

# Lab4, 70

May 14, 2021

**Author:** *Golovko Eugene Olegovich*

**Group:** *K-12*

**Variant:** *70*

**Lab instructor:** *Efremov Mykola Serhiiovych*

```
[1]: import numpy as np
import networkx as nx
import matplotlib.pyplot as plt

from queue import Queue
```

```
[2]: FILE_PATH = "./graph.txt"

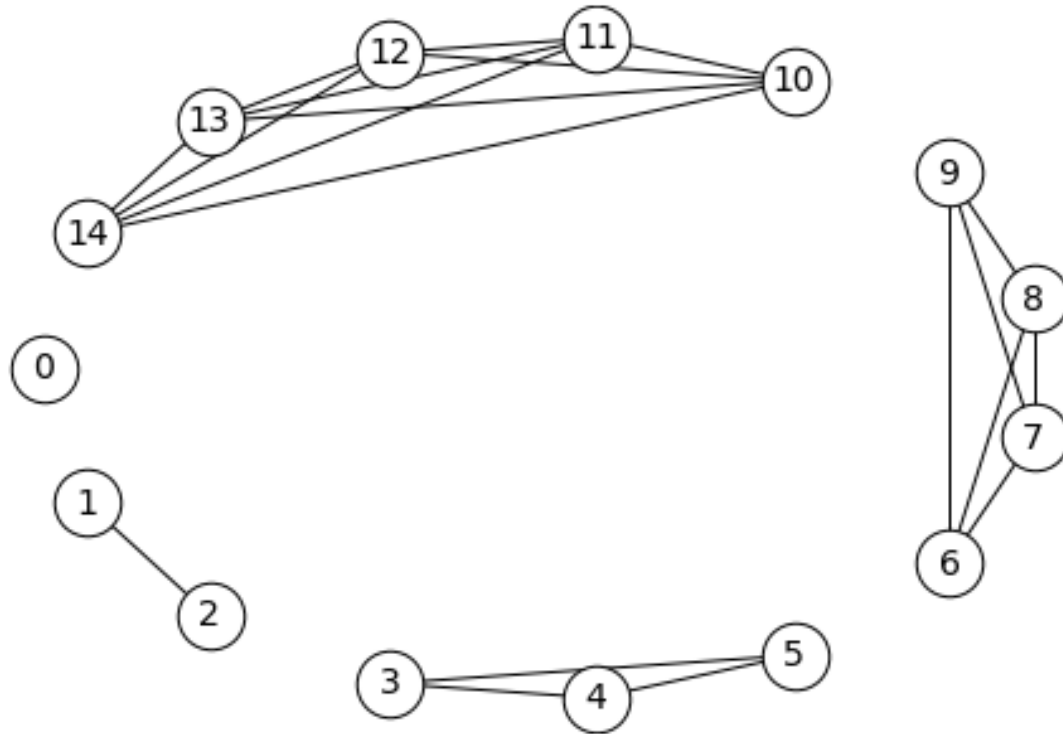
N = 15
V = 20
K = 5

SIZES = {'node_size': 2000, 'font_size': 25, 'width': 3}
BASE_COLORS = {'font_color': 'black', 'node_color': 'white', 'with_labels': 1
               ↪ True}
BLACK = {'edgecolors': 'black', 'edge_color': 'black'} | BASE_COLORS
```

## 0.1 Task 2

```
[3]: g = nx.read_adjlist(FILE_PATH, nodetype=int, create_using=nx.Graph)
```

```
[4]: nx.draw_shell(g, **BLACK, node_size=700, font_size=14)
plt.savefig("Task 2.png")
```



