Прикладні алгоритми. Завдання 2, звіт

Михайло Голуб

20 вересня 2024 р.

Реалізація класу граф:

Створено клас *Graph* що при ініціалізації приймає на вхід матрицю суміжності ребер. Цей клас має наступні методи:

to_ll – повертає масив списків зв'язності;

clear – встановлює усі клітинки матриці в False;

from_ll – створює нову матрицю зв'язності з масиву списків зв'язності;

add_vertice – додає одну колонку та один рядок в матрицю суміжності;

add_edge – встановлює відповідну пару клітинок в True;

remove_vertice – видаляє колонку та рядок з вказаним індексом;

remove_edge – встановлює відповідну пару клітинок в False.

Реалізація інших класів:

Клас *OrientedGraph* є нащадком (не дитиною; від англ. inherit – наслідувати / успадковувати) *Graph* з перевизначенням додання та видалення ребер: значення встановлюється не для пари клітинок, а для однієї клітинки.

Клас *WeightedGraph* є нащадком *Graph* з перевизначенням методів: методи працюють не з булевими значеннями, а з числовими.

Клас *OrientedWeightedGraph* є нащадком *WeightedGraph* з перевизначенням додання та видалення ребер: значення встановлюється не для пари клітинок, а для однієї клітинки.

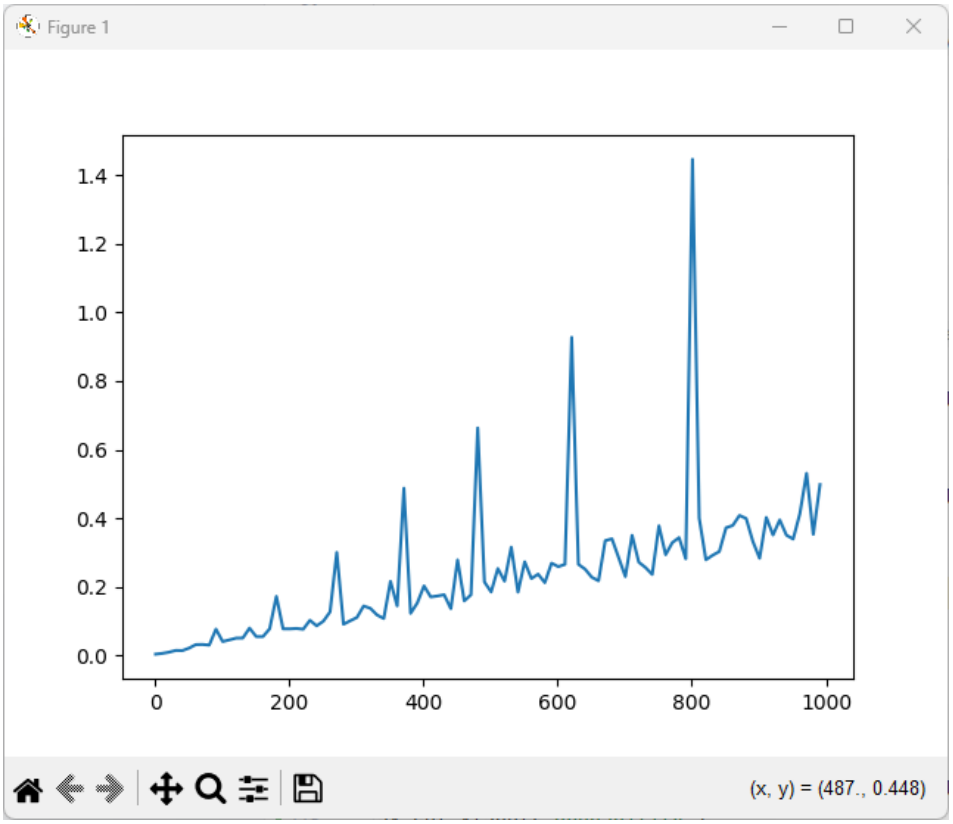
Реалізація рандомізованих класів:

Класи *RandomGraph, RandomOrientedGraph, RandomWeightedGraph, RandomOrienedWeightedGraph* є нащадками відповідних класів з перевизначенням *__init__*: створюється пуста (заповнена False) матриця суміжності потрібного розміру, і через неї проходить цикл що з вказаною ймовірністю записує True або випадкове число. Для того щоб в неорієнтованих графах кожне ребро було пройдено циклом лише один раз, ітератор другої координати працює від i до n, де i – позиція ітератора першої координати та n – кількість вершин. В орієнтованих графах обидва ітератори працюють від 0 до n

