НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

До захисту допущено
Завідувач кафедри
_____ Оксана ТИМОЩУК
«07» 06 2021 р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Системний аналіз і управління»
спеціальності 124 "Системний аналіз"

Виконав: Використання спектральної теорії графів для розв'язання комбінаторних задач
Студент IV курсу, групи КА-73
Расторгуєв Роман Олексійович _____

Керівник:
доцент, к.ф.-м.н., доцент кафедри ММСА
Олександр Вікторович Стусь _____

Консультант з економічного розділу:
доцент, к.е.н., доцент кафедри ТТПЕ
Рощіна Надія Василівна _____

Консультант з нормоконтролю:
доцент, к.т.н., доцент кафедри ММСА
Коваленко Анатолій Єпіфанович _____

Рецензент:
доцент, к. ф.-м. н., доцент кафедри МА та ТЙ
Ільєнко Андрій Борисович _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ-2021

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 124 "Системний аналіз"

Освітньо-професійна програма «Системний аналіз і управління»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____Оксана ТИМОЩУК
«___»_____20__ р.

ЗАВДАННЯ
на дипломну роботу студенту
Расторгуєву Роману Олексійовичу

1. Тема роботи «Використання спектральної теорії графів для розв'язання комбінаторних задач», керівник роботи к. ф.-м. н., доцент Стусь Олександр Вікторович, затверджені наказом по університету від 26.05. 2021 р. №1344-с .

2. Термін подання студентом роботи 07.06.2021 р.

3. Вихідні дані до роботи: матриця суміжності графа

4. Зміст роботи: розробка програмного продукту для обчислення кількості кістякових дерев графа

5. Перелік ілюстративного матеріалу: (із зазначенням плакатів, презентацій тощо): презентація

6. Консультанти розділів роботи:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Рощіна Н.В., доцент кафедри ТПЕ		

7. Дата видачі завдання: 20.02.2021

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Затвердження теми ДР	12.04.2021-20.04.2021	виконано
2	Ознайомлення зі структурою БДР згідно з Положенням про державну атестацію студентів НТУУ «КПІ ім. І. Сікорського»	12.04.2020-27.04.2021	виконано
3	Ознайомлення з ДСТУ 3008-95 та стандарти ЄСПД	20.04.2021-27.04.2021	виконано
4	Проведення дослідження за темою БДР під керівництвом керівника	20.04.2021-04.05.2021	виконано
5	Завершення роботи над першим варіантом частини БДР	05.05.2021-12.05.2021	виконано
6	Проведення роботи над експериментальною частиною БДР	05.05.2021-16.05.2021	виконано
7	Проведення роботи над програмним продуктом	16.05.2021-25.05.2021	виконано
8	Оформлення БДР та аналіз отриманих результатів	16.05.2021-25.05.2021	виконано

Студент Роман РАСТОРГУЄВ

Керівник Олександр СТУСЬ

РЕФЕРАТ

Дипломна робота містить 89 с., 8 таблиць, 31 рис., 2 дод. та 33 джерела.

ГРАФ, РЕГУЛЯРНИЙ ГРАФ, ПОВНИЙ ГРАФ, МАТРИЦЯ СУМІЖНОСТІ, ХАРАКТЕРИСТИЧНИЙ ПОЛІНОМ, СПЕКТРАЛЬНА ТЕОРІЯ ГРАФІВ, КІЛЬКІСТЬ КІСТЯКОВИХ ДЕРЕВ ГРАФА.

Об’єкт дослідження: кількість кістякових дерев графа.

Предмет дослідження: методи спектральної теорії графів.

Мета роботи: аналіз методів спектральної теорії графів для розв’язання задач комбінаторики та обчислення кількості кістякових дерев графа. Подальша розробка програмного продукту для обчислення кількості кістякових дерев графа на основі проведеного аналізу.

Метод дослідження: аналіз методів спектральної теорії графів.

Розроблено алгоритм обчислення кількості кістякових дерев графа, розрахований на регулярні і нерегулярні графи з довільною розмірністю. Для подальшого вдосконалення програмного продукту можна впровадити функціонал покрокового відображення роботи алгоритму, а також графічний інтерфейс користувача для покращення зручності користування.

ABSTRACT

Thesis contains 89 p., 8 tables, 31 fig., 2 add. and 33 references.

GRAPH, REGULAR GRAPH, COMPLETE GRAPH, ADJACENCY MATRIX, CHARACTERISTIC POLYNOMIAL, SPECTRAL GRAPH THEORY, NUMBER OF SPANNING TREES IN A GRAPH.

Object of research: number of spanning trees in a graph.

Subject of research: methods of spectral graph theory.

The purpose of the work: to analyze the methods of spectral graph theory to solve combinatorial problems and calculate number of spanning trees in a graph. Further development of a software product for calculating number of spanning trees in a graph based on the analysis.

Method of research: analysis of methods of spectral graph theory.

An algorithm for calculating the number of skeletal trees of a graph, designed for regular and irregular graphs with arbitrary dimensions, has been developed. To further improve the software product, functionality of step-by-step display of the algorithm could be added, as well as a graphical user interface to improve usability.

ЗМІСТ

ВСТУП

Теорія графів – хоч і досить давній розділ математики, періодом започаткування якого вважають початок XVIII сторіччя, у сучасному світі є дуже актуальним. Методи, що використовуються у теорії графів, як і весь розділ цілком, є досить універсальними, тому знаходять місце у багатьох сферах сучасного світу, хоч ми і не завжди це помічаємо. Теорія графів є досить великим розділом математики, який доволі часто використовують у якості інструмента для пошуку рішень.

Головне завдання теорії графів – вивчення властивостей графів для подальшого їх використання при вирішені задач. Також важливою частиною є малювання графів для відображення відносин на множинах, що є додатковим плюсом у сенсі наочності усього процесу.

Комбінаторика, у свою чергу, - також дуже популярна і затребувана наука, оскільки вона, хоч і не має такої універсальності на відміну від теорії графів, дозволяє вирішити велику кількість проблем з вибором та розташуванням. Задачі, які комбінаторика покликана вирішити нерідко зустрічаються при розробці або тестуванні алгоритмів.

Ця робота має за мету розробити алгоритм для вирішення комбінаторних задач за допомогою спектральної теорії графів, а також проаналізувати доцільність вибору саме методів спектральної теорії графів для вирішення поставленого питання.

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність задачі

Комбінаторика – це область математики, яка пов’язана зі задачами вибору, розташування і маніпуляцій у скінченній і дискретній системах. Вона допомагає вирішити задачі з різних областей математики, таких як алгебра, геометрія і теорія ймовірностей, а також широко використовується у комп’ютерних науках з ціллю отримання формул і оцінок при аналізі алгоритмів – для оцінки кількості потрібних алгоритму ітерацій.

Також комбінаторика включає комбінаторну оптимізацію, яка має велику різноманітність застосувань – складання авіаційних мап, алгоритми служб таксі і доставки для вибору оптимального водія для виконання замовлення, оптимізація шляху.

Комбінаторика тісно пов’язана з теорією графів, через що частим вибором для вирішення комбінаторних задач є методи теорії графів. Їх перевагою можна вважати потенціал для візуалізації процесу, що надалі здатне покращити досвід використання продукту.

Сукупність вищеперерахованих факторів визначила метод, який ми надалі будемо використовувати у цій роботі.

1.2 Основні поняття

1.2.1 Теорія графів

Графом $G = (X, U)$ називають скінчену множину X (її елементи називають вершинами) сумісно зі множиною U двоелементних підмножин цього X (елементи множини U називають ребрами). Аналогічно, орієнтовний граф (X, U) – це скінченна множина X і множина U впорядкованих пар елементів множини X, які називаються орієнтованими ребрами або дугами. [1]

Графи з неорієнтованими або орієнтованими кратними ребрами називаються мультиграфами и мультиорграфами. У цьому випадку допускається наявність петель – ребер чи дуг, обидві вершини яких співпадають (Рисунок 1.1). [1], [16]

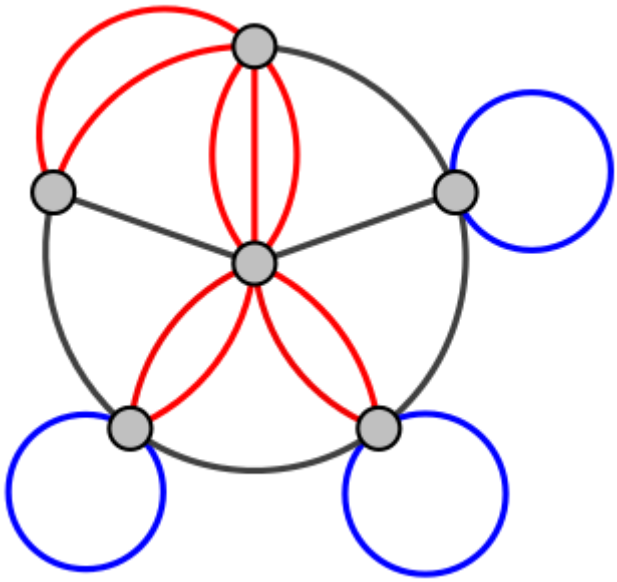


Рисунок 1.1 – Мультиграф з кратними ребрами (червоні) і петлями (сині)

Дві вершини називаються суміжними, якщо вони поєднані ребром. Якщо вершина x є кінцем ребра u, то x і u є інцидентними. Число ребер, які є інцидентними до даної вершини неорієнтованого графа, називають ступенем або валентністю даної вершини.

Якщо усі вершини графа мають одну й ту саму валентність, яка дорівнює r, то граф називають регулярним до степені r (Рисунок 1.2). [1], [5]

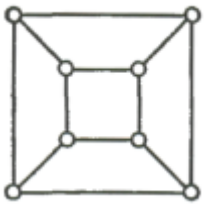


Рисунок 1.2 – Граф, регулярний до степені 3

Матриця суміжності А мультиграфа G з множиною вершин $\{x_1, x_2, \dots, x_n\}$ – це квадратна матриця порядку n, у котрій значення a_{ij} елемента, розташованого на місці (i, j) дорівнює числу ребер (дуг), що починаються у вершині x_i і закінчуються у вершині x_j.

Мультиграфи $G = (X, U)$ і $H = (Y, V)$ називають ізоморфними, якщо існує (1,1)-відображення $y = \varphi(x)$ X на Y таке, що для будь-якої пари вершин $x', x' \in X$ у H є стільки ж ребер, що прямують із $y' = \varphi(x')$ до $y'' = \varphi(x'')$, скільки у G прямує із x' у x'' (Рисунок 1.3).

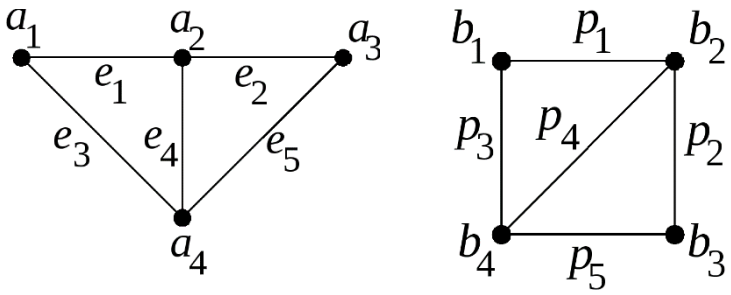


Рисунок 1.3 – Ізоморфні графи

Характеристичний поліном $|\lambda I - A|$ матриці суміжності A графа G називається характеристичним поліномом графа G. Власні значення матриці A і спектр матриці A називаються відповідно власними значеннями і спектром графа G.

Граф $H = (Y, V)$ є підграфом графа $G = (X, U)$, якщо $Y \subset X$ і $V \subset U$. Граф H називається кістяковим підграфом чи частковим графом графа G, якщо $Y = X$. Якщо множина V складається із усіх таких ребер множини U, що вони з'єднують вершини із множини Y, то H називається породженим підграфом.

За допомогою K_n будемо позначати повний граф на n вершинах – граф, у якого будь-які дві різні вершини з'єднані одним ребром (Рисунок 1.4).

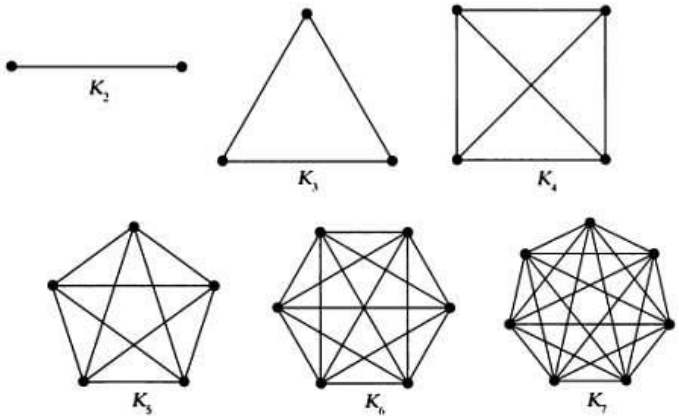


Рисунок 1.4 – Приклади повних графів

Будь-яка послідовність прямуючих один за одним ребер мультиграфа називається маршрутом. Довжина маршруту – число його ребер. Маршрут може містити одне і те ж ребро більш ніж один раз. Простий ланцюг P_n довжини $n-1, n \geq 2$, – це граф, який має n вершин $x_1, \dots, x_n$ і $n-1$ ребер, у якому вершини x_i і $x_{i+1}, i = 1, \dots, n-1$, з'єднані ребром.

Вершини a_1 і a_2 називаються зв'язаними, якщо у графі є шлях між ними

1.2.2 Спектральна теорія графів

Спектральна теорія графів – це наука про властивості графу у зв'язку з характеристичним поліномом, власними числами та власними векторами матриць, що пов'язаний з графом, такими як матриця суміжності або матриці Кірхгофа. [15]

Спектром графа G називають спектр його матриці суміжності. Дві матриці суміжності A і A' належать одному класу суміжних матриць тоді і тільки тоді, коли існує матриця перестановки P така, що $A' = P^{-1}AP$. Таким чином, теорію графів G можна ототожнити з теорією цих матричних класів $\mathcal{A}$ і їх інваріантами. Важливими інваріантами класу $\mathcal{A}$ є характеристичний поліном $P_G(\lambda) = |\lambda I - A|, A \in \mathcal{A}(G)$ і спектр $S_p(G) = [\lambda_1, \lambda_2, \dots, \lambda_n]$, де λ_i – корінь рівняння $P_G(\lambda) = 0$.

Два графи називаються ізоспектральними, якщо матриця суміжності графів є ізоспектральною, тобто, якщо вона має однакові мультимножини власних значень. Ізоспекральні графи не обов'язково є ізоморфними, проте ізоморфні графи завжди ізоспектральні.

Кажуть, що граф G визначається своїм спектром, якщо будь-який інший граф, що має такий же спектр, як і G, є ізоморфним до G.

1.3 Операції над графами

Операції над графами можна поділити за вхідними даними: унарні операції – одномісні операції, які створюють новий граф із старого та бінарні операції – двомісні операції, які створюють нових граф із двох вхідних графів $G_1(V_1, E_1)$ і $G_2(V_2, E_2)$.

Унарні операції:

– Реберний граф:

Реберний граф $L(G)$ неорієнтованого графа G, який представляє сусідство ребер графа G [18]

– Двоїстий граф:

Граф G', двоїстий до планарного графа G – граф, у якого усі вершини відповідають граням графа G (Рисунок 1.5).[20], [18]

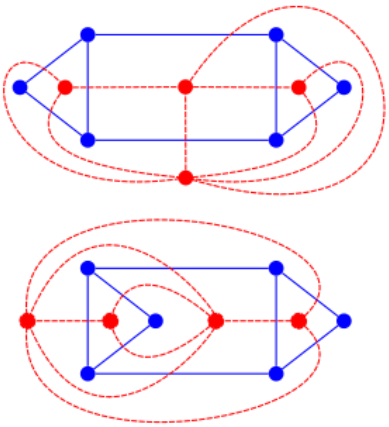


Рисунок 1.5 – червоні графи є двоїстими до синього графа

– Доповнення:

Граф H, обернений до графа G (доповнення графа G) – граф на тих самих вершинах, поєднаних ребрами тоді і тільки тоді, коли вони несуміжні в G (Рисунок 1.6).[4]

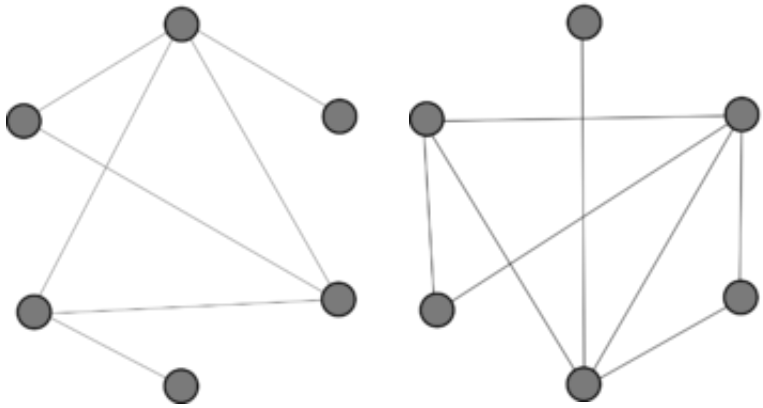


Рисунок 1.6 – граф (зліва) і його доповнення (справа)

– Мінор графа:

Граф H – мінор для заданого графа G, який може бути утворений з G видаленням ребер і вершин і стягуванням ребер [18]

– Стягування ребер:

Операція, яка видаляє ребро з графа, а вершини, які він зв'язував до цього, зливаються у одну

– Ступінь графа:

Ступінь G^k неорієнтованого графа G – це інший граф, який має той самий набір вершин і дві вершини цього графа суміжні, якщо відстань між цими вершинами у початковому графі G не перевищує k. [21]

Бінарні операції:

– Об'єднання графів:

Граф, що містить об'єднання множин вершин V_1 і V_2 графів і множин дуг. [4]

– З’єднання:

Об’єднання двох графів, до яких додані усі дуги, що з’єднують вершини обох графів (Рисунок 1.7). [4], [7]

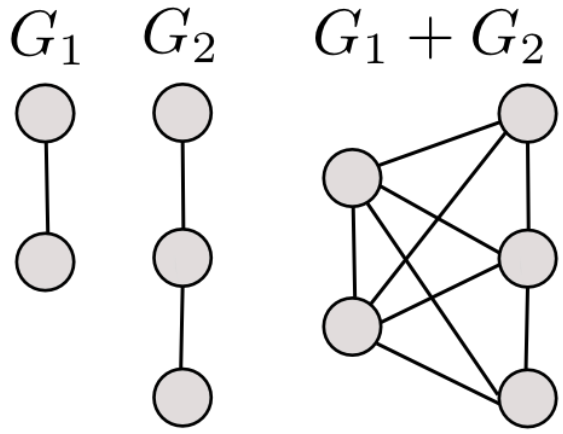


Рисунок 1.7 – Приклад з’єднання графів G_1 і G_2.