## 0.2 Task 3

```
[5]: shift = {
    0: np.array((0, 0)),
    1: np.array((3, 0)),
    2: np.array((3, 0)),
    3: np.array((6, 0)),
    4: np.array((6, 0)),
    5: np.array((6, 0)),
    6: np.array((9, 0)),
    7: np.array((9, 0)),
    8: np.array((9, 0)),
    9: np.array((9, 0)),
    10: np.array((12, 0)),
    11: np.array((12, 0)),
    12: np.array((12, 0)),
    13: np.array((12, 0)),
    14: np.array((12, 0))
}

center_pos = {
    0: np.array([6.123234e-17, 1.000000e+00]),
    1: np.array([6.123234e-17, 1.000000e+00]),
```

```

2: np.array([ 6.123234e-17, -1.000000e+00]),
3: np.array([6.123234e-17, 1.000000e+00]),
4: np.array([ 0.8660254, -0.5]),
5: np.array([-0.8660254, -0.5]),
6: np.array([0, 0]),
7: np.array([ 0.8660254, -0.5]),
8: np.array([-0.8660254, -0.5]),
9: np.array([-1.8369702e-16, 1.0000000e+00]),
10: np.array([6.123234e-17, 1.000000e+00]),
11: np.array([0.25105652, 0.00901699]),
12: np.array([0.95105652, -0.80901699]),
13: np.array([-0.95105652, -0.80901699]),
14: np.array([-0.25105652, 0.00901699])
}

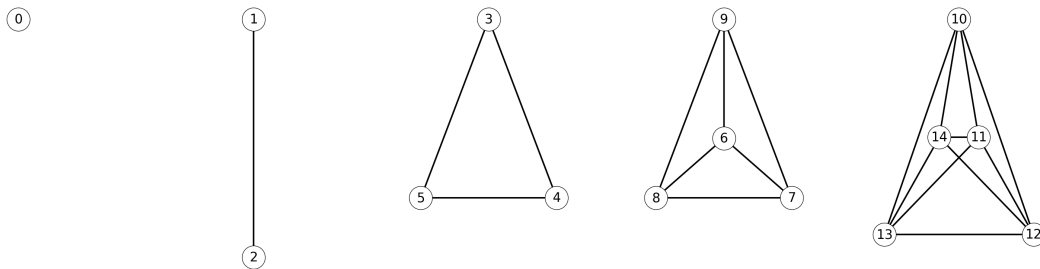
positions = {i: (center_pos[i] + shift[i]) for i in range(N)}

```

```

[6]: plt.figure(figsize=(30, 7), dpi=100)
nx.draw(g, pos=positions, **(BLACK|SIZES))
plt.savefig("Task 3.png")

```



### 0.3 Task 4

```

[7]: for component_nodes in nx.connected_components(g):
    subgraph = g.subgraph(component_nodes)
    eccentricities = nx.eccentricity(subgraph)
    n_nodes = subgraph.number_of_nodes()
    start_index = n_nodes * (n_nodes-1) // 2

    plt.figure(figsize=(3, 3), dpi=100)
    nx.draw(subgraph, pos=center_pos, **BLACK, node_size=400, font_size=12)
    plt.show()

    print("Count of nodes:", subgraph.number_of_nodes())
    print("Count of edges:", subgraph.number_of_edges())
    print("Radius:", nx.radius(subgraph))

```

```

print("Diameter", nx.diameter(subgraph))

print("Node", "Degree", "Eccentricity", sep="\t")
for node in component_nodes:
    print(node, end="\t")
    print(g.degree(node), end="\t")
    print(eccentricities[node])

print()
print()

```

①

```

Count of nodes: 1
Count of edges: 0
Radius: 0
Diameter 0
Node    Degree  Eccentricity
0       0       0

