НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці

ДО ЗАХИСТУ ДОПУЩЕНО
В.о. завідувача кафедри
_____Олександр Коваль
«_____»_____2024р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Інженерія програмного забезпечення інтелектуальних кібер-фізичних систем в енергетиці»
спеціальності 121 Інженерія програмного забезпечення
на тему: «Програмне забезпечення для обчислення параметрів функціональної стійкості мережевих інформаційних систем на основі методу структурних перетворень та методу прямого перебору станів елементів системи»

Виконав:
студент IV курсу, групи ТВ-01
Курносенко Сергій Віталійович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник:
професор, д.т.н., професор Барабаш О.В. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент:
професор, д.т.н., професор Отрох С.І. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

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

Київ – 2024

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Навчально-науковий інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 Інженерія програмного забезпечення
Освітньо-професійна програма «Інженерія програмного забезпечення інтелектуальних кібер-фізичних систем в енергетиці»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Олександр КОВАЛЬ
(підпис)
«____»_____ 2024р

ЗАВДАННЯ

на дипломну роботу студенту

Курносенку Сергію Віталійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Програмне забезпечення для обчислення параметрів функціональної стійкості мережевих інформаційних систем на основі методу структурних перетворень та методу прямого перебору станів елементів системи
керівник роботи професор, д.т.н., Барабаш О.В.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від "30" травня 2024 року № 2197-с
2. Строк подання студентом роботи "12" червня 2024 року
3. Вихідні дані до роботи мови програмування PHP та JavaScript, платформи Symfony, React та Api Platform, база даних MariaDB, система контейнеризації Docker
4. Зміст (дипломної роботи) пояснювальної записки (перелік завдань, які потрібно розробити) розробити серверний застосунок для обчислення параметрів функціональної стійкості мережевих інформаційних систем, розробити інтерфейс для введення графів та відображення результатів, порівняти результати обчислень обох методів
5. Перелік ілюстративного матеріалу блок-схеми алгоритмів обчислень параметрів функціональної стійкості, порівняння продуктивності методів, інтерфейс користувача

6. Дата видачі завдання «30» жовтня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Строки виконання етапів роботи	Примітка
1	Отримання завдання	30.10.2023	
2	Дослідження предметної області	31.10.2023-31.12.2024	
3	Дослідження існуючих рішень	01.01.2024-15.02.2024	
4	Постановка вимог до проєктування системи	16.02.2024-14.04.2024	
5	Розробка програмного продукту	15.04.2024-12.05.2024	
6	Тестування програмного продукту	12.05.2024-19.05.2024	
7	Захист програмного продукту	14.05.2024	
8	Оформлення дипломної роботи	20.05.2024-31.05.2024	
9	Передзахист	04.06.2024	
10	Захист	18.06.2024	

Студент _____ Курносенко Сергій
(підпис) (ім’я, прізвище)

Керівник роботи _____ Олег Барабаш
(підпис) (ім’я, прізвище)

РЕФЕРАТ

Структура та обсяг дипломної роботи. Робота містить 51 сторінку, 22 рисунки, 6 таблиць, 2 додатки та 5 посилань.

Метою роботи є розробка та програмного забезпечення для обчислення параметрів функціональної стійкості мережевих інформаційних систем на основі методу структурних перетворень та методу прямого перебору станів елементів системи.

Для досягнення поставленої мети було проведено аналіз обох методів, що включав оцінку ефективності обчислення параметрів для різноманітних вхідних даних.

Розроблено систему, здатну обчислювати параметри функціональної стійкості мережевих інформаційних систем двома методами і яка надає користувачеві базовий графічний інтерфейс для побудови графів і перегляду результатів виконання.

Практичне значення одержаних результатів полягає в отриманні системи, що здатна обчислювати параметри функціональної стійкості мережевих інформаційних систем двома методами. Проведено огляд результатів роботи обох методів, та надано рекомендації що їх використання. Реалізовано базовий інтерфейс користувача, а також надана можливість інтеграції системи з іншими застосунками.

Ключові слова: мережеві інформаційні системи , функціональна стійкість, теорія графів, серверний застосунок.

ABSTRACT

Structure and scope of the thesis. The work is performed on 51 sheets, it contains 22 figures, 6 tables, 2 appendices, and 5 references.

The purpose of the work is the development of software for calculating parameters of functional stability of network information systems based on the method of structural transformations and the method of direct enumeration of the states of system elements.

To achieve the goal, an analysis of both methods was carried out, which included an assessment of the effectiveness of parameter calculation for various input data.

A system capable of calculating parameters of functional stability of network information systems by two methods and providing the user with a basic graphical interface for constructing graphs and viewing performance results has been developed.

The practical significance of the results lies in obtaining a system capable of calculating the parameters of functional stability of network information systems by two methods. The results of both methods were reviewed, and recommendations for their use were provided. The basic user interface and the possibility of integrating the system with other applications have been implemented.

Keywords: network information systems, functional stability, graph theory, server application.

ЗМІСТ

ВСТУП

У сучасному цифровому світі, де мережеві інформаційні системи відіграють важливу роль у різноманітних сферах, виникає нагальна потреба в забезпеченні їх функціональної стійкості та надійності. З впровадженням нових технологій і зростанням обсягів обміну даними, збільшується і ризик виникнення проблем, пов'язаних зі збоїв у роботі систем, вразливістю до атак та іншими загрозами безпеці.

Одним із ефективних підходів до забезпечення стійкості мережевих інформаційних систем є аналіз їх структури та внутрішніх параметрів. У цьому контексті методи обчислення параметрів функціональної стійкості набувають особливого значення. Вони дозволяють систематично досліджувати можливі конфігурації системи, виявляти слабкі місця та встановлювати шляхи їх виправлення.

Використання таких методів є важливим етапом у процесі розробки та підтримки мережевих інформаційних систем, оскільки воно дозволяє виявляти та усувати потенційні проблеми ще на етапі проектування та тестування.

Дана робота спрямована на розробку програмного забезпечення, яке використовує зазначені методи для вирішення конкретних завдань, пов'язаних з підвищенням функціональної стійкості мережевих інформаційних систем.

Подальший розвиток цього напряму досліджень та практичного застосування обраного підходу може сприяти покращенню безпеки та надійності інформаційного середовища, що стає все більш важливим у світі, де цифрові технології проникають у всі сфери життя.

1 ТЕОРЕТИЧНЕ ОБҐРУНТУВАННЯ

Під функціональною стійкістю розподілених інформаційних систем розуміється властивість системи зберігати протягом заданого часу виконання своїх основних функцій в межах, встановлених нормативними вимогами, при впливі потоку експлуатаційних відмов, збоїв, пошкоджень, умисного шкідництва, втручання в обмін та обробку інформації, а також при помилках обслуговуючого персоналу.

1.1 Загальні відомості щодо показників функціональної стійкості

Для розподіленої інформаційної системи приймаємо, що система має виконувати дві основні функції [1,4]:

- обробку, зберігання, відображення інформації.
- передачу інформації між вузлами комутації.

Звідси випливає основна вимога до функціонально стійких розподілених інформаційних систем:

1. Забезпечити працездатність усіх вузлів комутації.
2. Забезпечити передачу інформації між вузлами комутації за основними або резервними маршрутами.

Ознаки функціональної стійкості структури

1. Структура розподіленої інформаційної системи є функціонально стійкою, якщо граф структури є однокомпонентним і не має мостів та вузлів з'єднання. Зворотне визначення дозволяє зумовити функціональну нестійкість структури.

2. Структура розподіленої інформаційної системи є функціонально нестійкою, якщо її граф є багатокомпонентним та незв'язним.

Таким чином, за виглядом графа, а саме за кількістю компонентів, наявністю мостів і вузлів з'єднання графа можна судити про функціональну стійкість структури, тобто про закладену в ній здатність компенсувати відмови та пошкодження. Однак

для великих графів з великою кількістю ребер здійснити оцінювання на вигляд складно [1, 2].

Показники функціональної стійкості структури

1. Число вершинної зв'язності $\chi(G)$ – це найменша кількість вершин, видалення яких разом з інцидентними ним ребрами призводить до незв'язного або одновершинного графа.

2. Число реберної зв'язності $\lambda(G)$ – це найменша кількість ребер, видалення яких призводить до незв'язного графа.

3. Імовірність зв'язності $P_{ij}(t)$ – це ймовірність того, що повідомлення з вузла i у вузол j буде передано за час не більше ніж t.

Аналіз даних показників дозволяє виділити такі особливості [1, 3]:

- числа вершинної та реберної зв'язності характеризують лише поточну структуру, незалежно від надійності вузлів комутації або ліній зв'язку
- ймовірність зв'язності $P_{ij}(t)$ дозволяє враховувати надійність комутаційного обладнання, вид фізичного каналу передачі інформації, наявність резервних каналів та маршрутів, а також зв'язність розподіленої структури
- ймовірність зв'язності характеризує лише зв'язок між однією парою вершин. Для того, щоб характеризувати зв'язність між усіма парами вершин, необхідно оперувати з матрицею ймовірностей зв'язності:

$$P_{\text{СВ}} = \left\| P_{ij} \right\| \quad i, j = 1,2, \dots, n \tag{1.1}$$

На основі запропонованих ознак та показників можна розробити критерії функціональної стійкості структури:

Критерії функціональної стійкості структури

1. Структурний критерій. Структура буде функціонально стійкою, якщо число вершинного зв'язку та число реберного зв'язку задовольняють умовам:

$$\chi(G) \geq 2 \ \cup \ \lambda(G) \geq 2 \tag{1.2}$$

2. Імовірнісний критерій. Структура буде функціонально стійкою, якщо ймовірність зв'язності між кожною парою вершин буде не менш заданої:

$$P_{ij}(t) \geq P_{ij}^{\text{зад}}, i \ \neq j, \ \ i,j = 1,2,\dots,n \tag{1.3}$$

1.2 Огляд методу прямого перебору станів елементів системи

Найочевиднішим способом побудови матриці ймовірностей зв'язності є перебір всіх станів елементів системи. В цьому алгоритмі, для кожної пари вершин перебираються всі можливі комбінації присутності та відсутності ребер, після чого додаються ймовірності всіх комбінацій, при яких існує шлях з вершини *i* до вершини *j* [1].

Розглянемо алгоритм детальніше, для графа, що зображений на рисунку 1.1. Припустимо, що вузли комутації в наведеній розподіленій інформаційній системі безвідмовні $p(v_i)=1$, а надійність ліній зв’язку визначається ймовірністю передачі інформації $p(l_i)=1-q(l_i)$.. У цьому випадку граф матиме $m_L=5$ елементів, а число всіх можливих станів графа дорівнюватиме $2^5= 32$ (таблиця 1.1). Справному стану елемента відповідає значення булевої змінної 1, а несправному – 0. Символом «+» відзначені стани елементів графа що відповідають подіям зв'язності та незв'язності.

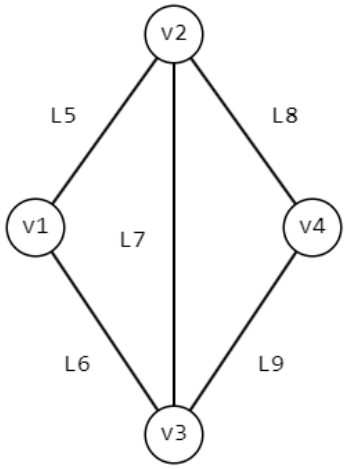


Рисунок 1.1 - Граф елементарної структури РІС

Таблиця 1.1 - Визначення показника функціональної стабільності структури, представленої на рис. 1.1

№	Конституенти елементів					$E_{1,4}$	$\overline{E}_{1,4}$
	l_5	l_6	l_7	l_8	l_9		
1	0	0	0	0	0		+
2	0	0	0	0	1		+
3	0	0	0	1	0		+
4	0	0	0	1	1		+
5	0	0	1	0	0		+
6	0	0	1	0	1		+
7	0	0	1	1	0		+
8	0	0	1	1	1		+
9	0	1	0	0	0		+
10	0	1	0	0	1	+	
11	0	1	0	1	0		+
12	0	1	0	1	1	+	
13	0	1	1	0	0		+
14	0	1	1	0	1	+	
15	0	1	1	1	0	+	
16	0	1	1	1	1	+	

№	Конституенти елементів					$E_{1,4}$	$\overline{E}_{1,4}$
	l_5	l_6	l_7	l_8	l_9		
17	1	0	0	0	0		+
18	1	0	0	0	1		+
19	1	0	0	1	0	+	
20	1	0	0	1	1	+	
21	1	0	1	0	0		+
22	1	0	1	0	1	+	
23	1	0	1	1	0	+	
24	1	0	1	1	1	+	
25	1	1	0	0	0		+
26	1	1	0	0	1	+	
27	1	1	0	1	0	+	
28	1	1	0	1	1	+	
29	1	1	1	0	0		+
30	1	1	1	0	1	+	
31	1	1	1	1	0	+	
32	1	1	1	1	1	+	

Кожна конституента є добутком булевих змінних елементів графа. Наприклад, конституента 4 із таблиці 1.1 може бути записана у вигляді $k_4 = \{\overline{5}\cdot\overline{6}\cdot\overline{7}\cdot 8\cdot 9\}$. Значення ймовірності конституенти: $P(k_4)=q_5\cdot q_6\cdot q_7\cdot p_8\cdot p_9$.

Визначити значення ймовірності зв’язності можна наступним чином:

$$P_{x,y} = \sum_{i,k_i \in K_{x,y}} \prod_{j=1}^{n} r_j \qquad r_j = \begin{cases} p_j, \text{при } k_{ij} = 1 \\ q_j, \text{при } k_{ij} = 0 \end{cases} \tag{1.4}$$

де $K_{x,y}$, - підмножина конституент що відповідають події $E_{x,y}$.