Тестування швидкості роботи:

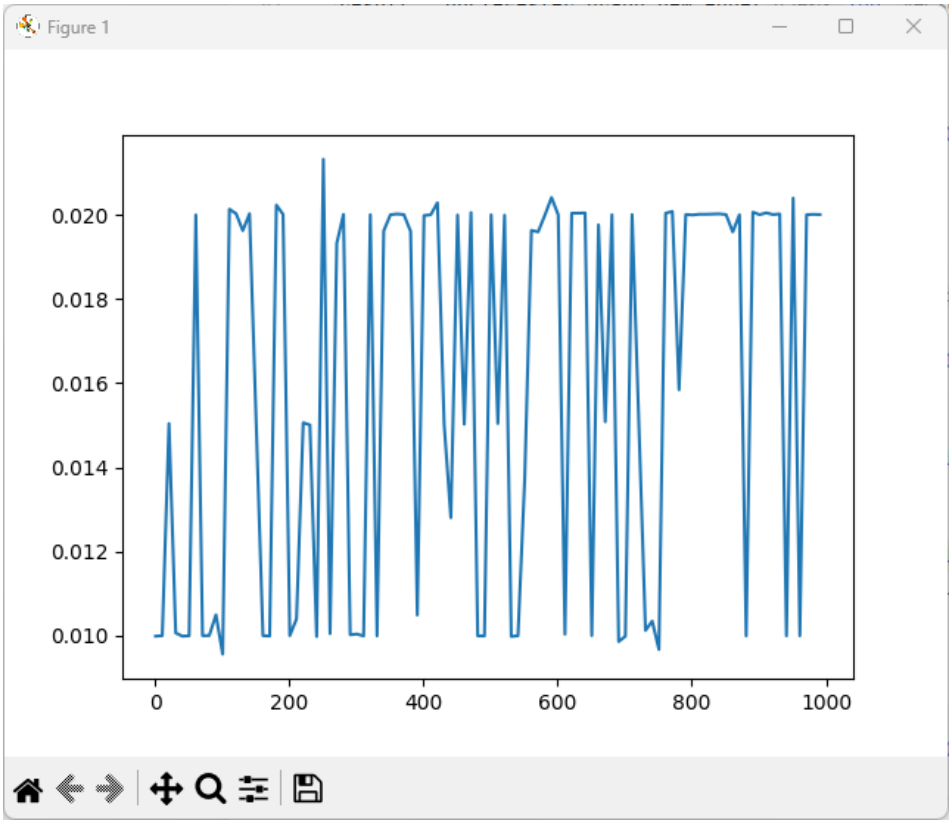
Дядько Фестер (Аддамс), представник класу *SkilledTester*, проводить тестування наступним чином: генерує вказану кількість випадкових графів, виконує на ній потрібну операцію і записує час роботи. Тестуються виключно неорієнтовані незважені графи, оскільки різниця складності роботи методів знехтовно мала. На графіках вказано час роботи в мс в залежності від кількості вершин

Результати тестування *add_vertice*:



