Звіт до лабораторної роботи

Ярослав Грунда
Фі-21, ФТІ КПІ

22 вересня 2024 р.

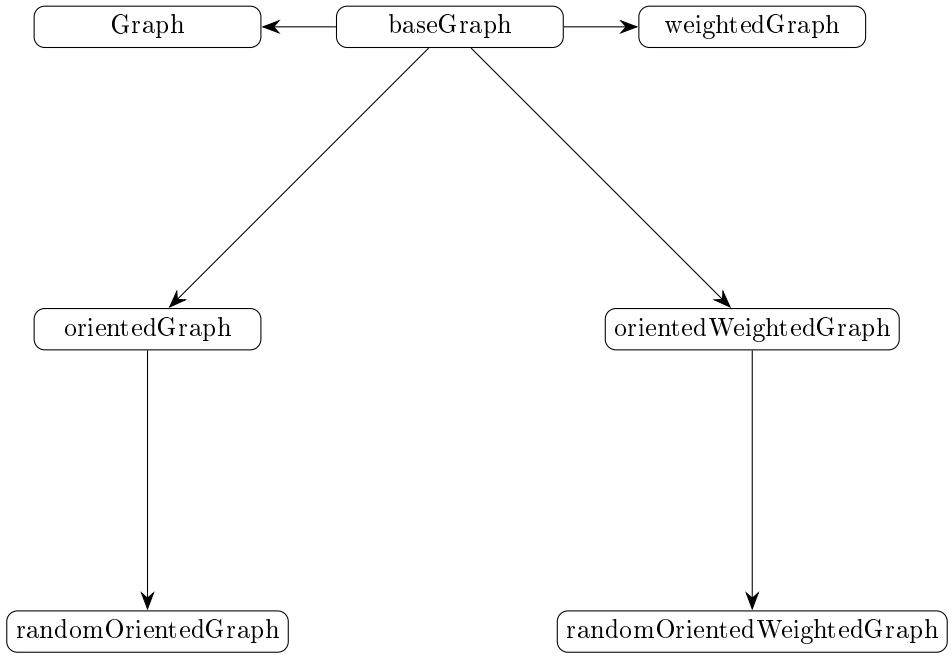
Зміст

1 Мета роботи

Опанувати способи реалізації представлення рызних типів графів (неорієнтовані, орієнтовані, зважені) та їх ефективної реалізації.

2 Опис класів

2.1 Структура коду



2.2 Клас `baseGraph`

Цей клас є базовим для всіх графів. Він ініціалізує порожній граф з заданою кількістю вершин і містить методи для додавання та видалення вершин, а також для перетворення графа між списком і матрицею суміжності.

- `def __init__(self, n):` - Ініціалізація порожнього графа з n вершинами.
- `def add_vertex(self):` - Додає нову вершину до графа.
- `def del_vertex(self, v):` - Видаляє вершину v і всі пов'язані з нею ребра.
- `def to_adjacency_matrix(self):` - Перетворює граф зі списку суміжності у матрицю суміжності.
- `def to_adjacency_list(self, matrix, weighted=False):` - Перетворює граф з матриці суміжності у список суміжності.
- `def show(self):` - Виводить граф у вигляді списку суміжності.
- `def show_matrix(self):` - Виводить граф у вигляді матриці суміжності.
- `def makeNonOriented(self):` - Робить граф неорієнтованим.

Швидкості функцій:

- `add_vertex()`: $O(1)$
- `del_vertex(v)`: $O(V + E)$, де V — кількість вершин, E — кількість ребер.

- `to_adjacency_matrix()`: $O(V^2)$
- `to_adjacency_list(matrix, weighted)`: $O(V^2)$
- `show()`: $O(V)$
- `show_matrix()`: $O(V^2)$

2.3 Клас `Graph`

Цей клас реалізує неорієнтований граф. Він має методи для додавання та видалення ребер.

- `def add_edge(self, u, v):` - Додає ребро між вершинами u і v.
- `def del_edge(self, u, v):` - Видаляє ребро між вершинами u і v.

Швидкості функцій:

- `add_edge(u, v)`: $O(V)$
- `del_edge(u, v)`: $O(V)$

2.4 Клас `weightedGraph`

Цей клас реалізує зважений неорієнтований граф.

- `def add_edge(self, u, v, w):` - Додає зважене ребро між вершинами u і v з вагою w.
- `def del_edge(self, u, v):` - Видаляє зважене ребро між вершинами u і v.

Швидкості функцій:

- `add_edge(u, v, w)`: $O(V)$
- `del_edge(u, v)`: $O(V)$

2.5 Клас `orientedGraph`

Цей клас реалізує орієнтований граф.