```



Count of nodes: 2

Count of edges: 1

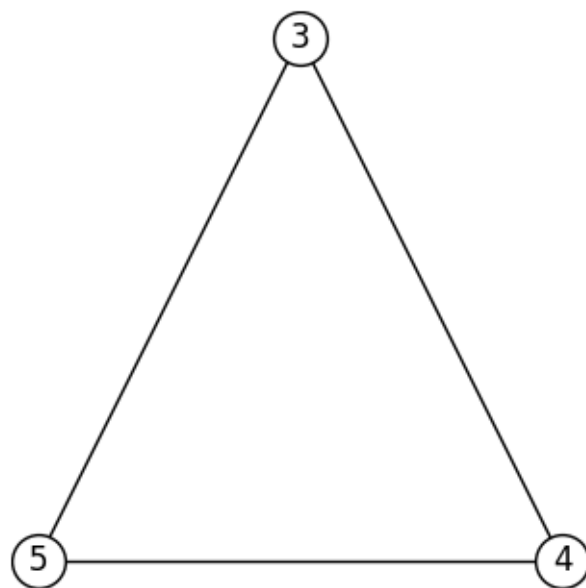
Radius: 1

Diameter 1

Node	Degree	Eccentricity
------	--------	--------------

1	1	1
---	---	---

2	1	1
---	---	---



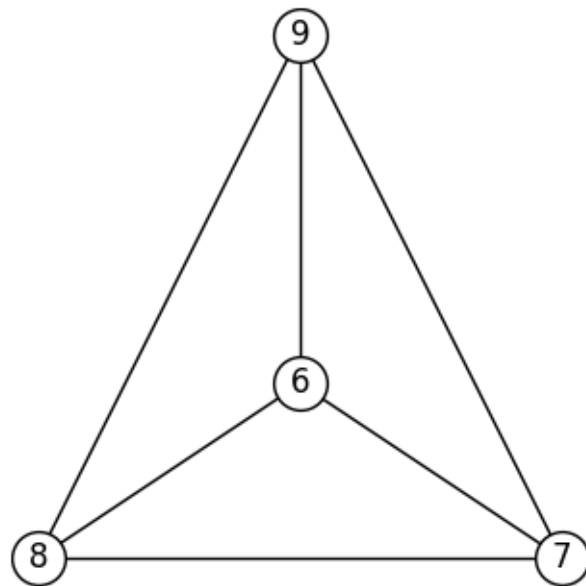
Count of nodes: 3

Count of edges: 3

Radius: 1

Diameter 1

Node	Degree	Eccentricity
3	2	1
4	2	1
5	2	1



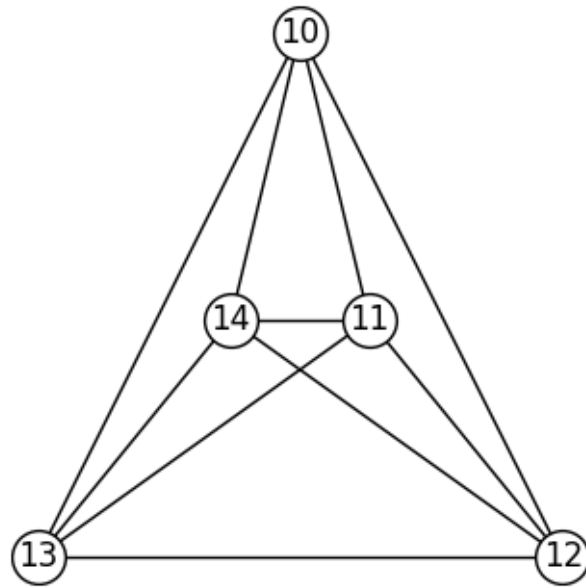
Count of nodes: 4

Count of edges: 6

Radius: 1

Diameter 1

Node	Degree	Eccentricity
8	3	1
9	3	1
6	3	1
7	3	1



Count of nodes: 5

Count of edges: 10

Radius: 1

Diameter 1

Node	Degree	Eccentricity
10	4	1
11	4	1
12	4	1
13	4	1
14	4	1

#### 0.4 Task 5

```

[8]: def bfs(graph, start_node):
      """
      Find paths from selected vertex to all other

      Parameters
      graph : networkx.Graph
          Connected graph in which the search will be performed.

      start_node : int
          Label of starting node

      Returns
  
