Візуалізація даних

**Лабораторна робота №4**

# Візуалізація графів

Однією з поширених бібліотек для роботи з графами є NetworkX. NetworkX – це пакет Python для створення, маніпулювання та вивчення структури, динаміки та функцій складних графів (мереж).

Для того, щоб встановити цю бібліотеку необхідно виконати у терміналі наступну команду:

pip install networkx

Щоб створити порожній графік без вузлів і без ребер:

import networkx as nx

G = nx.Graph()

За визначенням, a граф являє собою сукупність вузлів (вершин) разом з виявленими парами вузлів (називаються ребрами, посиланнями тощо). У NetworkX вузлами може бути будь-який об'єкт хешування, наприклад, текстовий рядок, зображення, об'єкт XML, інший граф, набір об'єктів вузлів тощо.

Граф може бути створений кількома способами. Зокрема NetworkX включає в себе велику кількість функцій генератора графів і засобів для читання та створення графів у різних форматах. Щоб розпочати роботу, розглянемо прості маніпуляції з графом, наприклад можна створити один вузол

G.add\_node(1)

або додати список вузлів

G.add\_nodes\_from([2, 3])

Граф також може бути створений шляхом додавання одного ребра за один раз

G.add\_edge ( 1 , 2 )

е = ( 2 , 3 )

G.add\_edge( \* e ) # розпакувати крайній кортеж \*

Або через додавання списку ребер

G . add\_edges\_from ([( 1 , 2 ), ( 1 , 3 )])

NetworkX можна додавати нові вузли / ребра і NetworkX тихо ігнорує ті, які вже присутні.

G.add\_edges\_from ([( 1 , 2 ), ( 1 , 3 )])

G.add\_node ( 1 )

G.add\_edge ( 1 , 2 )

G.add\_node ( "спам" ) # додає вузол "спам"

G.add\_nodes\_from ( "spam" ) # додає 4 вузла: 's', 'p', 'a', 'm'

G.add\_edge (3 , 'м' )

На цьому етапі граф G складається з 8 вузлів та 3 ребер, що видно з:

G.number\_of\_nodes () # == 8

G.number\_of\_edges () # == 3

У NetworkX є чотири основних методи для роботи з властивостями графів: G.nodes, G.edges, G.adjі та G.degree. Це набір методів дозволяє переглянути вузли, краї, сусіди (суміжності) та ступені вузлів графу.

list(G.nodes) # [ 'а', 1, 2, 3, 'спам', 'т', 'р', 'S']

list(G.edges) # [(1, 2 ), (1, 3), (3, 'т')]

list(G.adj[1]) # або список (G.neighbors (1)) # [2, 3]

G.degree[1] # кількість ребер, що падають на 1 # -- 2

Більше прикладів з використання бібліотеки можна знайти в туторіалі <https://networkx.github.io/documentation/stable/tutorial.html>

У якості прикладу пропонується розглянути побудову дерева файлової системи та її візуалізацію за допомогою бібліотеки NetworkX (лістинг 1). За допомогою вказаної бібліотеки у прикладі виконана візуалізація дерева (рисунок 1) різними алгоритмами.

Лістинг 1 – Приклад з візуалізації дерева каталогів

# -\*- coding: utf-8 -\*-

**import** networkx **as** nx

**import** matplotlib**.**pyplot **as** plt

**import** os

**import** matplotlib

**def** get\_tree**(**tree**=[**u"e:\\Music\\"**,** **],** G**=**nx**.**Graph**(),** itr**=**0**,** max\_itr**=**900**):**

point **=** tree**.**pop**(**0**)**

itr **=** 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**(**map**(lambda** b**:** **(**point**,** b**),** sub\_tree**),** lablel**=**sub\_tree**)**

**if** tree **and** itr **<=** max\_itr**:**

**return** get\_tree**(**tree**,** G**,** itr**)**

**else:**

**return** G

**def** is\_hidden\_dir**(**d**):**

**import** sys**,** subprocess

**if** sys**.**platform**.**startswith**(**"win"**):**

p **=** subprocess**.**check\_output**([**"attrib"**,** d**.**encode**(**'cp1251'**,** errors**=**'replace'**)** **if** isinstance**(**d**,** unicode**)** **else** d**])**

**return** **True** **if** 'H' **in** p**[:**12**]** **else** **False**

**else:**

**return** **True** **if** os**.**path**.**basename**(**d**)[**0**]** **==** '.' **else** **False**

**def** main**():**

G **=** get\_tree**()**

options **=** **{**

'node\_color'**:** 'blue'**,**

'node\_size'**:** 30**,**

'width'**:** 0.5**,**

'with\_labels'**:** **True,**

'alpha'**:** 0.6**,**

'font\_size'**:** 3,

'font\_family': 'Arial'

