Data visualization course

Laboratory work 5

Graph visualization

One of the common libraries for working with graphs is NetworkX. NetworkX is a Python package for creating, manipulating, and studying the structure, dynamics, and functions of complex graphs (networks).
In order to install this library, you need to execute the following command in the terminal:

```
pip install networkx
```

To create an empty graph with no nodes and no edges:

```
import networkx as nx
G = nx.Graph()
```

By definition, a graph is a collection of nodes (vertices) together with discovered pairs of nodes (called edges, links, etc.). In NetworkX, nodes can be any hashing object, such as a text string, an image, an XML object, another graph, a set of node objects, etc.

A graph can be created in several ways. In particular, NetworkX includes a large number of graph generator functions and tools for reading and creating graphs in various formats. To get started, consider simple manipulations with a graph, for example, you can create one:

```
G.add_node(1)
```

Or add new nodes:

```
G.add_nodes_from([2, 3])
```

A graph can also be created by adding one edge at a time:

```
G.add_edge ( 1 ,  2 )
e = ( 2 , 3 )
G.add_edge( * e )   unpack tuple with * operator
```

Or by adding a list of edges:

```
G . add_edges_from ([( 1 ,  2 ),  ( 1 ,  3 )])
```

NetworkX can add new nodes/edges and NetworkX silently ignores those already present.

```
G.add_edges_from ([( 1 ,  2 ),  ( 1 ,  3 )])
```

```
G.add_node ( 1 )
G.add_edge ( 1 ,  2 )
G.add_node ( "spam" )       # add node with string value
G.add_nodes_from ( "spam" ) # add 4 nodes with values: 's', 'p', 'a', 'm'
G.add_edge (3 , 'м' )
```

At this stage, the graph G consists of 8 nodes and 3 edges, as can be seen from:

```
G.number_of_nodes () # == 8
G.number_of_edges () # == 3
```

NetworkX has four main methods for working with graph properties: G.nodes, G.edges, G.adji, and G.degree. This set of methods allows viewing the nodes, edges, neighbors (adjacencies) and degrees of the nodes of a graph.

```
list(G.nodes)  # [ 'a', 1, 2, 3, 'spam', 'т', 'p', 'S']
list(G.edges)  # [(1, 2 ), (1, 3), (3, 't')]
list(G.adj[1]) # or list (G.neighbors (1)) # [2, 3]
G.degree[1]    # list of nodes falling at 1 # -- 2
```

More examples of using the library can be found in the tutorial https://networkx.github.io/documentation/stable/tutorial.html

As an example, it is proposed to consider the construction of a file system tree and its visualization using the NetworkX library (Listing 1). With the help of the specified library, in the example, tree visualization (Figure 1) was performed using various algorithms.

Listing 1 – An example of visualizing a directory tree

```
import networkx as nx
import matplotlib.pyplot as plt
import os
import subprocess
import sys


def get_tree(tree, G=nx.Graph(), itr=0, max_itr=900):
    point = tree.pop(0)
    itr += 1
    sub_tree = [
        os.path.join(point, x) for x in os.listdir(point)
```

```
        if os.path.isdir(os.path.join(point, x)) and not
is_hidden_dir(os.path.join(point, x))
    ]

    if sub_tree:
        tree.extend(sub_tree)
        G.add_edges_from((point, b) for b in sub_tree)

    if tree and itr <= max_itr:
        return get_tree(tree, G, itr)
    else:
        return G


def is_hidden_dir(d):
    if sys.platform.startswith("win"):
        p = subprocess.check_output(["attrib", d.encode('cp1251',
errors='replace')])
        p = p.decode('cp1251', errors='replace')  # decode the bytes to
str
        return 'H' in p[:12]
    else:
        return os.path.basename(d).startswith('.')


def main(root_dir: str):
    G = get_tree(tree=[root_dir])
    options = {
        'node_color': 'blue',
        'node_size': 30,
        'width': 0.5,
        'with_labels': True,
        'alpha': 0.6,
        'font_size': 3,
        'font_family': 'Arial'
    }
    plt.figure(figsize=(10, 10))

    plt.subplot(221)
    plt.title('kamada_kawai')
    nx.draw_kamada_kawai(G, **options)

    plt.subplot(222)
    plt.title('circular')
    nx.draw_circular(G, **options)

    plt.subplot(223)
    plt.title('spectral')
    nx.draw_spectral(G, **options)

    plt.subplot(224)
```

```
    plt.title('spring')
    nx.draw_spring(G, **options)

    plt.tight_layout()
    plt.savefig('graphs.png', dpi=600)
    plt.show()


if __name__ == "__main__":
    root_dir = r"D:\Docs"
    main(root_dir)
```