Так як метод прямого перебору обчислює ймовірність P_{ij} зв’язності лише між однією парою вершин, то для побудови матриці ймовірностей зв'язності необхідно повторити алгоритм n(n-1) разів, де n – кількість вершин графа.

1.3 Огляд методу структурних перетворень

Обрахунок P_{ij} методом структурних перетворень являє собою розклад структури розподіленої інформаційної системи на послідовні та паралельні з’єднання ліній зв’язку [1].

Для цього структура РІС розкладається відносно якогось елемента. Наприклад, для обчислення P_{14} вихідний граф G (рисунок 1.2) перетворюється на два графи G_1 і G_2.

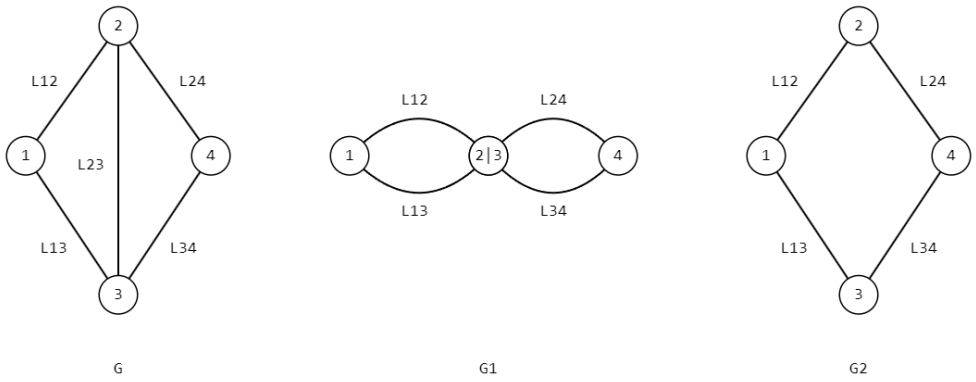


Рисунок 1.2 - Перетворення графа у послідовно-паралельне з'єднання ребер

Граф G_1 отримано стягуванням ребра l_{23}, що відповідає справному стану ребра l_{23}. Граф G_2 отримано після розриву l_{23}, що відповідає його несправному стану. Імовірність зв'язності $P_{1,4}$ для графа G можна обчислити за основною формулою розкладання:

$$P_{14}(G) = p_{23} * P_{14}(G_1) + q_{23} * P_{14}(G_2) \quad (1.5)$$

де p_{23}=1–q_{23} – ймовірність передачі інформації по лінії зв’язку, відповідній ребру l_{23};

Для кожної ймовірності P_{ij} розклад графа на послідовно-паралельне з'єднання ребер продовжується доти, доки не закінчяться ребра. Також для випадків, коли розрив ребра призводить до відсутності шляху з вершини *i* до вершини *j* послідовність не продовжується.

Як і для методу прямого перебору станів системи, для кожної пари вершин алгоритм виконується окремо, що означає необхідність виконати його n(n-1)/2 разів, де n – кількість вершин графа [1].

Висновки до розділу 1

Таким чином, в цьому розділі розглянуто основні відомості про параметри функціональної стійкості та методи їх обчислення, в тому числі надано необхідні формули та наведено приклади.

Визначено, що необхідно дослідити дані методи та порівняти їх продуктивність і точність результатів.

2 ВИБІР ЗАСОБІВ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка застосунку для обчислення параметрів функціональної стійкості мережевих інформаційних систем складається з декількох складових частин: вибір мов програмування, вибір платформ та інструментів для розробки застосунку та інтерфейсу для нього, а також вибір технологій для створення інфраструктури застосунку.

2.1 Мова програмування PHP

Мова програмування PHP є однією з найпопулярніших мов для веб-розробки. Вона має простий синтаксис, велику кількість вбудованих функцій і підтримує різноманітні платформи для швидкого створення веб-додатків. Вибір пав саме на цю мову програмування, тому що завдяки своїй гнучкості, вона дозволяє легко створювати застосунки з сучасною архітектурою та багатим функціоналом, а також легко підтримувати та покращувати їх.

2.2 Платформа Symfony

Symfony - це високопродуктивний PHP-фреймворк, який дозволяє розробникам створювати складні веб-додатки швидко і ефективно. Він забезпечує гнучкість, розширюваність і надійність завдяки великому набору компонентів і зручному інтерфейсу. Ця платформа була обрана завдяки своїм розширюваним компонентам, високій продуктивності та великій кількості документації і підтримки.

2.3 Платформа Api Platform

API Platform - це набір інструментів для швидкої розробки API-інтерфейсів на основі Symfony. Вона надає зручний спосіб створення REST API з мінімальними

зусиллями. API Platform автоматично генерує документацію, значно спрощує створення кінцевих точок доступу та їх налаштування, а також легко інтегрується з Symfony.

2.4 Система контейнеризації Docker

Docker - це платформа для розробки, доставки і виконання програмного забезпечення в контейнерах. Контейнеризація дозволяє ізолювати додатки та їх залежності, забезпечуючи консистентне середовище в різних областях розробки, тестування та виробництва. Docker спрощує розгортання та масштабування додатків, забезпечуючи їхню портативність і надійність та дозволяє легко інтегрувати застосунок в існуючий кластер мікросервісів.

2.5 База даних MariaDB

MariaDB - це відкрита реляційна база даних, яка є розгалуженням MySQL і надає високу продуктивність, надійність та розширюваність для зберігання та управління даними. Обрано MariaDB як основну базу даних для проекту з причини простоти її налаштування у системі Docker.

2.6 Мова програмування JavaScript

JavaScript є основною мовою програмування для розробки інтерактивних інтерфейсів веб-додатків. Вона дозволяє створювати динамічні і швидкі інтерфейси користувача, що покращують досвід взаємодії з веб-застосунками.

Використання JavaScript в парі з PHP дозволяє створювати повнофункціональні серверні та клієнтські рішення, що забезпечують високу продуктивність і зручність використання. Завдяки своїй гнучкості, JavaScript дозволяє легко інтегрувати сторонні бібліотеки, що прискорює процес розробки та покращує якість кінцевого продукту.

2.7 Платформа React

React - це популярна бібліотека для створення користувацьких інтерфейсів, розроблена компанією Facebook. Вона дозволяє будувати складні UI з використанням компонентного підходу, що сприяє повторному використанню коду і знижує складність управління станом додатку.

Використання React забезпечує високу продуктивність за рахунок віртуального DOM та ефективного рендерингу компонентів. React легко інтегрується з іншими технологіями, такими як API Platform, що дозволяє швидко створювати сучасні, масштабовані і зручні для користувачів веб-додатки.

Висновки до розділу 2

Таким чином, вищезазначені технології та інструменти дозволяють створити ефективний, надійний і масштабований серверний застосунок, що відповідає сучасним вимогам та стандартам розробки програмного забезпечення, базовий веб-інтерфейс для нього та побудувати для них спільну інфраструктуру, включно із базою даних.

3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

Створення застосунку для обчислення параметрів функціональної стійкості мережевих інформаційних систем складається з декількох складових частин, серед яких: вибір архітектури серверного застосунку, визначення формату вхідних та вихідних даних, реалізація алгоритмів обчислення параметрів функціональної стійкості та створення інтерфейсу користувача, а також впровадження документації, яка дозволить користувачам розуміти можливі варіанти використання системи.

3.1 Архітектура серверної частини застосунку

Архітектура серверного застосунку ґрунтується на принципах DDD (Domain Driven Design) та RESTful дизайну, що надає цілий ряд переваг, які сприяють створенню гнучкого та розширюваного програмного рішення.

Підхід DDD дозволяє розробникам глибше розуміти бізнес-логіку застосунку шляхом фокусування на основних доменних областях. Це допомагає зменшити складність системи, розділивши її на окремі модулі, кожен з яких відповідає за певний аспект бізнесу. Крім того, DDD сприяє покращенню комунікації між розробниками та експертами з галузі, оскільки вони спільно працюють над визначенням та реалізацією доменної моделі.

RESTful дизайн дозволяє створювати легко зрозумілі та зручні для використання API. Завдяки використанню стандартних HTTP методів та URI, клієнти можуть легко взаємодіяти з ресурсами застосунку. Це спрощує розробку клієнтських додатків та дозволяє розробникам змінювати та розширювати API без значних зусиль.

Комбінація цих двох підходів створює міцну основу для розробки сучасних веб-додатків. Вона дозволяє розробникам швидко реагувати на зміни в бізнесі та вимоги користувачів, забезпечуючи при цьому високу якість та надійність програмного забезпечення.

Для взаємодії з базою даних використовується Doctrine ORM, що дозволяє взаємодіяти з базою даних, використовуючи об'єктно-орієнтований підхід. Це

спрощує роботу з базою даних, забезпечуючи консистентність та ефективність взаємодії між даними та API.

Для забезпечення масштабованості, портативності та консистентнсті середовища розробки та виробництва використовується система контейнеризації Docker, яка дозволяє упаковувати додатки та їх залежності в контейнери, які можуть бути легко перенесені між різними середовищами без втрати функціональності. Це дозволило зосередитися на розробці програмного забезпечення, уникнувши проблем, пов'язаних з різницею у середовищах від розробки до виробництва.

3.2 Реалізація обчислення параметрів функціональної стійкості

В першу чергу, для реалізації методів обчислення параметрів функціональної стійкості потрібно визначити кінцеві точки серверного застосунку, формат та зміст вхідних і вихідних даних, а також налаштувати застосунок для прийому HTTP запитів.

Кінцеві точки були визначені шляхом аналізу майбутньої функціональності застосунку. В результаті було створено 3 кінцеві точки:

- Кінцева точка для обчислення методом прямого перебору станів системи.
- Кінцева точка для обчислення методом структурних перетворень.
- Кінцева точка для отримання збережених результатів по ідентифікатору.

Після цього було створені та налаштовані Docker контейнери, які згодом було оркестровано за допомогою Docker Compose. Ці контейнери включають в себе сам застосунок, веб-сервер Caddy та базу даних MariaDB, можуть вільно спілкуватися один з одним, але максимально ізольовані від середовища, в якому вони запущені.

В якості формату обміну даними було обрано JSON. Використання JSON (JavaScript Object Notation) стало стандартом для обміну даними між клієнтом та сервером у веб-додатках. Він є легко зрозумілим для людини і легко оброблюється комп'ютерами, що робить його ідеальним вибором для передачі структурованих даних.

Структура запиту включає в себе список вершин, список ребер з їх ваговими коефіцієнтами та цільовий шанс успішної доставки повідомлення для кожного значення з матриці ймовірностей зв`язності. Приклад запиту можна побачити на рисунку 3.1.

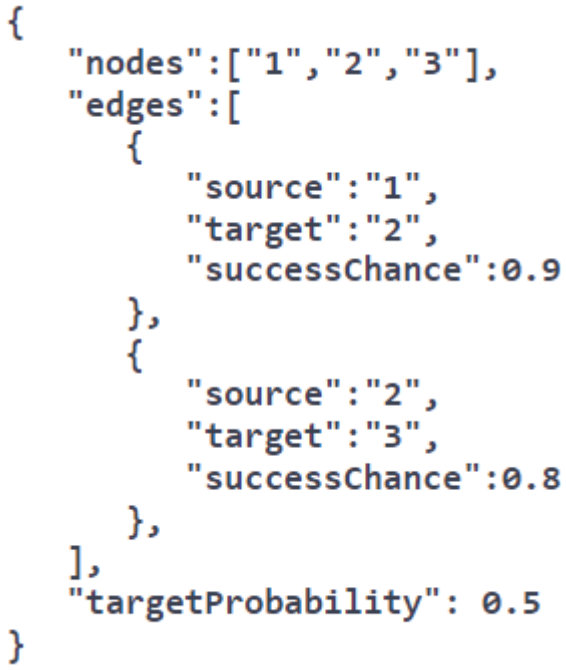


Рисунок 3.1 - Приклад запиту

Структура відповіді включає в себе час, який було витрачено на обрахунки, ступені вершинної та реберної зв’язності, матрицю ймовірностей зв’язності, унікальний ідентифікатор, використовуючи який можна повторно отримати результати, та булеве значення, яке вказує, чи є даний граф функціонально стійким. Приклад відповіді застосунку можна побачити на рисунку 3.2.

```
"isStable": false,
"id": "018f6bee-2356-7860-bf63-1bc5c4f9bad8",
"execTimeMilliseconds": 4.26611328125,
"x(G)": 1,
"λ(G)": 1,
"probabilityMatrix": [
  {
    "source": "1",
    "target": "2",
    "probability": 0.9
  },
  {
    "source": "1",
    "target": "3",
    "probability": 0.7200000000000001
  },
  {
    "source": "2",
    "target": "3",
    "probability": 0.8
  }
]
```


Рисунок 3.2 - Приклад відповіді застосунку

Також передбачена валідація вхідних запитів, яка включає такі правила:

- Кожне значення у списку вершин має бути рядком.
- Кожне значення початкової або цільової вершини у списку ребер має бути одним із значень списку вершин.
- Значення вагових коефіцієнтів ребер та цільовий шанс успішної доставки повідомлення для кожного значення з матриці ймовірностей зв`язності мають бути більше 0 та більше або рівні 1.

Метод прямого перебору станів елементів системи

Для обчислення параметрів функціональної стійкості методом прямого перебору станів системи, спочатку знаходяться всі можливі пари вершин, та всі можливі комбінації присутності та відсутності ребер. Далі, для кожної пари вершин всі можливі комбінації ребер перевіряються на наявність шляху між початковою та цільовою вершиною, якщо так, то ймовірність існування цієї комбінації ребер

додається до результуючої ймовірності. Блок схема даного алгоритму наведена на рисунку 3.3.