**}**

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\_\_"**:**

main**()**

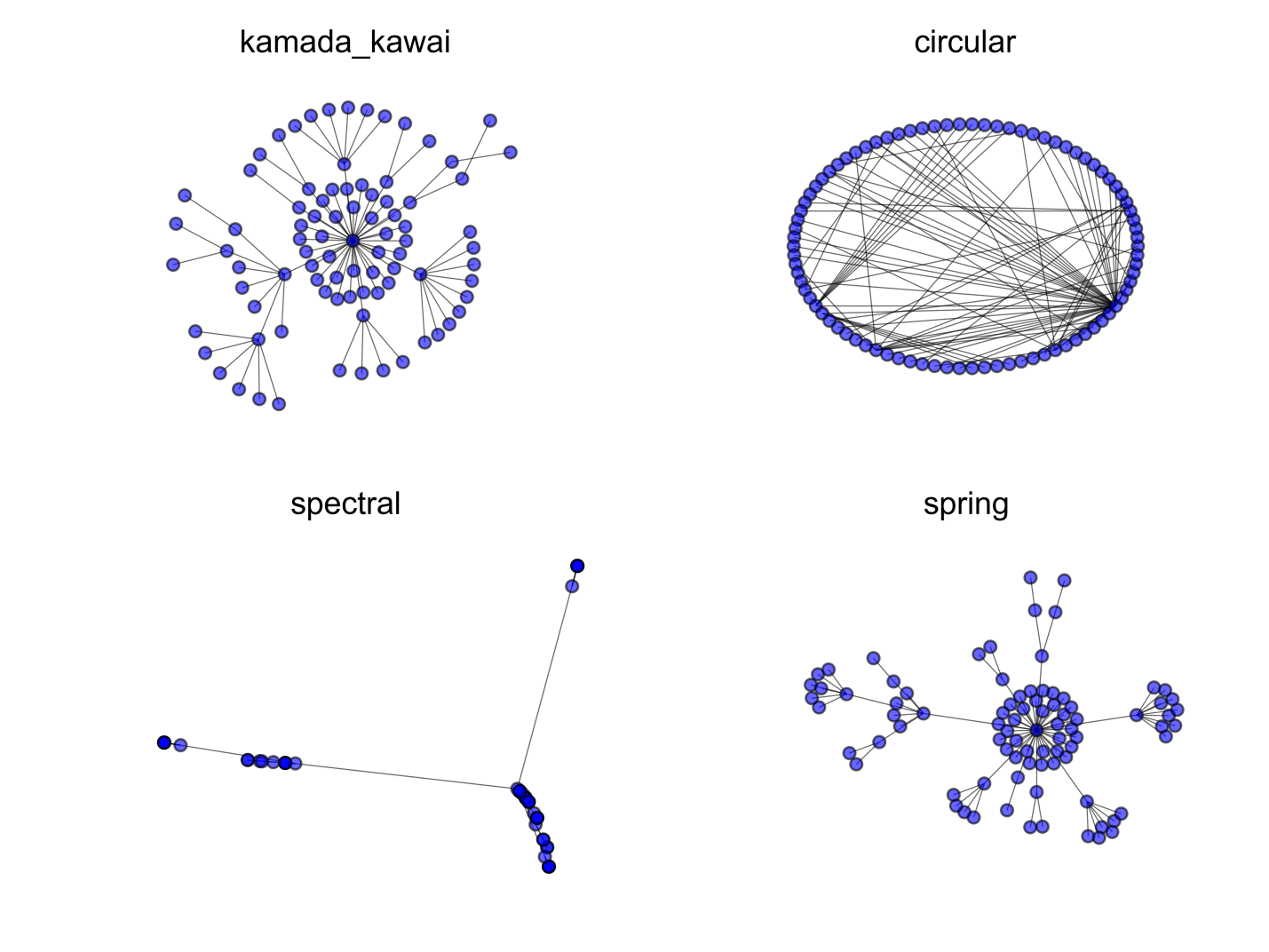


Рисунок 1 – Результат візуалізації структури файлової системи різними алгоритмами

# Варіанти завдань

Побудувати граф, що включає файли та каталоги, починаючи з певного рівня (каталогу). Побудувати його візуалізацію, враховуючи особливості згідно з варіантом. Файли та каталоги візуалізуються різним кольором.

**Варіант 1**

Розмір вершини є пропорційним до розміру файлу;

**Варіант 2**

Колір вершини пов’язаний з розміром файлу (від синього до червоного);

**Варіант 3**

Колір вершини залежить від типу файлу;

**Варіант 4**

Колір вершини залежить від дати створення файлу;

**Варіант 5**

Розмір вершини залежить від кількості файлів в каталозі;

**Варіант 6**

Колір вершини залежить від кількості файлів в каталозі;;

**Варіант 7**

Колір залежить від рівня (на одному рівні – один колір);

**Варіант 8**

Розмір вершини залежить від рівня (чим далі – тим менше);

**Варіант 9**

Розмір вершини залежить від типу файлу;