**Прізвище:** Муравський

**Ім'я:** Владислав

**Група:** КН-408

**Варіант:** 17

**GitHub:**

**Кафедра:** САПР

**Дисципліна:** Дискретні моделі в САПР

**Перевірив:** Кривий Р. З.

**ЗВІТ**

до лабораторної роботи №5

«Ізоморфізм графів»

**Мета:** вивчення і дослідження основних підходів до встановлення ізоморфізму графів.

**Теоретичні відомості:**

**1. Алгоритми рішення задачі встановлення ізоморфізму графів**

a) З точки зору реалізації програмного забезпечення, одним із найпростіших алгоритмів є той, який перевіряє ізоморфізм графа шляхом повного обходу (можливо перенумерації вершин), але складність цього алгоритму факторна.

б) Граф «підвішений» чергуванням вершин (всі ребра збалансовані). Суть алгоритму полягає в знаходженні однакових «підвішених» графів (для будь-якої вершини), ізоморфізм яких є детермінованим. Більш того, в одному з графів вершини, на яких він «висить», чергуються. Ізоморфізм графів визначається їх матрицями суміжності, які формуються за однотипними правилами: вершина, для якої граф закріплений («плаває»), має в матриці індекс, що дорівнює 1, кортеж вершин у матриця задається Рівнем сусідів; кортеж вершин у кожному сусідньому рівні визначається ступенем вершини та кількістю ребер над і під нею.

в) Метод побудови оптимального графового коду. Метод базується на алгоритмі, який формує еквівалентні матриці зв’язності шляхом ідентифікації вершин з однаковими топологічними характеристиками. Відповідно до алгоритму таким же методом перетворюється матриця зв'язку графа. У випадку, коли матриці модифікації однакові, граф є ізоморфним. Ізоморфізм між двома чи більше графами можна визначити за допомогою їх оптимальних кодів.

d) З використанням згортки графа.

**2. Рішення теоретичної задачі встановлення ізоморфізму простих графів**

Рішення теоретичної задачі встановлення ізоморфізму простих графів: повного перебору і почергового “підвішування” графів за вершини.

**3. Ідея рішення задачі встановлення ізоморфізму наближеними методами**

На даний час створена певна кількість алгоритмів встановлення ізоморфізму графів, які за рахунок різних евристичних прийомів знижують складність задачі від факторіальної до степеневої функції.

**4. Основні кроки алгоритму побудови оптимального коду ненаправленого нерегулярного графа**

Алгоритм побудови оптимального коду графа представлений такими процедурами:

1) Будуємо матрицю суміжності MSUM(N\*N) та матрицю найменшої відстані MNV(N\*N) для графу G (N,L), де N - кількість вершин в графі, а L кількість зв'язків між вершинами. Приймаємо k = 1.

2) З головної діагоналі матриці найменшої відстані MNV знаходимо вершину (вершини) найбільшої степені. Якщо є єдина вершина з найбільшим степенем, то позначаємо її як Ak та переходимо на 10-й крок. В противному випадку (якщо маємо більше ніж одну вершину з найбільшим степенем) переходимо на 3-й крок.

3) Нехай, скажемо, є j вершин з найбільшим степенем (M1,M2,...,Mj). Для кожної вершини знаходимо число MW. Якщо є єдина вершина з найбільшим числом MW, то позначаємо цю вершину як Ak та переходимо на 10-й крок. Якщо маємо більше, ніж одну вершину з найбільшим числом MW, то переходимо на 4-й крок.

4) Нехай є j вершин з найбільшим числом MW (M1,M2,...,Mj). Для кожної вершини знаходимо число VN. Якщо є єдина вершина з найбільшим числом VN, то позначимо цю вершину як Ak та переходимо на 10-й крок. Якщо є більше ніж одна вершина з найбільшим числом VN, то переходимо на 5-й крок.