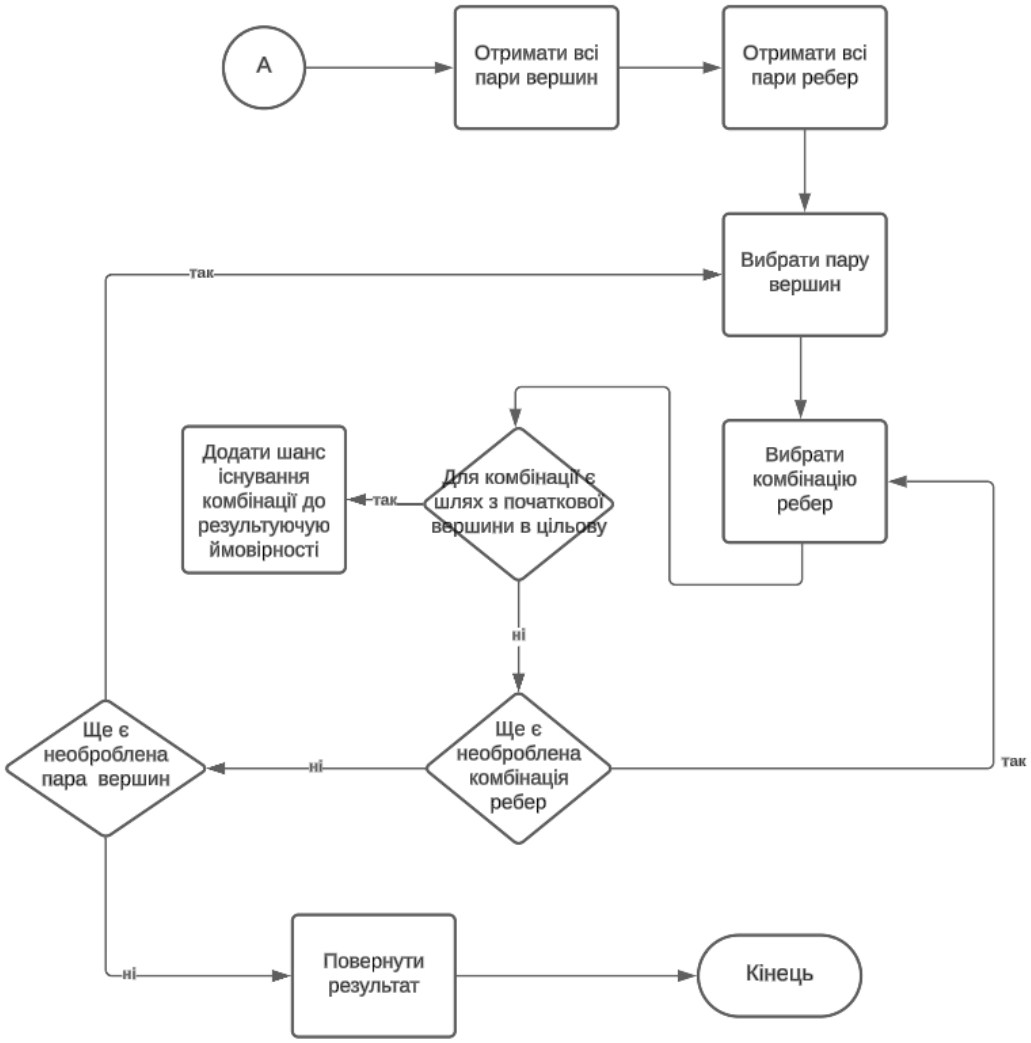


Рисунок 3.3 – Блок схема алгоритму методу прямого перебору станів системи

Метод структурних перетворень

Для обчислення параметрів функціональної стійкості методом структурних перетворень, спочатку знаходяться всі можливі пари вершин, після чого для графа вибирається довільне ребро. Довільне ребро стягується та розривається, що утворює два графи. Для графа зі стягнутою вершиною алгоритм рекурсивно повторюється, якщо ще залишились ребра. Для графа з розірваною вершиною, алгоритм повторюється тільки якщо після розривання ребра між початковою і цільовою

вершиною залишився шлях, і якщо ще залишились ребра. Після всіх рекурсивних викликів повертається результат. Блок схема даного алгоритму наведена на рисунку 3.4.

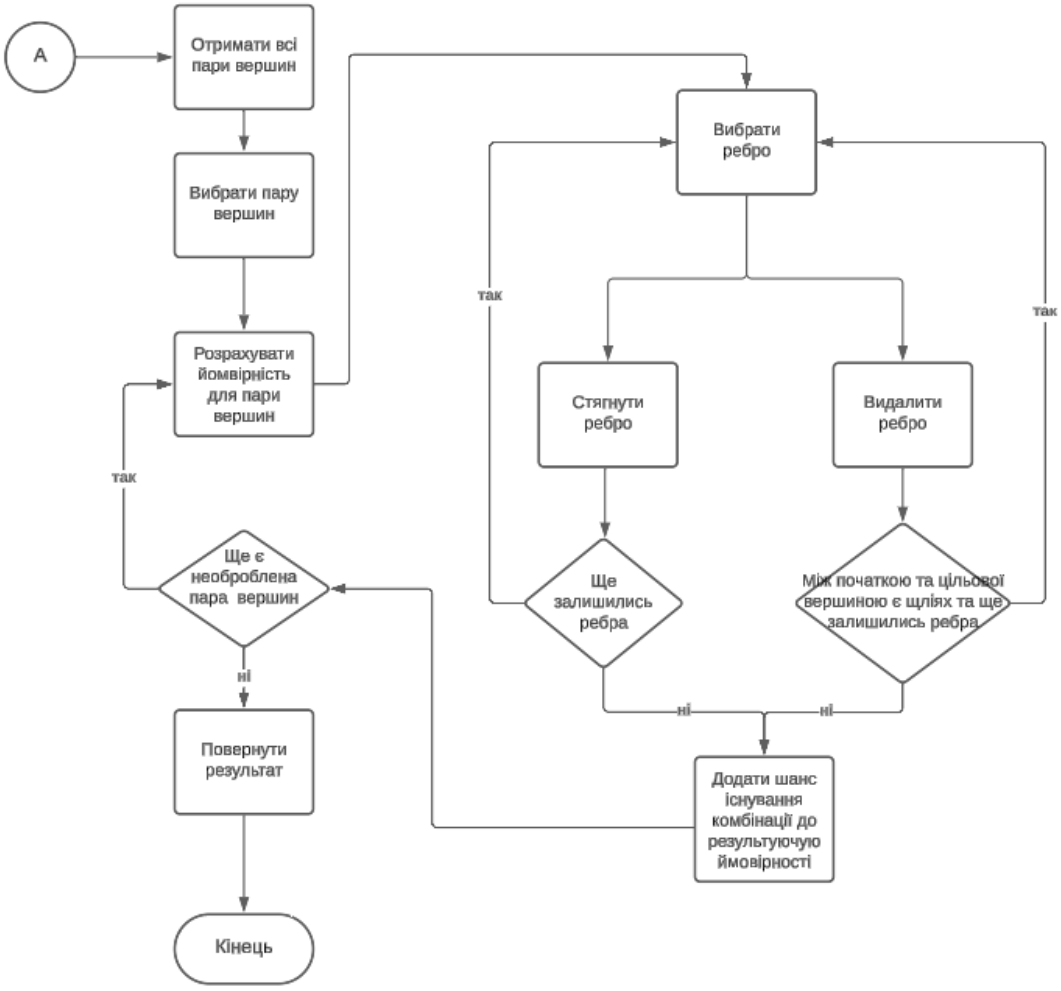


Рисунок 3.4 – Блок схема алгоритму методу структурних перетворень

Обчислення реберної зв’язності

Для обчислення реберної зв’язності, з графа по черзі видаляється кожне ребро і перевіряється, чи залишився граф зв’язним. Якщо граф залишається зв’язним без кожного ребра, результат збільшується на одиницю, і алгоритм повторюється для графа без одного ребра. Блок схема даного алгоритму наведена на рисунку 3.5.

Рисунок 3.5 – Блок схема алгоритму обчислення реберної зв’язності

Обчислення вершинної зв’язності

Для обчислення вершинної зв’язності, з графа по черзі видаляється кожна і всі інциденті їх ребра і перевіряється, чи залишився граф зв’язним. Якщо граф залишається зв’язним, результат збільшується на одиницю, і алгоритм повторюється для графа без однієї вершини. Блок схема даного алгоритму наведена на рисунку 3.6.

Рисунок 3.6 – Блок схема алгоритму обчислення вершинної зв’язності

3.3 Створення інтерфейсу користувача

Інтерфейс користувача був розроблений за допомогою бібліотеки React, яка є однією з найпопулярніших технологій для створення користувацьких інтерфейсів. React базується на компонентній архітектурі, що дозволяє розбити інтерфейс на невеликі, повторно використовувані компоненти. Це спрощує розробку та утримання коду, а також забезпечує більшу гнучкість і швидкість розробки нового функціоналу.

Одним з ключових аспектів розробки інтерфейсу є використання концепції односторінкового застосунку. SPA (Single Page Application) дозволяє завантажити всі необхідні ресурси, такі як HTML, CSS, та JavaScript, один раз під час першого відкриття сторінки, а потім динамічно оновлювати вміст сторінки без

перезавантаження при подальших взаємодіях користувача. Це робить користувацький досвід більш плавним та ефективним, оскільки не виникає затримок у завантаженні нових сторінок.

Однією з переваг SPA є зменшення часу завантаження, оскільки всі ресурси завантажуються лише один раз під час першого відкриття сторінки. Це полегшує роботу з додатком та забезпечує швидку відповідь на дії користувача.

Крім того, SPA дозволяє створювати більш динамічні інтерфейси, оскільки вони можуть взаємодіяти з сервером без перезавантаження сторінки, що покращує користувацький досвід. Наприклад, це може бути миттєве оновлення списку елементів без необхідності повного перезавантаження сторінки. Такий підхід забезпечує високу продуктивність та зручність використання додатку для користувача.

Інтерфейс користувача являє окремий веб-застосунок, який взаємодіє з серверною частиною через HTTP запити. Він має окремі залежності та власний веб-сервер, але є складовою частиною інфраструктури проекту, та запускається разом із іншими її компонентами.

3.4 Впровадження інтерактивної документації

API Platform використовує анотації або анотаційні PHP-докблоки для визначення схеми API прямо в коді. Після цього, на основі цієї інформації, API Platform автоматично генерує документацію у форматі OpenAPI, що надає зручний спосіб ознайомлення з функціоналом API для розробників та інших зацікавлених сторін.

OpenAPI - це стандарт для опису RESTful API, який надає можливість документувати, використовувати та автоматизувати взаємодію з API. OpenAPI використовує мову опису API на основі JSON або YAML для забезпечення чіткої та структурованої документації.

Використання платформи API Platform для генерації документації у форматі OpenAPI дозволяє швидко та ефективно створювати опис API з використанням інформації, визначеної у вихідному коді застосунку. Структурована документація у форматі OpenAPI надає зручний спосіб ознайомлення з функціоналом API, включаючи доступні ресурси, їх параметри та методи. Крім того, OpenAPI дозволяє автоматизувати процеси тестування та інтеграції API, що спрощує розробку та підтримку API у різних проектах.

Загальний вигляд документації можна побачити на рисунку 3.7.

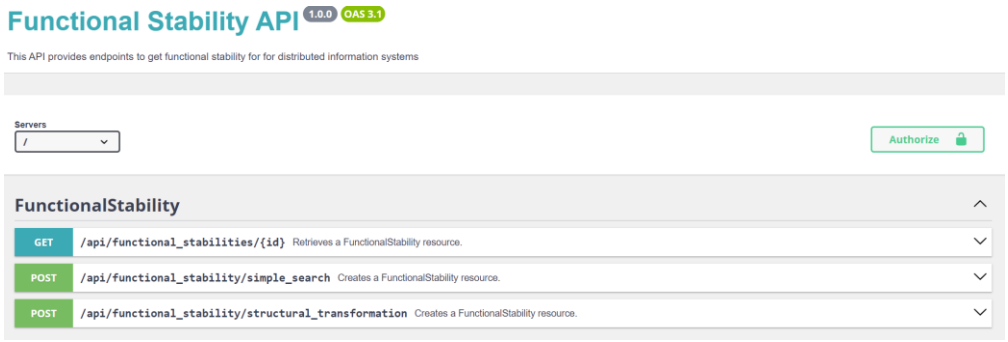


Рисунок 3.7 – Загальний вигляд документації застосунку

Використання API Platform гарантує, що документація завжди буде актуальною і відповідатиме реальному стану коду. Це знижує ризик виникнення помилок через невідповідність між кодом та документацією, що є важливим для забезпечення надійності та підтримуваності системи

Висновки до розділу 3

В цьому розділі було розглянуто програмну реалізацію застосунку, включно з його архітектурою, алгоритмами обчислення параметрів функціональної стійкості мережевих інформаційних систем, інтерфейсом користувача та документацією.

Було надано блок-схеми для демонстрації алгоритмів обчислення параметрів, а також приклади форматування вхідних та вихідних даних.

4 РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

Для роботи з програмною системою потрібно встановити її, а також розуміти основні функції та можливості системи, які будуть наведені у документації застосунку та подальших інструкціях.

4.1 Інструкції щодо встановлення

Для того щоб успішно встановити та запустити застосунок, потрібно, щоб на комп'ютері бути встановлені наступні інструменти:

- Docker
- Docker Compose
- Git
- Make

Ці утиліти встановлюються досить просто, і є стандартними інструментами для розробки і запуску застосунків, а також доступні на будь-якій операційній системі.

Рекомендовано використовувати застосунок з операційною системою Linux, для успішної роботи у операційній системі Windows треба виконати наступну послідовність команд:

1. git config --global core.autocrlf input
2. git config --global core.eol lf

Ці команди запобігають автоматичній заміні символів закінчення рядку з LF, що використовується у Linux, на CRLF у Windows, і забезпечать коректну роботу застосунку у Docker контейнерах.

Далі треба клонувати сам застосунок, використовуючи систему контролю версій Git, виконавши наступну команду "git clone https://github.com/kukuruzvelt/functional-stability-analyzer.git" у потрібній директорії.