```

```

    dict
    Dictionary with vertex labels as keys and chain of vertexes as path
    ↳ from start node to other
    """

    nodes = dict.fromkeys(graph.nodes, None)
    nodes[start_node] = [start_node]

    queue = Queue()
    visited = []

    queue.put(start_node)
    visited.append(start_node)

    while not queue.empty():
        curr_node = queue.get()
        for neighbor_node in graph.neighbors(curr_node):
            if neighbor_node not in visited:
                nodes[neighbor_node] = nodes[curr_node] + [neighbor_node]
                visited.append(neighbor_node)
                queue.put(neighbor_node)
    return nodes

def diameter(graph):
    """
    Find the diameter of connected graph

    Parameters
    graph : networkx.Graph
        Connected graph in which the search will be performed.

    Returns:
    list
        Chain of connected nodes.
    """
    eccentricities = nx.eccentricity(graph)
    start_node = max(eccentricities, key=lambda x: eccentricities.get(x))
    paths = bfs(graph, start_node)
    path_key = max(paths, key=lambda x: len(paths.get(x)))
    return paths[path_key]

def nodes_to_chain_edges(nodes):
    """
    Convert list of connected nodes to list of edges

    Parameters

```



```

nodes : list
    List of connected nodes

Returns
list
    List of edges
"""
edges = []
for i in range(len(nodes)-1):
    edges.append((nodes[i], nodes[i+1]))
return edges

```

```

[9]: diameter_nodes = []
diameter_edges = []

for component_nodes in nx.connected_components(g):
    subgraph = g.subgraph(component_nodes)
    current_diameter_nodes = diameter(subgraph)
    diameter_nodes.extend(current_diameter_nodes)
    diameter_edges.extend(nodes_to_chain_edges(current_diameter_nodes))

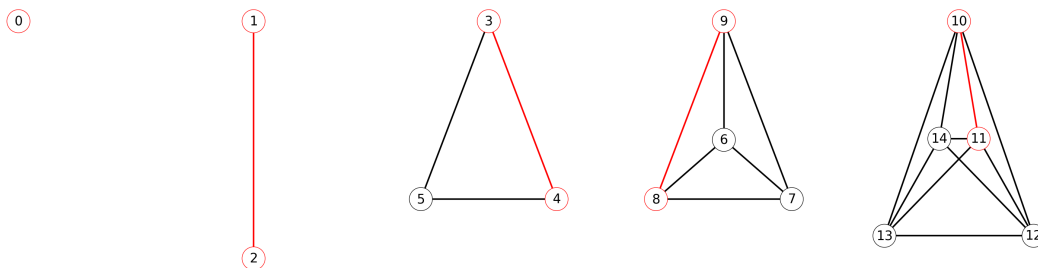
diam_border_colors = ["red" if node in diameter_nodes else "black" for node in g.nodes]
diam_edge_colors = ["red" if (u, v) in diameter_edges or (v, u) in diameter_edges else "black"
                    for u, v in g.edges]

```

```

[10]: plt.figure(figsize=(30, 7), dpi=100)
nx.draw(g, pos=positions, **(BASE_COLORS|SIZES),
        edgecolors=diam_border_colors, edge_color=diam_edge_colors)
plt.savefig("Task 5.png")

```



## 0.5 Task 6

```
[11]: tree_edges = set()

for component_nodes in nx.connected_components(g):
    subgraph = g.subgraph(component_nodes)
    current_tree_edges = nx.dfs_edges(subgraph)
    tree_edges.update(current_tree_edges)

tree_edge_colors = ["red" if (u, v) in tree_edges or (v, u) in tree_edges else
                    ↪ "black"
                    for u, v in g.edges]
```

```
[12]: plt.figure(figsize=(30, 7), dpi=100)
nx.draw(g, pos=positions, **(BASE_COLORS|SIZES),
        edgecolors="black", edge_color=tree_edge_colors)
plt.savefig("Task 6.png")
```