– Добуток графів :

Добутком $G_1 \times G_2$називають граф з множиною вершин V, яка дорівнює декартовому добутку $V_1 \times V_2$. Вершини $u = (u_1, u_2)$ і $v = (v_1, v_2)$ з $V = V_1 \times V_2$ суміжні у $G = G_1 + G_2$ тоді і тільки тоді, коли $(u_1 = v_1, \text{а } u_2 \text{ і } v_2 - \text{суміжні})$ чи $(u_2 = v_2, \text{а } u_1 \text{ і } v_1 - \text{суміжні})$ (Рисунок 1.8). [4], [7]

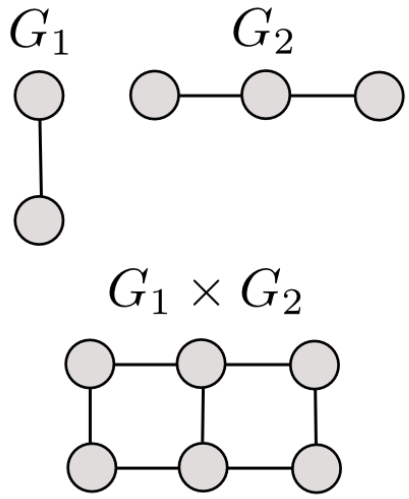


Рисунок 1.8 – Приклад добутку графів G_1 і G_2.

– Композиція графів :

Композиція $G_1(G_2)$ графів G_1 і G_2 – це граф з множиною вершин E, у якому існує дуга (x_i, x_j) тоді і тільки тоді, коли існує дуга (x_i, x_k), яка належить множині E_1 і дуга (x_k, x_j), яка належить множині E_2 (рисунок 1.9). [4], [7]

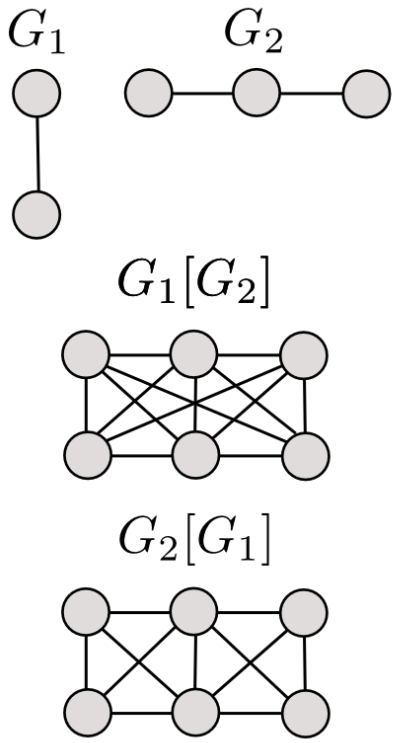


Рисунок 1.9 – Приклад композиції графів G_1 і G_2.

– Перетин:

Перетин $G_1 \cap G_2$ графів G_1 і G_2 – це граф з множиною вершин $X_1 \cap X_2$ з множиною ребер $E = \mathrm{E}_1 \cap E_2$. [12]

1.4 Основні теореми

Теорема 1. [22]

Не існує регулярних графів степені r, які мають $r^2 + 2$ вершини і обхват, що дорівнює 5.

Теорема 2. [23]

Нехай p і q – прості числа, $p \equiv q \equiv 3(mod\ 4)$ і s – додатне ціле число. Не існує циклічних самодвоїстих графів з $n = p^{2s}$ чи з $n = pq$вершинами.

Теорема 3. [24]

Для твірної функції $H_G(t)$ числа маршрутів у графі G правдиві наступні формули:

$$H_{G'}(t) = \frac{1}{1-t} H_G\left(\frac{1}{1-t}\right);$$

$$H_{\vec{G}}(t) = \frac{H_G\left(-\frac{t}{t+1}\right)}{t+1-tH_G\left(-\frac{t}{t+1}\right)};$$

$$H_{G_1+G_2}(t) = H_{G_1}(t) + H_{G_2}(t);$$

$$H_{G_1 \nabla G_2}(t) = \frac{H_{G_1}(t) + H_{G_2}(t) 2tH_{G_1}(t)H_{G_2}(t)}{1-t^2 H_{G_1}(t)H_{G_2}(t)}.$$

Теорема 4. [25], [26]

Нехай $N_k^j \sum c_{ji_j} \lambda_{ji_j}^k (j = 1, \dots, n)$ позначає число маршрутів довжини k у графі G_j. Тоді NEP-сума з базисом $\mathcal{B}$ графів $G_1, \dots, G_n$містить $N_k = \sum_{i_1,\dots,i_n} c_{1i_1} \dots c_{ni_n} \left(\sum_{\beta \in \mathcal{B}} \lambda_{1i_1}^{\beta_1} \dots \lambda_{ni_n}^{\beta_n}\right)^k$ маршрутів довжини k

Теорема 5. [24]

Функція $G(t) = (-1)^n \frac{P_{\overline{A}}(-1/t)}{P_A(1/t)}$ є твірною функцією для чисел $\overline{V}_n^k(A)$ A-перестановок k-го класу з повтореннями, при цьому правдивим є співвідношення $\overline{V}_n^k(A) = \frac{1}{k} G^{(k)}(0), k = 1,2, \dots$ при k=0 $\overline{V}_n^0(A) = 1$.

Теорема 6. [1]

Число $\mathcal{M}-$ поєднань дорівнює

$$\frac{n}{p}\left(\sum_{\mathcal{J}'}\frac{p!}{j_0!j_1!\dots j_\mu!}-1\right),$$

де $\mathcal{J}'=\{(j_0,j_1,\dots,j_\mu)|\sum_{k=0}^{\mu}j_k=p\},n|\sum_{k=1}^{\mu}m_kj_k$.

Теорема 7. [1], [27]

Якщо G – регулярний граф степені r з n вершинами і m ребрами, то

$$t(L(G))=2^{m-n+1}r^{m-n-1}t(G).$$

Теорема 8. [28]

Граф G з n вершинами і більш ніж $[n^2/4]$ ребрами містить принаймні один трикутник. Єдиними графами без трикутників, що мають n вершин і $[n^2/4]$ ребер є тільки графи $K_{l,l}$ для $n=2l$ і $K_{L=1,l}$ для $n=2l+1$.

Теорема 9. [1]

Якщо граф G має $2n$ вершин і n^2+p ребер, то він не містить менше ніж $\frac{pn}{2}+\frac{p^2}{2n}+\frac{p^2}{6n^3}$ трикутників.

Теорема 10. [29]

Якщо G – регулярний граф степені r з n вершинами $\left(r\geq\frac{1}{2}(n-2);n\geq 2\right)$, то для числа $t(G)$ кістякових дерев графа G правдиві наступні нерівності:

$$\left(\frac{2r+2-n}{n}\right)^{n-1}n^{n-2}\leq t(G)\leq\left(\frac{r}{n-1}\right)^{n-1}n^{n-2}.$$

1.5 Висновки за розділом

У даному розділі були розглянуті об’єкти предметної області. Були визначені основні поняття для звичайної теорії графів, а також для спектральної теорії графів. Окрім цього були розглянуть операції над графами для розуміння загального процесу, а також введені основні теореми, які у подальшому будуть використовуватися в ході роботи

РОЗДІЛ 2 РОЗВ'ЯЗАННЯ КОМБІНАТОРНИХ ЗАДАЧ ЗА ДОПОМОГОЮ СПЕКТРАЛЬНОЇ ТЕОРІЇ ГРАФІВ

2.1 Пошук спектрів і характеристичних поліномів для графів деяких спеціальних типів

– для графа G з n вершинами без ребер чи петель характеристичний поліном $P_G(\lambda) = \lambda^n$, тобто спектр, складається з n чисел, які дорівнюють 0

– повний граф K_n з n вершинами є доповненням вищенаведеного графа. Для нього $P_{K_n}(\lambda) = (\lambda - n + 1)(\lambda + 1)^{n-1}$, тобто спектр графа K_n складається з числа $n-1$ і $n-1$ чисел, що дорівнюють -1.

– якщо G – регулярний граф степені 1, кожна компонента якого є ізоморфною до K_2, то згідно попередньому пункту $P_{K_2}(\lambda) = \lambda^2 - 1$. Відповідно, має місце співвідношення $P_G(\lambda) = (\lambda^2 - 1)^k$, якщо G містить $2k$ вершин.

– при переході до додаткового графу графа у попередньому пункті, знаходимо характеристичний поліном регулярного графа H степені $n-2$ з $n = 2k$ вершинами у вигляді

$$P_H(\lambda) = (\lambda - 2k + 2) \times \lambda^k \times (\lambda + 2)^{k-1}.$$

– для повного дводольного графа K_{n_1,n_2} правдивим є співвідношення $K_{n_1,n_2} = G_1 \nabla G_2$, де G_1 і G_2 – графи, які відповідно містять n_1 і n_2ізольованих вершин. Оскільки $P_{G_1}(\lambda) = \lambda^{n_1}$ і $P_{G_2}(\lambda) = \lambda^{n_2}$, то $P_{K_{n_1,n_2}}(\lambda) = (\lambda^2 - n_1 n_2)\lambda^{n_1+n_2-2}$, тобто спектр графа K_{n_1,n_2}складається з чисел $\sqrt{n_1 n_2}$, $-\sqrt{n_1 n_2}$ і $n_1 + n_2 - 2$ чисел, що дорівнюють нулю. Якщо $n_1 = n$, а $n_2 = 1$, то виходить зірка з $n+1$ вершинами, характеристичним поліномом якої є $P_{K_{n,1}}(\lambda) = (\lambda^2 - n)\lambda^{n-1}$.

– спектр простого циклу C_n складається з чисел $2\cos\frac{2\pi}{n}$ $(i = 1, \dots, n)$. Нескладно побачити, що справедливим є співвідношення:

$$P_{C_n}(\lambda) = 2\left(T_n\left(\frac{\lambda}{2}\right) - 1\right) = -2 + \sum_{k=0}^{[n/2]}(-1)^k \frac{n}{n-k}\binom{n-k}{k}\lambda^{n-2k} = 2\cos\left(n \arccos\frac{\lambda}{2}\right) - 2,$$

де $T_n(x) = \sum_{k=0}^{\left[\frac{n}{2}\right]}(-1)^k \frac{n}{n-k}\binom{n-k}{k} 2^{n-2k-1}x^{n-2k} = \cos(n \arccos x)$

– многочлен Чебишева першого роду.

– з попередніх результатів можемо вивести характеристичний поліном і спектр простого ланцюга P_n з n вершинами. Усі підграфи простого циклу C_n, що були породжені $n-1$ вершинами, ізоморфні до простого ланцюга P_{n-1}. Отже, $P_{P_{n-1}}(\lambda) = \frac{1}{n}P'_{C_n}(\lambda)$. За допомогою полінома Чебишева:

$$U_n(x) = \frac{\sin[(n+1)\arccos x]}{\sqrt{1-x^2}},$$

отримуємо

$$P_{P_n}(\lambda) = U_n\left(\frac{\lambda}{2}\right) = \sum_{k=0}^{[n/2]}(-1)^k \frac{n}{n-k}\binom{n-k}{k}\lambda^{n-2k}.$$

Звідси маємо, що спектр простого ланцюга P_n складається з чисел $2\cos\frac{\pi}{n+1}i$ $(i = 1, \dots, n)$.

2.2 Огляд комбінаторних задач, які розв’язуються за допомогою спектральної теорії графів

2.2.1 Існування або неіснування комбінаторних об’єктів

Існування деяких комбінаторних об’єктів можна дослідити за допомогою спектрів графу. Сенс методу полягає у припущені існування досліджуваного об’єкта і подальшому його співставленні з деяким графом. Використовуючи властивості комбінаторного об’єкта знаходять спектр співставленого графа, після чого за допомогою знайденого спектра знаходять відповідний граф, або спростовують існування такого, що навпаки доводить неіснування досліджуваного комбінаторного об’єкта.

В загалі, задача існування графа з заданим спектром не є легкою і має декілька способів для вирішення. Наприклад, застосовується побудування: спочатку будують графи з відомими спектрами, після чого на них застосовують деякі графові операції з ціллю отримати шуканий граф із заданим спектром. Іншим метод є отримання усієї можливої інформації за допомогою теорем про спектри графа, після чого побудова графа за допомогою неспектральних методів.

Для того щоб довести неіснування комбінаторного об’єкта, можна скористатися спектральними методами і отримати із заданого спектра деякі структурні деталі, що будуть суперечити запропонованим властивостям об’єкта, що розглядається.

Нажаль, на теперішній час не існує методів для вирішення описаної задачі у загальному випадку. Більш того, така задача є досить складною навіть за наявності окремого конкретного випадку.

2.2.2 Визначення кількості маршрутів

Кількість маршрутів у графі пов’язана з його спектром. Твірна функція для кількості маршрутів отримується через характеристичні поліноми графа і його

доповнення і позначається як $H_G(t)$. Для визначення твірної функції для будь-якого графа (за умови відомих твірних функцій для елементарних графів) використовується теорема 3 з пункту 1.4 цієї роботи.

Для того щоби показати застосування даного методу у вирішенні комбінаторних задач, наведемо приклад: Необхідно знайти кількість перестановок k-го класу з повторенням елементі з множини $\mathcal{X} = \{x_1, \dots, x_n\}$, які є довільною впорядкованою k-кой $(x_1, \dots, x_{i_k})$, де $i_i \in \{1, \dots, n\}$ $(j = 1, \dots, k)$.

Під час утворення підстановок з повторенням можливе введення деяких обмежень. Розглянемо Задамо такі типи обмежень: Нехай $\mathcal{X}_i (i = 1, \dots, n)$ – сімейство підмножин множини $\mathcal{X}$. Пара (x_i, x_i) називається допустимою тоді і тільки тоді, коли $x_i \in \mathcal{X}_i$. Квадратна матриця $A = (a_{ij})^n$, де $a_{ij} = 1$, якщо (x_i, x_i) – допустима пара і $a_{ij} = 0$ в іншому випадку, називається матрицею допустимих пар. Матрицю обмежень $\overline{A}$ отримують із A у результаті взаємної заміни 0 і 1.

A-перестановкою з повторами k-го класу називають k-тка $(x_i, \dots, x_{i_k})$, де $x_{i_j} \in \mathcal{X}_{i_{j-1}} (j = 2, \dots, k)$.

Ми маємо можливість перевести дану задачу з задачею пошуку кількості маршрутів у графі. Якщо A інтерпретується як матриця сумісності орграфа G з вершинами $x_1, \dots, x_n$, то можемо показати, що число $\overline{V}_n^k(A)$, яке є відповіддю на нашу задачу.

2.2.3 Визначення кількості кістякових дерев

Для цього розділу знадобляться додаткові твердження:

Твердження 1. Якщо G – зв'язний мультиграф, то :

$$t(G) = \frac{1}{n} \Pi \mu,$$

де μ пробігає усі відмінні від нуля власні значення матриці $C = D - A$.

Цей результат може будити виражений наступним чином:

$$t(G) = \frac{(-1)^{n-1}}{n} C'_G(0) = \frac{1}{n} B_0^n(G).$$

Твердження 2. Для будь-якого регулярного мультиграфа G степені r

$$t(G) = \frac{1}{n} \prod_{i=2}^{n} (r - \lambda_i) = \frac{1}{n} P'_G(r),$$

де λ_i – звичайне власне значення мультиграфа G. [33]

Спектральний метод визначення кількості кістякових дерев $t(G)$ графа G заснований на вищезазначених твердженнях 1 і 2.