Після цього, перейдіть у директорію, в яку було клоновано застосунок, та виконайте команду "make start".

Ця команда запустить Docker контейнери застосунку, встановить в них усі необхідні залежності та проведе міграції до бази даних.

Для того щоб зупинити застосунок, виконайте команду “make stop”.

Для того щоб видалити створені Docker контейнери, передбачена команда “make down”

4.2 Взаємодія користувача з системою

Після успішного встановлення системи, можна перейти по посиланню “http://localhost:3000”, інтерфейсу користувача, який складається з робочого простору, призначеному для створення графів, а також елементів керування та відображення результатів. Загальний вигляд інтерфейсу можна побачити на рисунку 4.1.

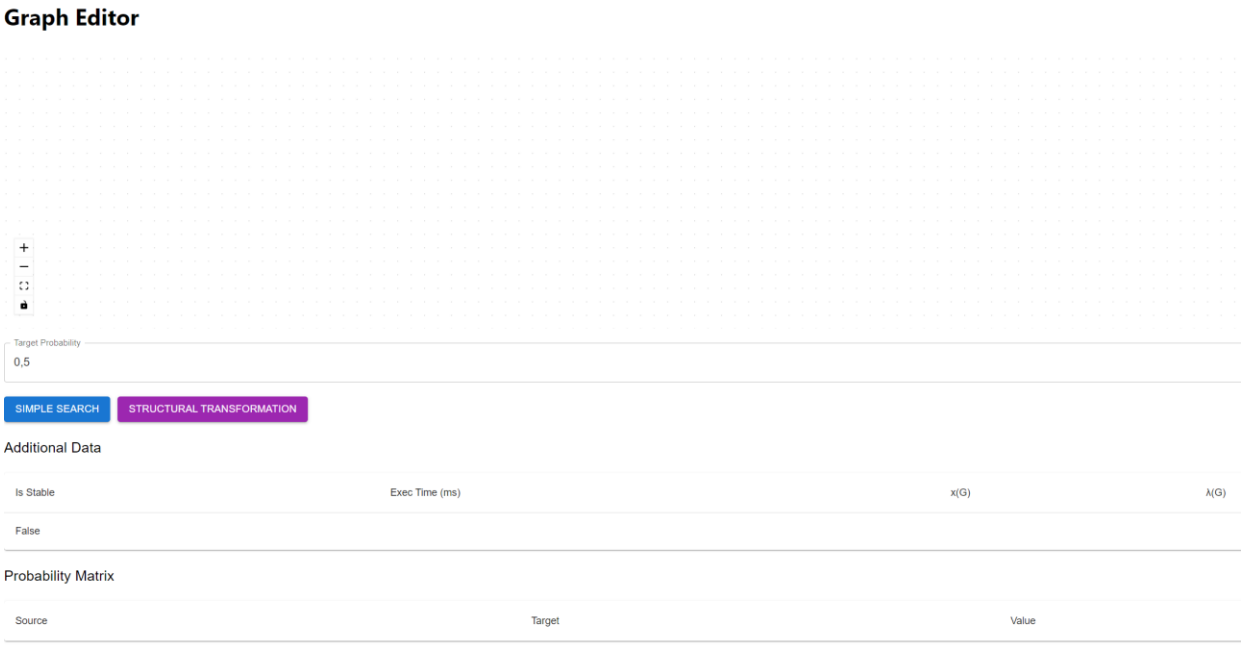


Рисунок 4.1 – Загальний вигляд інтерфейсу користувача

Робочий простій підтримує створення та редагування графів мережевих інформаційних систем, для яких будуть вираховуватись параметри функціональної стійкості. Передбачена можливість створення та видалення нових вершин, а також створення, видалення, та редагування ребер графа.

Створення нової вершини відбувається після натискання на вільне місце у робочому просторі, після чого з’являється нова вершина з автоматично згенерованим ім’ям, що відповідає її порядковому номеру. Для видалення вершини необхідно двічі натиснути на неї, що також видалить і всі інцидентні вершині ребра.

Для того, щоб створити ребро, необхідно спочатку натиснути на початкову вершину, а потім на цільову, після чого з’явиться діалогове вікно, яке можна побачити на рисунку 4.2.

Рисунок 4.2 – Діалогове вікно створення ребра

Для ймовірності зв'язності між вершинами у діалоговому вікні передбачена валідація, число повинно бути більше 0 і не більше 1, а при введенні неправильних даних користувач побачить повідомлення, яке можна побачити на Рисунку 4.3.

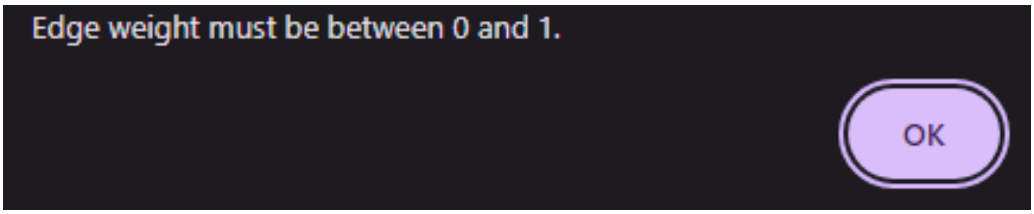


Рисунок 4.3 – Повідомлення про неправильно введену ймовірність

Для редагування або ж видалення ребра слід натиснути на число ймовірності зв’язності, після чого з’явиться діалогове вікно з рисунку 4.4.

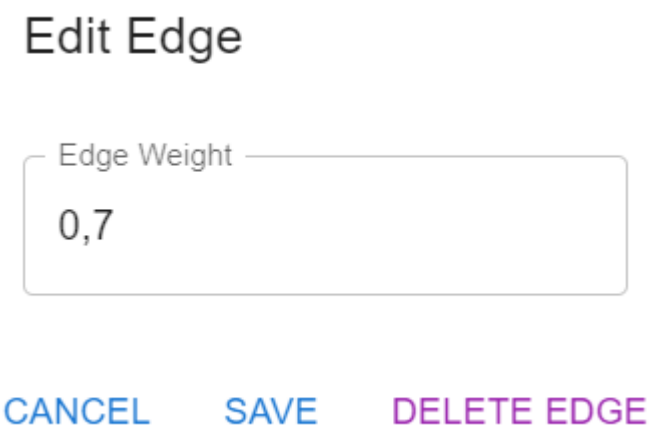


Рисунок 4.4 – Діалогове вікно редагування ребра

Приклад елементарного графа, створеного за допомогою інтерфейсу, наведено на Рисунку 4.5.

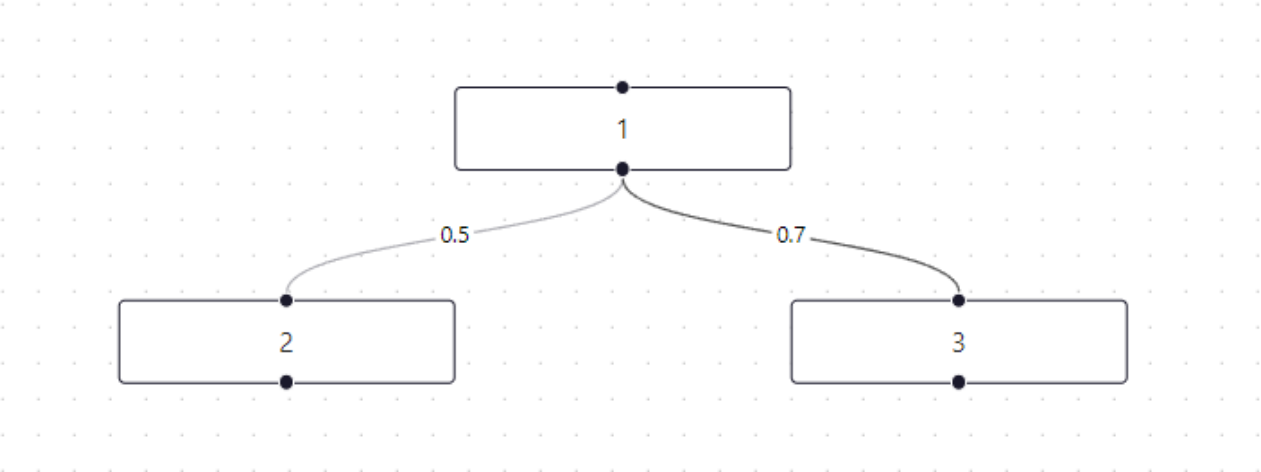


Рисунок 4.5 – Приклад елементарного графа

Для отримання результатів слід натиснути на кнопку із потрібним методом, після чого на сторінці з’являться обчислені параметри функціональної стійкості. Приклад можна побачити на рисунку 4.6.

Additional Data

Is Stable	Exec Time (ms)	χ(G)	λ(G)
False	3.322998046875	2	2

Probability Matrix

Source	Target	Value
1	2	0.536
1	3	0.6239999999999999
1	4	0.344
2	3	0.38399999999999995
2	4	0.454
3	4	0.384

Рисунок 4.6 – Приклад обчислених параметрів функціональної стійкості

Граф, для його будуть обчислюватись параметри функціональної стійкості має бути зв’язним, саме тому для графів теж присутня валідація, яка перевіряє, що у ньому немає ізольованих вершин. В інакшому випадку, користувач побачить помилку, наведену на рисунку 4.7.

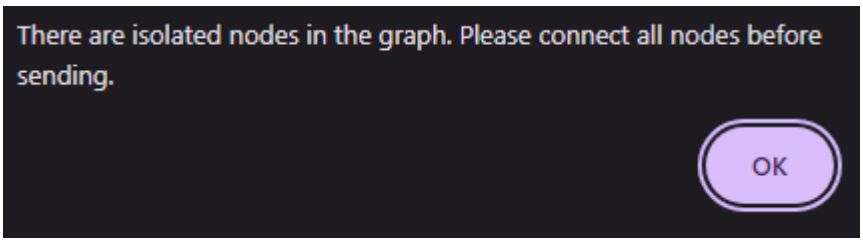


Рисунок 4.7 – Повідомлення про неправильно введений граф

Після отримання результатів, можна продовжити роботу з системою та отримати результати іншим методом, або ж для іншого графа, і після повторного запиту на екрані з’являться нові значення.

4.3 Можливості інтеграції системи

Так як серверний застосунок та інтерфейс користувача є різними програмами, то обчислювати параметри можна не лише за допомогою наданого інтерфейсу, а й іншим чином, наприклад із мобільного застосунку, або ж іншої серверної платформи. Для спрощення можливості інтеграції було розроблено інтерактивну документацію, що дозволяю чітко розуміти можливості системи та способи взаємодії з нею. Для

доступу до документації, після успішного встановлення системи слід перейти по посиланню “https://localhost/api/docs”.

Там можна знайти всю інформацію про наявні кінцеві точки застосунку, їх HTTP методи, сигнатури запитів та можливі відповіді. Приклад наведено на рисунку 4.8.

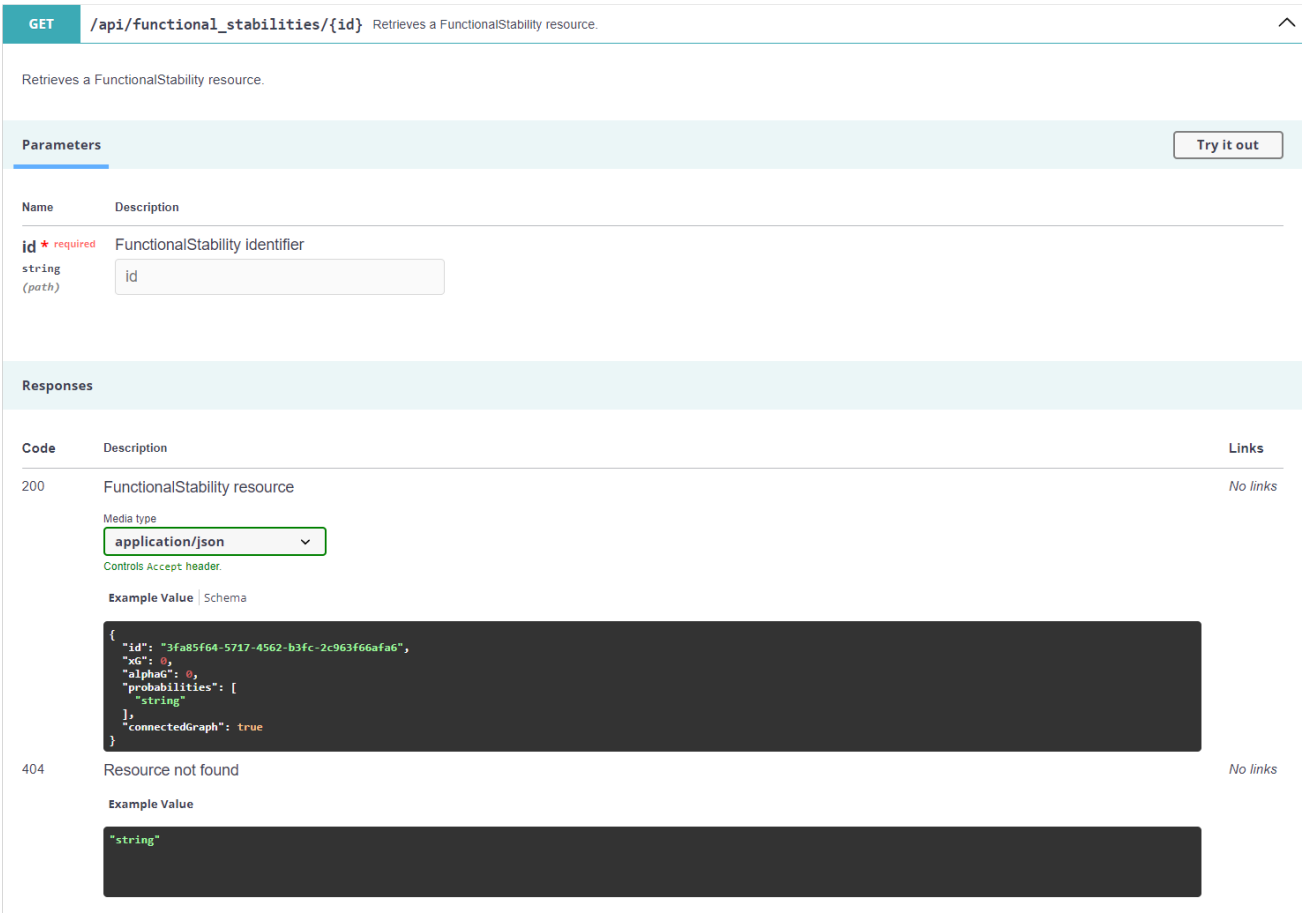


Рисунок 4.8 – Документація до однієї з кінцевих точок

З цієї сторінки можна надсилати запити до застосунку, та бачити відповіді, які надходять, вводячи вхідні дані у необхідні поля. Це надає розробникам можливість легко працювати з кінцевими точками, оскільки вони можуть безпосередньо взаємодіяти з API у режимі реального часу. Вводячи необхідні параметри в спеціальні поля форми та надсилаючи запити, розробники отримують миттєві відповіді від сервера, що дозволяє їм перевіряти коректність роботи API, тестувати різні сценарії використання та швидко виявляти та виправляти помилки.

