Звіт

Опис проекту

У цьому проекті реалізовано бібліотеку для роботи з графами, яка виконує операції над графами. Основні функції бібліотеки:

1. Читання графу з файлу.
2. Пошук Гамільтонового циклу - Бугір Єлизавета
3. Пошук Ейлерового циклу - Клюсик Анастасія
4. Перевірка на дводольність - Зубрицька Богдана
5. Перевірка ізоморфізму графів - Кібєрєва Вікторія
6. Розфарбування графа у три кольори - Михайлищук Назар

Опис алгоритмів

1. **Перевірка на дводольність:**

   Функція `is_bipartite` перевіряє, чи є граф двочастковим. Двочастковий граф — це граф, вершини якого можна розбити на дві множини так, щоб всі ребра з'єднували вершини з різних множин.

Використовується підхід з перевіркою на розфарбування у два кольори:

· Починаємо з будь-якої вершини, яку фарбуємо в колір 0.

· Сусідів цієї вершини фарбуємо в колір 1, сусідів сусідів — знову в 0, і так далі.

· Якщо в процесі фарбування знаходимо дві суміжні вершини з однаковим кольором, граф не є двочастковим.

Граф може складатися з кількох компонент незв'язаних частин, тому функція перевіряє кожну компоненту окремо. Для орієнтованих графів враховується напрямок ребер під час перевірки кольорування.

Параметри:

graph (dict):
Граф, заданий у вигляді списку суміжності. Ключі словника — вершини, значення — список суміжних вершин.

directed (bool):
Вказує, чи граф є орієнтованим.

Знання із дискретної математики:
Дводольний граф — це граф, вершини якого можна розбити на дві множини так, щоб всі ребра з'єднували вершини з різних множин.

2. Пошук Гамільтонового циклу

Опис:

Функція **hamiltonian_cycle** знаходить Гамільтонів цикл у графі за допомогою використання рекурсивного підходу із застосуванням методу “backtraking”, якщо не вдається знайти на певному етапі Гамільтонів шлях. Для цього було прописано одну основну функцію **hamiltonian_cycle** та дві допоміжні **is_valid** та **hamiltonian_util** . У задачі пошуку Гамільтонового циклу в графі основна функція **hamiltonian_cycle** відповідає за запуск процесу пошуку та вибір стартової вершини. Її роль - це забезпечення виконання алгоритму і повернення результату — знайдений цикл або **None**, якщо циклу немає.

Допоміжна функція **is_valid** перевіряє, чи можна додати вершину до шляху, перевіряючи дві умови: чи є вершина сусідом останньої в шляху та чи не була вона відвідана раніше.

Функція **hamiltonian_util** є рекурсивною і намагається побудувати шлях, додаючи вершини. Якщо шлях неможливо знайти таким чином, то

відбувається, в певному плані, “відкат” і надалі підбирається інша вершина. Якщо цикл знайдений,то рекурсія завершується. А якщо не вдалося його знайти, то вертає **None**.

Параметри:

graph: словник, в якому ключ — певна вершина графу, а значення — це сусідні до нього вершини**.**

directed (bool | None): параметр для визначення, чи є граф орієнтованим.

1. Якщо значення None, орієнтація графа визначається за допомогою атрибута **directed** об’єкта.
2. Якщо True, граф вважається орієнтованим (ребра мають напрямок).
3. Якщо False, граф вважається неорієнтованим (ребра не мають напрямку).

(За замовчуванням використовується значення None, тобто орієнтація графа встановлюється через атрибут об'єкта.)

Повертає:

1. Список цілих чисел, що представляють вершини Гамільтонового циклу, якщо такий цикл існує. Перший і останній елементи цього списку — це одна й та сама вершина, що вказує на цикл.
2. **None**, якщо Гамільтонів цикл не знайдений.

Алгоритм роботи:

1. Функція починає пошук з довільної вершини (зазвичай першої в отриманому словнику).
2. Алгоритм намагається побудувати шлях, рекурсивно додаючи до нього вершини, що відповідають умовам, за яких це можливо (цю роль виконує функція **is_valid**)
3. Кожен крок перевіряє, чи є поточний шлях правильним:
    a. Поточна вершина повинна бути суміжною з останньою доданою вершиною.

    b. Поточна вершина не повинна бути вже в шляху.
4. Якщо шлях включає всі вершини і повертається до початкової, знайдено Гамільтонів цикл.
5. Якщо поточний шлях не може бути продовжений (або через те, що вершина не є суміжною з попередньою, або вже була відвідана), алгоритм відкачується, видаляючи останню вершину з шляху і пробуючи іншу вершину. Якщо жоден валідний цикл не знайдений, функція повертає **None**.

Знання із дискретної математики:
Гамільтонів цикл — це цикл, який проходить через кожну вершину графа рівно один раз і повертається до початкової вершини.

3. Пошук Ейлерового циклу

Реалізовано дві функції для знаходження ейлерового циклу: find_eulerian_cycle_general і find_eulerian_cycle_uniform. Вони виконують пошук Ейлерового циклу для однорідних та неоднорідних графів. Обидві функції базуються на ітеративному алгоритмі для побудови циклу. Нижче розглянемо реалізацію та функціональні аспекти кожної з них.

find_eulerian_cycle_general - функція для знаходження Ейлерового циклу в **неоднорідному** графі.

Параметри:

graph: словник, який представляє граф у вигляді списків суміжності. Ключі — вершини, значення — списки сусідніх вершин.