За допомогою другого твердження можуть бути одразу отримані кількості кістякових дерев для деяких графів:

– для повного графа K_n з n вершинами маємо:

$$t(K_n) = n^{n-2}.$$

– для регулярного графа G з $n = 2k$ вершинами степені $n - 2$:

$$t(G) = 2^{2k-2}(k-1)^k k^{k-2}.$$

– так як $P_{K_{n,n}}(\lambda) = (\lambda^2 - n^2)\lambda^{2n-2}$, то:

$$t\left(K_{n,.n}\right) = n^{2n-2}.$$

– нехай є граф $G^* = G + K_2$, де G – довільний граф і K_2 – повний граф на двох вершинах. Спектр графа K_2 містить числа 1, -1. Якщо $\lambda_1, \dots, \lambda_n$ – власні значення заданого графа G, то власними значеннями графа G^* є $\lambda_1 + 1, \dots, \lambda_n + 1, \lambda_1 - 1, \dots, \lambda_n - 1$, тобто:

$$P_{G^*}(\lambda) = P_G(\lambda - 1)P_G(\lambda + 1).$$

– характеристичний поліном $P_{G^*_{k,n}}(\lambda)$ графа Гуда – де Брюйна з обмеженнями $G^*_{k,n}$ можна отримати зі співвідношення

2.3 Огляд існуючих рішень головної задачі роботи

Існує велика кількість бібліотек, пакетів і надбудов, які дійсно значно полегшують роботу з графами і матрицями, дозволяючи використовувати вже створені функції для економії часу на розробку. До таких відноситься пакет NetworkX[13] для мови програмування Python, яка дозволяє створювати графи у вигляді об’єктів, використовувати операція над графами, а також малювати графи за допомогою бібліотеки MatPlotLib[30]. Окрім цього існують пакети Combinatorica` і IGraph/M для мови програмування Wolfram Language, які мають такі функції як побудова кістякових дерев, чи знаходження кількості кістякових дерев графа[31], [32].

Не дивлячись на це, на даний момент у відкритому доступі немає надбудов або готових рішень для розв’язання комбінаторних задач за допомогою спектральної теорії графів.

Також, у відкритому доступі немає готових рішень для покрокової візуалізації процесу вирішення комбінаторних задач. Варто зазначити, що такий функціонал здатний у великій мірі допомогти користувачам в розумінні роботи

продукту, а також в розумінні безпосередньо алгоритму, який застосовується для вирішення проблеми.

2.4 Висновки за розділом

У даному розділі був продемонстрований пошук спектрів, а також характеристичних поліномів для графів деяких спеціальних типів, результати котрого також використовувалися у ролі прикладу у другій частині цього розділу. Друга частина містить огляд комбінаторних задач, які можуть бути розв’язані за допомогою спектральної теореми графів, а також деякі приклади розв’язання таких задач на прикладі результатів з попередньої частини.

РОЗДІЛ 3 РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

3.1 Вступ до розділу

У даному розділі буде йтися про практичну частину, тобто розробку програмного продукту. Основною задачею програми є рахування кількості кістякових дерев графа, що був введений користувачем. Окрім цього до програми додано графічний модуль для виведення безпосередньо графа і декількох його кістякових дерев у ролі прикладу. Для розрахунку кількості кістякових дерев у програмі реалізовані методи Кірхгофа та метод на основі твердження Хученройтера.

3.2 Архітектура програми

На вхід до програми подається матриця суміжності графа, для якого необхідно знайти кількість кістякових дерев. Оскільки програма передбачає роботу з симетричною матрицею суміжності, поле для вводу було реалізовано такими чином, що ввести несиметричну матрицю неможливо. Розмірність матриці не є фіксованою і визначається у програмі після введення користувачем даних.

Таблиця 3.1 – Опис основних змінних програми

Змінна	Опис
adjacency_matrix	Симетрична матриця суміжності графа
G	Граф
r	Максимальна валентність графа
char_pol	Характеристичний поліном
eigen_values	Масив власних значень графа
eigen_vectors	Масив власних векторів графа

Таблиця 3.2 – Опис функцій програми

Функція	Опис
yes_or_no	Визначення відповіді, що була введена користувачем до консолі
draw_graph	Виведення малюнку графа на основі матриці суміжності
draw_some_of_spanning_trees	Виведення малюнку деяких кістякових дерев графа
huts	Знаходження числа кістякових дерев графа за методом на основі твердження Хученройтера
kirchhoff	Знаходження числа кістякових дерев графа за методом Кірхгофа
get_characteristic_polynomial	Знаходження характеристичного полінома графа
get_matrix	Функція введення симетричної матриці суміжності графа і її перевірки
get_max_valency	Знаходження максимальної валентності ребра графа
graph_is_regular	Перевірка графу на регулярність
graph_is_complete	Перевірка графу на повноту
add_loops	Додавання петель до графу шляхом зміни його матриці суміжності
get_eigens	Визначення власних значень і власних векторів графа
_expand	Знаходження усіх можливих розширень explored_nodes і explored_edges (Допоміжна функція функції get_all_spanning_trees)

Кінець таблиці 3.2

get_all_spanning_trees	Знаходження усіх кістякових дерев графа
cayley	Знаходження числа кістякових дерев графа за методом Келі

3.3 Інструкція з експлуатації

Для того, щоб запустити програму, достатньо відкрити файл з розширенням .exe. Хоча цей спосіб і має недоліки у вигляді часу, який витрачається на запуск програми і обмеженої доступності на система з ОС Linux або MacOS, він має перевагу, яка полягає у відсутності необхідності встановлювати додаткове програмне забезпечення на комп’ютер.

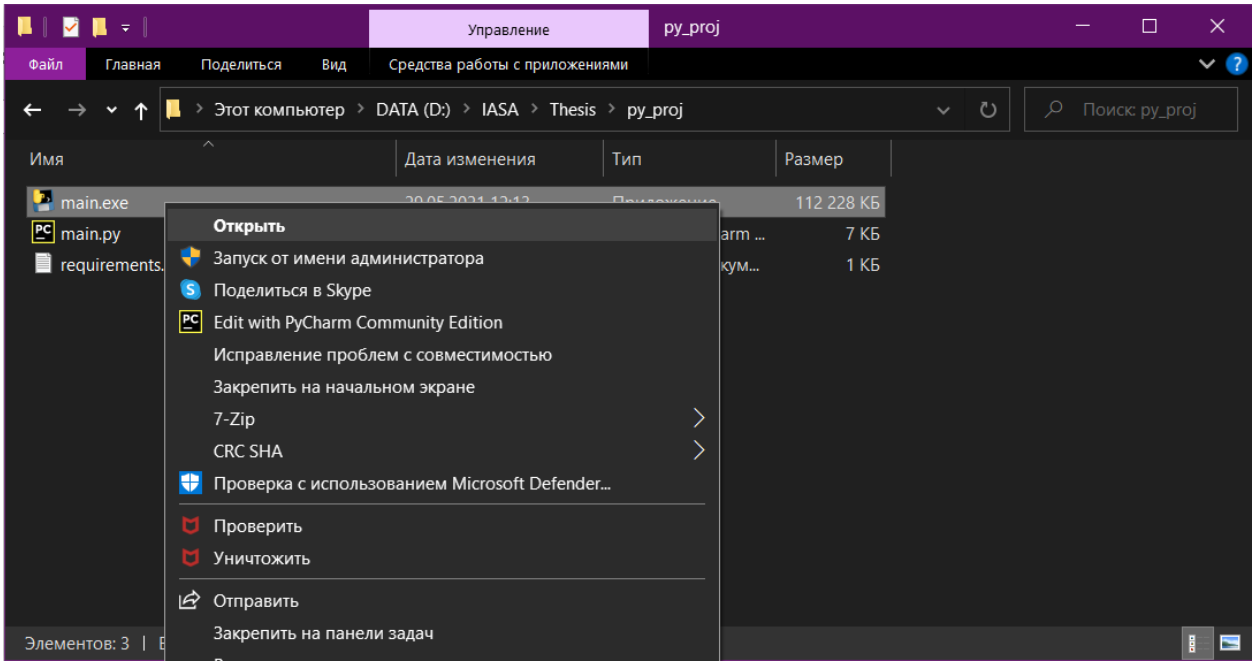


Рисунок 3.1 – Запуск файлу main.exe з файлової системи

Відкриється нове вікно, у якому автоматично розпочне свою роботу програма.

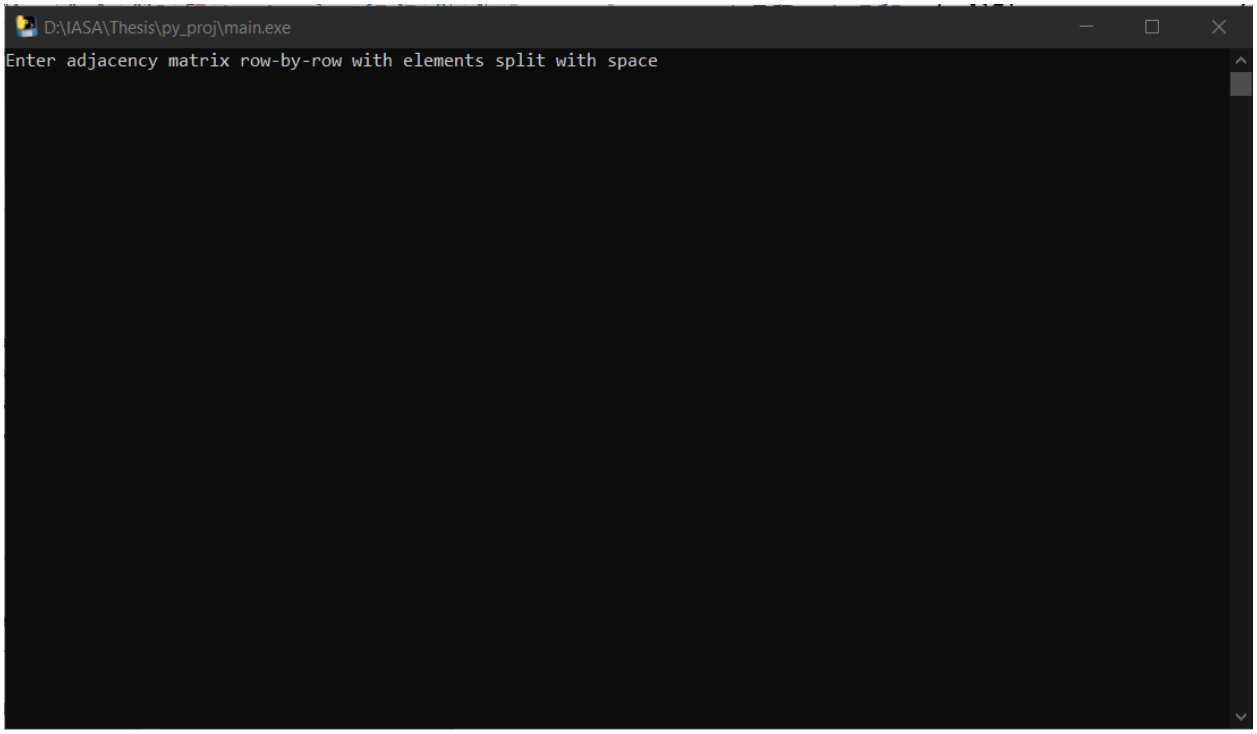


Рисунок 3.2 – Вікно програми після запуску

3.4 Алгоритм розв’язання задачі

Розроблена програма працює за таким алгоритмом:

а) Користувач вводить симетричну матрицю суміжності графа построково, розділяючи елементи кожного рядка пробілами. Розмірність матриці не є фіксованою і визначається довжиною першого введеного рядку. Окрім цього, частину матриці під головною діагоналлю програма заповнює сама, що виключає можливість ввести несиметричну матрицю.

б) Програма визначає максимальну валентність вершини графа, виводить для ознайомлення малюнок графа, що відповідає введеній матриці суміжності і проводить перевірку графа на регулярність, після чого, за потреби, додає до графа петлі і виводить малюнок оновленого графа.

в) Програма рахує характеристичний поліном і виводить його у загальному вигляді, а також виводить таблицю з власними значеннями графа і відповідними їм власними векторами графа.

г) Обчислюється кількість кістякових дерев введеного графа і виводиться зображення деяких кістякових дерев цього графа для ознайомлення.

3.5 Приклади роботи програми

3.5.1 Приклад регулярного графа

```
Enter adjacency matrix row-by-row with elements split with space
0 1 0 1 1 0
1 0 1 1 0 0
0 1 0 0 1 1
1 1 0 0 0 1
1 0 1 0 0 1
0 0 1 1 1 0
Matrix read successfully
[[0 1 0 1 1 0]
 [1 0 1 1 0 0]
 [0 1 0 0 1 1]
 [1 1 0 0 0 1]
 [1 0 1 0 0 1]
 [0 0 1 1 1 0]]
r= 3
n= 6
```

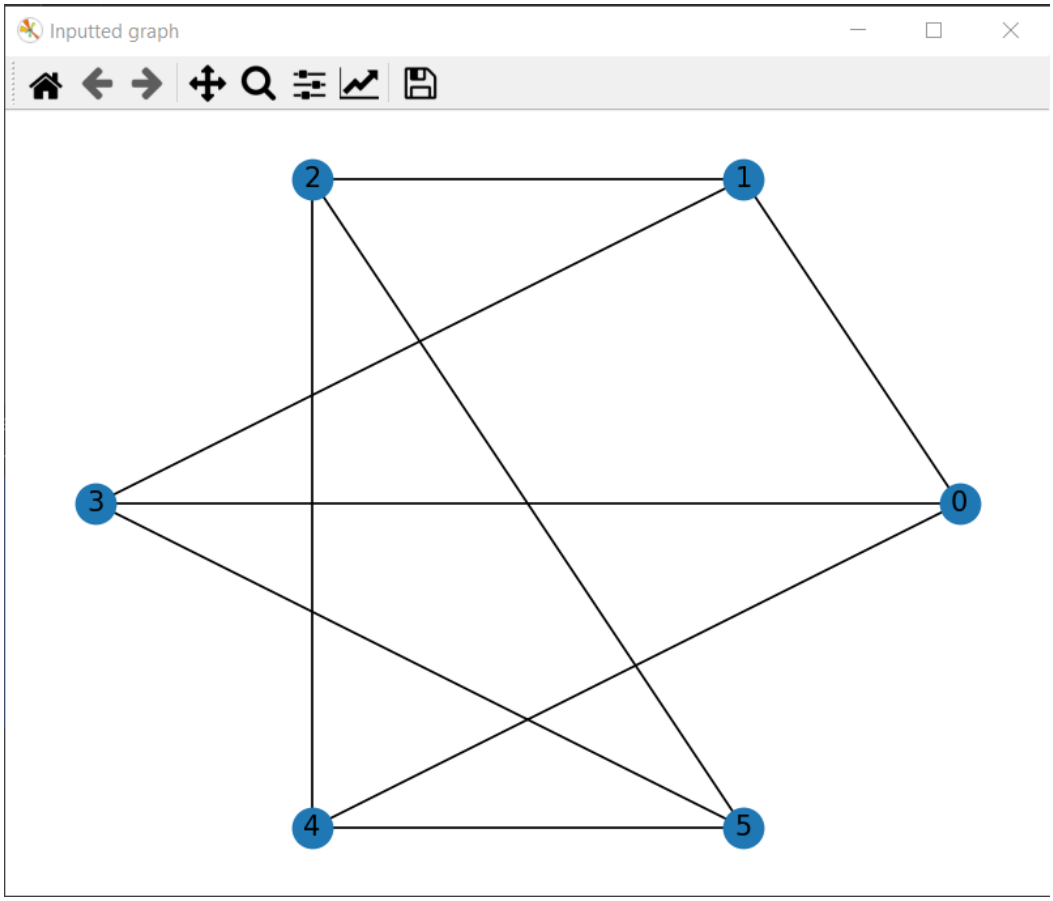


Рисунок 3.3 – Регулярний граф, побудований на основі матриці суміжності, яка була введена користувачем

Graph regular but not complete

Characteristic polynomial for adjacency matrix looks like:

$$\lambda^2 \cdot (\lambda - 3) \cdot (\lambda - 1) \cdot (\lambda + 2)^2$$

Рисунок 3.4 – Вигляд характеристичного полінома введеної матриці суміжності графа

Eigen value	Eigen vector
3.0	[-0.40824829 -0.40824829 -0.57735027 0.10989286 -0.55814831 -0.02909111]
1.0	[-0.40824829 -0.40824829 0.28867513 0.43591269 0.15119589 -0.48481932]
-2.0	[-0.40824829 0.40824829 -0.28867513 -0.43591269 0.15119589 -0.48481932]
-2.0	[-0.40824829 -0.40824829 0.28867513 -0.54580554 0.40695241 0.51391043]
0.0	[-0.40824829 0.40824829 0.57735027 -0.10989286 -0.55814831 -0.02909111]
-0.0	[-0.40824829 0.40824829 -0.28867513 0.54580554 0.40695241 0.51391043]

Рисунок 3.5 – Таблиця власних значень і відповідних їх власних векторів введеного графу

Huts' method:

$$t(G) = \frac{r^2 \cdot (r-1)^2 \cdot (r+2)^2}{n} = 75$$

Рисунок 3.6 – Вирішення задачі методом метод на основі твердження Хученройтера

Kirchhoff's method: t(G)=75

The Huts method takes an average of 3.229186000000013e-05 seconds to execute.

