Лабораторна робота №10 Основи роботи з графами у NetworkX

Мета заняття

Лабораторна робота спрямована на формування базових умінь створювати, аналізувати та візуалізувати графові структури за допомогою бібліотеки **NetworkX**. Студент має набути навичок роботи з орієнтованими та неорієнтованими графами, додавання та вилучення вузлів і ребер, обчислення найважливіших метричних характеристик графів, а також побудови простих візуалізацій.

Завдання

1. Створення та базові операції з графами

– Побудувати неорієнтований граф із 7 вершинами.
– Додати ребра так, щоб граф мав хоча б одну компоненту зв’язності з циклом.
– Вивести множини вершин та ребер.
– Видалити одну вершину та проаналізувати, як зміниться структура графа.

2. Метричні характеристики графа

– Обчислити ступінь кожної вершини.
– Знайти діаметр та радіус графа (якщо граф незв’язний — виконати для найбільшої компоненти).
– Обчислити центральність: ступеневу, близькості та міжцентральність.
– Пояснити, які вершини виявилися найбільш “важливими” та чому.

3. Пошук шляхів

– Знайти найкоротший шлях між двома заданими вершинами.
– Визначити всі найкоротші шляхи між цими ж вершинами.
– Побудувати остове дерево за допомогою алгоритму BFS або DFS.

4. Орієнтований граф

– Створити орієнтований граф із 6–8 вершин.
– Додати до нього ребра так, щоб існували вершини з ненульовим вхідним та вихідним ступенями.
– Побудувати матрицю суміжності.
– Перевірити, чи існує цикл у цьому графі.

5. Візуалізація

– Побудувати візуалізацію неорієнтованого графа із застосуванням одного з доступних layout-алгоритмів (spring, circular, spectral).
– Побудувати окрему візуалізацію орієнтованого графа зі стрілками.

Теоретичні відомості

Поняття графа посідає провідне місце в аналізі складних систем, моделюванні мережевої структури та в алгоритмах пошуку зв'язності. У сучасних застосуваннях графи використовуються для моделювання соціальних мереж, транспортних маршрутів, комунікаційних каналів, структур баз даних, взаємодій білків, рекомендаційних систем та багатьох інших об'єктів. Бібліотека **NetworkX** забезпечує повнофункціональне середовище для роботи з такими структурами в мові Python, включаючи створення графів, редагування їхньої структури, обчислення метричних характеристик, реалізацію класичних алгоритмів теорії графів та побудову візуалізацій.

У найзагальнішому формулюванні граф визначається як впорядкована пара

$$G = (V, E),$$

де V — скінченна множина вершин, а $E \subseteq V \times V$ — множина ребер. Ребра можуть бути неорієнтованими або орієнтованими. У першому випадку граф називається неорієнтованим, і ребро (u, v) вважається тотожним ребру (v, u). У другому випадку граф є орієнтованим, що означає наявність напрямку переходу між вершинами.

Важливим різновидом графів є графи з вагами, коли кожному ребру ставиться у відповідність деяке невід'ємне число

$$w : E \to \mathbb{R}_{\geq 0},$$

що інтерпретується як відстань, вартість або пропускна здатність. Це дає змогу моделювати транспортні мережі, мережі потоків та засоби оптимального комбінаторного планування.

У теорії графів істотне місце займають структурні характеристики вершин і графа в цілому. Ступінь вершини у неорієнтованому графі визначається загальною кількістю ребер, інцидентних цієї вершині. В орієнтованих графах розглядають два окремі показники:

$$\deg^+(v) = |\{u : (v, u) \in E\}|, \quad \deg^-(v) = |\{u : (u, v) \in E\}|,$$

де $\deg^+$ характеризує вихідні, а $\deg^-$ – — вхідні зв'язки вершини. Ці значення надають початкову оцінку того, наскільки вершина активно впливає на інші та наскільки вона підпадає під їхній вплив.

Ключовим аспектом аналізу мереж є вивчення шляхів. Послідовність вершин

$$P = (v_0, v_1, \ldots, v_k)$$

називається шляхом, якщо кожна пара сусідніх вершин пов'язана ребром. Якщо граф ваговий, то довжина шляху обчислюється як сума ваг усіх ребер, що входять до цього шляху. Найкоротшим шляхом між вершинами s і t називається шлях, що має мінімальну довжину серед усіх можливих. Алгоритмічно він може бути визначений за допомогою BFS (для невагових графів), алгоритму Дейкстри (для графів з невід'ємними вагами), або алгоритму Беллмана—Форда (для графів із довільними вагами).

Метричні характеристики графа, такі як діаметр та радіус, дозволяють оцінити його «масштаб». Діаметр визначається як

$$\operatorname{diam}(G) = \max_{u,v \in V} d(u, v),$$

де $d(u, v)$ — довжина найкоротшого шляху між вершинами u і v. Радіус графа, навпаки, визначається як

$$\operatorname{rad}(G) = \min_{v \in V} \max_{u \in V} d(u, v),$$

тобто відображає позицію найбільш «центральної» вершини з погляду максимальних відстаней до інших.

