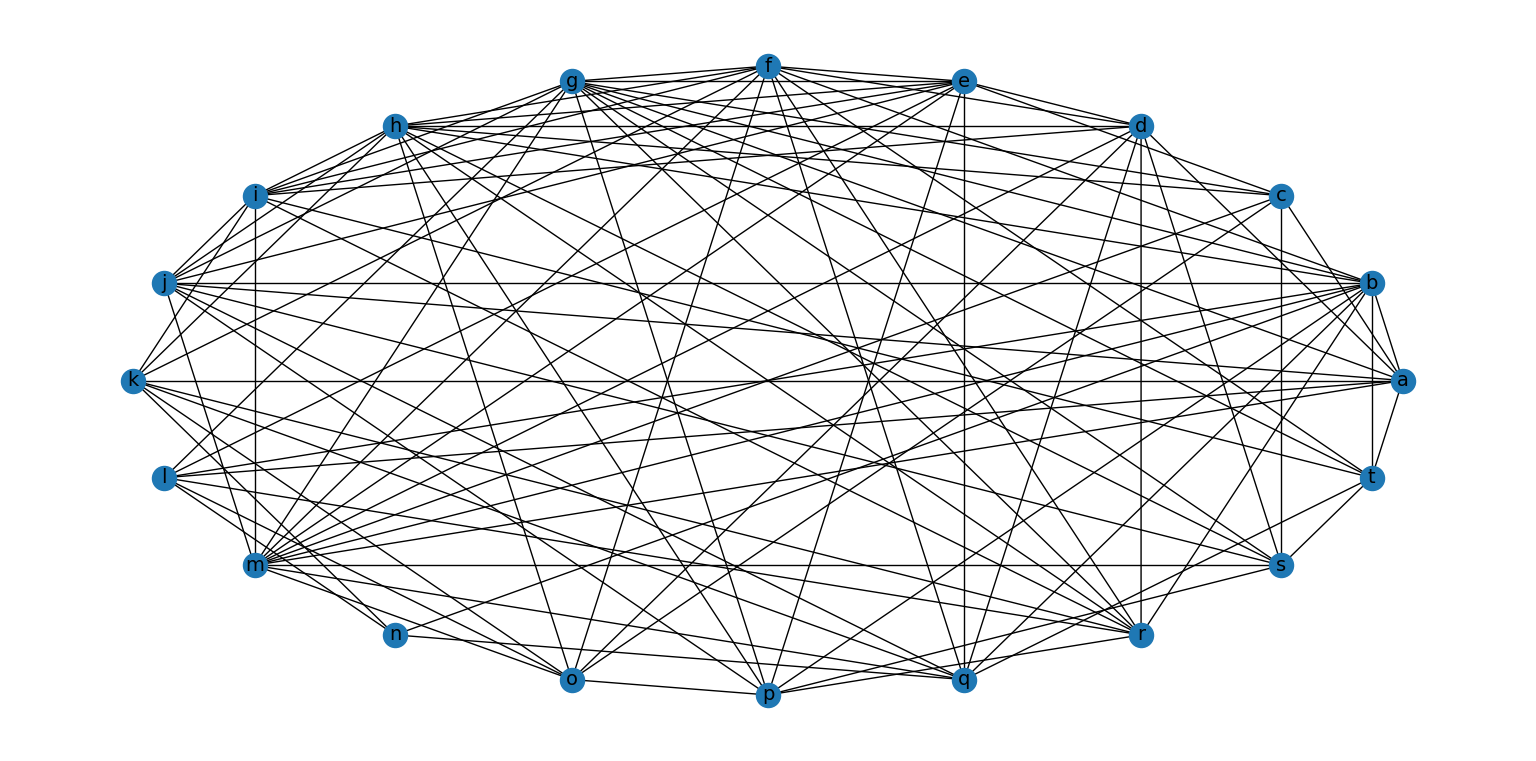
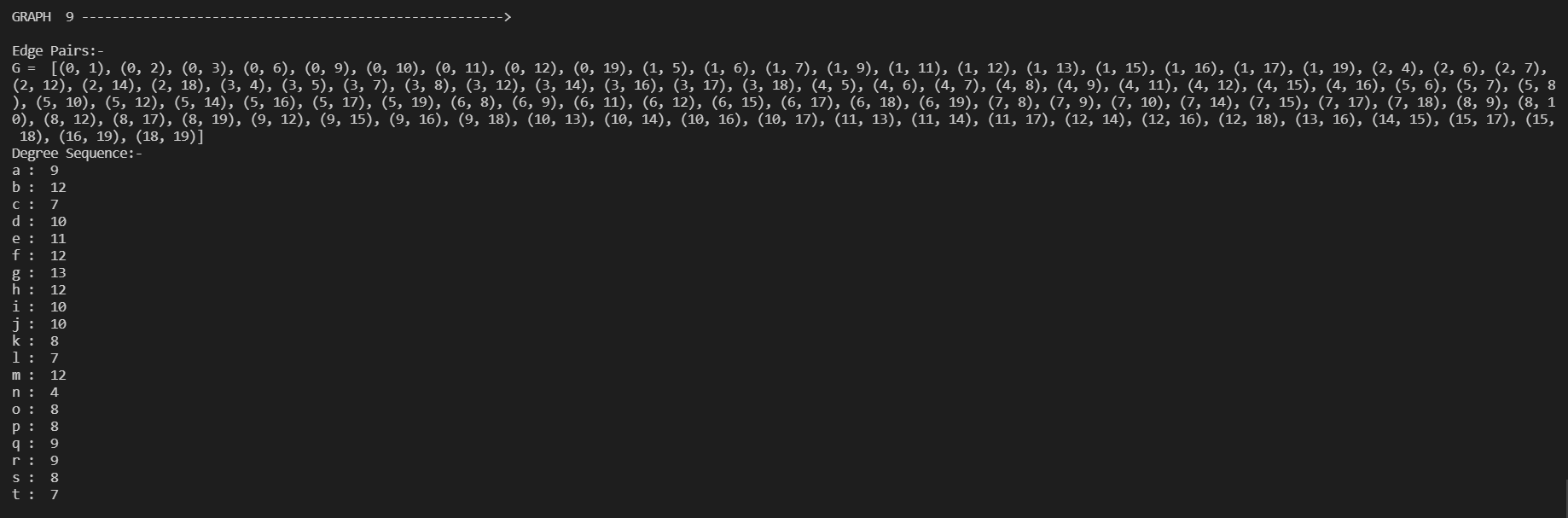
****