Kirchhoff's method:

t(G)=M(D-A), where M - any minor of matrix, D - Degree Matrix, A - Adjacency Matrix

A=

[[0 1 0 1 1 0]

[1 0 1 1 0 0]

[0 1 0 0 1 1]

[1 1 0 0 0 1]

[1 0 1 0 0 1]

[0 0 1 1 1 0]]

D=

[[3. 0. 0. 0. 0. 0.]

[0. 3. 0. 0. 0. 0.]

[0. 0. 3. 0. 0. 0.]

[0. 0. 0. 3. 0. 0.]

[0. 0. 0. 0. 3. 0.]

[0. 0. 0. 0. 0. 3.]]

L=

[[3. -1. 0. -1. -1. 0.]

[-1. 3. -1. -1. 0. 0.]

[0. -1. 3. 0. -1. -1.]

[-1. -1. 0. 3. 0. -1.]

[-1. 0. -1. 0. 3. -1.]

[0. 0. -1. -1. -1. 3.]]

t(G)=M(L)=75

The Kirchhoff method takes an average of 2.0045610000000024e-05 seconds to execute.

Press any key to continue …

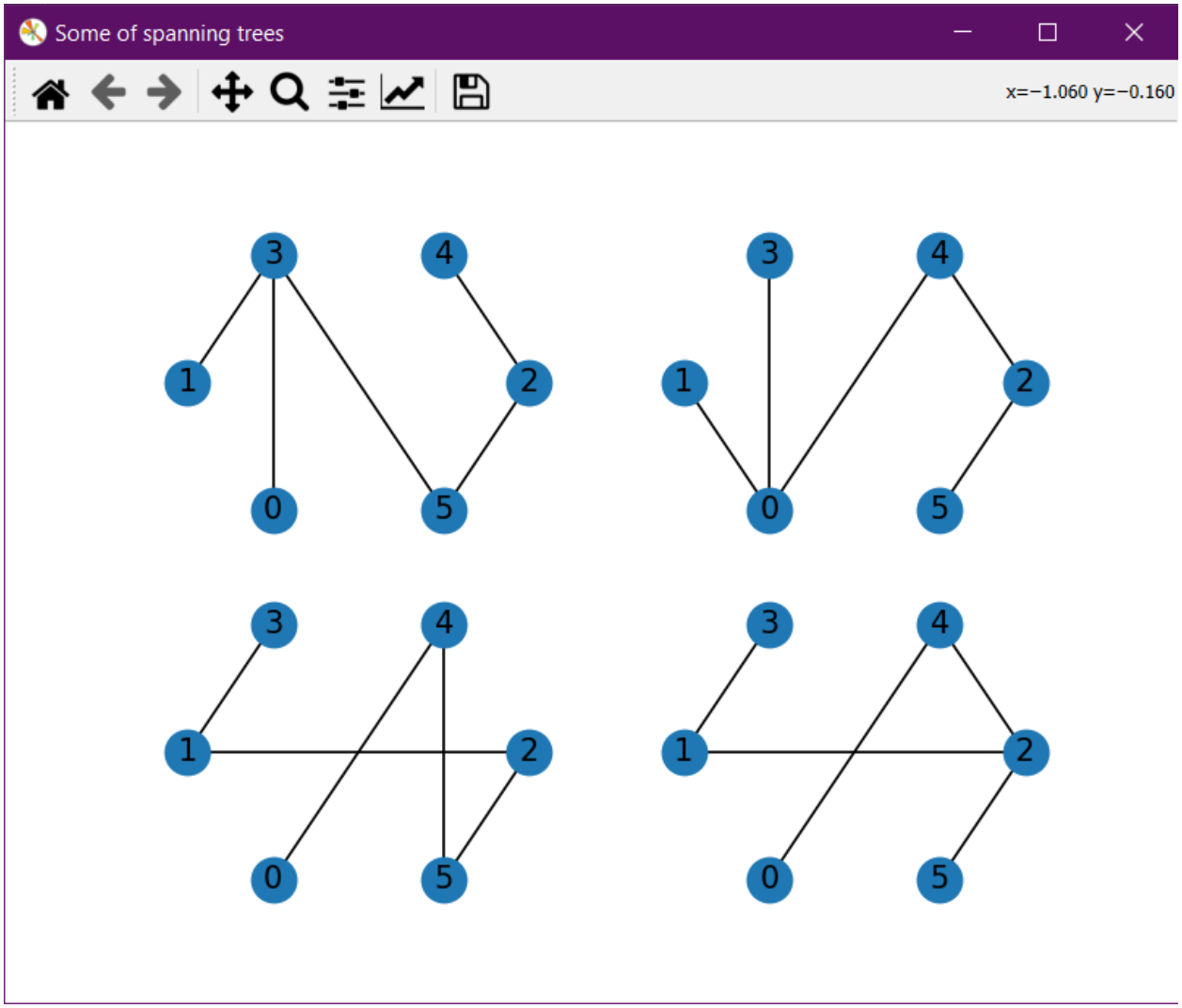


Рисунок 3.7 – Деякі кістякові дерева введеного графа

Process finished with exit code 0

3.5.2 Приклад нерегулярного графа

Enter adjacency matrix row-by-row with elements split with space

0 0 1 1

0 0 1 1

1 1 0 1

1 1 1 0

Matrix read successfully

[[0 0 1 1]

[0 0 1 1]

[1 1 0 1]

[1 1 1 0]]

r= 3

n= 4

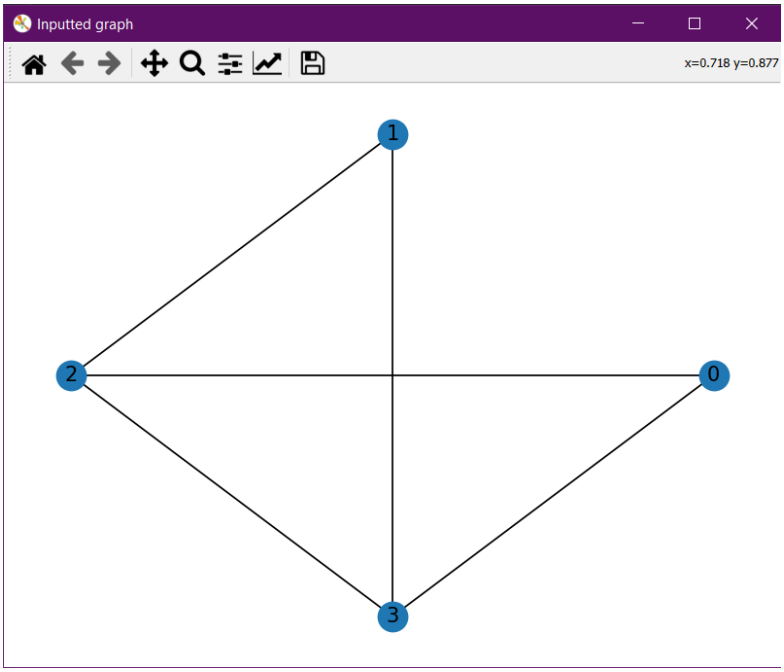


Рисунок 3.8 – Нерегулярний граф, побудований на основі матриці суміжності, яка була введена користувачем

Graph not regular. Adding self-loops...

New graph for Huts method:

[[1 0 1 1]

[0 1 1 1]

[1 1 0 1]

[1 1 1 0]]

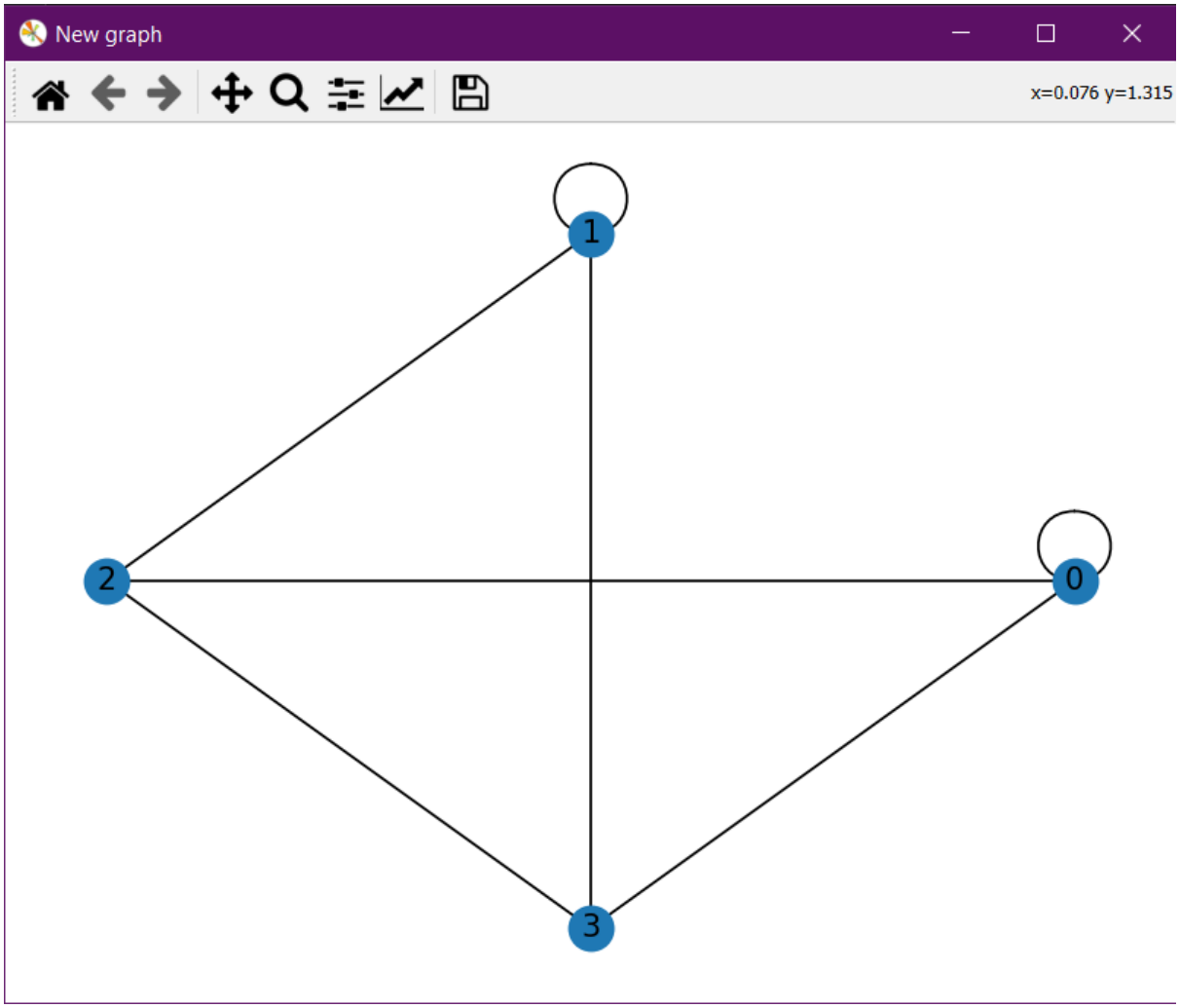


Рисунок 3.9 – Вхідний граф з доданими петлями

Characteristic polynomial for adjacency matrix looks like:

$$(\lambda - 3)\cdot(\lambda - 1)\cdot(\lambda + 1)^2$$

Рисунок 3.10 – Вигляд характеристичного полінома введеної матриці суміжності графа

Eigen value	Eigen vector
3.0	[-0.5 -0.70710678 0.5 0.12126781]
1.0	[-0.5 0.70710678 0.5 0.12126781]
-1.0	[-5.00000000e-01 -2.66704499e-16 -5.00000000e-01 -8.07262153e-01]
-1.0	[-5.00000000e-01 -8.19779219e-17 -5.00000000e-01 5.64726528e-01]

Рисунок 3.11 – Таблиця власних значень і відповідних їх власних векторів введеного графу

Huts' method:

$$t(G) = \frac{(r - 1) \cdot (r + 1)^2}{n} = 8$$

Рисунок 3.12 – Вирішення задачі методом метод на основі твердження Хученройтера

The Huts method takes an average of 2.5244069999999396e-05 seconds to execute.

Kirchhoff's method:

t(G)=M(D-A), where M - any minor of matrix, D - Degree Matrix, A - Adjacency Matrix

A=

[[0 0 1 1]

[0 0 1 1]

[1 1 0 1]

[1 1 1 0]]

D=

[[2. 0. 0. 0.]

[0. 2. 0. 0.]

[0. 0. 3. 0.]

[0. 0. 0. 3.]]

L=

[[2. 0. -1. -1.]

[0. 2. -1. -1.]

[-1. -1. 3. -1.]

[-1. -1. -1. 3.]]

t(G)=M(L)=8

The Kirchhoff method takes an average of 2.019069999999914e-05 seconds to execute.

Press any key to continue …

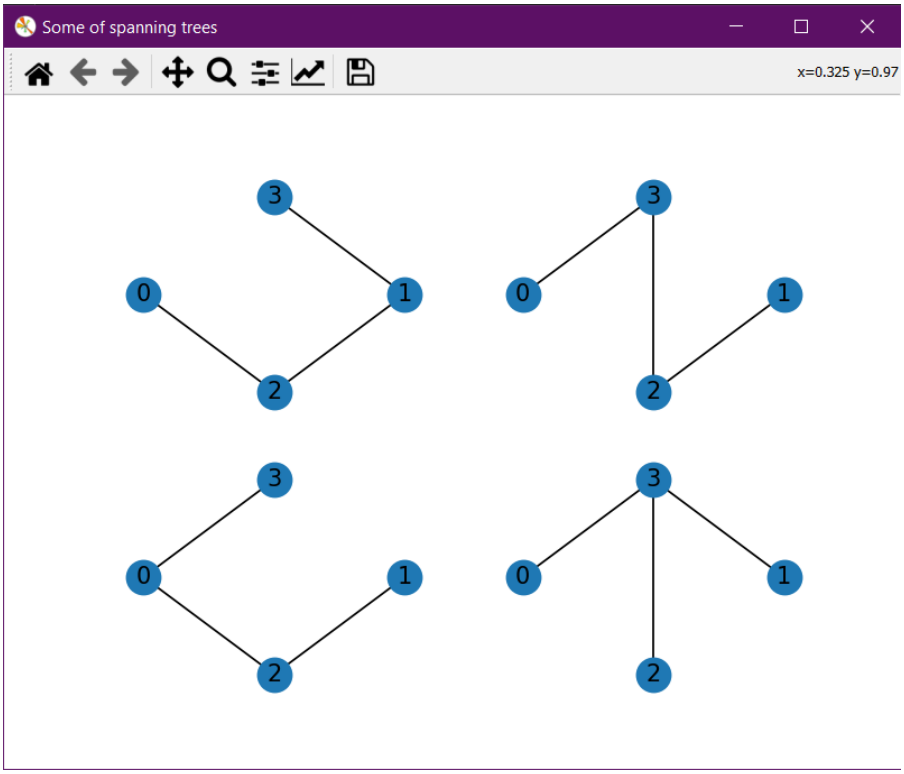


Рисунок 3.13 – Деякі кістякові дерева введеного графа

Process finished with exit code 0

3.5.3 Приклад повного графа

Enter adjacency matrix row-by-row with elements split with space

0 1 1 1 1 1 1 1

1 0 1 1 1 1 1 1

1 1 0 1 1 1 1 1

1 1 1 0 1 1 1 1

1 1 1 1 0 1 1 1

1 1 1 1 1 0 1 1

1 1 1 1 1 1 0 1

1 1 1 1 1 1 1 0

Matrix read successfully

[[0 1 1 1 1 1 1 1]

[1 0 1 1 1 1 1 1]

[1 1 0 1 1 1 1 1]

[1 1 1 0 1 1 1 1]

[1 1 1 1 0 1 1 1]

[1 1 1 1 1 0 1 1]

[1 1 1 1 1 1 0 1]

