Звіт до лабораторної роботи

Ярослав Грунда
Фі-21, ФТІ КПІ

6 жовтня 2024 р.

Зміст

1 Мета роботи

Метою даної роботи є реалізація структури даних UnionFind за допомогою дерев та застосування її для алгоритму Краскала. Алгоритм Краскала використовується для побудови мінімального кістякового дерева неорієнтованого зваженого графа. У звіті розглянуто особливості реалізації алгоритму, а також проаналізовано його експериментальну продуктивність.

2 Реалізація UnionFind

- **Клас Node**: кожен екземпляр має `value`, `parent` і `size`.
- **Ініціалізація**: створюється масив з `n` кількістю об'єктів класу Node. Кожен Node має своє значення (`value` від 1 до `n` відповідно до порядку створення) і є своїм власним батьком (є коренем дерева), розмір дерева при цьому дорівнює 1.
- **Операція Find**: виконує пошук кореня дерева для певного вузла (поки батько Node не є він самий, шукаємо батька Node). Під час пошуку виконується компресія шляху: всі вузли, які знаходяться на шляху від початкового вузла до кореня, безпосередньо прикріпляються до кореня, тим самим зменшуючи глибину дерева для наступних пошуків. Це значно пришвидшує подальші операції пошуку.
- **Операція Union**: виконує об'єднання двох дерев, корені яких визначаються за допомогою операції `find`. Для оптимізації операції об'єднання застосовується **правило зваженого об'єднання**: менше дерево приєднується до більшого. Це допомагає мінімізувати глибину дерев і зменшує час на виконання подальших операцій пошуку.

Операції в UnionFind мають такі складності:

- Операція `find` — $O(\log n)$;
- Операція `union` — $O(\log n)$.

Алгоритм Крускала будує мінімальне кістякове дерево за допомогою таких кроків:

1. Сортуємо ребра графа за вагою.
2. Ініціалізуємо структуру Union-Find для n вершин. Кожна вершина є коренем одного дерева.
3. Ітеруємо через всі ребра в порядку зростання ваги.
4. Перевіряємо, чи вершини u і v знаходяться у різних компонентах (не є з'єднаними) за допомогою методу find класу UnionFind (output якого є коренем дерева, тому якщо корені однакові — це одна компонента). Якщо так, то:
   (а) Додаємо це ребро до мінімального кістякового дерева.
   (б) Додаємо вагу цього ребра до загальної вартості MST.
5. Завершуємо роботу, якщо кількість компонентів дорівнює 1.

3 Теоретична оцінка продуктивності алгоритму Крускала

Теоретична складність алгоритму складається з:

- Сортування ребер: $O(E \log E)$;
- Виконання операцій об'єднання і пошуку без компресії шляху: $O(V \log V)$;
- Виконання операцій об'єднання і пошуку з компресією шляху: $O(V \cdot G(V))$, де $G(V)$ — функція Акермана, яка для практичних значень є константою.

Можемо вважати складність виконання алгоритму без сортування — $O(V)$.