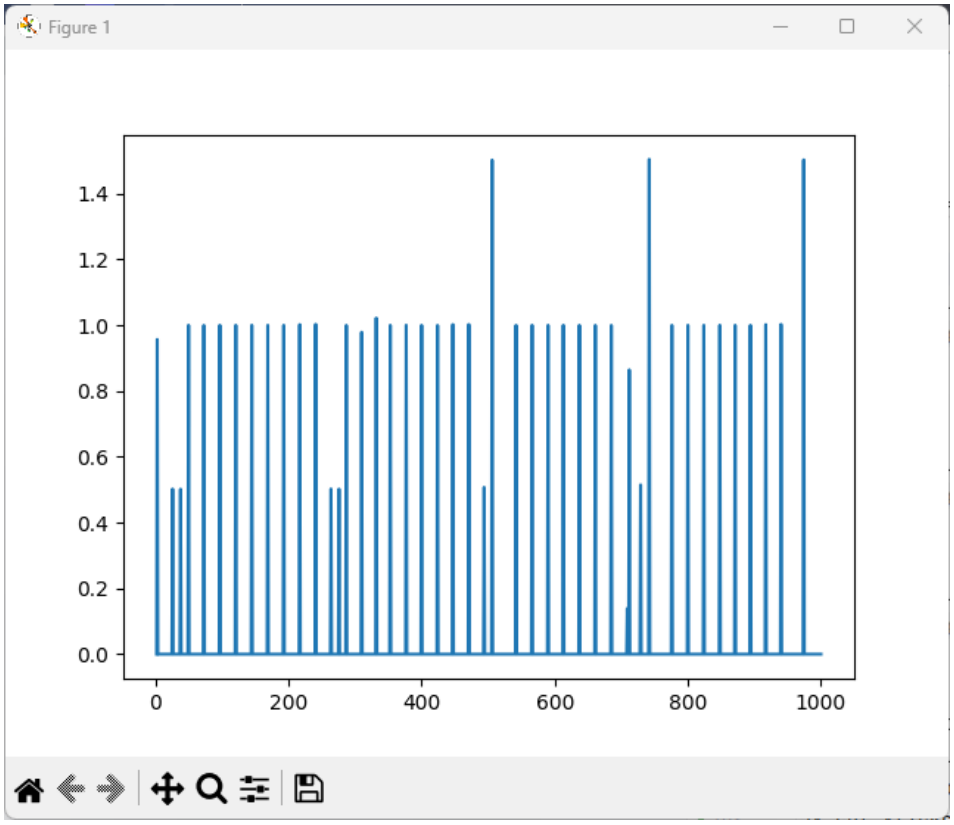
Явно видно, що складність методу співпадає з очікуваною складністю O(n). Проте, видно значні відхилення від прямої, скоріше за все це пов'язано з випадковою природою графів, що тестуються.

Результати тестування *add_edge*:



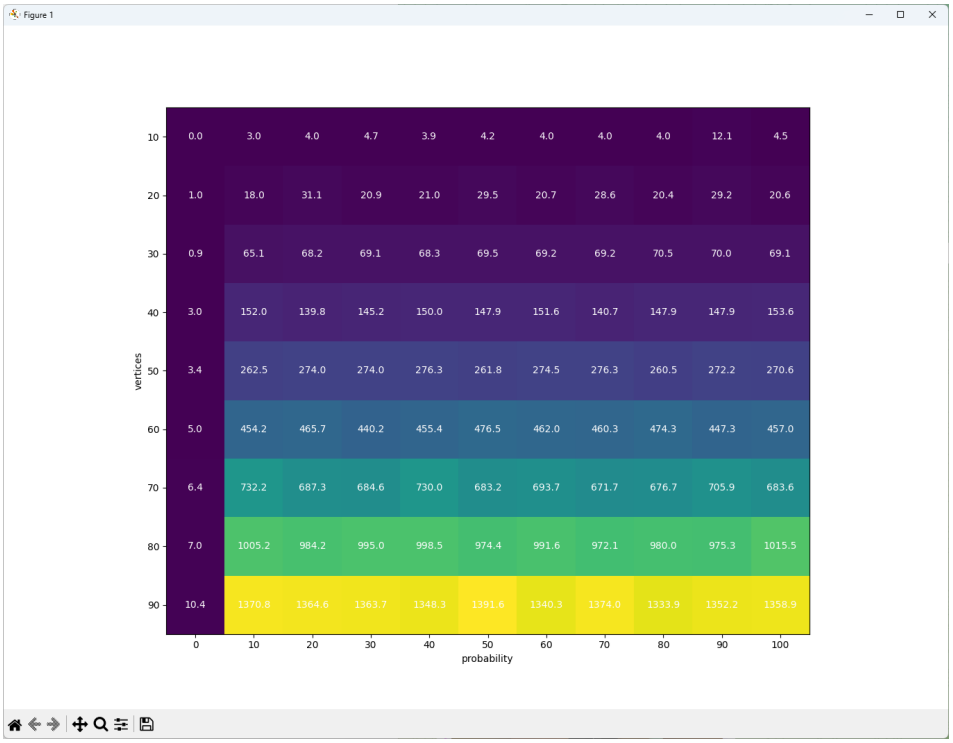
Цей метод виконується за час від 90мкс до 30мкс, що співпадає з очікуваною складністю O(1). Видалення ребер працює так само, лише записує в клітинки False замість True, тож час роботи додання та видалення ребра однаковий.