Висновки до розділу 4

Таким чином, було розглянуто взаємодію користувача з системою та її можливості, зокрема керування застосунком за допомогою інтерфейсу, а також можливості інтеграції до інших застосунків та систем.

Було визначення необхідні для встановлення програми інструменти та утиліти, а також надано алгоритм дій для запуску, завершення та налаштування роботи системи для різних операційних систем.

5 ТЕСТУВАННЯ СИСТЕМИ

Тестування програмної системи є критично важливим етапом у процесі розробки, оскільки воно дозволяє виявити та усунути помилки, забезпечуючи таким чином високу якість та надійність продукту. Для тестування даної системи були використані кінцеві (end-to-end, e2e) тести, що дозволяють перевірити роботу всієї системи від початку до кінця з точки зору користувача.

Основним інструментом для тестування був обраний Behat – платформа для поведінкового тестування (Behavior Driven Development, BDD) на PHP. Behat дозволяє описувати поведінку системи на основі сценаріїв, написаних мовою Gherkin, яка є читабельною як для розробників, так і для інших зацікавлених сторін та забезпечує зрозумілість та прозорість тестів.

5.1 Поведінкове тестування

BDD – це методологія розробки програмного забезпечення, яка зосереджується на описі бажаної поведінки системи з точки зору користувача. За допомогою BDD розробники можуть писати сценарії, що описують конкретні ситуації використання системи, що дозволяє перевірити її роботу у реальних умовах.

Основна ідея BDD полягає у створенні зрозумілих і читабельних специфікацій, які описують поведінку системи за допомогою прикладів. Ці специфікації пишуться на природній мові, доповненій спеціальними ключовими словами (Given, When, Then), що робить їх доступними і зрозумілими для всіх учасників проекту. Приклад можна побачити на рисунку 5.1.

```
Scenario: Test results with simple graph
  Given node with name "1"
  And node with name "2"
  And node with name "3"
  And edge with source "1" target "2" and success chance 0.9
  And edge with source "2" target "3" and success chance 0.8
  And edge with source "1" target "3" and success chance 0.7
  And target probability 0.5
  When graph is send to Simple Search endpoint
  And graph is send to Structural Transformation endpoint
  Then the results should be equal
```

Рисунок 5.1 – Приклад сценарію тестування

Ці сценарії служать як документацією, так і тестовими кейсами. Вони описують конкретні ситуації використання системи, що дозволяє перевірити її роботу у реальних умовах. Використання BDD допомагає зосередитися на потребах користувачів і розробити систему, яка відповідає їхнім очікуванням.

5.2 Тестування за допомогою Behat

Behat дозволяє писати e2e тести, які моделюють реальну взаємодію користувача з системою. Це включає в себе перевірку коректності роботи інтерфейсу, взаємодії з сервером, обробки введених даних та інших аспектів роботи системи. Сценарії тестування пишуться у вигляді файлів, що містять опис кроків, які користувач має виконати, та очікуваний результат для кожного кроку.

Далі сценарії обробляється у PHP коді, і кожен крок відповідає виклику окремої функції. Behat знає, які функції йому потрібно викликати завдяки спеціальним аннотаціям, що присутні у функціях. В контексті e2e тестування, Behat відправляє запити до реального застосунку, але у тестовому середовищі, тобто зміни внесені тестами ніяк не вплинуть на стан працюючого застосунку.

Було створено наступні сценарії тестування:

- Для елементарного графа результати обох методів будуть збігатися, враховуючи незначну похибку.
- При передачі незв’язного графа у вхідних аргументах система поверне відповідну помилку.
- При використанні у вхідних аргументах у записі ребра неіснуючої вершини система поверне відповідну помилку.
- При використанні у вхідних аргументах у записі ребра неправильної ймовірності система поверне відповідну помилку.
- При використанні у вхідних аргументах неправильної цільової ймовірності система поверне відповідну помилку.

Для запуску e2e тестів, необхідно виконати команду “make start” у директорії з кодом застосунку, після чого у консолі з’являться етапи виконаних сценаріїв, а також підсумки та сумарний час виконання. Приклад можна побачити на рисунку 5.2.

```
  Scenario: Test invalid target probability                                                    #
    Given node with name "1"                                                                   #
    And node with name "2"                                                                     #
    And node with name "3"                                                                     #
    And edge with source "1" target "2" and success chance 0.5                                 #
    And edge with source "2" target "3" and success chance 0.8                                 #
    And target probability 2                                                                   #
    When graph is send to Simple Search endpoint                                               #
    Then validation error should be "Value of targetProbability must be between 0 and 1" #

5 scenarios (5 passed)
40 steps (40 passed)
0m0.08s (41.76Mb)
```

Рисунок 5.2 – Приклад успішного виконання тестів

В разі успішного виконання, усі етапи сценаріїв буде відмічено зеленим кольором. Якщо ж на певному етапі виникне помилка або неочікуваний результат, то його буде відмічено червоним.

Висновки до розділу 5

Таким чином, в цьому розділі розглянуто основні відомості про підходи та інструменти, що використовувались для тестування системи, та надано приклади сценаріїв тестування. Також було надано перелік наявних сценаріїв тестування, та надано інструкції щодо їх запуску

Таким чином, використання BDD і Behat не лише підвищило якість розробленого програмного забезпечення, але й забезпечило його відповідність вимогам. Це дозволило створити надійну систему, яка ефективно виконує свої функції та відповідає високим стандартам якості, та автоматизувати її тестування.

6 ПОРІВНЯЛЬНА ОЦІНКА МЕТОДІВ

Одним із завдань даної роботи було проаналізувати метод прямого перебору станів системи та метод структурних перетворень графа, а саме їх продуктивність та точність результатів. В цьому розділі буде проведено перевірку достовірності обох методів на модельному прикладі та аналіз їх ефективності в залежності від вхідних параметрів.

Для аналізу ефективності роботи методів використовувався комп'ютер з наступними компонентами:

- Процесор на 6 ядер та 12 потоків з частотою 4.5-5.2 ГГц
- 32 Гб оперативної пам'яті з частотою 5200 МГц
- 1 Тб SSD зі швидкістю 6000-7000 Мб/с

Треба розуміти, що характеристики ефективності методів значною мірою залежать від апаратних компонентів, зокрема вони напряму впливають на швидкість обчислень та час виконання обрахунків, а також на максимальний об'єм пам'яті, виділений алгоритмам.

6.1 Перевірка достовірності двох методів на основі модельного прикладу

Для перевірки достовірності методів було виконано порівняння результатів на декількох модельних прикладах. В першу чергу, розглянемо результати для графа, який продемонстровано на рисунку 6.1.

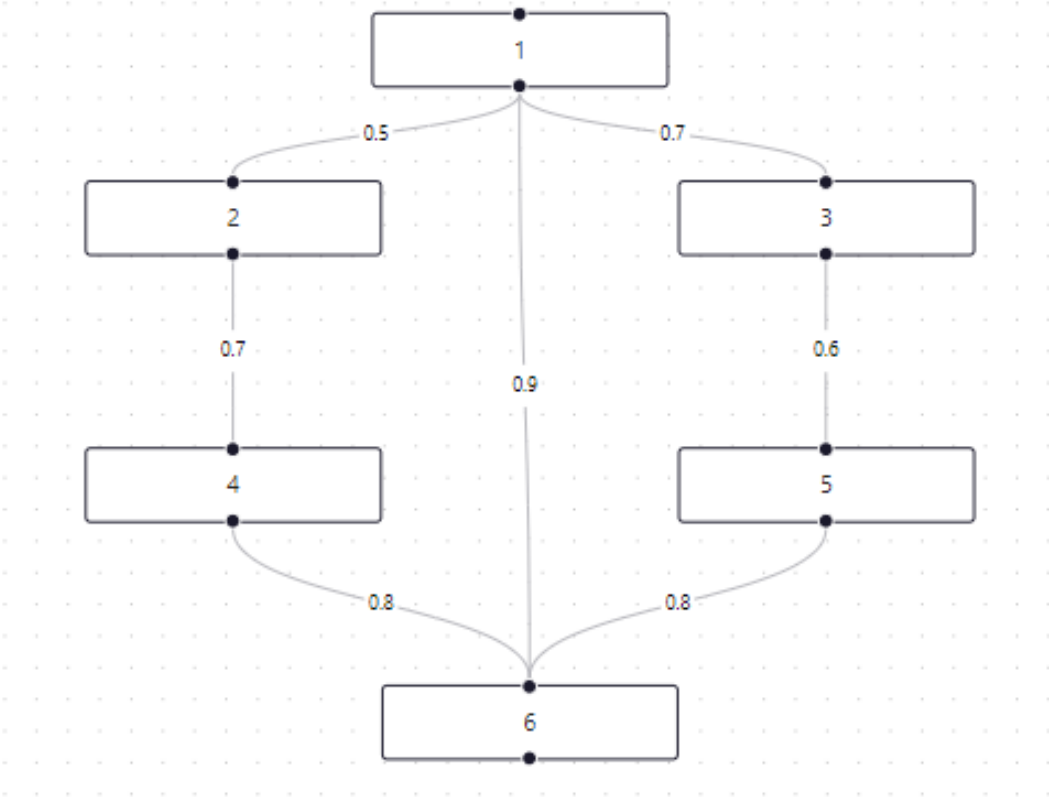


Рисунок 6.1 – Граф модельного прикладу

Матриця ймовірностей зв’язності, обраховані методом прямого перебору станів системи можна побачити у таблиці 6.1.

Таблиця 6.1 - Матриця ймовірностей зв’язності обрахована методом прямого перебору

№	1	2	3	4	5	6
1	0	0.761	0.833	0.835	0.850	0.952
2	0.761	0	0.644	0.812	0.677	0.765
3	0.833	0.644	0	0.714	0.807	0.817
4	0.835	0.812	0.714	0	0.761	0.865
5	0.850	0.677	0.807	0.761	0	0.877
6	0.952	0.765	0.817	0.865	0.877	0

Матриця ймовірностей зв’язності, обраховані методом структурних перетворень графа можна побачити у таблиці 6.2.

Таблиця 6.2 - Матриця ймовірностей зв’язності обрахована методом структурних перетворень

№	1	2	3	4	5	6
1	0	0.761	0.833	0.835	0.850	0.952
2	0.761	0	0.644	0.812	0.677	0.765
3	0.833	0.644	0	0.714	0.807	0.817
4	0.835	0.812	0.714	0	0.761	0.865
5	0.850	0.677	0.807	0.761	0	0.877
6	0.952	0.765	0.817	0.865	0.877	0

Можна побачити, що параметри функціональної стійкості обраховані обома методами повністю збігаються, за виключенням незначної похибки для значень у матриці зв’язності, якою можна знехтувати, тож можна зробити висновок, що результати обох методів є повністю достовірними, та алгоритми обрахунку були реалізовані коректно.

6.2 Аналіз ефективності методу прямого перебору

Для аналізу ефективності методу прямого перебору станів системи створимо нетривіальний граф (рисунок 6.2) зі значно більшою кількістю вершин та ребер, ніж у модельному прикладі. Це дозволить точніше відстежити, як час обрахунку параметрів даних методом змінюється в залежності від вхідних даних.