[1 1 1 1 1 1 1 0]]

r= 7

n= 8

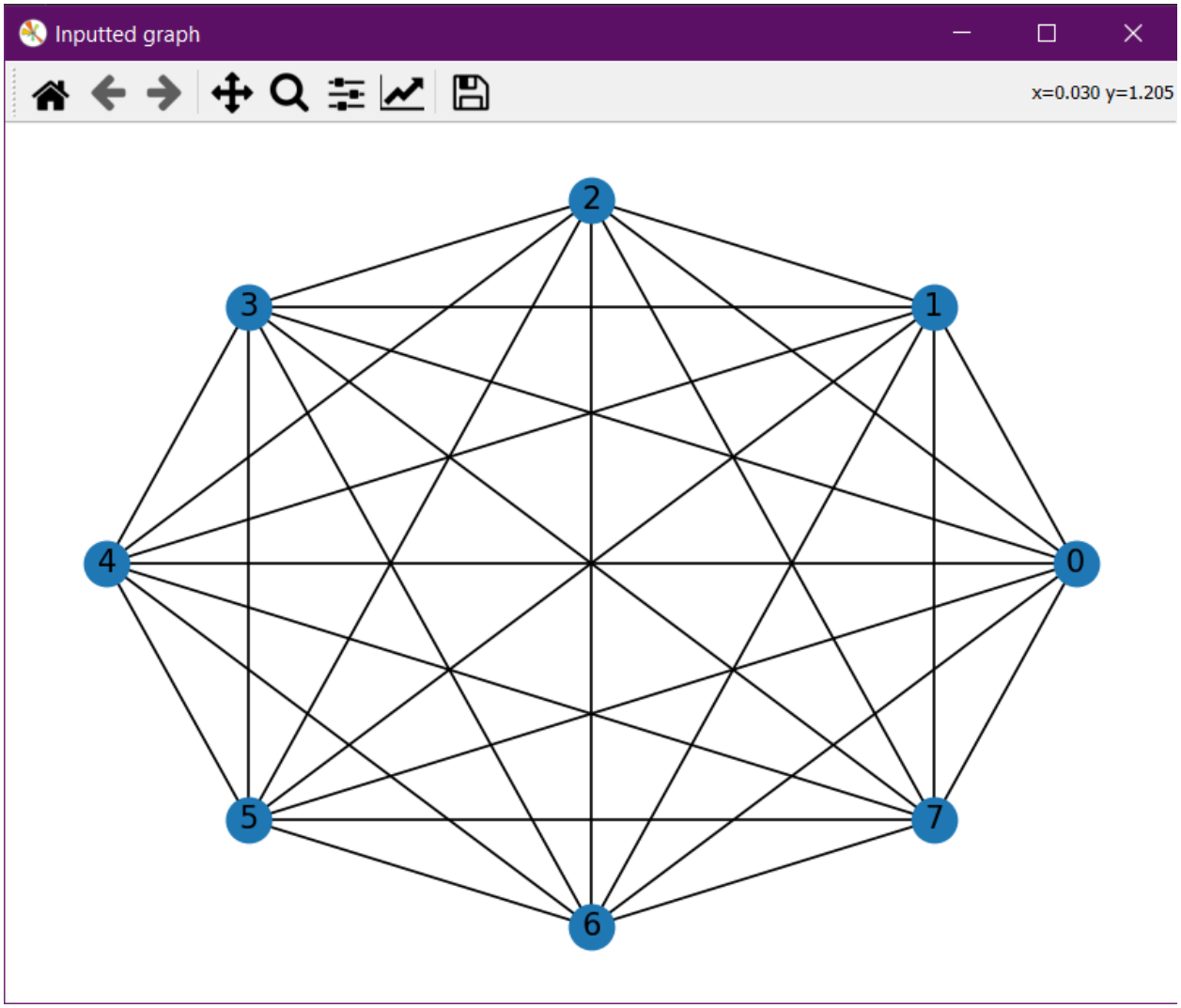


Рисунок 3.14 – Повний граф, побудований на основі матриці суміжності, яка була введена користувачем

Graph regular and complete

Characteristic polynomial for adjacency matrix looks like:

$$(\lambda - 7)\cdot(\lambda + 1)^{7}$$

Рисунок 3.15 – Вигляд характеристичного полінома введеної матриці суміжності графа

Eigen value	Eigen vector
-1.0	[-0.93541435 0.35355339 0.0673326 -0.13963183 -0.12519645 0.28379149 0.13658387 -0.2034466]
7.0	[0.13363062 0.35355339 -0.3174251 0.49868509 0.16096687 0.28379149 -0.17560783 0.2615742]
-1.0	[0.13363062 0.35355339 -0.50123695 -0.71434353 0.12849191 -0.09968639 0.07054886 -0.04911624]
-1.0	[0.13363062 0.35355339 0.3131416 -0.02800839 -0.6777343 0.14952961 -0.44505459 0.47689832]
-1.0	[0.13363062 0.35355339 -0.50123695 0.46732384 -0.41820156 -0.55114107 0.18686658 -0.25284609]
-1.0	[0.13363062 0.35355339 0.3131416 -0.02800839 0.25525492 0.26698604 -0.18266698 -0.69859049]
-1.0	[0.13363062 0.35355339 0.3131416 -0.02800839 0.42116371 -0.60025722 -0.34095117 0.22912106]
-1.0	[0.13363062 0.35355339 0.3131416 -0.02800839 0.25525492 0.26698604 0.75028126 0.23640585]

Рисунок 3.16 – Таблиця власних значень і відповідних їх власних векторів введеного графу

Cayley's method:

t(G)=262144

The Cayley method takes an average of 5.304099999989376e-07 seconds to execute.

Press any key to continue …

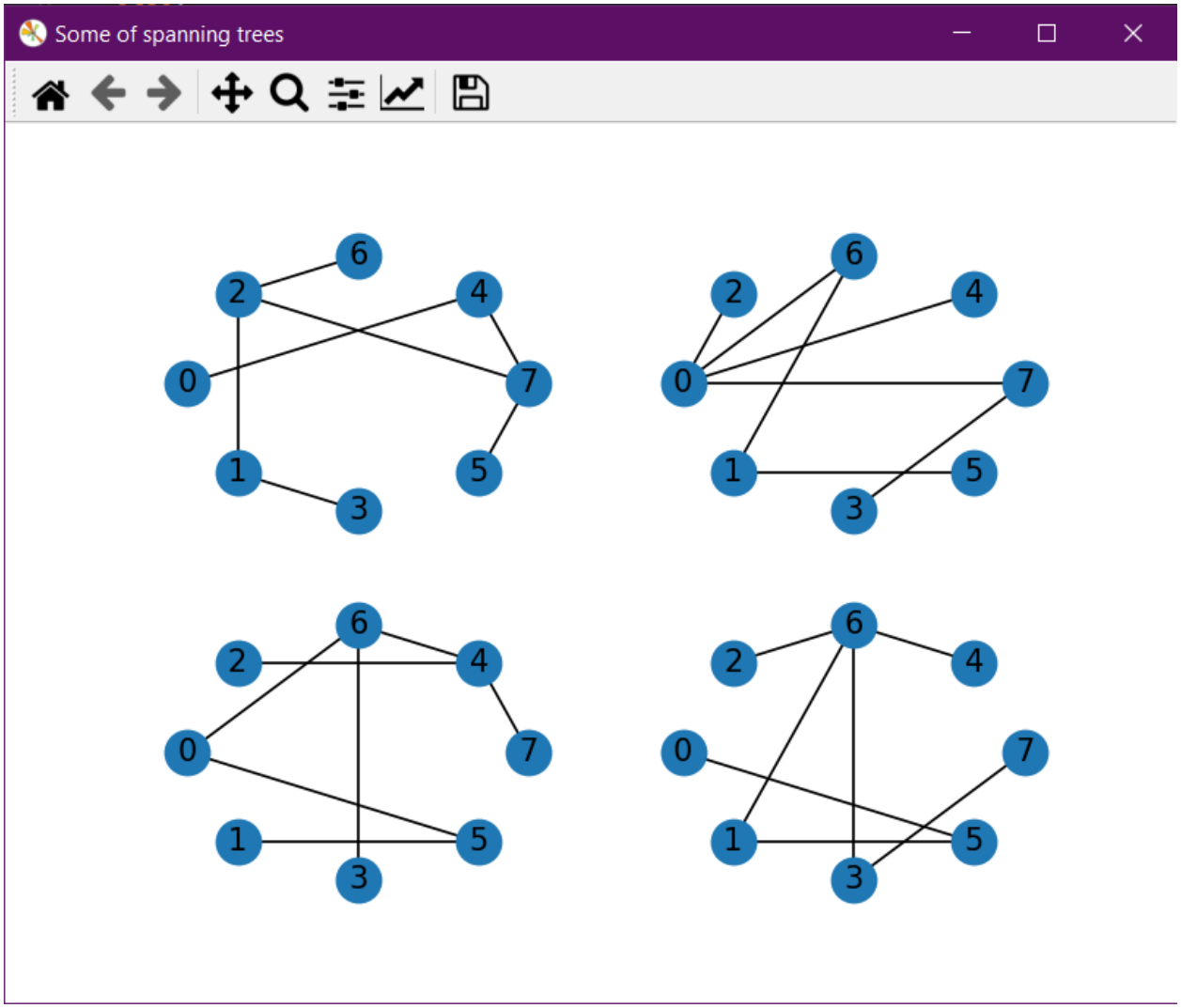


Рисунок 3.17 – Деякі кістякові дерева введеного графа

Process finished with exit code 0

3.6 Висновки за розділом

У цьому розділі було описано розроблений програмний продукт, його архітектуру і приведено інструкція з використання програми, а також був уточнений алгоритм, за яким програма була побудована щоб знаходити кількість кістякових дерев графа.

Програмний продукт було протестовано на графах різного вигляду з різними розмірностями і валентностями. Було доведено справну роботу алгоритму з різними вхідними даними і при використанні різних методів розрахунку

Надалі буде доцільно спрямувати дослідження на розробку графічного інтерфейсу системи, а також вивід покрокового обчислення кількості кістякових дерев графа, щоб надати користувачу більш зручний спосіб відслідковування ходу виконання.

РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ

4.1 Постановка завдання

У даному розділі проводиться оцінка основних характеристик програмного продукту, призначеного для знаходження кількості кістякових дерев графа. Інтерфейс користувача був розроблений за допомогою мови програмування Python у середовищі розробки PyCharm. Програмний продукт призначений для використання на персональних комп’ютерах під управлінням операційних систем Windows.

У цьому розділі будуть проаналізовані різні варіанти реалізації модулю, за допомогою чого буде обрано оптимальний, зважаючи на економічні фактори, а також на характеристики продукту, які впливають на продуктивність і сумісність з апаратним забезпеченням.

Для цього було використано апарат функціонально-вартісного аналізу (ФВА). Функціонально-вартісний аналіз – це технологія, яка дозволяє оцінити реальну вартість продукту або послуги незалежно від організаційної структури компанії. Як прямі, так і побічні витрати розподіляються по продуктам та послугам у залежності від потрібних на кожному етапі виробництва обсягів ресурсів. Виконані на цих етапах дії у контексті методу ФВА називаються функціями.

Мета ФВА полягає у забезпеченні правильного розподілу ресурсів, виділених на виробництво продукції або надання послуг, на прямі та непрямі витрати. У даному випадку – аналізі функцій програмного продукту і виявлення усіх витрат на реалізацію цих функцій.

На практиці цей метод працює за наступним алгоритмом:

а) Визначається послідовність функцій, що необхідних для виробництва продукту. Спершу – всі можливі, потім їх розподіляють на дві групи за впливом на вартість продукту: ті, що впливають, і ті, що не впливають. На цьому ж етапі

оптимізується сама послідовність скорочення кроків, що не впливають на цінність і відповідно витрати.

б) Визначаються повні річні витрати і кількість робочих годин для кожної з функцій

в) Визначається кількісна характеристика джерела витрат для кожної функції (на основі попереднього пункту).

г) Проводиться кінцевий розрахунок витрат на виробництво продукту.

4.2 Постановка задачі техніко-економічного аналізу

У роботі застосовується методі ФВА для проведення техніко-економічного аналізу розробки.

Відповідно до цього варто обирати і систему показників якості програмного продукту. Технічні вимоги до продукту наступні:

– функціонувати на персональних комп’ютерах зі стандартним набором компонентів;

– забезпечувати високу швидкість обробки великих об’ємів даних у реальному часі;

– забезпечувати зручність і простоту взаємодії з користувачем або з розробником програмного забезпечення у випадку використання його як модуля;

– передбачати мінімальні витрати на впровадження програмного продукту.

4.2.1 Обґрунтування функцій програмного продукту

Головна функція F_0 – розробка програмного продукту, який визначає тип графу, приводить його до необхідного, якщо це потрібно, і рахує кількість

кістякових дерев. Виходячи з конкретної мети, можна виділити наступні основні функції ПП (програмного продукту):

1) F_1 – вибір мови програмування;
2) F_2 – метод репрезентації результатів;
3) F_3 – інформаційне вікно.

Кожна з основних функцій може мати декілька варіантів реалізації.

Функція F1:

а) мова програмування Python;

б) мова програмування Java;

Функція F2:

а) бібліотека NetworkX;

б) бібліотека JGraphT.

Функція F3:

а) Виведення результатів у окремий файл;

б) Розробка графічного інтерфейсу користувача.

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рисунок 4.1).

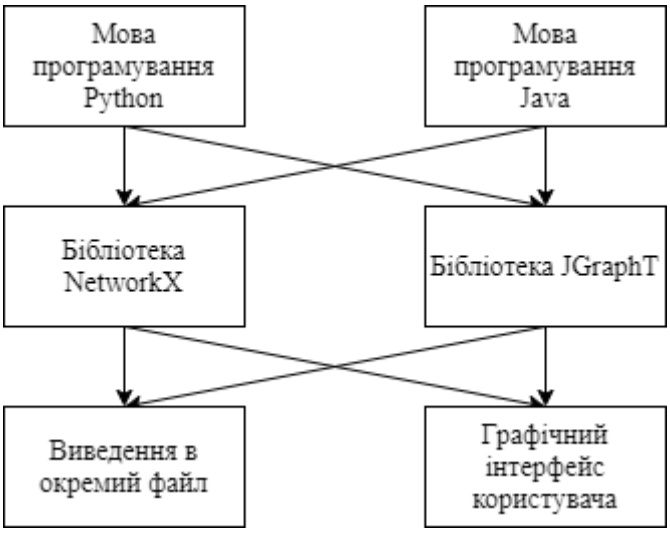


Рисунок 4.1 – Морфологічна карта

4.2.2 Варіанти реалізації основних функцій

Морфологічна карта відображає всі можливі комбінації варіантів реалізації функцій, які складають повну множину варіантів програмного продукту (рисунок 4.1). На основі цієї карти побудовано позитивно-негативну матрицю варіантів основних функцій. (таблиця 4.1)

Таблиця 4.1 – Позитивно-негативна матриця

Основні функції	Варіанти реалізації	Переваги	Недоліки
F1	*А*	Менший об’єм програмного коду, багато спеціалізованих бібліотек для обробки даних	Повільніше виконання програм
	Б	Висока швидкодія	Потрібно більше часу для написання коду
F2	*А*	Широкий функціонал, простота створенні	Складніший у вивченні
	Б	Простий у застосуванні	Менше функцій для використання
F3	*А*	Простий у реалізації	Відсутність візуалізації вихідних даних
	Б	Стабільний у використанні	Потребує більше часу для написання коду

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути

тому, що вони не відповідають поставленим перед програмним продуктом задачам. Ці варіанти відзначені у морфологічній карті.

Функція F1:

Оскільки розрахунки проводяться з використанням великої кількості операцій над графами і матрицями, при чому масиви вхідних даних не є занадто великими, є сенс віддати перевагу більшому функціоналу. Враховуючи це, відкидаємо варіант Б)

Функція F2:

Оскільки основною задачею продукту є знаходження кількості кістякових дерев графу, то більш зручним є варіант, який надає більше можливостей. Отже, варіант Б) має бути відкинутий.

Функція F3:

Вибір між варіантами А) і Б) не є очевидним, тож розглянемо детальніше обидва.

Таким чином, будемо розглядати такі варіанти реалізації ПП:

F1А – F2А – F3А

F1 – F2А – F3Б

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

4.3 Обґрунтування системи параметрів програмного продукту

4.3.1 Опис параметрів

Зважаючи на визначені функції, які повинен реалізовувати даний ПП, визначимо параметри, що будуть використовуватися для рахування коефіцієнта технічного рівня.

Для того, щоб охарактеризувати ПП, будемо використовувати наступні параметри:

а) X1 – час ознайомлення з мовою програмування;

б) X2 – об’єм пам’яті для збереження даних;

в) X3 – час обробки даних;

г) X4 – потенційний об’єм програмного коду.

X1: Відображає час, необхідний для написання програмного коду.

X2: Відображає об’єм пам’яті в оперативній пам’яті персонального комп’ютера, необхідний для збереження та обробки змінних, необхідних для виконання програми.

X3: Відображає час, який витрачається на дії з обробки даних.

X4: Показує розмір програмного коду який необхідно створити безпосередньо розробнику.

4.3.2 Кількісна оцінка параметрів

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію ПП як показано у таблиці 4.2.

Таблиця 4.2 – Основні параметри програмного продукту

Назва Параметра	Умовні позначення	Одиниці виміру	Значення параметра		
			гірші	середні	кращі
Час ознайомлення з мовою програмування	X1	год	10	6	4
Об’єм пам’яті для збереження даних	X2	Мб	32	16	8
Час обробки даних алгоритмом	X3	мс	1000	500	100

Кінець таблиці 4.2

Потенційний об’єм програмного коду	X4	кількість рядків коду	1000	800	500

За даними таблиці 4.2 будуються графічні характеристики параметрів – рис. 4.2 – рис. 4.5.