5) Нехай маємо i вершин з найбільшим числом VN(M1,M2,...,Mi). На даному етапі ми не можемо виділити з I вершин одну кращу вершину на місце Ak. Всі I вершин є можливими претендентами на Ak, отже ми повинні тепер виконувати процедури пошуку наступних вершин для I можливих претендентів на Ak, поки та чи інша єдина вершина не стане кращою вибіркою для Ak, або поки не завершиться алгоритм. Переходимо на 10-й крок.

6) Знаходимо число B1 для всіх непозначених сусідів вершини Aq (визначення подано на 11-тому кроці). Якщо є єдина вершина, яка має найменше число B1, то позначаємо її як Ak та переходимо на 10-й крок. В противному випадку підраховуємо числа Bd (2 <= d <= k-1) для тих вершин, які мають мінімальне число Bd-1. Якщо знайдена єдина вершина з найменшим числом Bd, то позначаємо цю вершину як Ak та переходимо на 10-й крок. Якщо є більше ніж одна вершина з найменшим числом Bd, та якщо d = k-1, то перейти на 7-й крок. Якщо d < k-1, то збільшити d на одиницю та повторити обчислення.

7) Підраховуємо число U для кожних перших сусідів вершини Aq, що мають найменше число Bd. Якщо є єдина вершина з найбільшим числом U, то позначити цю вершину як Ak та перейти на 10-й крок. В противному випадку переходимо на 8-й крок.

8) Обчислюємо степінь кожного першого непозначеного сусіда Aq, який має найменше число Bd та найбільше число U. Якщо є єдина вершина з найбільшим степенем, то позначити цю вершину як Ak та перейти на 10-й крок. В противному разі перейти на 9-й крок.

9) Підраховуємо число C для кожного першого сусіда Aq, який має найменше число Bd та найбільші число U і степінь. Якщо є єдина вершина з найбільшим числом C то позначаємо цю вершину як Ak та переходимо на 10-й крок. Нехай, скажемо, є L вершин з найбільшим числом C (M1,M2,...,Ml). Тоді всі ці L вершин є претендентами на Ak, отже ми повинні тепер виконувати процедури пошуку наступних вершин для L можливих претендентів на Ak. Переходимо на 10-й крок.

10) Якщо k = 1, то перейти на 11-й крок. Якщо є тільки одна вибірка для Ak-1, то перейти на 11-й крок. Припустимо що є P вибірок для Ak-1. Якщо вершина Ak не позначена для всіх вибірок, то визначаємо Aq та переходимо на 6-й крок для позначення Ak для ще однієї вибірки Ak-1 (порядок не важливий). В противному випадку для кожної вибірки виводимо числа Bd, число U, степінь та число c вершин, позначених як Ak у вигляді підкодів. Найкращими варіантами є підкоди з найменшими числами Bd та найбільшим числом U, степенем та числом C. Припустимо, що тільки R підходів є оптимальними. Далі ми будемо розглядати тільки ці підкоди. Переходимо на 11-й крок, щоб призначити Ak+1 для вершин Ak, що залишились.

11) Приймаємо k = k+1. Якщо k = N, то переходимо на 12-й крок. Якщо k = N, то знаходимо Aq, де Aq - вже позначена вершина з найнижчим індексом, яка має принаймні одного непозначеного першого сусіда. Якщо Aq має тільки одного непозначеного першого сусіда, то призначити цього сусіда як Ak та перейти на 10-й крок. Якщо Aq має більше, ніж одного непозначеного першого сусіда, то перейти на 6-й крок.

12) Коли k=N, то вершина, яка залишається непозначеною, позначається як Ak для всіх вибірок Ak-1. Якщо ми маємо P вибірок, то тепер будуємо P оптимальних кодів. Якщо ці коди не є тотожними, то код з найбільшою вагою вибирається як оптимальний код графа.

**5) Сутність методів згортки (редукції) графа**

Редукція полягає в згортці (послідовному спрощені) графа за допомогою замін фрагментів графа розробленими спрощувальними підстановками.