- **class orientedGraph(baseGraph):**
  - `def add_edge(self, u, v):` - Додає орієнтоване ребро між вершинами u і v.
  - `def del_edge(self, u, v):` - Видаляє орієнтоване ребро між вершинами u і v.

Швидкості функцій:

- `add_edge(u, v)`: $O(V)$
- `del_edge(u, v)`: $O(V)$

2.6 Клас `orientedWeightedGraph`

Цей клас реалізує зважений орієнтований граф.

- `def add_edge(self, u, v, w):` - Додає орієнтоване зважене ребро (u –w–> v) до графа.
- `def del_edge(self, u, v):` - Видаляє орієнтоване зважене ребро (u –w–> v) з графа.

Швидкості функцій:

- `add_edge(u, v, w)`: $O(V)$
- `del_edge(u, v)`: $O(V)$

2.7 Клас `randomOrientedGraph`

Цей клас реалізує випадковий орієнтований граф у моделі Ердеша-Шеньї.

- `def __init__(self, n, p):` - Ініціалізує випадковий орієнтований граф з n вершинами і ймовірністю p для створення ребра.
- `def generate_random_graph(self, p):` - Генерує випадковий орієнтований граф за імовірністю p для кожної пари вершин.

Швидкості функцій:

- `generate_random_graph(p)`: $O(V^2)$

2.8 Клас `randomOrientedWeightedGraph`

Цей клас реалізує випадковий орієнтований граф у моделі Ердеша-Шеньї.

- `def __init__(self, n, p, min_weight=1, max_weight=10):` - Ініціалізує випадковий зважений граф з n вершинами, ймовірністю p для створення ребра, і вагою між min_weight та max_weight.
- `def generate_random_graph(self, p, min_weight, max_weight):` - Генерує випадковий зважений граф за ймовірністю p для кожної пари вершин з випадковою вагою.

Швидкості функцій:

- `generate_random_graph(p, min_weight, max_weight)`: $O(V^2)$

3 Тестування швидкості

3.1 Результати ініціювання графа без ребер

Рис. 1: Бачимо лінійну складність, що пояснюється тим що створення кожної вершини і порожнього списку сусідів має постійну складність $O(1)$, оскільки таких ітерацій n, загальна складність цієї операції — $O(degV)$.

3.2 Результати додавання та видалення (видалення для орієнтованого графа) вершин

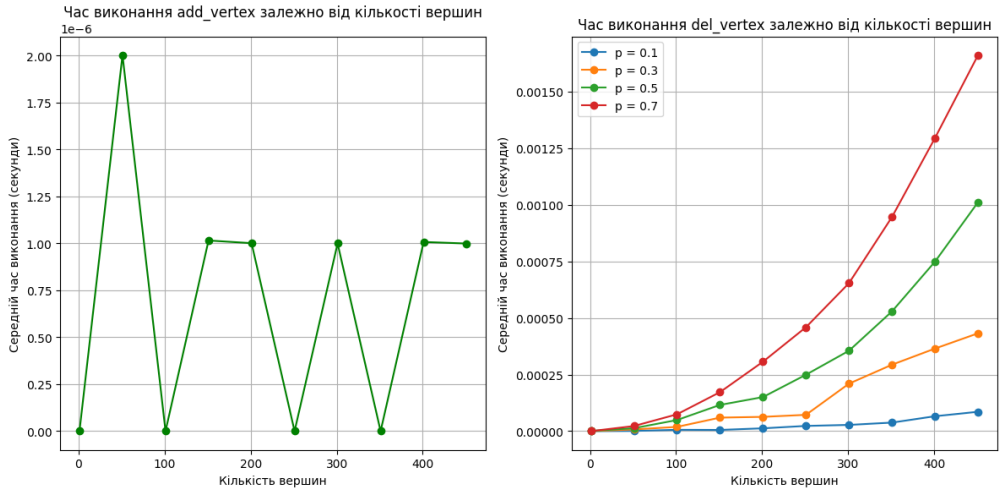


Рис. 2: Для додавання очевидно складність $O(1)$, оскільки це просто створення нового пустого списку. Для видалення бачимо ріст при збільшенні кількості вершин і залежність від щільності графа, тому $O(degV)$

3.3 Результати додавання та видалення ребра (для орієнтованого графа)