Поширеним є аналіз центральності вершин. Ступенева центральність відображає відносну міру локального впливу, оскільки вона нормується за формулою

$$C_D(v) = \frac{\deg(v)}{|V| - 1}.$$

Центральність близькості ґрунтується на оберненому середньому значенні довжин найкоротших шляхів:

$$C_C(v) = \frac{|V| - 1}{\sum_{u \neq v} d(v, u)}.$$

Вона оцінює, наскільки ефективно вершина «досягає» інших. Міжцентральність відображає глобальну значущість вершини, оскільки вона враховує частку найкоротших шляхів, що проходять через цю вершину:

$$C_B(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}},$$

де σ_{st} — кількість найкоротших шляхів між s і t, а $\sigma_{st}(v)$ — кількість таких шляхів, що проходять через v. Високе значення міжцентральності вказує на важливе «посередницьке» положення вершини у структурі мережі.

Граф може складатися з декількох незалежних частин, які називаються компонентами зв’язності. У неорієнтованому графі компоненту зв’язності визначають як максимальну підмножину вершин, усередині якої існує шлях між будь-якими двома вершинами. В орієнтованих графах аналогами є сильні та слабкі компоненти зв’язності.

У прикладних задачах значну роль відіграє візуалізація графів. Найпоширеніший спосіб — геометричне відображення вершин у площині з подальшим з’єднанням ребрами. Механізм вибору координат вершин задається layout-алгоритмами, серед яких spring layout (модель сил пружин), spectral layout (використання власних векторів матриці Лапласа) та circular layout (розміщення вершин по колу). NetworkX пропонує прямий доступ до цих алгоритмів і дозволяє створювати інформаційно насичені візуалізації.

Таким чином, теорія графів формує багатий математичний фундамент, що забезпечує глибоке розуміння структурної організації складних мереж. Засоби бібліотеки NetworkX надають практичний інструментарій для ефективного застосування цих принципів у задачах аналізу даних, моделювання та оптимізації.

Приклади коду

Наведені фрагменти коду є орієнтовними та можуть бути використані як шаблон.

Створення та робота з графом

```
import networkx as nx

G = nx.Graph()
G.add_nodes_from([1, 2, 3, 4, 5, 6, 7])
G.add_edges_from([(1,2), (2,3), (3,1), (3,4), (4,5), (5,6)])

print("Вершини:", G.nodes())
print("Ребра:", G.edges())
```

Обчислення характеристик

```
degrees = dict(G.degree())
print("Ступені вершин:", degrees)

largest_cc = max(nx.connected_components(G), key=len)
H = G.subgraph(largest_cc)
print("Діаметр:", nx.diameter(H))
print("Радіус:", nx.radius(H))

print("Ступенева центральність:", nx.degree_centrality(G))
print("Центральність близькості:", nx.closeness_centrality(G))
print("Міжцентральність:", nx.betweenness_centrality(G))
```

Орієнтований граф

```
DG = nx.DiGraph()
DG.add_edges_from([(1,2), (2,3), (3,1), (3,4), (4,5)])

A = nx.adjacency_matrix(DG).todense()
print(A)
print("Чи є цикл:", nx.is_directed_acyclic_graph(DG))
```

Візуалізація

```
import matplotlib.pyplot as plt

plt.figure(figsize=(6,4))
nx.draw(G, with_labels=True, node_color='lightblue', node_size=800)
plt.show()
```

Хід роботи

1. Ініціалізувати робоче середовище Python, імпортувати бібліотеки `networkx` та `matplotlib`.
2. Створити неорієнтований граф, додати вершини та ребра, перевірити його базові властивості.
3. Обчислити ключові метричні характеристики та прокоментувати отримані результати.
4. Здійснити пошук найкоротших шляхів між вибраними вершинами та побудувати остове дерево.
5. Створити орієнтований граф, додати ребра, отримати матрицю суміжності, проаналізувати наявність циклів.
6. Побудувати візуалізацію обох графів у зручній формі.
7. Підготувати звіт, що містить графічні ілюстрації, код, отримані результати та аналітичні висновки.

Методичні поради щодо виконання роботи

Під час виконання лабораторної роботи рекомендується дотримуватися логічної послідовності операцій, звертаючи особливу увагу на коректність побудови графової структури та інтерпретацію отриманих характеристик. Доцільно починати роботу зі створення базових прикладів графів, що дозволяє краще зрозуміти відмінності між орієнтованими та неорієнтованими графами, а також між ваговими та неваговими

структурами. Будь-які помилки у визначенні множини вершин або ребер призводять до некоректності подальших обчислень, тому етап ініціалізації варто проводити уважно, використовуючи виведення структури на екран.