4 Приклад виконання алгоритму Крускала

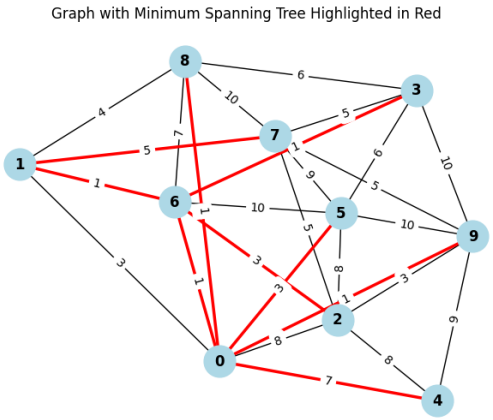

Рис. 1: Кількість вершин = 10, щільність = 0.7

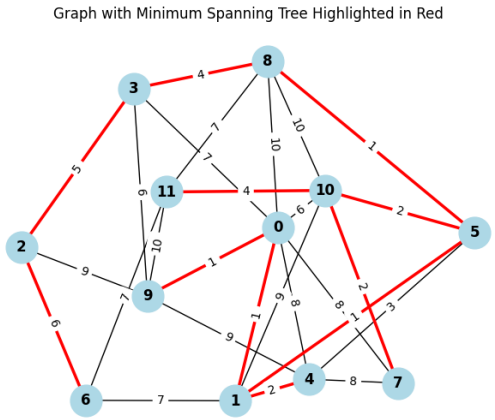

Рис. 2: Кількість вершин = 12, щільність = 0.4

5 Експериментальні дані алгоритму Крускала

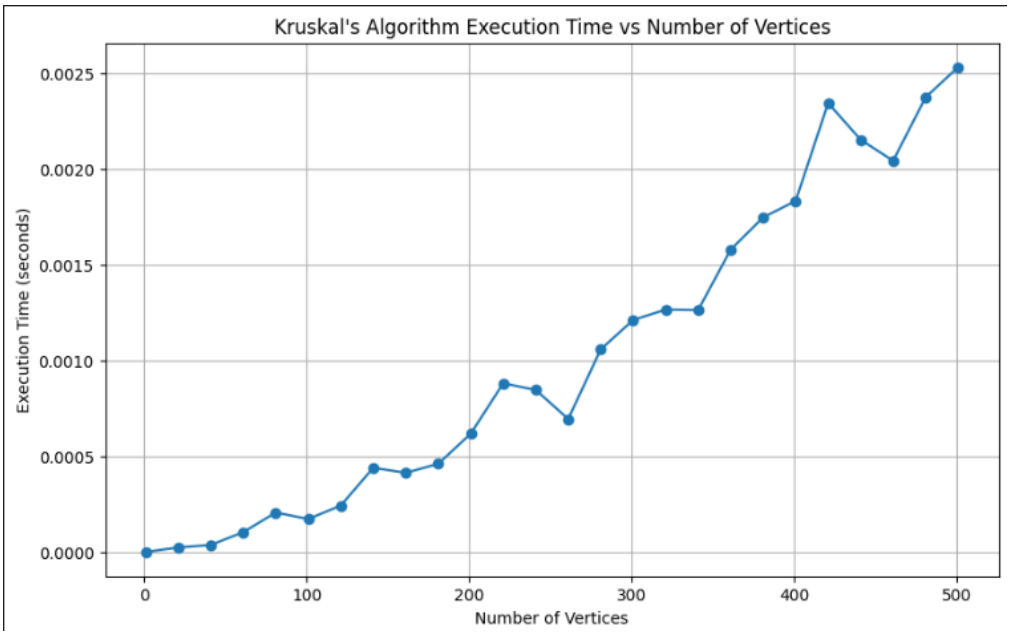

Рис. 3: На рисунку з кроком 20 вершин показано залежність часу виконання алгоритму Крускала (з компресією шляху) від кількості вершин графа при щільності 0.7. На кожну кількість вершин взято середній час 1000 експериментів.