Алгоритм роботи:

1. **Перевірка парності ступенів вершин**:
   Використовується умова: якщо у графі є вершина із непарним ступенем (кількістю сусідніх вершин), то Ейлеровий цикл неможливий.
2. Для збереження початкового графу створюється його локальна копія (local_graph).

3. **Пошук циклу** (вкладена функція find_cycle):
   Алгоритм побудови циклу реалізовано за допомогою стека:
   - Стек використовується для збереження шляху.
   - Якщо у вершини є сусіди, обирається один із них, видаляється ребро між ними і додається сусід у стек.
   - Коли у вершини більше немає сусідів, вона додається до циклу.
4. Після завершення побудови циклу перевіряється, чи всі ребра графу були використані.
5. Повертається список вершин у порядку проходження Ейлерового циклу.

find_eulerian_cycle_uniform - функція призначена для роботи з **однорідними** графами. Однорідним вважається граф, у якого всі вершини мають однаковий ступінь.

Параметри:

graph: словник, який представляє граф у вигляді списків суміжності.

Алгоритм роботи:

1. Використовується перевірка, чи всі вершини мають однаковий ступінь.

2. Створюється копія графу.

3. Логіка реалізації вкладеної функції find_cycle аналогічна до find_eulerian_cycle_general.

4. Якщо після побудови циклу у графі залишилися невикористані ребра, повертається повідомлення: "The graph does not have an Euler cycle".

5. Повертається список вершин Ейлерового циклу.

Знання із дискретної математики:
Ейлеровий цикл — це шлях у графі, який проходить через кожне ребро рівно один раз і повертається до початкової вершини.

4. **Перевірка на ізоморфність**

Функція **isomorphic** приймає два графи і перевіряє їх на ізоморфність.

Для цього вона використовує алгоритм Вайсфейлера-Лемана.

Для цього вона викорисовує допоміжну функцію **label,** яка кожній вершині графу присвоює її ім'я:

1. На першій ітерації кожній вершині присовоюється ім'я 0
2. Далі для кожної вершини до її імені додаються імена усіх її сусідів
3. Так ми повторюємо стільки разів, скільки є вершин у графі

Функцію **labels** виконуємо для обох графів. Порівнюємо множини імен усіх вершин. Якщо вони відрізняються, то графи не ізоморфні. Якщо ж вони рівні, то графи мають однакові канонічні форми, проте ми не можемо повністю виключити ймовірність того, що графи не ізоморфні.

Знання з дискретної математики

Прості графи G1 та G2 називаються ізоморфними, якщо існує така бієкція $\phi: V1 \to V2$, що вершини u та v суміжні в G1 тоді і тільки тоді, коли вершини $\phi(u)$ та $\phi(v)$ суміжні в G2 для всіх $u,v \in V1$

5. **Розфарбування графа у 3 кольори**

Функція `three_color_graph` приймає граф та повертає розмальований, або ж повідомлення про неможливість його розмалювати

Для цього вона проводить певні махінації:

- Сортує граф за його степенем
- Від більшого до меншого степеня пробує розфарбувати граф
    - Перевіряє чи не стикується перший можливий колір з сусідніми вершинами
    - Якщо стикується, пробує розмалювати в інший колі
- Робить це до поки він повністю не розмалює граф, або ж дізнається що його неможливо розфарбувати

Допоміжні функції допомагають розфарбувати граф.

Функція **is_avalible** перевіряє чи не стикається колір у який ми хочемо розмалювати вершину з сусідами

Функція **color_graph** рекурсивно намагається зафарбувати граф. Якщо виходять якісь стики функція повертається на місце де граф не мав ніяких стичок і пробує інакше розмалювати граф.

Знання з дискретної математики

Знання про види та типи графів.

Процес виконання проекту

На початку проекту була створена структура бібліотеки та визначено, які функції реалізовуватимуться.

Кожен учасник команди реалізував свою частину проекту, після чого виконувалася інтеграція та тестування всіх функцій. Було протестовано всі функції на графах різних розмірів і типів (орієнтованих, неорієнтованих, зв’язних і незв’язних). Використовується модуль argparse. Після відгуку ментора ми додали більш детальну документацію та чіткіший поділ на клас.

Враження:

Проект дав змогу поглибити знання в теорії графів, вивчити нові алгоритми та навчитись їх застосовувати на практиці. Проте складність викликала оптимізація перевірки ізоморфізму графів, оскільки немає чіткого правильного алгоритму перевірки на цю властивість, окрім будування бієктивного зображення, яке важко зробити на великих об’ємах.

Фідбек викладачам та асистентам

Дякуємо за цікаве завдання, яке дозволило застосувати теоретичні знання на практиці та надало можливість краще засвоїти знання, отримані протягом вивчення цього матеріалу.

Проте було би добре, якби нам дали навчальні матеріали про графи швидше,

оскільки ми почали їх вивчати 3 тижні після того, як розпочали роботу над проєктом, і потрібно було самостійно шукати інформацію. Окрема подяка нашому ментору – пану Андрію, який завжди готовий прийти на допомогу, та який давав слушні поради щодо покращення та оптимізації проєкту.

Висновок

Ми вдосконалили знання алгоритмів теорії графів, а також їх реалізацію у вигляді зручної для використання бібліотеки. Розроблена бібліотека є функціональною, а отримані знання та досвід — корисними для подальшого навчання та роботи.