Результати тестування *rem_ vertice*:



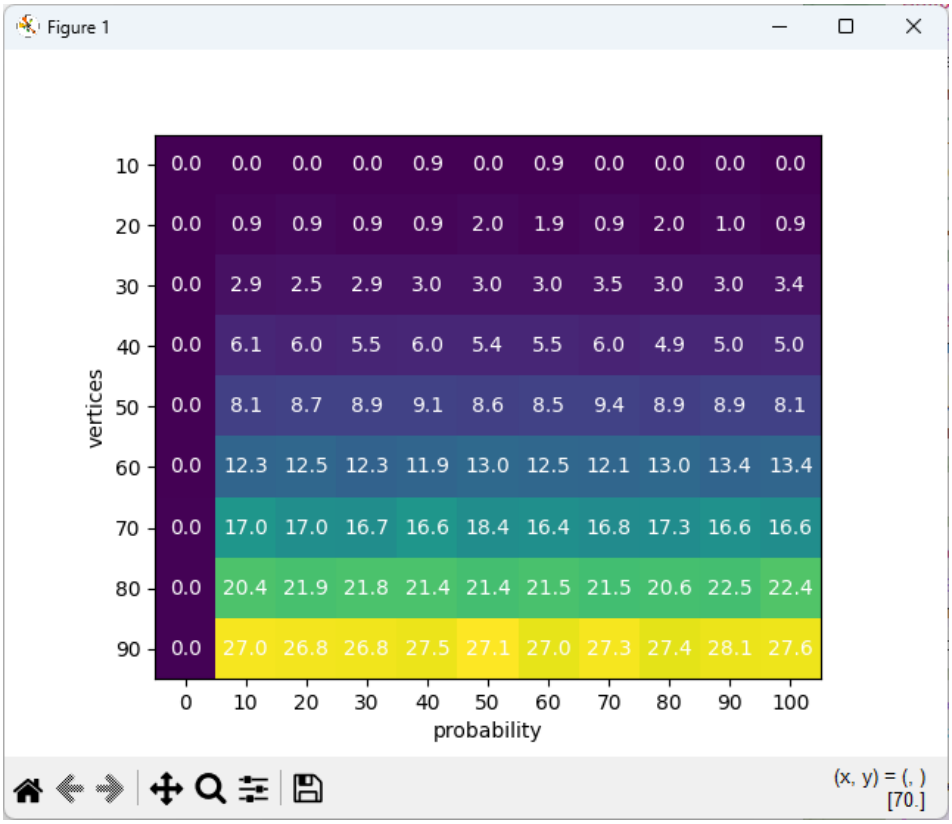
Цей метод виконується моментально, або за час менший 2мс, отримана складність – O(1). Очікувана складність O(n^2) не співпадає з отриманими даними, оскільки Python може видалити рядок та стовпець матриці суміжності не перестворюючи її.

Результати тестування *to_ll*:



Явно видно, що за відсутності ребер складність O(n). Час роботи залежить від кількості вершин значно більше ніж від кількості наявних ребер: можна знехтувати кількістю ребер та сказати що складність роботи залежить виключно від кількості вершин. Складність перетворення матричної форми. Отримана складність – O(n^2), очікувана складність залежить виключно від кількості ребер. При використанні об'єктів list, замість списків, для списків суміжності можна було б покращити час роботи.

Результати тестування *from_ll*:



Явно видно, що за відсутності ребер складність O(1). При наявності ребер складність O(n^2). Очікувана складність залежить від кількості ребер.

Додаткові тестування перетворень з ймовірністю ребер від 0 до 10 показують, що час роботи не залежить від ймовірності якщо вона більше 5% і час роботи наближається до нуля якщо ймовірність наближається до 0%. Додаткові тестування з більшою кількістю вершин показують, що час роботи не залежить від ймовірності, якщо ймовірність більше або рівна 1%, на графах з більш ніж 400 вершинами. Можна зробити висновок, що складність роботи перетворень матриць суміжностей на списки, та навпаки, має середню складність O(n^2) та не залежить від кількості ребер.

Viva Python та його "швидкодія"яка перетворює адекватні алгоритми на алгоритми з експоненційною складністю.

Посилання на репозиторій практикуму:
https : *//github.com/MINIAProgramStudio/applied_algorythms/tree/main/task_2*