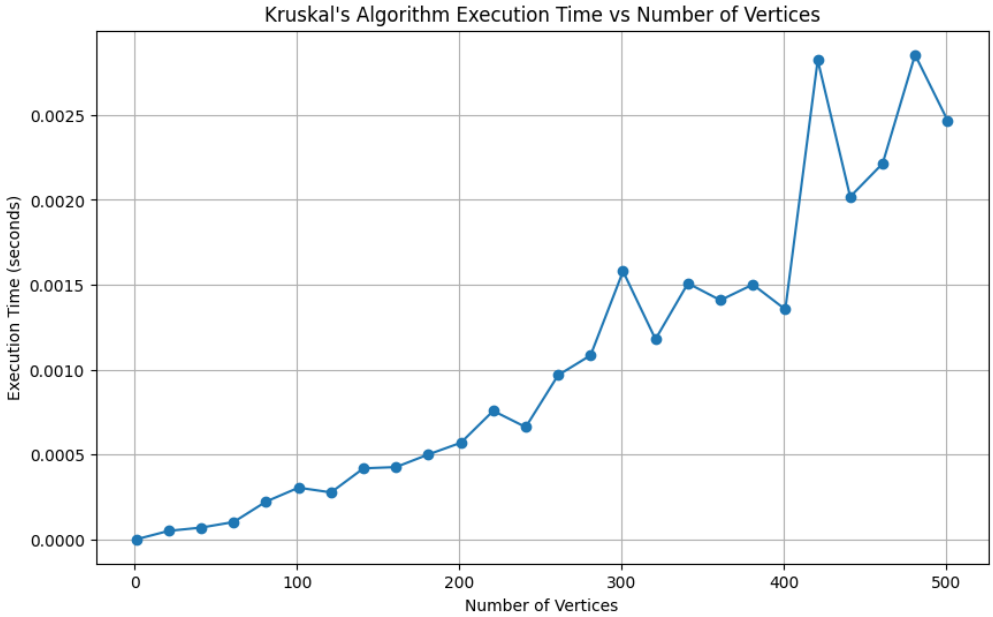

Рис. 4: На рисунку з кроком 20 вершин показано залежність часу виконання алгоритму Крускала (з компресією шляху) від кількості вершин графа при щільності 0.5. На кожну кількість вершин взято середній час 1000 експериментів.

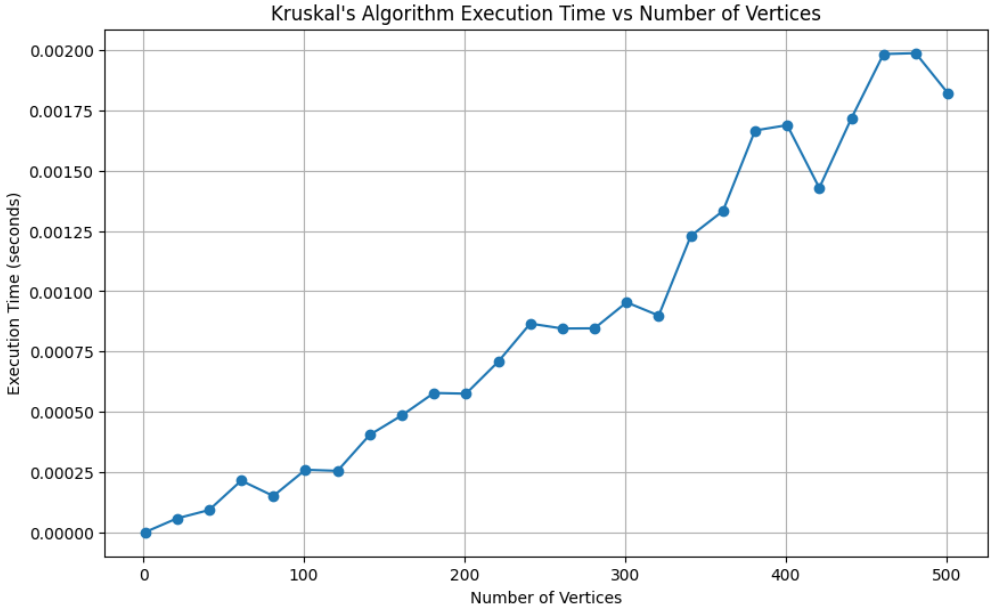

Рис. 5: На рисунку з кроком 20 вершин показано залежність часу виконання алгоритму Крускала (з компресією шляху) від кількості вершин графа при щільності 0.2. На кожну кількість вершин взято середній час 1000 експериментів.

Бачимо досить лінійну залежність часу, з деякими скачками, які можна пояснити характеристиками згенерованих графів. Цей результат відповідає теоретичній оцінці $O(V)$.

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