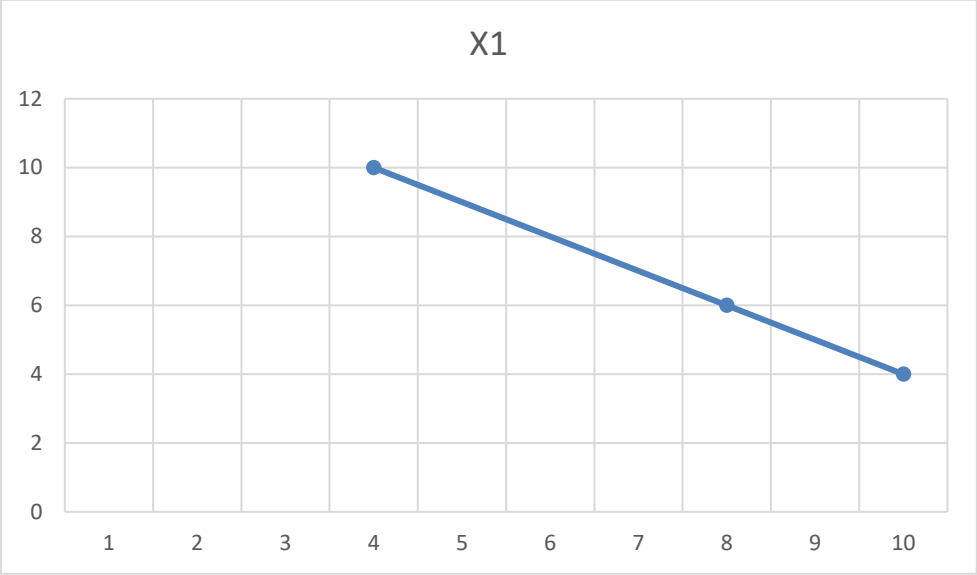


Рисунок 4.2 – X1, час ознайомлення з мовою програмування

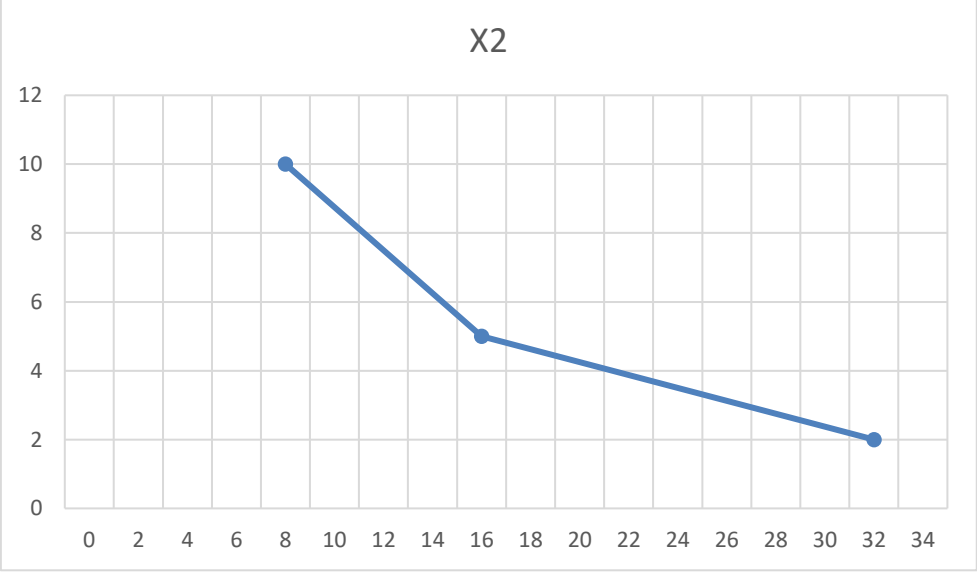


Рисунок 4.3 – X2, об’єм пам’яті для збереження даних

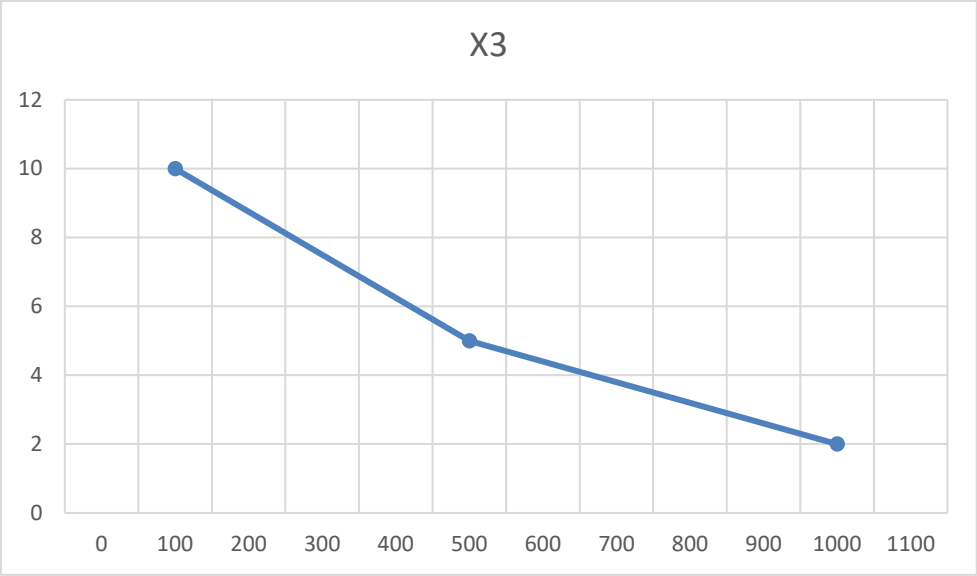


Рисунок 4.4 – X3, час обробки даних алгоритмом

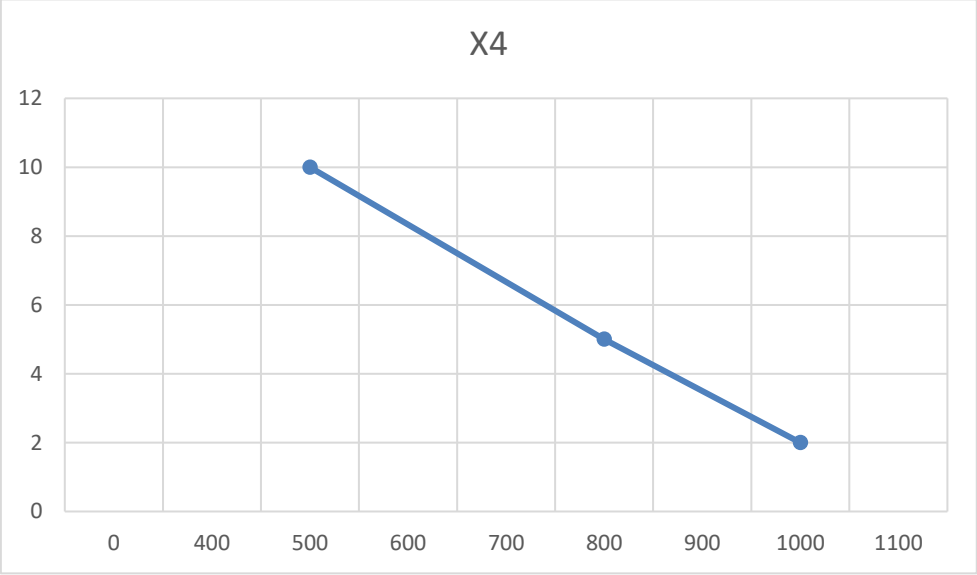


Рисунок 4.5 – X4, потенційний об’єм програмного коду

4.3.3 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка програмного продукту, який дає найбільш точні результати при знаходженні кількості кістякових дерев графа.

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Визначення коефіцієнтів значимості передбачає:

а) визначення рівня значимості параметра шляхом присвоєння різних рангів;

б) перевірку придатності експертних оцінок для подальшого використання;

в) визначення оцінки попарного пріоритету параметрів;

г) обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 4.3.

Таблиця 4.3 – Результати ранжування параметрів

Позначення параметра	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхилення Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
X1	Час ознайомлення з мовою програмування	год	1	2	1	2	1	2	2	11	-6,5	42,25
X2	Об’єм пам’яті для збереження даних	Мб	2	1	2	1	2	1	1	10	-7,5	56,25
X3	Час обробки даних алгоритмом	Мс	4	3	4	3	4	3	3	24	60,5	42,25
X4	Потенційний об’єм програмного коду	кількість рядків коду	3	4	3	4	3	4	4	25	7,5	56,25
	Разом		10	10	10	10	10	10	10	70	0	197

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 70, \quad (4.1)$$

де N – число експертів;

n – кількість параметрів.

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 17{,}5; \quad (4.2)$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T; \quad (4.3)$$

Сума відхилень по всім параметрам повинна дорівнювати 0;

г)загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 197. \quad (4.4)$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3-n)} = \frac{12 \cdot 197}{7^2(4^3-4)} = 0.8 > W_k = 0{,}67. \quad (4.5)$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування з таблиці 4.3, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 4.4.

Таблиця 4.4 – Попарне порівняння параметрів

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	<	>	<	>	<	>	>	>	1,5
X1 і X3	<	<	<	<	<	<	<	<	0,5
X1 і X4	<	<	<	<	<	<	<	<	0,5
X2 і X3	<	<	<	<	<	<	<	<	0,5
X2 і X4	<	<	<	<	<	<	<	<	0,5
X3 і X4	>	<	>	<	>	<	<	<	0,5

Числове значення, що визначає ступінь переваги *i*–го параметра над *j*–тим, a_{ij} визначається наступним чином:

$$a_{ij} = \begin{cases} 1.5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j. \\ 0.5 \text{ при } X_i < X_j \end{cases} \quad (4.6)$$

З отриманих числових оцінок переваги складемо матрицю $A = \|a_{ij}\|$.

Для кожного параметра зробимо розрахунок вагомості K_{b_i} за наступними формулами:

$$K_{b_i} = \frac{b_i}{\sum_{i=1}^{n} b_i}, \text{ де } b_i = \sum_{i=1}^{N} a_{ij}. \quad (4.7)$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 5%). На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K_{b_i}{}' = \frac{b_i'}{\sum_{i=1}^{n} b_i'}, \text{ де } b_i' = \sum_{i=1}^{N} a_{ij} b_j. \quad (4.8)$$

Як видно з таблиці 4.5, різниця значень коефіцієнтів вагомості не перевищує 5%, тому більшої кількості ітерацій не потрібно.

Таблиця 4.5 – Розрахунок вагомості параметрів

Параметри x_i	Параметри x_j				Перша ітерація		Друга ітерація		Третя ітерація	
	X1	X2	X3	X4	b_i	$K_{b\text{i}}$	b_i^1	$K_{\text{вi}}^1$	b_i^2	$K_{\text{вi}}^2$
X1	1,0	1,5	0,5	0,5	3,5	0,219	12,25	0,208	44,875	0,207
X2	0,5	1,0	0,5	0,5	2,5	0,156	9,25	0,157	34,125	0,158
X3	1,5	1,5	1,0	0,5	4,5	0,281	16,25	0,275	59,125	0,274
X4	1,5	1,5	1,5	1,0	5,5	0,344	21,25	0,306	77,875	0,361
Всього:					16	1	59	1	216	1

4.3 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів *X2* (об’єм пам’яті для збереження даних) та *X1* (швидкодія мови програмування) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра *X3 (*час обробки даних) обрано не найгіршим (не максимальним), тобто це значення відповідає або варіанту б) 500 мс або варіанту в) 100 мс.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується наступним чином (таблиця 4.6):

$$K_K(j) = \sum_{i=1}^{n} K_{вi,j} B_{i,j}\,, \tag{4.9}$$

де n – кількість параметрів;

$K_{вi}$– коефіцієнт вагомості i–го параметра;

B_i – оцінка i–го параметра в балах.

Таблиця 4.6 – Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	Варіант реалізації функції	Параметри	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1	А	X1	6	8	0,207	1,656
F2	А	X2	16	5	0,158	0,79
F3	А	X3	500	5	0,274	1,37
		X4	500	10	0,361	3,61
	Б	X3	300	7	0,274	1,918
		X4	900	3	0,361	1,083

За даними з таблиці 4.6 за формулою:

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \ldots + K_{\text{ТУ}}[F_{zk}], \tag{4.10}$$

визначаємо рівень якості кожного з варіантів:

$$K_{K1} = 1{,}656 + 0{,}79 + 1{,}37 + 3{,}61 = 7{,}426, \quad (4.11)$$

$$K_{K2} = 1{,}656 + 0{,}79 + 1{,}918 + 1{,}083 = 5{,}447. \quad (4.12)$$

Як видно з розрахунків, кращим є перший варіант, для якого коефіцієнт технічного рівня має найбільше значення.

4.4 Економічний аналіз варіантів розробки програмного продукту

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1. обробка вхідних даних та програмна реалізація методу знаходження кількості кістякових дерев графа;

2. практична реалізація методу знаходження кількості кістякових дерев графа на прикладі складання розкладу;

За ступенем новизни перше і друге завдання відносяться до групи А. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 – до групи 2.

Для реалізації завдання 1 та завдання 2 використовується інформація у вигляді даних, які вводяться у матрицю суміжності.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань. Загальна трудомісткість обчислюється як

$$T_O = T_P \cdot K_П \cdot K_{СК} \cdot K_M \cdot K_{СТ} \cdot K_{СТ.М}, \quad (4.13)$$

де T_P – трудомісткість розробки програмного продукту;

$K_П$ – поправочний коефіцієнт;

$K_{СК}$ – коефіцієнт на складність вхідної інформації;

K_M – коефіцієнт рівня мови програмування;

$К_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм;

$К_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення.

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру ступеня новизни А та групи складності алгоритму 1, трудомісткість дорівнює: $Т_Р$ = 90 людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $К_П$ = 0,9. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівний 1: $К_{СК}$ = 1. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $К_{СТ}$ = 0,8. Тоді загальна трудомісткість програмування першого завдання дорівнює:

$$Т_1 = 90 \cdot 0,9 \cdot 0,8 = 64,8 \text{ людино-днів.} \quad (4.14)$$

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм другої групи складності, ступінь новизни А), тобто $Т_Р$ = 36 людино-днів, $К_П$ =0,9, $К_{СК}$ = 1, $К_{СТ}$ =0,8:

$$Т_2 = 36 \cdot 0,9 \cdot 0,8 = 25,92 \text{ людино-днів.} \quad (4.15)$$

Складаємо трудомісткість відповідних завдань реалізації програми для кожного з обраних варіантів, щоб отримати їх трудомісткість:

$$Т_0 = (64,8 + 25,92) \cdot 8 = 725,76 \text{ людино-годин.} \quad (4.16)$$

В розробці беруть участь два програмісти: один з окладом 10000 грн., інший – 13000 грн. Визначимо середню зарплату за годину за формулою:

$$С_Ч = \frac{М}{T_m \cdot t} грн., \quad (4.17)$$

де М – місячний оклад працівників;

T_m – кількість робочих днів тиждень;

t – кількість робочих годин в день.

$$С_Ч = \frac{10000+13000}{2 \cdot 21 \cdot 8} \approx 68{,}45 \text{ грн.} \quad (4.18)$$

Тоді, розрахуємо заробітну плату за формулою:

$$С_{ЗП} = С_Ч \cdot Т_i \cdot К_Д, \quad (4.19)$$

де $С_Ч$ – величина погодинної оплати праці програміста;

$Т_i$ – трудомісткість відповідного завдання;

$К_Д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за виконання завдання становить:

$$С_{ЗП} = 68{,}45 \cdot 725{,}76 \cdot 1{,}2 \approx 59616{,}00 \text{ грн.} \quad (4.20)$$

Відрахування на соціальний внесок становить 22%:

$$С_{ВІД} = С_{ЗП} \cdot 0{,}22 = 59616{,}00 \cdot 0{,}22 = 13115{,}52 \text{ грн.} \quad (4.21)$$

Тепер визначимо витрати на оплату однієї машино-години. ($С_М$)

Так як одна ЕОМ обслуговує одного програміста з коефіцієнтом зайнятості 0,2 то для однієї машини в середньому отримаємо:

$$С_Г = 12 \cdot М \cdot К_З = \frac{12 \cdot (10000+13000) \cdot 0{,}2}{2} = 27600 \text{ грн.} \quad (4.22)$$

З урахуванням додаткової заробітної плати:

$$С_{ЗП} = С_Г \cdot (1 + К_З) = 27600 \cdot (1 + 0,2) = 33120 \text{ грн.} \quad (4.23)$$

Відрахування на соціальний внесок:

$$С_{ВІД} = С_{ЗП} \cdot 0,22 = 33120 \cdot 0,22 = 7286,4 \text{ грн.} \quad (4.24)$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 10000 грн.

$$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1,15 \cdot 0,25 \cdot 10000 = 2875 \text{ грн.}, \quad (4.25)$$

де $К_{ТМ}$ – коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

$К_А$ – річна норма амортизації;

$Ц_{ПР}$ – договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1,15 \cdot 10000 \cdot 0,05 = 575 \text{ грн.}, \quad (4.26)$$

де $К_Р$ – відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$Т_{ЕФ} = (Д_К – Д_В – Д_С – Д_Р) \cdot t_З \cdot К_В = (365 – 104 – 11 – 16) \cdot 8 \cdot 0,95 = 1778,4 \text{ годин}, \quad (4.27)$$

де $Д_К$ – календарна кількість днів у році;

$Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів;

$Д_Р$ – кількість днів планових ремонтів устаткування;

t – кількість робочих годин в день;

$К_В$ – коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 1778,4 \cdot 0,22 \cdot 0,78 \cdot 2,7515 = 839,68 \text{ грн.}, \quad (4.28)$$

де $N_С$ – середньо-споживча потужність приладу;

$К_З$ – коефіцієнтом зайнятості приладу;

$Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$С_Н = Ц_{ПР} \cdot 0.67 = 10000 \cdot 0,67 = 6700 \text{ грн.} \quad (4.29)$$

Тоді, річні експлуатаційні витрати будуть:

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н, \quad (4.30)$$

$$С_{ЕКС} = 33120 + 7286,4 + 2875 + 575 + 839,68 + 6700 = 51396,08 \text{ грн.}$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$С_{М\text{-}Г} = С_{ЕКС} / Т_{ЕФ} = 51396,08 / 1778,4 = 28,9 \text{ грн/год.} \quad (4.31)$$

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$С_М = С_{М\text{-}Г} \cdot Т_0, \quad (4.32)$$

$$С_М = 28,9 \cdot 725,76 = 18826,21 \text{ грн.}$$

Накладні витрати складають 67% від заробітної плати:

$$С_Н = С_{ЗП} \cdot 0,67, \quad (4.33)$$
$$С_Н = 33120 \cdot 0,67 = 22190,4 \text{ грн.}$$

Отже, вартість розробки ПП за варіантами становить:

$$С_{ПП} = С_{ЗП} + С_{ВІД} + С_М + С_Н, \quad (4.34)$$
$$С_{ПП} = 33120 + 7286,4 + 18826,21 + 22190,4 = 81423,01 \text{ грн.}$$

4.5 Вибір кращого варіанту програмного продукту техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$К_{ТЕРj} = К_{Кj} / С_{Фj}, \quad (4.35)$$
$$К_{ТЕР1} = 7,426 / 81423,01 \approx 9,12 \times 10^{-5}$$
$$К_{ТЕР2} = 5,447 / 81423,01 \approx 6,69 \times 10^{-5}.$$

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1} = 9,12 \times 10^{-5}$.