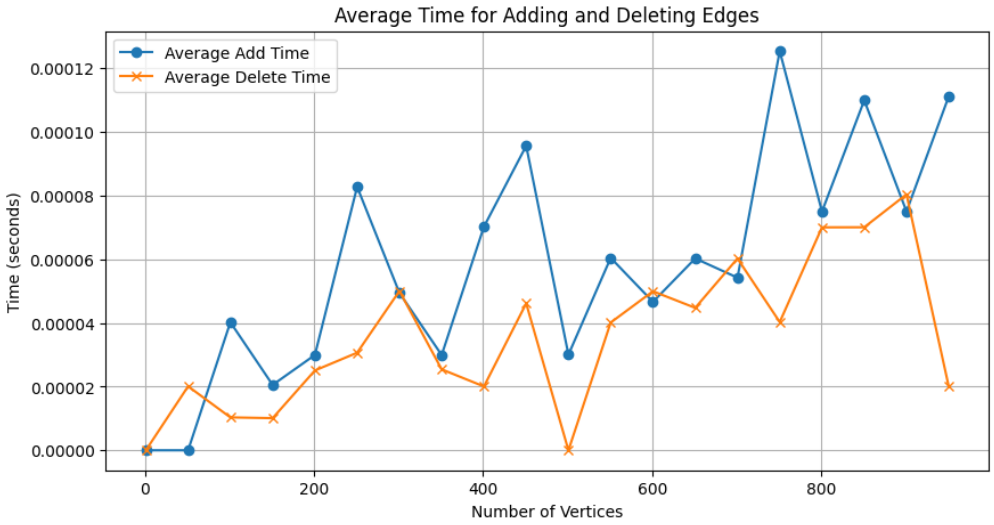


Рис. 3: Бачимо ріст при збільшенні вершин, але оскільки додавання і видалення залежать від щільності графа, тобто $O(deg_V)$
, бо потрібно перевірити чи існує ребро перед тим як додати чи прибрати його, тому значення доволі не лінійні.

3.4 Результати генерування випадкових орієнтованих графів

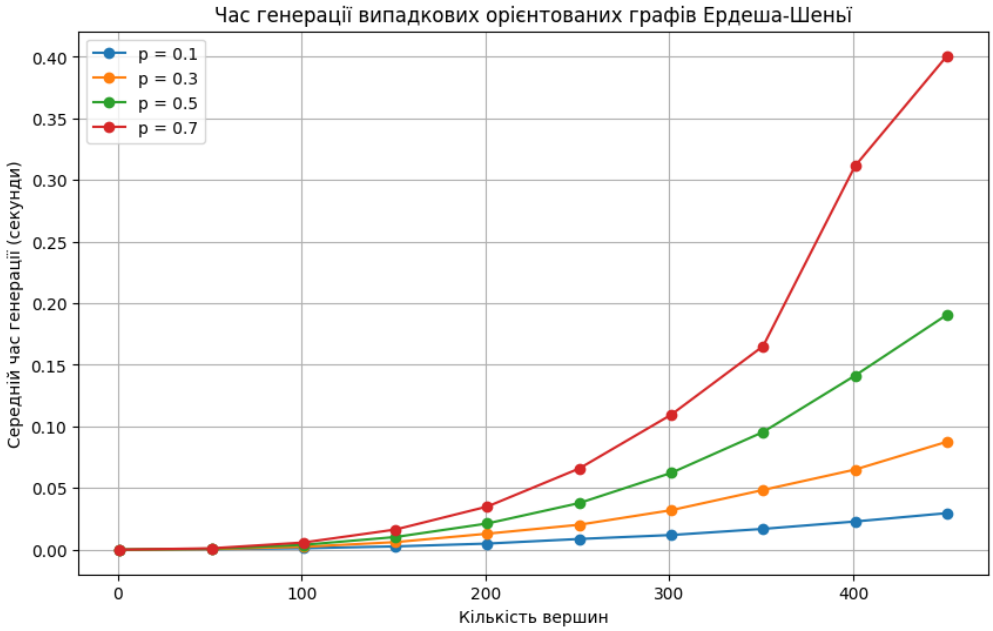


Рис. 4: Бачимо для щільного графа $O(V^2)$, для розрідженого схоже на лінійну складність $O(V)$

4 Висновки

У процесі виконання лабораторної роботи ми дослідили різні способи представлення графів, включаючи неорієнтовані, орієнтовані та зважені графи. Реалізація класів дозволила зрозуміти структуру та ефективність різних операцій, таких як додавання та видалення вершин і ребер.

Аналіз швидкостей функцій показав, що:

- Додавання нових вершин має постійну складність $O(1)$.
- Видалення вершин і ребер залежить від кількості сусідів, що веде до складності $O(deg\, V)$.
- Генерація випадкових графів демонструє квадратичну складність $O(V^2)$ для щільних графів, тоді як для розріджених графів складність наближається до лінійної.

Для більш детальної інформації, будь ласка, відвідайте наступний ресурс: Перейти до репозиторію.