Figure 3.2. Output graph 9

Figure 3.2. Output graph 10

1. **Applications of the program**

Graph theory has many applications in different fields like computer science, electrical engineering, transportation, networking etc.

**4.1 In computer science**

* used for the study of algorithms like Dijkstra's Algorithm, Prim's Algorithm, Kruskal's Algorithm
* Graphs are used to define the **flow of computation**.
* Graphs are used to represent **networks of communication**.
* Graphs are used to represent **data organization**.

**4.2 In Navigation**

* Graph theory is used to find the shortest path in a road or a network.
* In Google Maps, various locations are represented as vertices or nodes and the roads are represented as edges and graph theory is used to find the shortest path between two nodes.

**4.3 In electrical engineering**

Graph theory is used in designing circuit connections.

**4.4 In Linguistics**

Graphs are mostly used for parsing of a language tree and grammar of a language tree.

**4.5 In Computer Network**

* The relationships among interconnected computers within the network follow the principles of graph theory.
* Graph theory is also used in network security.
* We can use the vertex coloring algorithm to find a proper coloring of the map with four colors.

1. **Conclusion**

This article aims to give a flavor of how a field of random graphs can be generated with a rich mathematical theory with applications across many scientific and real-life disciplines.

**5.1 Graphs and Probability**

From the theory point of view, a special focus was put on the concept of digraphs and simple graphs. A directed graph or digraph G is defined as ***(V, E)*** if E is a subset of ***VxV***. They have both in and out degrees. They are used to represent relationships between vertices. they can be non-directed or multi directed.

Probability is also discussed which is important in the emergence of a randomly generated graph.

**5.2 Software**

Python offers multiple great graphing libraries that come packed with lots of different features. Here we have used networkx and matplotlib.pyplot for visualizing and manipulating graphs. In addition to this, we have also used string.

**5.3 Applications**

From the viewpoint of application, the theory of random graphs has proven to be appropriate for the description and analysis of complex structures arising everywhere from nature. On the other hand, diverse applications like navigation, data organization, flow of computation, linguistic semantics, network security, parse trees, circuits, etc., continue to motivate and inform the study of random graphs.

The expansion of random graph theory and its applications shows us again how much abstract mathematical ideas can teach us about the “real world”.

1. **References**

[1]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

[2]<https://www.javatpoint.com/graph-theory-applications>

[3]<https://networkx.org/documentation/stable/reference/classes/generated/networkx.Graph.nodes.html>

[4]<https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.html#:~:text=pyplot%20is%20a%20state%2Dbased,%2C%200.1>)%20y%20%3D%20np

[5]<https://mathworld.wolfram.com/SimpleGraph.html>

[6]Prunescu, Mihai. (2015). Re: What is the relationship between randomness and probability?. Retrieved from:

<https://www.researchgate.net/post/What-is-the-relationship-between-randomness-and-probability/550c60b1f15bc76c3d8b4639/citation/download>