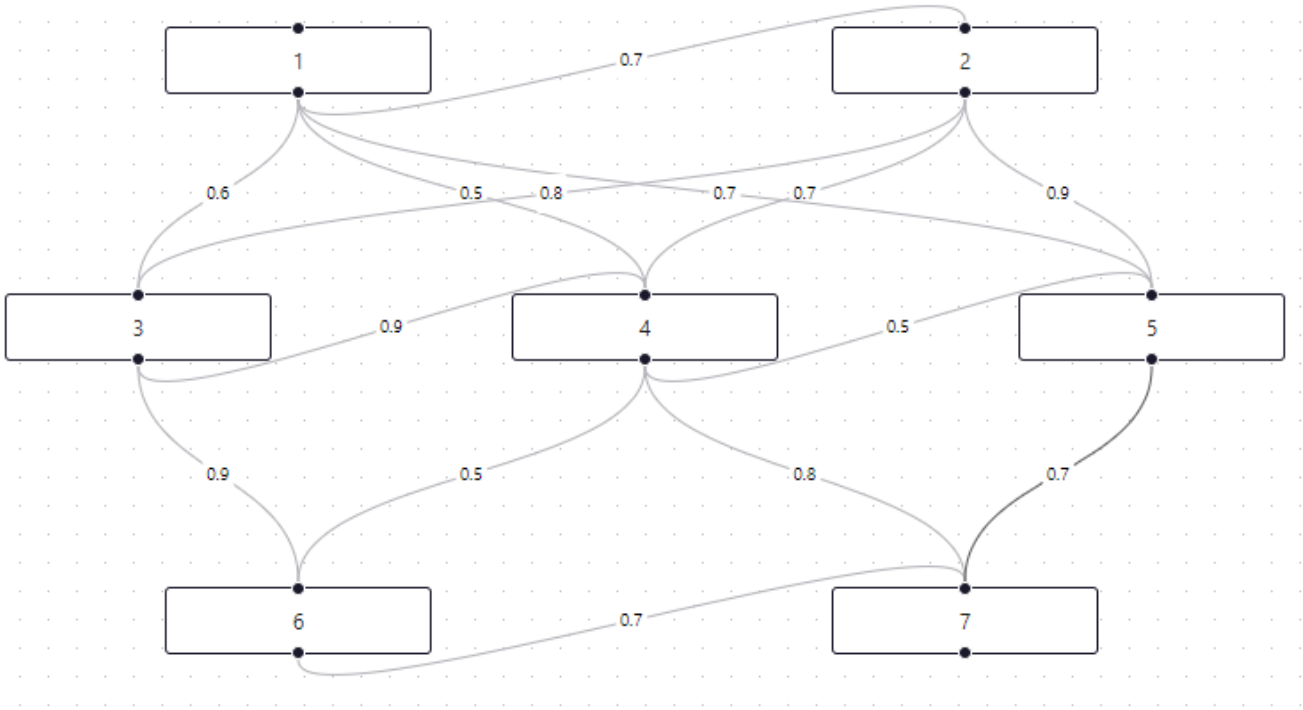


Рисунок 6.2 – Нетривіальний граф

Часова складність обрахунку ймовірності зв'язності для однієї пари вершин методом прямого перебору дорівнює двійці у степені, що дорівнює кількості ребер, і даний процес необхідно повторити для кожної унікальної пари вершин, тому остаточну часову складність алгоритму можна виразити наступним чином:

$$O\left(2^n * \frac{m\,(m-1)}{2}\right) \tag{6.1}$$

де n – кількість ребер,
m – кількість вершин.

Спираючись на формулу 6.1, можна очікувати, що при сталій кількості вершин, кожне нове додане ребро буде збільшувати час виконання вдвічі, а при сталій кількості ребер, кожна додана вершина буде збільшувати час виконання в квадратичному порядку.

Для перевірки очікуваних значень, обчислимо параметри функціональної стійкості для нетривіального графа, а також для його модифікацій, в які було додано ребро та вершину. Час виконання обчислень наведено у таблиці 6.3.

Таблиця 6.3 – Час виконання різних модифікацій нетривіального графа для методу прямого перебору

	Нетривіальний граф	Нетривіальний граф з додатковим ребром	Нетривіальний граф з додатковою вершиною
Час виконання, мс	415	789	509

Таким чином, можна побачити, що для методу прямого перебору зміна часу в залежності від кількості вершин та ребер цілком відповідає очікуваній.

6.3 Аналіз ефективності методу структурних перетворень

Для аналізу ефективності методу структурних перетворень графа також використаємо нетривіальний граф, наведений на рисунку 6.2, що дозволить не тільки провести аналіз методу, а й порівняти його продуктивність із методом прямого перебору.

Не зважаючи на те, що метод структурних перетворень значно відрізняється від методу прямого перебору, він має таку саму часову складність, яка може бути розрахована за формулою 6.1. Різниця між методами полягає в тому, що метод прямого перебору виконує розрахунки для усіх можливих комбінацій присутності та відсутності ребер, в той час як метод структурних перетворень обчислює лише ймовірності комбінацій, при яких існує шлях із початкової вершини в цільову.

Завдяки цій відмінності, метод структурних перетворень буде працювати ефективніше у більшості випадків. Результати часу виконання при обчисленні параметрів функціональної стійкості методом структурних перетворень наведено у таблиці 6.4.

Таблиця 6.4 – Час виконання різних модифікацій нетривіального графа для методу структурних перетворень

	Нетривіальний граф	Нетривіальний граф з додатковим ребром	Нетривіальний граф з додатковою вершиною
Час виконання, мс	209	429	323

Таким чином, можна побачити, що для методу структурних перетворень зміна часу в залежності від кількості вершин та ребер цілком відповідає очікуваній, та є меншою за метод прямого перебору завдяки меншій кількості обчислень.

Варто зазначити, що навіть для повних графів метод структурних перетворень буде мати більше продуктивність, завдяки рекурсивній природі алгоритму, що дозволяє виконувати менше обчислень, порівняно з ітеративним алгоритмом прямого перебору. Перевіримо це, запустивши обидва алгоритми для повного графа з рисунку 6.3.

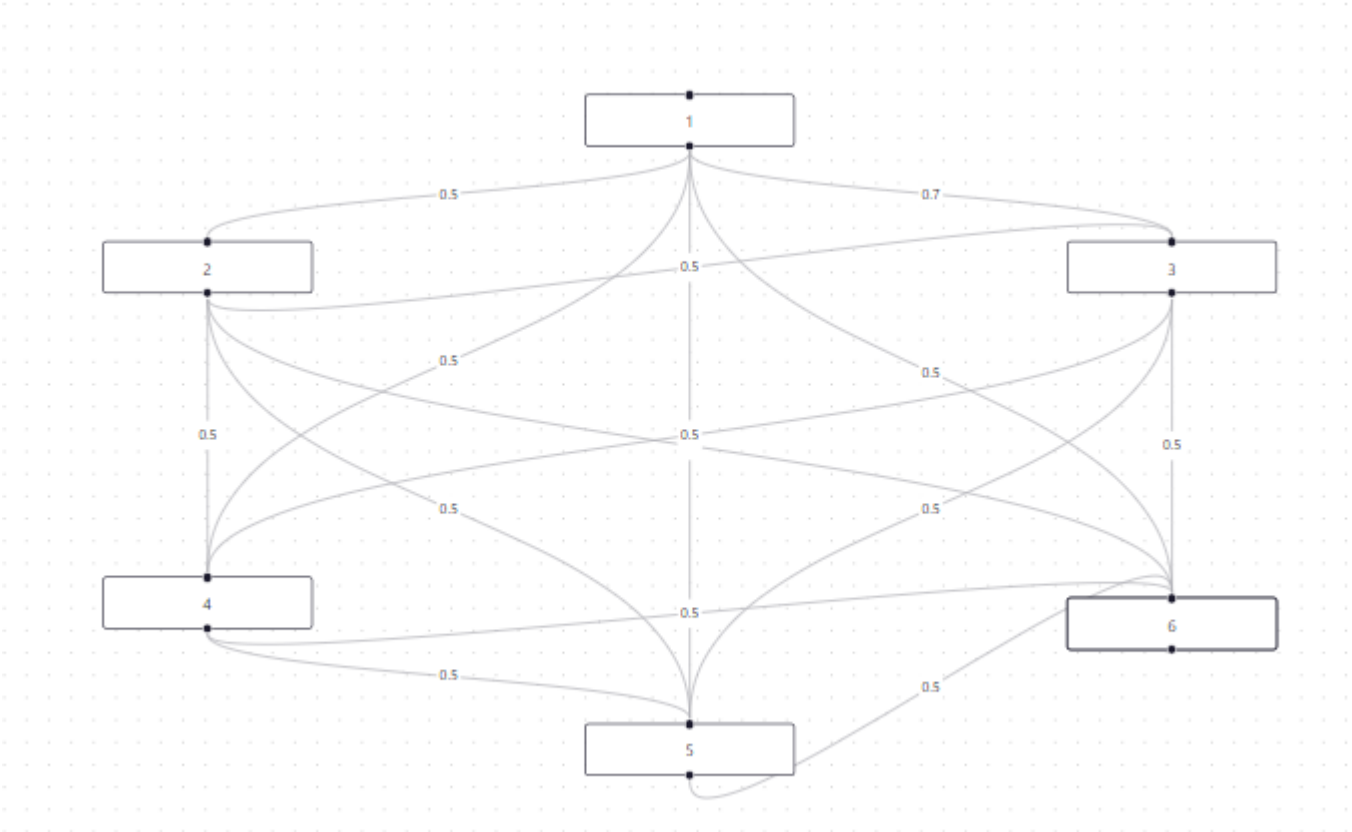


Рисунок 6.3 – Повний граф

Час виконання обчислень обома методами наведено у таблиці 6.5.

Таблиця 6.5 – Час виконання обчислень обома методами для повного графа

	Метод прямого перебору станів системи	Метод структурних перетворень графа
Час виконання, мс	587	301

Таким чином, можна побачити, що навіть при обчисленні параметрів функціональної стійкості повних графів, метод структурних перетворень має більшу продуктивність завдяки більш досконалому алгоритму.

Висновки до розділу 6

У цьому розділі було проведено перевірку достовірності двох методів на основі модельного прикладу, в результаті чого було виявлено, що результати обчислень обома методами повністю співпадають, за виключенням незначної похибки.

Також було проведено аналіз обох методів, визначено часову складність алгоритмів обчислень параметрів функціональної стійкості та продемонстровано залежність часу виконання від вхідних параметрів.

7 ПРАКТИЧНІ РЕКОМЕНДАЦІЇ ЩОДО ВИБОРУ МЕТОДУ

Одним із завдань даної роботи було порівняння між собою методі прямого перебору станів системи та структурних перетворень графа. В цьому розділі буде зроблено висновки щодо продуктивності методів на основі порівняльної оцінки та надано рекомендації щодо їх використання.

7.1 Висновки на основі порівняльної оцінки

Після проведення порівняльної оцінки, можна чітко побачити, що метод структурних перетворень графа є ефективнішим за метод прямого перебору станів системи, завдяки тому, що метод структурних перетворень не витрачає час на обрахунок ймовірності існування комбінацій присутності та відсутності ребер, при яких не існує шляху із початкової вершини в цільову.

Було визначено, що навіть для повних графів метод структурних перетворень має значно більшу продуктивність. Цей виграш у часі виконання досягається за рахунок рекурсивної природи алгоритму, що значно зменшує кількість обчислень.

Варто зазначити, що рекурсивний алгоритм має і свій недолік, а саме витрату пам’яті. Мова програмування PHP, яка використовувалась для створення програми, не має вбудованих інструментів для оптимізації рекурсій, тому в порівнянні з методом прямого перебору, для великих графів із кількістю ребер більше 20, окрім виграшу у часі виконання буде зростати і обсяг зайнятої пам’яті.

Також варто зазначити, що продуктивність методів, як і обмеження системи, та час виконання алгоритмів значною мірою залежать від апаратних компонентів. Хоча метод структурних перетворень графа і є більш продуктивним, при запуску застосунку на комп’ютері з дуже малою обчислюваною здатністю, різниця в продуктивності може бути непомітною.

7.2 Рекомендації щодо використання одного з методів або їх комбінації

Зробивши висновки на основі порівняльної оцінки методів, можна надати рекомендації щодо використання одного з них або їх комбінації.

Так як обидва методи мають однакову точність обрахунку параметрів функціональної стійкості, слід обирати метод на основі його продуктивності та ресурсів, необхідних для його роботи. Метод структурних перетворень графа дає значний виграш в часі виконання, але може потребувати значно більше пам’яті для великих графів, в той час як метод прямого перебору станів системи буде працювати повільніше, але має меншу залежність від продуктивних апаратних компонентів.

На основі цього надаємо наступні рекомендації щодо використання методів або їх комбінацій:

- Метод структурних перетворень графа рекомендовано використовувати як основний, через значно більше продуктивність у порівнянні з методом прямого перебору станів системи, за винятком випадків, коли комп’ютер має малу кількість оперативної пам’яті, що може призвести то відсутності виграшу в часі, або навіть гіршій продуктивності ніж у методу прямого перебору.
- Метод прямого перебору станів системи рекомендовано використовувати в якості перевірочного до методу структурних перетворень графа, а також у випадках, коли обсяг оперативної пам’яті замалий, щоб користуватися методом структурних перетворень, або ж обсяг зайнятої пам’яті важливіший, ніж час виконання обчислень.
- Рекомендовано використовувати обидва методи у випадках, коли надзвичайно важливою є точність обчислень, що дозволить порівняти обчислені параметри функціональної стійкості обох методів та нівелювати похибку.

Висновки до розділу 7

Таким чином, у цьому розділі було зроблено висновки на основі порівняльної оцінки методів, щодо їх ефективності та її залежності від використовуваних апаратних компонентів.

Також було надано рекомендації щодо використання методів або їх комбінацій, в залежності від цілей обчислення параметрів та наявних апаратних компонентів.

ВИСНОВКИ

Дипломна робота, присвячена розробці програмного забезпечення для обчислення параметрів функціональної стійкості мережевих інформаційних систем на основі методу структурних перетворень та методу прямого перебору станів елементів системи, дозволила отримати значний досвід у сфері розробки програмного забезпечення для вирішення складних інженерних завдань.

Під час виконання цієї роботи було досягнуто кілька важливих результатів. Зокрема, було виявлено, що метод структурних перетворень та метод прямого перебору станів елементів системи є ефективними інструментами для обчислення параметрів функціональної стійкості мережевих інформаційних систем. Ці методи дозволяють не лише проводити детальний аналіз рівня стійкості системи до впливу різноманітних зовнішніх подразників, але й оцінювати вплив змін внутрішньої структури системи на її стійкість.

Метод структурних перетворень, зокрема, має значно більшу продуктивність розрахунків для більшості сценаріїв виконання, але потребує великої кількості оперативної пам’яті. Завдяки цьому він гарно підходить для випадків, коли швидкість розрахунків стоїв в пріоритеті. Метод прямого перебору станів, в свою чергу, обчислює параметри повільніше, та потребує менше ресурсів, що робить його підходяшим для використання на старих комп’ютерах або в якості перевірочного.