4.6 Висновки за розділом

В даному розділі було проведено функціонально-вартісний аналіз програмного продукту, який був створений в рамках дипломної роботи. Повний

процес аналізу можливо умовно поділити на два: аналіз технічної складової продукту, де були визначені основні функції програмного продукту і сформована множина варіантів їх реалізації, і другий – аналіз економічної складової, у якому увага приділялася вибору найбільш економічно обґрунтованого варіанту.

Після виконання такого аналізу можна зробити висновок, що з варіантів функції F3, що залишилися після відбору варіантів для перших двох функцій, найкращим є перший варіант – виведення результатів в окремий файл, оскільки саме він має найбільший показник техніко-економічного рівня якості $К_{ТЕР1} = 9{,}12 \times 10^{-5}$.

Цей варіант реалізації програмного продукту має такі параметри:

– Мова програмування Python;
– Бібліотека NetworkX;
– Виведення результатів у окремий файл;

ВИСНОВКИ

У даній дипломній роботі було проведено аналіз методів спектральної теорії графів і їх застосуванні при вирішеннях комбінаторних задач, зокрема, задачі обчислення кількості кістякових дерев графа, а також розроблено алгоритм на основі зазначеного аналізу.

Проведені обчислення кількості кістякових дерев регулярних і нерегулярних графів різних розмірностей за допомогою методів спектрального аналізу, виконаний аналіз отриманих результатів.

Було розроблено програмний продукт, який спрощує користування кодом алгоритму, проводить покращені перевірки введених даних і розрахований на можливість помилки користувача.

Для подальшого вдосконалення програмного продукту варто впровадити функціонал покрокового відображення роботи алгоритму, графічний інтерфейс, який дозволить використовувати програму менш досвідченим користувачам, а також проаналізувати оптимальність програмної реалізації алгоритму і за можливості покращити її.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Цветкович Д., Дуб М., Захс Х. Спектры графов. Теория и применение. Киев: Наукова думка, 1984. 384 с.

2. Мельников О. И. Теория графов в занимательных задачах. Москва: Книжный дом «ЛИБРОКОМ», 2009. 232с.

3. Оре О. Теория графов. Москва «НАУКА», 1980. 336с.

4. Харари Ф. Теория графов. Москва: Едиториал УРСС, 2003. 296с.

5. Примеры графов. URL: https://studbooks.net/2257357/matematika_himiya_fizika/primery_grafov

6. Текст лекции по курсу «Дискретная математика», Самара: СГТУ, 2015. 93. URL: https://studfile.net/preview/4528848

7. Теоретико-множественные операции над графами/ URL: https://neerc.ifmo.ru/wiki/index.php?title=Теоретико-множественные_операции_над_графами

8. Карпов Д., Теория графов, 2017. 553 с. URL: https://logic.pdmi.ras.ru/~dvk/graphs_dk.pdf

9. Combinatorics. URL: https://brilliant.org/wiki/combinatorics/

10. Combinatorics. URL: https://www.britannica.com/science/combinatorics

11. Теоретико-множественные операции над графами, URL: https://neerc.ifmo.ru/wiki/index.php?title=Теоретико-множественные_операции_над_графами

12. Конспект по дискретной математике. Белорусско-Российский университет, 2016. 119. URL:https://studfile.net/preview/5615402/

13. Networkx tutorial. URL: https://networkx.org/documentation/stable/tutorial.html

14. Теория графов. URL:https://ru.wikipedia.org/wiki/Теория_графов

15. Спектральная теория графов. URL:https://ru.wikipedia.org/wiki/Спектральная_теория_графов

16. Мультиграф. URL:https://ru.wikipedia.org/wiki/Мультиграф

17. Combinatorics. URL: https://en.wikipedia.org/wiki/Combinatorics

18. Операции над графами. URL: https://ru.wikipedia.org/wiki/Операции_над_графами

19. Spectral graph theory. URL: https://en.wikipedia.org/wiki/Spectral_graph_theory

20. Weisstein, Eric W. "Dual Graph." From MathWorld--A Wolfram Web Resource. URL: https://mathworld.wolfram.com/DualGraph.html

21. Степень графа. URL: https://ru.wikipedia.org/wiki/Степень_графа

22. Brown W. On the non-existence of a type of regular graphs of girth 5. – Canad. J. Math., 1967, p. 644-648.

23. Sachs H. Über selbstkomplementäre Graphen // Publicationes Mathematicae Debrecen. — 1962. — T. 9. — C. 270—288.

24. Cvetković D. M. The generating function for variations with restrictions and paths of the graph and self-complementary graphs. – Univ. Beograd Publ. Elektrotehn. Fak., Ser. Mat. Fiz., 1971, N320 – N328, p.27 – 34

25. Cvetković D. M. A note on paths in the p-sum of graphs, Univ. Beograd, Publ. Elektrotehn. Fak., Ser. Mat. Fiz., 1970, N 302 – 319, p. 49 – 51.

26. Cvetković D., Graphs and their spectra (Thesis). - Univ. Beograd, Publ. Elektrotehn. Fak. Ser. Mat. Fiz., 1971, N 354 – 356, p. 1 – 50.

27. Кельманс А. К. ОГ числе деревьев графа. II. – Автоматика и телемеханика, 1966, № 2, с. 56 – 65.

28. Turán P. Egy gráfelmèleti szélsöértékfeladatról. – Mat. Fiz. Lapok, 1941, 48, p. 436 – 452.

29. Nosal E. Eigenvalues of Graphs. – Masterthesis. Univ. of Calgary, 1970.

30. MatPlotLib user’s guide. URL: https://matplotlib.org/stable/users/index.html

31. IGSpanningTreeCount exapmles http://szhorvat.net/mathematica/IGDocumentation/#igspanningtreecount

32. NumberOfSpanningTrees function documentation https://reference.wolfram.com/language/Combinatorica/ref/NumberOfSpanningTrees.html

33. Hutschenreuther H. Einfacher Beweis des Matrix-Gerüst-Satzes der Netzwerktheorie. – Wiss. Z. TH Ilmenau, 1967, 13, S. 403 – 404.

ДОДАТОК А ЛІСТИНГ ПРОГРАМНОГО ПРОДУКТУ

```
import sys
import timeit
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
import pylab
from click._compat import raw_input
from sympy import Matrix, symbols, factor, Eq, Function
from sympy.printing.pretty.pretty import pretty_print, pretty
import os
from datetime import datetime
from prettytable import PrettyTable, ALL



def create_log_dir():
    now = datetime.now()
    path = os.getcwd().replace('\\', '/') + f'/{str(now).replace(":", ".")} log'
    try:
        os.mkdir(path)
    except OSError:
```

```
        print("Creation of the directory %s failed" % path)
    else:
        print("Successfully created the directory %s " % path)
        return path


def yes_or_no(question):
    reply = str(raw_input(question + ' (y/n): ')).lower().strip()
    if reply[0] == 'y':
        return True
    if reply[0] == 'n':
        return False
    else:
        return yes_or_no("Please enter ")


def draw_graph(array, window_name='Figure'):
    G = nx.from_numpy_matrix(array, create_using=nx.MultiGraph)
    pos = nx.circular_layout(G)
    fig = pylab.gcf()
    fig.canvas.manager.set_window_title(window_name)
    nx.draw(G, pos, with_labels=True, arrows=False)
```

```
        plt.show()
        return G


    def draw_some_of_spanning_trees(G):
        ST = get_all_spanning_trees(G)
        pos = nx.circular_layout(ST[0])
        fig, axs = plt.subplots(2, 2)
        fig.canvas.manager.set_window_title('Some of spanning trees')
        for i, ax in zip(range(min(4, len(ST))), np.ndenumerate(axs)):
            nx.draw(ST[i], pos, with_labels=True, arrows=False, ax=ax[1])
        plt.show()


    def huts(n, r, eigen_values, char_pol, printing=False):
        if printing:
            x, r_symbol, n_symbol, lamda, G = symbols('x r n lamda G')
            t = Function('t')(G)
            expr = (char_pol/(lamda-r)*(1/n_symbol)).subs(lamda, r_symbol)
            print(pretty(Eq(Eq(t, expr), expr.subs([(r_symbol, r), (n_symbol, n)]),
evaluate=False)))
```

```
            t = [(r - eigen_value) for eigen_value in eigen_values[np.around(eigen_values) != r]]
            t.append(1 / n)
            return round(np.product(t))


        def kirchhoff(array, printing=False):
            degree_matrix = np.zeros(array.shape)
            np.fill_diagonal(degree_matrix, array.sum(axis=1))
            if printing:
                print('t(G)=M(D-A), where M - any minor of matrix, D - Degree Matrix, A - Adjacency Matrix')
                print(f'A=\n{array}')
                print(f'D=\n{degree_matrix}')
                print(f'L=\n{degree_matrix - array}')
                print(f't(G)=M(L)={round(np.linalg.det((degree_matrix - array)[1:, 1:]))}')
            laplacian_matrix = degree_matrix - array
            return round(np.linalg.det(laplacian_matrix[1:, 1:]))


        def cayley(n):
            return n ** (n - 2)
```

```
def get_characteristic_polynomial(array):
    matrix = Matrix(array)
    lamda = symbols('lamda')
    return factor(matrix.charpoly(lamda).as_expr())



def get_matrix(message):
    length_check = False
    while not length_check:
        print(message)
        array = np.array([[x for x in input().split(' ') if x != '']], int)
        if array.shape[1] < 2:
            # clear_console()
            if yes_or_no('n should be ≥2. Want re-write input?'):
                continue
            else:
                sys.exit('Program got adjacency matrix with n<2')
        else:
            length_check = True
    for i in range(1, array.shape[1]):
```

```
            print(*array[:, i], end=' ')
            array = np.append(array, [np.append(array[:, i], list(map(int, [x for x in input().split(' ') if x != ''])))],
                              0)
        print('Matrix read successfully')
        return array


    def get_max_valency(array):
        return sum(array[array.sum(axis=1).argmax()])


    def graph_is_regular(array):
        degrees = np.sum(array, axis=1, dtype=int)
        return min(degrees) == max(degrees)


    def graph_is_complete(array):
        expected_array = np.ones(array.shape)
        np.fill_diagonal(expected_array, 0)
        return np.array_equal(expected_array, array)
```

```
def add_loops(array, r):
    np.fill_diagonal(array, (r - array.sum(axis=1))+array.diagonal())
    return array



def get_eigens(array):
    return np.linalg.eig(array)



def _expand(G, explored_nodes, explored_edges):
    frontier_nodes = list()
    frontier_edges = list()
    for v in explored_nodes:
        for u in nx.neighbors(G, v):
            if not (u in explored_nodes):
                frontier_nodes.append(u)
                frontier_edges.append([(u, v), (v, u)])
    return zip([explored_nodes | frozenset([v]) for v in frontier_nodes],
               [explored_edges | frozenset(e) for e in frontier_edges])
```

```
def get_all_spanning_trees(G, root=0):
    explored_nodes = frozenset([root])
    explored_edges = frozenset([])
    solutions = [(explored_nodes, explored_edges)]
    for ii in range(G.number_of_nodes() - 1):
        solutions = [_expand(G, nodes, edges) for (nodes, edges) in solutions]
        solutions = set([item for sublist in solutions for item in sublist])
    return [nx.from_edgelist(edges) for (nodes, edges) in solutions]



if __name__ == '__main__':
    adjacency_matrix = get_matrix("Enter adjacency matrix row-by-row with elements split with space")
    adjacency_matrix_Huts = np.copy(adjacency_matrix)
    print(adjacency_matrix)

    r = get_max_valency(adjacency_matrix)
    print('r=', r, "\nn=", adjacency_matrix.shape[0])
    G = draw_graph(adjacency_matrix, 'Inputted graph')
    if graph_is_regular(adjacency_matrix):
        print("Graph regular", end='')
        if graph_is_complete(adjacency_matrix):
```

```
            print(' and complete')
        else:
            print(' but not complete')
    else:
        adjacency_matrix_Huts = add_loops(adjacency_matrix_Huts, r)
        print(f"Graph not regular. Adding self-loops...\nNew graph for Huts method:\n{adjacency_matrix_Huts}\n")
        draw_graph(adjacency_matrix_Huts, 'New graph')
    char_pol = get_characteristic_polynomial(adjacency_matrix_Huts)
    print('Characteristic polynomial for adjacency matrix looks like:')
    pretty_print(char_pol)

    eigen_values, eigen_vectors = get_eigens(adjacency_matrix_Huts)
    eigen_table = PrettyTable()
    eigen_table.add_column('Eigen value', eigen_values.round(2))
    eigen_table.add_column('Eigen vector', eigen_vectors)
    eigen_table.hrules = ALL
    eigen_table.horizontal_char = '~'
    eigen_table.junction_char = '*'
    print(eigen_table)

    iterations = 10000
```

```
        if graph_is_complete(adjacency_matrix):
            print('Cayley\'s method:\nt(G)=', end='')
            print(cayley(adjacency_matrix.shape[0]))
            time = timeit.timeit(stmt='cayley(adjacency_matrix.shape[0])',
                                 number=iterations, globals=globals()) / iterations
            print(f"The Cayley method takes an average of {time} seconds to execute.\n")
        else:
            print('Huts\' method:')
            huts(adjacency_matrix_Huts.shape[0], r, eigen_values, char_pol, printing=True)
            time = timeit.timeit(stmt='huts(adjacency_matrix_Huts.shape[0], r, eigen_values, char_pol, '
                                      'printing=False)', number=iterations, globals=globals()) / iterations
            print(f"The Huts method takes an average of {time} seconds to execute.\n")
            print('Kirchhoff\'s method:')
            kirchhoff(adjacency_matrix, printing=True)
            time = timeit.timeit(stmt='kirchhoff(adjacency_matrix, printing=False)',
                                 number=iterations, globals=globals()) / iterations
            print(f"The Kirchhoff method takes an average of {time} seconds to execute.")
        os.system("pause")
```

draw_some_of_spanning_trees(G)

ДОДАТОК Б ПРЕЗЕНТАЦІЯ

ВИКОРИСТАННЯ СПЕКТРАЛЬНОЇ ТЕОРІЇ ГРАФІВ ДЛЯ ВИРІШЕННЯ КОМБІНАТОРНИХ ЗАДАЧ

Виконав:
Студент 4 курсу групи КА-73
Расторгуєв Роман

Науковий керівник:
доцент, канд. фіз.- мат. наук Стусь О. В.

АКТУАЛЬНІСТЬ ДОСЛІДЖЕННЯ

- Комбінаторика включає комбінаторну оптимізацію, яка має велику різноманітність застосувань – складання авіаційних мап, алгоритми служб таксі і доставки для вибору оптимального водія для виконання замовлення, оптимізація шляху.
- Кістякові дерева використовуються для мінімізації витрат на проведення електромереж, дротових з'єднань, трубопроводів тощо.
- На кістякових деревах побудовано алгоритм приведення мережі до деревоподібної топології у телекомунікаційних мережах STP, який використовується і досі.

- **Об’єкт дослідження:** кількість кістякових дерев графа.
- **Предмет дослідження:** методи спектральної теорії графів.
- **Мета роботи:** аналіз методів спектральної теорії графів для розв’язання задач комбінаторики та обчислення кількості кістякових дерев графа. Подальша розробка програмного продукту для обчислення кількості кістякових дерев графа на основі проведеного аналізу.