Слід враховувати, що ступені вершин та інші локальні характеристики є чутливими до зміни навіть одного ребра, тому перед обчисленням метрик варто переконатися в правильності моделі. Особливу увагу необхідно приділити роботі з компонентами зв'язності: у випадку незв'язного графа глобальні метрики, такі як діаметр чи радіус, мають сенс лише для його найбільшої компоненти. У NetworkX зручно використовувати функцію `connected_components`, яка дозволяє виділити підграф для аналізу.

Під час пошуку найкоротших шляхів варто пам'ятати про різницю між алгоритмами для вагових та невагових графів. У простих навчальних прикладах достатньо використовувати реалізації `nx.shortest_path` та `nx.shortest_path_length`, однак при переході до вагових структур потрібно передавати параметр ваги. Рекомендується експериментувати зі зміною ваг ребер, щоб дослідити вплив на найкоротші маршрути.

Орієнтовані графи потребують особливої уваги при розрахунку ступенів, оскільки значення вхідного та вихідного ступенів визначають напрям потоків у мережі. Перед перевіркою на наявність циклів необхідно пересвідчитись, що структура графа коректно відтворює відношення напрямків. Функція `nx.is_directed_acyclic_graph` дає можливість швидко встановити факт відсутності циклів у моделі.

При виконанні візуалізації важливо враховувати, що різні layout-алгоритми дають різне уявлення про структуру графа. Рекомендується порівняти декілька методів розташування вершин, зокрема spring layout та circular layout, щоб сформувати інтуїтивне розуміння переваг кожного підходу. Візуалізація має відображати ключові структурні властивості графа, тому корисно варіювати параметри розміру вузлів, товщини ребер та підписів вершин.

Для забезпечення повноти та якості звіту потрібно додати кожний етап роботи: код, текстові пояснення, інтерпретацію отриманих характеристик, візуалізації та аналітичні висновки. Особливо важливо обґрунтувати значення центральності окремих вершин, оскільки саме ці показники демонструють глибинне розуміння структури мережі. Завершальним етапом має стати критичний аналіз: чи є отриманий граф типовим, які зміни структури суттєво впливають на результати, та яким чином ці спостереження можуть бути корисні у реальних прикладних задачах.

Контрольні запитання

1. Яким чином формально визначається граф як математичний об'єкт? Які відмінності між орієнтованим і неорієнтованим графом?
2. Що таке ступінь вершини у неорієнтованому графі та як визначаються вхідний і вихідний ступені у орієнтованому графі?
3. Які відмінності між шляхом, циклом та компонентами зв'язності? Як їх можна обчислити або знайти за допомогою NetworkX?

4. Якою є математична інтерпретація найкоротшого шляху між двома вершинами та які алгоритми його визначають? Як NetworkX реалізує пошук найкоротших шляхів?
5. Як визначаються діаметр та радіус графа? Чому їх обчислення має сенс лише для зв'язних графів?
6. Що таке ступенева центральність, центральність близькості та міжцентральність? Яку структурну інформацію вони відображають?
7. Чим відрізняються слабкі та сильні компоненти зв'язності в орієнтованих графах? Які функції NetworkX використовуються для їх пошуку?
8. Які переваги та недоліки мають різні layout-алгоритми (spring, circular, spectral)? Яке їхнє математичне або інтуїтивне підґрунтя?
9. Які структури даних використовуються NetworkX для представлення графів (списки суміжності, матриці суміжності тощо)? У яких випадках доцільно використовувати той чи інший формат?
10. Яким чином у NetworkX створюються вагові графи? Як передавати вагу ребра у функції пошуку найкоротших шляхів?

Ресурси для поглибленого вивчення теми

1. **NetworkX — Documentation**. https://networkx.org/documentation/stable/ (accessed 26.11.2025).
2. **NetworkX GitHub Repository**. https://github.com/networkx/networkx (accessed 26.11.2025).
3. **Hagberg A., Schult D., Swart P. — Exploring network structure, dynamics, and function using NetworkX**. https://conference.scipy.org/proceedings/SciPy2008/paper_2/ (accessed 26.11.2025).
4. **Python Graph Gallery — Network charts**. https://python-graph-gallery.com/network-chart/ (accessed 26.11.2025).
5. **GeeksforGeeks — NetworkX Tutorials**. https://www.geeksforgeeks.org/python-graph-viz-using-networkx/ (accessed 26.11.2025).
6. **Towards Data Science — Network Analysis with Python and NetworkX**. https://towardsdatascience.com/network-analysis-with-python-ffec5fdf7a (accessed 26.11.2025).
7. **MIT OpenCourseWare — Graph Theory Lectures**. https://ocw.mit.edu/courses/18-217-graph-theory-fall-2019/ (accessed 26.11.2025).
8. **Coursera — Algorithms on Graphs**. https://www.coursera.org/learn/algorithms-on-graphs (accessed 26.11.2025).