Крім того, під час виконання роботи було набуто важливий досвід у розробці та використанні програмних засобів для автоматизації обчислень та аналізу функціональної стійкості систем. Зокрема, було розроблено програмне забезпечення, яке дозволяє автоматизувати процеси обчислення та аналізу, що значно знижує час і зусилля, необхідні для проведення подібних досліджень. Це програмне забезпечення може бути корисним не лише в академічних дослідженнях, але й у практичному застосуванні для аналізу та оптимізації реальних мережевих інформаційних систем.

Таким чином, результати цієї дипломної роботи мають вагоме значення як для подальших наукових досліджень у сфері функціональної стійкості мережевих інформаційних систем, так і для практичної реалізації розроблених методів та

програмного забезпечення в реальних умовах. Це дозволить покращити загальний рівень надійності та стійкості сучасних мережевих інформаційних систем, що є критично важливим в умовах постійного зростання складності та взаємозалежності інформаційних технологій.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Барабаш О.В. Построение функционально устойчивых распределенных информационных систем. Монография. К.: НАОУ, 2004. С. 47–57.
URL: http://www.irbis-nbuv.gov.ua/publ/REF-0000073745

2. Барабаш О.В. Понятійний апарат функціональної стійкості розподілених інформаційно-керуючих систем / О. В. Барабаш, С. В. Козелков, О. А. Машков // Збірник наукових праць НЦ ВПС ЗС України. – К., 2005. – Вип. № 7. –С. 87–95.URL:http://dspace.nbuv.gov.ua/bitstream/handle/123456789/20873/16-Mashkov.pdf?sequence=1

3. Барабаш О.В., Мусієнко А.П. Основні поняття функціональної стійкості бездротової сенсорної мережі // Актуальні проблеми інформаційних технологій: Науково-технічної конференції молодих учених. – К.: КНУ, 2017. – С. 39–40.
URL:https://tit.dut.edu.ua/index.php/telecommunication/article/view/2255/2153

4. Собчук В.В., Барабаш О.В., Мусієнко А.П., Лаптєв О.А. Аналіз основних підходів та етапів щодо забезпечення властивості функціональної стійкості інформаційних систем підприємства. Sciences of Europe, Praha: Sciences of Europe, 2019. Vol 1, No 42. P. 41–44. URL:https://www.europe-science.com/wp-content/uploads/2020/10/VOL-1-No-42-2019.pdf

ДОДАТОК А

Програмне забезпечення для обчислення параметрів функціональної стійкості мережевих інформаційних систем на основі методу структурних перетворень та методу прямого перебору станів елементів системи

Фрагмент лістингу програми

Аркушів 8

Київ – 2024

```php
<?php

declare(strict_types=1);

namespace App\FunctionalStability\Domain\Entity;

use App\FunctionalStability\Infrastructure\FunctionalStabilityRepository;
use Doctrine\DBAL\Types\Types;
use Doctrine\ORM\Mapping as ORM;
use Symfony\Bridge\Doctrine\Types\UuidType;
use Symfony\Component\Uid\Uuid;

#[ORM\Entity(repositoryClass: FunctionalStabilityRepository::class)]
final class FunctionalStability
{
    private const SEPARATOR = '|';

    public function __construct(private array $graph)
    {
    }

    #[ORM\Id]
    #[ORM\Column(type: UuidType::NAME)]
    private ?Uuid $id = null;

    #[ORM\Column]
    private ?int $xG = null;

    #[ORM\Column]
    private ?int $alphaG = null;

    #[ORM\Column(type: Types::JSON)]
    private array $probabilities = [];

    public function getId(): ?Uuid
    {
        return $this->id;
    }

    public function getXG(): ?int
    {
        return $this->xG;
    }

    public function setXG(int $xG): static
    {
        $this->xG = $xG;

        return $this;
    }

    public function getAlphaG(): ?int
    {
        return $this->alphaG;
    }

    public function setAlphaG(int $alphaG): static
    {
        $this->alphaG = $alphaG;

        return $this;
    }
```

```
public function getProbabilities(): array
{
    return $this->probabilities;
}

public function setProbabilities(array $probabilities): static
{
    $this->probabilities = $probabilities;

    return $this;
}

public function setId(Uuid $id): static
{
    $this->id = $id;

    return $this;
}

public function countXG(array $graph = []): int
{
    $xG = 1;
    if (!$graph) {
        $graph = $this->graph;
    }

    // По черзі прибираємо кожну з вершин, аби пересвідчитись, що граф залишить з'язним
    for ($i = 0; $i < count($graph['nodes']); $i++) {
        $tempGraph = $this->removeNodeFromGraph($graph, $i);
        if (!$this->isConnectedGraph($tempGraph)) {
            return $xG;
        }
    }

    // Якщо так, збільшуємо показник на 1 та повторюємо алгоритм вже без одної вершини
    return $xG + $this->countXG($this->removeNodeFromGraph($graph, 0));
}

public function countAlphaG(array $graph = [])
{
    $alphaG = 1;
    if (!$graph) {
        $graph = $this->graph;
    }

    // По черзі прибираємо кожне з ребер, аби пересвідчитись, що граф залишить з'язним
    for ($i = 0; $i < count($graph['edges']); $i++) {
        $tempGraph = $this->removeEdgeByIndex($graph, $i);
        if (!$this->isConnectedGraph($tempGraph)) {
            return $alphaG;
        }
    }

    // Якщо так, збільшуємо показник на 1 та повторюємо алгоритм вже без одного ребра
    return $alphaG + $this->countAlphaG($this->removeEdgeByIndex($graph, 0));
}

public function countProbabilitiesSimpleSearch(): array
{
    $graph = $this->graph;
    $edges = $graph['edges'];
```

```
    $nodePairs = $this->getAllNodePairs($graph['nodes']);
    $edgeCombinations = $this->getAllEdgeCombinations($edges);

    $result = [];
    foreach ($nodePairs as $pair) {
        $source = $pair[0];
        $target = $pair[1];
        $probability = 0;
        foreach ($edgeCombinations as $combination) {
            if ($this->hasPathBetweenNodesWithEdgeCombination($edges, $source, $target, $combination)) {
                $temp = 1;
                for ($i = 0; $i < count($combination); $i++) {
                    if ($combination[$i]) {
                        $temp *= $edges[$i]["successChance"];
                    } else {
                        $temp *= 1 - $edges[$i]["successChance"];
                    }
                }
                $probability += $temp;
            }
        }

        $result[] = ['source' => $source, 'target' => $target, 'probability' => $probability];
    }

    return $result;
}

public function countProbabilitiesStructuralTransformation(): array
{
    $graph = $this->graph;
    $nodePairs = $this->getAllNodePairs($graph['nodes']);

    $result = [];
    foreach ($nodePairs as $pair) {
        $source = $pair[0];
        $target = $pair[1];
        $probability = $this->countProbabilityForNodePairStructuralTransformation($graph, $source, $target);

        $result[] = ['source' => $source, 'target' => $target, 'probability' => $probability];
    }

    return $result;
}

private function getAllNodePairs($nodes): array
{
    $pairs = [];
    $numNodes = count($nodes);

    for ($i = 0; $i < $numNodes; $i++) {
        for ($j = $i + 1; $j < $numNodes; $j++) {
            $pairs[] = [$nodes[$i], $nodes[$j]];
        }
    }

    return $pairs;
}

private function getAllEdgeCombinations($edges): array
{
    $numEdges = count($edges);
```

```
    $combinations = [];

    // Генеруємо всі числа від 0 до 2^numEdges - 1
    for ($i = 0; $i < pow(2, $numEdges); $i++) {
        $combination = [];
        // Перетворимо число на двійковий рядок довжиною numEdges
        $binary = str_pad(decbin($i), $numEdges, '0', STR_PAD_LEFT);
        // Для кожного ребра визначаємо його стан (присутнє чи відсутнє)
        for ($j = 0; $j < $numEdges; $j++) {
            $combination[] = $binary[$j] === '1'; // Перетворюємо '1' на true, '0' на false
        }
        $combinations[] = $combination;
    }

    return $combinations;
}

private function hasPathBetweenNodesWithEdgeCombination($edges, $source, $target, $edgeCombination): bool
{
    $graph = $this->buildGraph($edges, $edgeCombination);
    $visited = [];
    return $this->hasPathDFS($graph, $source, $target, $visited);
}

// Функція пошуку в глибину (DFS) для перевірки існування шляху між вершинами
private function hasPathDFS($graph, $source, $target, &$visited): bool
{
    if ($source === $target) {
        return true; // Найден путь
    }
    if (!key_exists($source, $graph)) {
        return false;
    }
    $visited[$source] = true;
    foreach ($graph[$source] as $neighbor) {
        if (!isset($visited[$neighbor])) {
            if ($this->hasPathDFS($graph, $neighbor, $target, $visited)) {
                return true; // Найден путь
            }
        }
    }
    return false; // Путь не найден
}

// Функція для побудови графа на основі списку ребер та комбінації ребер
private function buildGraph($edges, $edgeCombination): array
{
    $graph = [];
    foreach ($edges as $key => $edge) {
        if ($edgeCombination[$key]) { // Перевіряємо, чи є ребро в комбінації
            $source = $edge['source'];
            $target = $edge['target'];
            $graph[$source][] = $target;
            // Враховуємо і зворотний напрямок ребра
            $graph[$target][] = $source;
        }
    }
    return $graph;
}

private function hasPathDFSWithContractedNodes($graph, $source, $target, &$visited): bool
{
```

```
        // Перевіряємо, чи існує вершина $source у графі
        $found = false;
        foreach ($graph['nodes'] as $node) {
            if (str_contains($node, $source) && str_contains($node, $target)) {
                return true;
            }
            if ($node === $source || strpos($node, self::SEPARATOR) !== false && in_array($source,
explode(self::SEPARATOR, $node))) {
                $found = true;
                break;
            }
        }
        if (!$found) {
            return false;
        }

        // Перевіряємо, чи ми досягли цільової вершини
        if ($source === $target) {
            return true; // Найден путь
        }

        // Перевіряємо, чи є сусіди поточної вершини
        if (!isset($graph['edges']) || empty($graph['edges'])) {
            return false;
        }

        // Позначаємо поточну вершину як відвідану
        $visited[$source] = true;

        // Перебираємо всі ребра графа
        foreach ($graph['edges'] as $edge) {
            $edgeSource = $edge['source'];
            $edgeTarget = $edge['target'];

            // Перевіряємо, чи поточне ребро інцидентно нашій вершині
            if (strpos($edgeSource, $source) !== false || strpos($edgeTarget, $source) !== false) {
                // Определяем вершину, к которой ведет текущее ребро
                $neighbor = ($edgeSource === $source) ? $edgeTarget : $edgeSource;

                // Перевіряємо, чи відвідували ми вже цю вершину
                if (!isset($visited[$neighbor])) {
                    // Якщо ми ще не відвідували цю вершину, рекурсивно шукаємо шлях від неї до цільової вершини
                    if ($this->hasPathDFSWithContractedNodes($graph, $neighbor, $target, $visited)) {
                        return true; // Знайдено шлях
                    }
                }
            }
        }

        // Якщо ми дійшли до цієї точки, значить, шлях не знайдено
        return false;
    }

    private function countProbabilityForNodePairStructuralTransformation(
        array $graph,
        string $source,
        string $target,
    ): float {
        $probability = 0;
        $edgeIndex = 0;
        $edge = $graph['edges'][$edgeIndex];
```

```
        // Стягуємо ребро графа
        $graphWithContractedEdge = $this->contractEdge($graph, $edgeIndex);
        if (count($graphWithContractedEdge['edges']) > 0) {
            // Якщо залишилось більше одно ребра, рекурсивно повторюємо алгоритм
            $probability += ($edge["successChance"] * $this-
>countProbabilityForNodePairStructuralTransformation($graphWithContractedEdge, $source, $target));
        } else {
            $probability += $edge["successChance"];
        }

        // Прибираємо ребро графа
        $graphWithRemovedEdge = $this->removeEdgeByIndex($graph, $edgeIndex);
        $visited = [];
        if ($this->hasPathDFSWithContractedNodes($graphWithRemovedEdge, $source, $target, $visited)) {
            if (count($graphWithRemovedEdge['edges']) > 0) {
                // Якщо залишилось більше одно ребра, рекурсивно повторюємо алгоритм
                $probability += ((1 - $edge["successChance"]) * $this-
>countProbabilityForNodePairStructuralTransformation($graphWithRemovedEdge, $source, $target));
            } else {
                $probability += (1 - $edge["successChance"]);
            }
        }

        return $probability;
    }

    private function contractEdge(array $graph, int $edgeIndex): array
    {
        // Отримуємо ребро за індексом
        $edge = $graph['edges'][$edgeIndex];
        $source = $edge['source'];
        $target = $edge['target'];

        // Створюємо нову вершину, яка об'єднує вершини source та target
        $newNode = $source . self::SEPARATOR . $target;

        // Видаляємо вершини source і target зі списку вершин
        $nodes = array_diff($graph['nodes'], [$source, $target]);

        // Додаємо нову вершину до списку вершин
        $nodes[] = $newNode;

        // Створюємо нові ребра, що з'єднують нову вершину з вершинами, суміжними source та target
        $edges = [];
        foreach ($graph['edges'] as $currentEdgeIndex => $currentEdge) {
            $currentSource = $currentEdge['source'];
            $currentTarget = $currentEdge['target'];

            // Виключаємо ребро, яким здійснювалося стягування
            if ($currentEdgeIndex === $edgeIndex) {
                continue;
            }

            // Якщо поточне ребро інцидентно однією з вершин, що стягуються, замінюємо її новою вершиною
            if ($currentSource === $source || $currentSource === $target) {
                $currentEdge['source'] = $newNode;
            }
            if ($currentTarget === $source || $currentTarget === $target) {
                $currentEdge['target'] = $newNode;
            }

            // Додаємо ребро до списку, якщо воно не було виключено
```

```
            $edges[] = $currentEdge;
        }

        return [
            "nodes" => $nodes,
            "edges" => $edges
        ];
    }

    private function removeEdgeByIndex(array $graph, int $index): array
    {
        // Перевіряємо, чи існує ребро із зазначеним індексом у списку ребер
        if (isset($graph['edges'][$index])) {
            // Видаляємо ребро із зазначеним індексом зі списку ребер
            unset($graph['edges'][$index]);

            // Переіндексуємо масив ребер, щоб індекси починалися з 0
            $graph['edges'] = array_values($graph['edges']);
        }

        // Повертаємо новий граф із віддаленим рубом
        return $graph;
    }

    public function isConnectedGraph(array $graph = []): bool
    {
        if (!$graph) {
            $graph = $this->graph;
        }
        $nodes = $graph['nodes'];
        $edges = $graph['edges'];
        $n = count($nodes);

        if (count($nodes) < 1 || count($edges) < 1) {
            return false;
        }

        // Створюємо матрицю суміжності для графа
        $adjMatrix = [];
        foreach ($nodes as $node1) {
            $adjMatrix[$node1] = array_fill(0, $n, false);
        }
        foreach ($edges as $edge) {
            $source = $edge['source'];
            $target = $edge['target'];
            $adjMatrix[$source][$target] = true;
            $adjMatrix[$target][$source] = true; // Враховуємо двосторонні ребра
        }

        // Вибираємо довільну вершину як початкову для обходу
        $startNode = reset($nodes);

        // Перевіряємо зв'язок графа, запускаючи DFS від початкової вершини
        $visited = [];
        $this->dfs($adjMatrix, $startNode, $visited);

        // Якщо всі вершини відвідані, граф зв'язний
        return count($visited) === $n;
    }

    private function dfs($adjMatrix, $source, &$visited): void
    {
```

```
        // Позначаємо поточну вершину як відвідану
        $visited[$source] = true;

        // Рекурсивно обходимо всі суміжні вершини
        foreach (array_keys($adjMatrix[$source]) as $neighbor) {
            if ($adjMatrix[$source][$neighbor] && !isset($visited[$neighbor])) {
                $this->dfs($adjMatrix, $neighbor, $visited);
            }
        }
    }

    private function removeNodeFromGraph(array $graph, int $nodeIndex): array
    {
        $vertexToRemove = $graph['nodes'][$nodeIndex];

        // Видаляємо всі ребра, інцидентні заданій вершині
        $edgesToRemove = [];
        foreach ($graph['edges'] as $key => $edge) {
            if ($edge['source'] === $vertexToRemove || $edge['target'] === $vertexToRemove) {
                $edgesToRemove[] = $key;
            }
        }
        foreach ($edgesToRemove as $key) {
            unset($graph['edges'][$key]);
        }
        $graph['edges'] = array_values($graph['edges']); // Перенумеруємо ключі

        // Видаляємо саму задану вершину зі списку вершин графа
        unset($graph['nodes'][$nodeIndex]);
        $graph['nodes'] = array_values($graph['nodes']); // Перенумеруємо ключі

        return $graph;
    }
}
```