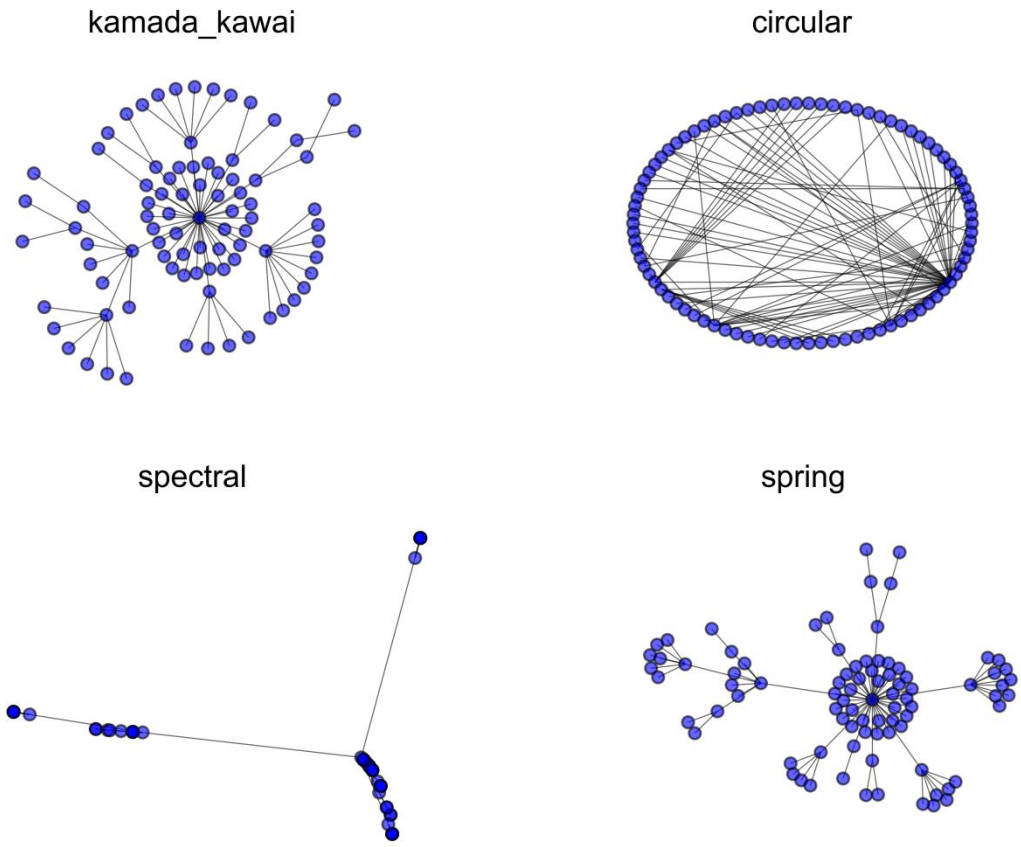

Figure 1 – The result of visualization of the file system structure by various algorithms

Task

Construct a graph including files and directories starting at a certain level (directory). Build its visualization, taking into account the feature according to the number in the student's list. Files and directories are visualized in different colors.
Features to realize (one per student):

1. The size of the node is proportional to the folder size;
2. The color of the node is related to the folder size;
3. The color of the node depends on the most common type of object (folder or some file extension) in folder;
4. The color of the node depends on the date the folder was created;
5. The size of the node depends on the number of folders and files in the directory;
6. The color of the node depends on the number of folders and files in the directory;
7. The color depends on the level (at one level - one color);
8. The size of the node depends on the level (the further, the smaller);

For options with color (2, 3, 4, 6, 7), add a legend to the plots.
For options with size (1, 5, 8), add text with the total size (round up where needed).