**Індивідуальне завдання:**

1. Отримати у викладача індивідуальне завдання.  
2. Підготувати програму для вирішення виданого завдання.  
3. Запустити на виконання програму відповідного методу.  
4. Проглянути результат роботи програм. Результат роботи може бути:  
ізоморфізм встановлено або не встановлено.  
5. У випадку, коли ізоморфізм встановлено (не встановлено), необхідно  
модифікувати граф, коректуючи два або три зв’язки, щоб знайти такий граф,  
на якому ізоморфізм не встановлюється (встановлюється).  
6. Зафіксувати результати роботи у викладача.  
7. Оформити і захистити звіт.

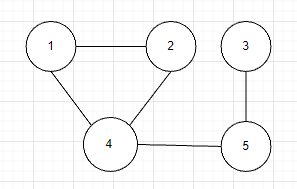
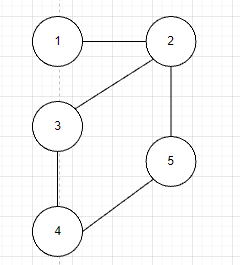
 

Рис. 1. Перший заданий граф. Рис. 2. Другий заданий граф.

По будові дані графи не є однаковими, і нумерація навіть різна, тому нам варто перевірити чи є дані графи ізоморфні. Складемо матриці суміжності для кожного з них:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **4** | **5** |
| **1** | 0 | 1 | 0 | 1 | 0 |
| **2** | 1 | 0 | 0 | 1 | 0 |
| **3** | 0 | 0 | 0 | 0 | 1 |
| **4** | 1 | 1 | 0 | 0 | 1 |
| **5** | 0 | 0 | 1 | 1 | 0 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **4** | **5** |
| **1** | 0 | 1 | 0 | 0 | 0 |
| **2** | 1 | 0 | 1 | 0 | 1 |
| **3** | 0 | 1 | 0 | 1 | 0 |
| **4** | 0 | 0 | 1 | 0 | 1 |
| **5** | 0 | 1 | 0 | 1 | 0 |

**Код алгоритму виконання:**

def check\_isom(gr1, gr2):  
 if len(gr1) != len(gr2) or len(gr1[0]) != len(gr2[0]):  
 return False  
  
 n = len(gr1)  
  
 vertices = list(range(n))  
  
 visited = set()  
  
 # перебирання всіх можливих перестановок вершин  
 for perm in itertools.permutations(vertices):  
 if perm not in visited:  
 visited.add(perm) # додання нової перестановки до відвіданого набору  
  
 # перевірка, чи є поточна перестановка дійсним ізоморфізмом  
 mapping = {}  
 is\_valid = True  
 for j in range(n):  
 for k in range(n):  
 if gr1[j][k] != gr2[perm[j]][perm[k]]:  
 is\_valid = False  
 break  
 if not is\_valid:  
 break  
 mapping.setdefault(gr1[j][j], gr2[perm[j]][perm[j]])  
  
 if mapping[gr1[j][j]] != gr2[perm[j]][perm[j]]:  
 is\_valid = False  
 break  
 if is\_valid:  
 return True  
  
 return False # графи не ізоморфні

**Результати виконання програми:**

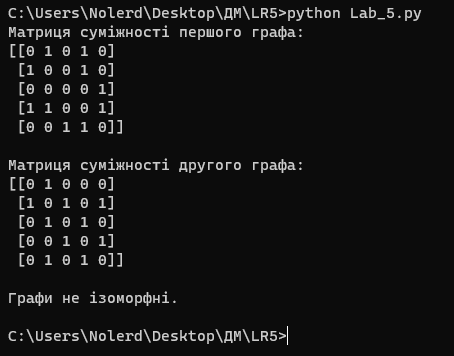


Рис. 3. Вивід даних на консоль

**Висновок:** під час виконання даної лабораторної роботи, проведено ознайомлення з основними підходами до встановлення ізоморфізму графів. У роботі, побудовано два графи, проведено їх перевірку на ізоморфізм, та складено програмну реалізацію, результати співпадають з ручними.