ДОДАТОК Б

Програмне забезпечення для обчислення параметрів функціональної стійкості мережевих інформаційних систем на основі методу структурних перетворень та методу прямого перебору станів елементів системи

Презентація

Аркушів 7

Київ – 2024

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці

Програмне забезпечення для обчислення параметрів функціональної стійкості мережевих інформаційних систем на основі методу структурних перетворень та методу прямого перебору станів елементів системи

Курносенко Сергій Віталійович, ТВ-01
Барабаш Олег Володимирович, професор, д.т.н.

2024

Актуальність теми

Мета дипломної роботи полягає у розробці програмного забезпечення для обчислення параметрів функціональної стійкості мережевих інформаційних систем на основі методу структурних перетворень та методу прямого перебору станів елементів системи.

Під функціональною стійкістю розподілених інформаційних систем розуміється властивість системи зберігати протягом заданого часу виконання своїх основних функцій в межах, встановлених нормативними вимогами, при впливі потоку експлуатаційних відмов, збоїв, пошкоджень, умисного шкідництва, втручання в обмін та обробку інформації, а також при помилках обслуговуючого персоналу.

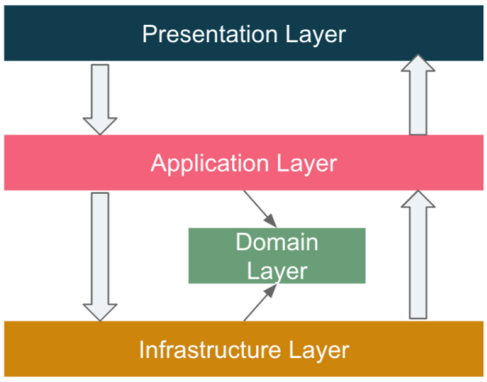
Постановка задачі

Для досягнення мети роботи потрібно було виконати наступні задачі:

- Розробити серверний застосунок для обчислення параметрів функціональної стійкості, а саме матриці ймовірності зв’язності, вершинної та реберної.
- Розробити інтуітивно зрозумілий інтерфейс користувача.
- Забезпечити можливість інтеграції з іншими системами
- Автоматизувати тестування застосунку.
- Порівняти ефективність методів та надати рекоменації щодо використання одного з них або їх комбінації.

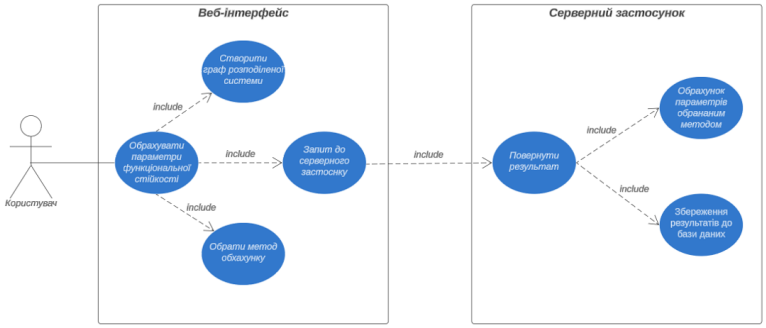
Проєктування системи

Архітектура серверного застосунку ґрунтується на принципах DDD (Domain Driven Design), що надає цілий ряд переваг, які сприяють створенню гнучкого та розширюваного програмного рішення, в той час як інтерфейс являє собою окремий веб застосунок розроблений за архітектурою SPA (Single Page Application)



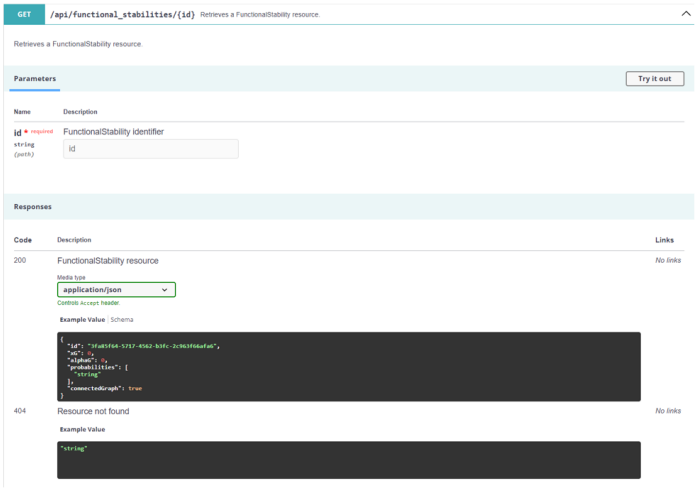
Проєктування системи

Користувач взаємодіє лише із веб-інтерфейсом, який надсилає запити до серверного застосунку, що обраховує параметри, зберігає результат у базу даних та повертає результат, який відображається в інтерфейсі після обробки даних.



Проєктування системи

Завдяки тому, що інтерфейс користувача є окремим веб-застосунком, який ніяк не пов’язаний з серверною частиною, та наявності документації, існує можливість інтеграції методів обчислення параметрів функціональної стійкості мережевиї інформаційних систем до інших застосунків або кластерів мікросевісів.



Проєктування системи

Взаємодія інтерфейсу та серверного застосунку відбувається у вигляді HTTP запитів, що надсилають дані у форматі JSON.

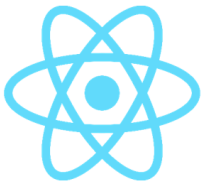

```
{
    "nodes":["0","1","2"],
    "edges":[
        {
            "source":"0",
            "target":"1",
            "successChance":0.65
        },
        {
            "source":"0",
            "target":"2",
            "successChance":0.78
        }
    ],
    "targetProbability": 0.5
}
```



```
"content": {
  "isStable": false,
  "id": "018ff1ca-a9af-777f-9fc2-86da7d8e06f7",
  "execTimeMilliseconds": "2.699",
  "λG": 1,
  "λG": 1,
  "probabilityMatrix": [
    {
      "source": "0",
      "target": "1",
      "probability": "0.650"
    },
    {
      "source": "0",
      "target": "2",
      "probability": "0.780"
    },
    {
      "source": "1",
      "target": "2",
      "probability": "0.507"
    }
  ]
},
```


Реалізація системи

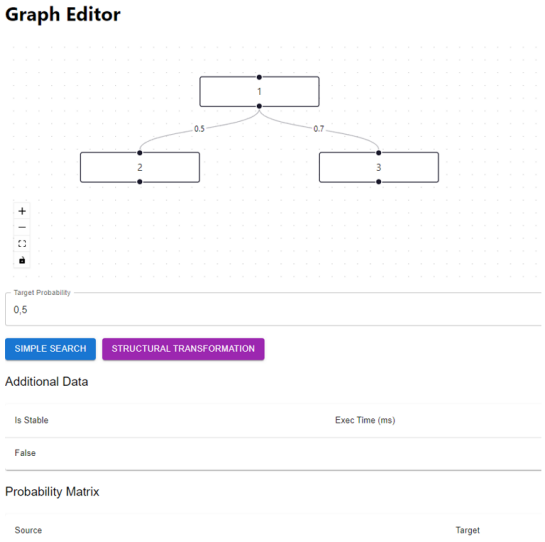
Для розробки системи було використано наступні технології:



Реалізація системи

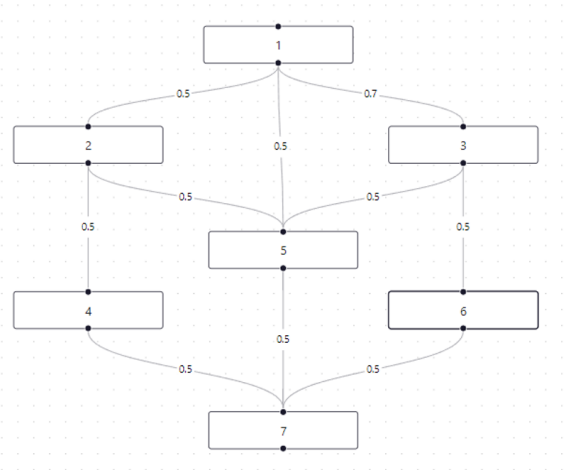
У інтерфейсі користувач може виконувати наступні дії:

- Створювати та видаляти вершини графа
- Створювати, видаляти та редагувати ребра
- Розраховувати параметри функціональної стійкості обома методами
- Вільно переміщати елементи графа в робочому просторі



Реалізація системи

Приклад створеного графу та результати обчислень параметрів



Additional Data

Is Stable	Exec Time (ms)	x(G)	λ(G)
False	64.004	3	3

Probability Matrix

Source	Target	Value
1	2	0.704
1	3	0.812
1	4	0.490
1	5	0.783
1	6	0.532

Реалізація системи

Стала кількості вершин (10)

Кількість ребер	Прямий перебір	Структурні перетворення
6 ребер	8-15мс	2-10мс
20 ребер	85с	28с
21 ребро	167с	53с
22 ребра	325с	112с

Стала кількість ребер (20)

Кількість вершин	Прямий перебір	Структурні перетворення
6 ребер	8-15мс	2-10мс
10 вершин	74с	28с
11 вершин	87с	38с
12 вершин	113с	46с

$$O\left(2^n * \frac{m\,(m-1)}{2}\right)$$

де n – кількість ребер,
m – кількість вершин.

Тестування

Для тестування використовувався фреймворк Behat, який дозволяє описувати сценарії тестування у стилі BDD на мові Gherkin. З його використанням було втілено тестування кінцевих точок застосунку, включно з перевіркою достовірності методів.

```
Feature:
  In order to prove that the Functional Stability Analyzer works correctly
  As a user
  I want to be able to get functional stability parameters

  Scenario: Test results with simple graph
    Given node with name "1"
    And node with name "2"
    And node with name "3"
    And edge with source "1" target "2" and success chance 0.9
    And edge with source "2" target "3" and success chance 0.8
    And edge with source "1" target "3" and success chance 0.7
    And target probability 0.5
    When graph is send to Simple Search endpoint
    And graph is send to Structural Transformation endpoint
    Then the results should be equal
```

Впровадження та супровід

Завдяки наявності спільної інфраструктури усіх компонентів системи, її розгортання зводиться до 2 простих кроків:

1. Клонування застосунку з платформи GitHub
2. Запуск команди “make start” у кореневій папці проекту

Система має й інші make команди, зокрема для запуску тесті, керування кешем та залежностями, а також налагодження застосунку.

У коді програми присутні коментарі, що пояснюють логіку роботи найважливіших компонентів.

Висновки

Таким чином, було розроблено систему, здатну обчислювати параметри функціональної стійкості мережевих інформаційних систем двома методами і яка надає користувачеві базовий графічний інтерфейс для побудови графів і перегляду результатів виконання.

Було створено можливість інтеграції системи до інших застосунків та автоматизовано тестування системи.

Також було проведено порівняння ефективності обох методів та надані рекомендації щодо використання одного з них або їх комбінації.