3

ПОСТАНОВКА ЗАДАЧІ

- Проаналізувати методи знаходження кількості кістякових дерев графа.
- Порівняти інструменти програмування для реалізації і візуалізації обраних алгоритмів.
- Розробити програму для знаходження кількості кістякових дерев для різних типів вихідних графів з впровадженням різних алгоритмів

4

ОСНОВНІ ПОНЯТТЯ

- **Спектральна теорія графів** – це наука про властивості графу у зв'язку з характеристичним поліномом, власними числами та власними векторами матриць, що пов'язаний з графом, такими як матриця суміжності або матриці Кірхгофа.
- **Матриця суміжності** – квадратна матриця порядку n (кількість вершин графа), a_{ij} – ті елементи дорівнюють числу ребер, що починаються у вершині x_i і закінчуються у вершині x_j.
- **Спектр графа** – множина власних значень матриці суміжності цього графа, що позначається як $S_P(G) = |\lambda_1, \dots, \lambda_n|$.
- **Кістякове дерево графа G** – підграф даного дерева, що є деревом, яке включає усі вершини графа G.

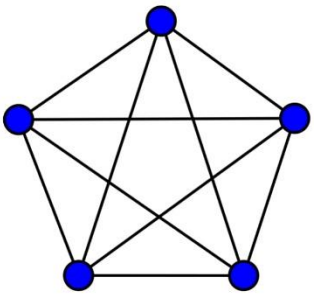
5

ВИЗНАЧЕННЯ КІЛЬКОСТІ КІСТЯКОВИХ ДЕРЕВ ГРАФА

Формулювання	Вимоги	Примітка
I. $t(G) = \frac{1}{n}\prod\mu$, μ – ненульові власні значення матриці $C = D - A$ (D – матриця валентностей)	Зв'язний мультиграф	
▪ $t(G) = \frac{(-1)^{n-1}}{n} C'_G(0) = \frac{1}{n} B_0^n(G)$		У термінах многочлена $C_G(\lambda)$ і многочлена Кельманса $B_\lambda^n(G)$
II. $t(G) = \frac{1}{n}\prod_{i=2}^{n}(r - \lambda_i) = \frac{1}{n} P'_G(r)$, де λ_i – власні значення мультиграфа G	Регулярний мультиграф	
III. $t(G) = M_{ij}, \forall\ 1 \le i, j \le n$, тобто мінор матриці Кірхгофа	Зв'язний мультиграф	Теорема Кірхгофа
IV. $t(K_n) = n^{n-2}$	Повний граф	Формула Келі

6

ГРАФ ДЛЯ ДЕМОНСТРАЦІЇ АЛГОРИТМІВ



Граф K_5

$$A = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix} \quad \mathrm{D} = \begin{pmatrix} 4 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 4 \end{pmatrix}$$

$$\mathrm{C} = \mathrm{D} - \mathrm{A} = \begin{pmatrix} 4 & -1 & -1 & -1 & -1 \\ -1 & 4 & -1 & -1 & -1 \\ -1 & -1 & 4 & -1 & -1 \\ -1 & -1 & -1 & 4 & -1 \\ -1 & -1 & -1 & -1 & 4 \end{pmatrix}$$

7

ДЕМОНСТРАЦІЯ АЛГОРИТМІВ

I. $t(K_5) = \frac{1}{5}\prod\mu$

$$C_G(\lambda) = |\lambda I - C| = \begin{vmatrix} \lambda+4 & -1 & -1 & -1 & -1 \\ -1 & \lambda+4 & -1 & -1 & -1 \\ -1 & -1 & \lambda+4 & -1 & -1 \\ -1 & -1 & -1 & \lambda+4 & -1 \\ -1 & -1 & -1 & -1 & \lambda+4 \end{vmatrix} =$$

$$= \lambda^5 + 20\lambda^4 + 150\lambda^3 + 500\lambda^2 + 625\lambda$$

$\mu_{1,2,3,4} = -5$ (μ – власні значення, що не дорівнюють 0)

$$t(K_5) = \frac{1}{5} \times (-5)^4 = \frac{5^4}{5} = \mathbf{125}$$

- $t(K_5) = \frac{1}{5}B_0^n(K_5)$

$$B_\lambda^n(K_5) = \frac{(-1)^n}{\lambda} C_G(-\lambda) = \frac{-1\left((-\lambda)^5 + 20\lambda^4 + 150(-\lambda)^3 + 500\lambda^2 + 625(-\lambda)\right)}{\lambda}$$

$$= \lambda^4 - 20\lambda^3 + 150\lambda^2 - 500\lambda + 625$$

$$t(K_5) = \frac{625}{5} = \mathbf{125}$$

8

ДЕМОНСТРАЦІЯ АЛГОРИТМІВ

II. $t(K_5) = \frac{1}{5}\prod_{i=2}^{n}(4-\lambda_i)$

$$|\lambda I - A| = \begin{vmatrix} \lambda & -1 & -1 & -1 & -1 \\ -1 & \lambda & -1 & -1 & -1 \\ -1 & -1 & \lambda & -1 & -1 \\ -1 & -1 & -1 & \lambda & -1 \\ -1 & -1 & -1 & -1 & \lambda \end{vmatrix} =$$

$$= \lambda^5 - 10\lambda^3 - 20\lambda^2 - 15\lambda - 4$$

$$\lambda_1 = 4, \lambda_{2,3,4,5} = -1$$

$$t(K_5) = \frac{(4+1)^4}{5} = 5^3 = \mathbf{125}$$

III. $t(K_5) = M_{ij}$ – мінор матриці C $\forall 1 \le i,j \le 5$

$$M_{11} = (-1)^2 \begin{vmatrix} 4 & -1 & -1 & -1 \\ -1 & 4 & -1 & -1 \\ -1 & -1 & 4 & -1 \\ -1 & -1 & -1 & 4 \end{vmatrix} = \mathbf{125}$$

IV. $t(K_5) = 5^{5-2} = 5^3 = \mathbf{125}$

9

ПРИКЛАДИ РОБОТИ ПРОГРАМНОГО ПРОДУКТУ

Регулярний граф

Введення матриці:

```
Enter adjacency matrix row-by-row with elements split with space
0 1 0 1 1 0
1 0 1 1 0 0
0 1 0 0 1 1
1 1 0 0 0 1
1 0 1 0 0 1
0 0 1 1 1 0
Matrix read successfully
[[0 1 0 1 1 0]
 [1 0 1 1 0 0]
 [0 1 0 0 1 1]
 [1 1 0 0 0 1]
 [1 0 1 0 0 1]
 [0 0 1 1 1 0]]
r= 3
n= 6
```

Розрахунок власних значень і векторів:

```
Graph regular but not complete
Characteristic polynomial for adjacency matrix looks like:
 2                    2
λ ·(λ - 3)·(λ - 1)·(λ + 2)
```

Eigen value	Eigen vector
3.0	[-0.40824829 -0.40824829 -0.57735027 0.10989286 -0.55814831 -0.02909111]
1.0	[-0.40824829 -0.40824829 0.28867513 0.43591269 0.15119589 -0.48481932]
-2.0	[-0.40824829 0.40824829 -0.28867513 -0.43591269 0.15119589 -0.48481932]
-2.0	[-0.40824829 -0.40824829 0.28867513 -0.54580554 0.40695241 0.51391043]
0.0	[-0.40824829 0.40824829 0.57735027 -0.10989286 -0.55814831 -0.02909111]
-0.0	[-0.40824829 0.40824829 -0.28867513 0.54580554 0.40695241 0.51391043]

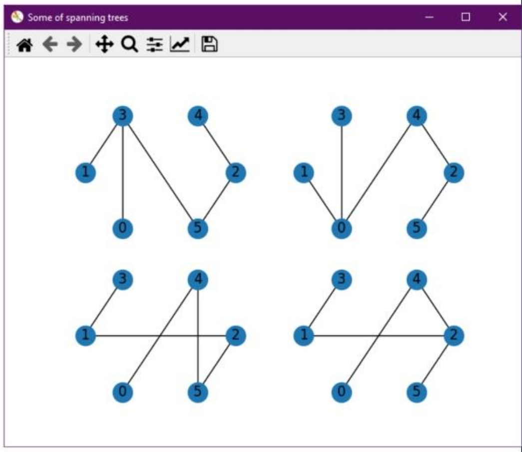
10

ПРИКЛАДИ РОБОТИ ПРОГРАМНОГО ПРОДУКТУ

Регулярний граф

Виведення кількості кістякових дерев графа, їх схематичних прикладів, а також часу виконання:

```
Huts' method:
         2              2
        r ·(r - 1)·(r + 2)
t(G) = ─────────────────── = 75
                n
The Huts method takes an average of 3.2352969999999456e-05 seconds to execute.
Kirchhoff's method:
t(G)=M(D-A), where M - any minor of matrix, D - Degree Matrix, A - Adjacency Matrix
A=                    D=
[[0 1 0 1 1 0]     [[3. 0. 0. 0. 0. 0.]
 [1 0 1 1 0 0]      [0. 3. 0. 0. 0. 0.]
 [0 1 0 0 1 1]      [0. 0. 3. 0. 0. 0.]
 [1 1 0 0 0 1]      [0. 0. 0. 3. 0. 0.]
 [1 0 1 0 0 1]      [0. 0. 0. 0. 3. 0.]
 [0 0 1 1 1 0]]     [0. 0. 0. 0. 0. 3.]]
L=
[[ 3. -1.  0. -1. -1.  0.]
 [-1.  3. -1. -1.  0.  0.]
 [ 0. -1.  3.  0. -1. -1.]
 [-1. -1.  0.  3.  0. -1.]
 [-1.  0. -1.  0.  3. -1.]
 [ 0.  0. -1. -1. -1.  3.]]
t(G)=M(L)=75
The Kirchhoff method takes an average of 1.9516280000001986e-05 seconds to execute.
```



11

ПРИКЛАДИ РОБОТИ ПРОГРАМНОГО ПРОДУКТУ

Нерегулярний граф

Введення матриці і додавання петель до графа:

```
Enter adjacency matrix row-by-row with elements split with space
0 0 1 1
0 0 1 1
1 1 0 1
1 1 1 0
Matrix read successfully
[[0 0 1 1]
 [0 0 1 1]
 [1 1 0 1]
 [1 1 1 0]]
r= 3
n= 4
Graph not regular. Adding self-loops...
New graph for Huts method:
[[1 0 1 1]
 [0 1 1 1]
 [1 1 0 1]
 [1 1 1 0]]
```

Розрахунок власних значень і векторів:

```
Characteristic polynomial for adjacency matrix looks like:
                         2
(λ - 3)·(λ - 1)·(λ + 1)
*~~~~~~~~~~~~*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*
| Eigen value |                           Eigen vector                          |
*~~~~~~~~~~~~*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*
|     3.0     |         [-0.5        -0.70710678  0.5          0.12126781]       |
*~~~~~~~~~~~~*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*
|     1.0     |         [-0.5         0.70710678  0.5          0.12126781]       |
*~~~~~~~~~~~~*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*
|     -1.0    | [-5.00000000e-01 -2.66704499e-16 -5.00000000e-01 -8.07262153e-01] |
*~~~~~~~~~~~~*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*
|     -1.0    | [-5.00000000e-01 -8.19779219e-17 -5.00000000e-01  5.64726528e-01] |
*~~~~~~~~~~~~*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*
```

12

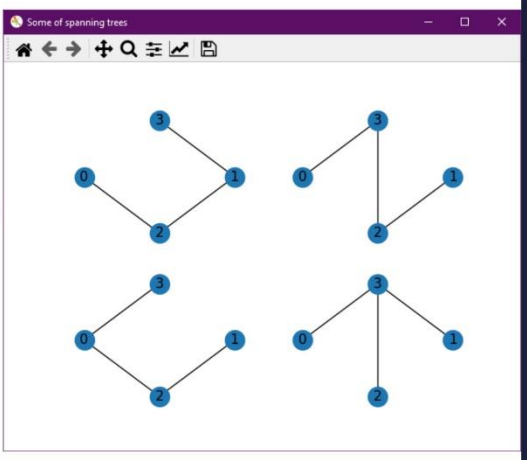
ПРИКЛАДИ РОБОТИ ПРОГРАМНОГО ПРОДУКТУ

Нерегулярний граф

Виведення кількості кістякових дерев графа, їх схематичних прикладів, а також часу виконання:

```
Huts' method:
                    2
          (r - 1)·(r + 1)
t(G) = ─────────────────── = 8
                  n
The Huts method takes an average of 2.645300000000006e-05 seconds to execute.

Kirchhoff's method:
t(G)=M(D-A), where M - any minor of matrix, D - Degree Matrix, A - Adjacency Matrix
A=              D=
[[0 0 1 1]    [[2. 0. 0. 0.]
 [0 0 1 1]     [0. 2. 0. 0.]
 [1 1 0 1]     [0. 0. 3. 0.]
 [1 1 1 0]]    [0. 0. 0. 3.]]
L=
[[ 2.  0. -1. -1.]
 [ 0.  2. -1. -1.]
 [-1. -1.  3. -1.]
 [-1. -1. -1.  3.]]
t(G)=M(L)=8
The Kirchhoff method takes an average of 2.4694909999999838e-05 seconds to execute.
```



ПРИКЛАДИ РОБОТИ ПРОГРАМНОГО ПРОДУКТУ

Повний граф

Введення матриці і додавання петель до графа:

```
Enter adjacency matrix row-by-row with elements split with space
0 1 1 1 1 1 1 1
1 0 1 1 1 1 1 1
1 1 0 1 1 1 1 1
1 1 1 0 1 1 1 1
1 1 1 1 0 1 1 1
1 1 1 1 1 0 1 1
1 1 1 1 1 1 0 1
1 1 1 1 1 1 1 0
Matrix read successfully
[[0 1 1 1 1 1 1 1]
 [1 0 1 1 1 1 1 1]
 [1 1 0 1 1 1 1 1]
 [1 1 1 0 1 1 1 1]
 [1 1 1 1 0 1 1 1]
 [1 1 1 1 1 0 1 1]
 [1 1 1 1 1 1 0 1]
 [1 1 1 1 1 1 1 0]]
r= 7
n= 8
Graph regular and complete
```

Розрахунок власних значень і векторів:

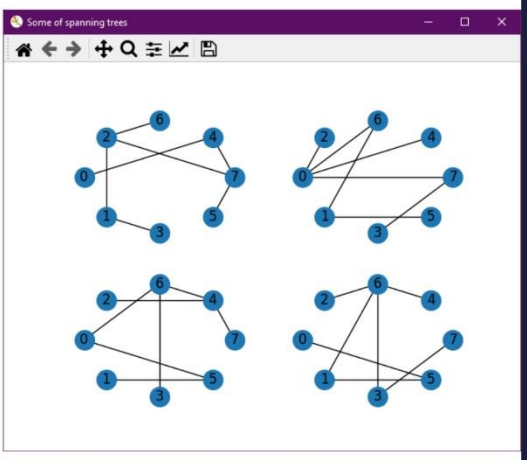
```
Characteristic polynomial for adjacency matrix looks like:
                7
(λ - 7)·(λ + 1)
*************************************************************************************
| Eigen value |                          Eigen vector                               |
*************************************************************************************
|     -1.0    | [-0.93541435  0.35355339  0.0673326  -0.13963183 -0.12519645  0.28379149 |
|             |                          0.13658387 -0.2034466 ]                    |
*************************************************************************************
|     7.0     | [ 0.13363062  0.35355339 -0.3174251   0.49868509  0.16096687  0.28379149 |
|             |                         -0.17560783  0.2615742 ]                    |
*************************************************************************************
|     -1.0    | [ 0.13363062  0.35355339 -0.50123695 -0.71434353  0.12849191 -0.09968639 |
|             |                          0.07054886 -0.04911624]                    |
*************************************************************************************
|     -1.0    | [ 0.13363062  0.35355339  0.3131416  -0.02800839 -0.6777343   0.14952961 |
|             |                         -0.44505459  0.47689832]                    |
*************************************************************************************
|             |                                  ...                                |
*************************************************************************************
|     -1.0    | [ 0.13363062  0.35355339  0.3131416  -0.02800839  0.25525492  0.26698604 |
|             |                         -0.18266698 -0.69859049]                    |
*************************************************************************************
```

ПРИКЛАДИ РОБОТИ ПРОГРАМНОГО ПРОДУКТУ

Повний граф

Виведення кількості кістякових дерев графа, їх схематичних прикладів, а також часу виконання:

```
Cayley's method:
t(G)=262144
The Cayley method takes an average of 3.67169999999999447e-07 seconds to execute.
```



15

ВИСНОВКИ

- Проаналізовано алгоритми знаходження кількості кістякових дерев графа
- Складено алгоритм на основі найбільш підходящих для програмної реалізації
- Розроблено програмний продукт для обчислення кількості кістякових дерев графа і порівняння швидкості реалізованих методів

16

ПОДАЛЬШИЙ РОЗВИТОК

- Створення графічного інтерфейсу користувача
- Оптимізація алгоритму для більш швидкої роботи з більшими об’ємами даних
- Покращена логіка обробки винятків для більш стабільної роботи і виключення можливості помилки зі сторони користувача
- Збільшення кількості додаткових функцій програми

17

ДЯКУЮ ЗА УВАГУ!

18