ДВНЗ «Донецький національний технічний університет»

Факультет комп'ютерно-інформаційних технологій і автоматизації
(повне найменування інституту, назва факультету)

Кафедра комп'ютерної інженерії
(повна назва кафедри)

«До захисту допущено»
Завідувач кафедри
_____ Сергій КОВАЛЬОВ
(підпис) (ініціали, прізвище)

«___»_____2022 р.

Випускна кваліфікаційна робота

бакалавра
(освітньо-кваліфікаційний ступінь)

На тему «Засоби топологічного аналізу MIMD-симулятора мережевого динамічного об'єкту з зосередженими параметрами»

Виконала: студентка 4 курсу, групи КІ-18
(шифр групи)

спеціальності 123 Комп'ютерна інженерія
(шифр і назва спеціальності)

Тихонова Ю. Г. _____
(прізвище та ініціали) (підпис)

Керівник Проф.каф. КІ, д.т.н., проф. Володимир СВЯТНИЙ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент Зав. кафедри ПМІ, д. т. н., проф. Ольга ДМИТРІЄВА _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Нормоконтроль:
_____ к.т.н., Юлія ДІКОВА

Засвідчую, що у цій випускній кваліфікаційній роботі немає запозичень з праць інших авторів без відповідних посилань.

Студентка _____
(підпис)

Луцьк– 2022 р.

ДВНЗ «Донецький національний технічний університет»

Факультет комп'ютерно-інформаційних технологій і автоматизації

Кафедра комп'ютерної інженерії

Освітній ступінь Бакалавр

Спеціальність 123 Комп'ютерна інженерія,
(шифр і назва)

ЗАТВЕРДЖУЮ:
Завідувач кафедри
_____ Сергій КОВАЛЬОВ
"____" _____2022 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТЦІ

Тихоновій Юлії Григорівни
(прізвище, ім'я, по батькові)

1. Тема роботи «Засоби топологічного аналізу MIMD-симулятора мережевого динамічного об'єкту з зосередженими параметрами»

Керівник роботи Володимир СВЯТНИЙ, д.т.н., проф.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом від 06.05.2022 р. № 180

2. Строк подання студенткою роботи 07.06.2022 р.

3. Вихідні дані до роботи результати науково-дослідної роботи
результати переддипломної практики

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити):

1. Проблема паралельного моделювання мережевого динамічного об'єкту з зосередженими параметрами (МДОЗП)
2. Топологічні характеристики МДОЗП
3. Алгоритми генерування топологічних характеристик
4. Програмна імплементація та експериментальні дослідження

Консультанти розділів кваліфікаційної роботи:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

6. Дата видачі завдання 22.04.2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	*Назва етапів випускної кваліфікаційної роботи*	*Строк виконання етапів роботи*	*Примітка*
1	Аналіз предметної області	22.04.22 – 24.04.22	
2	Постановка завдання	25.04.22 – 26.04.22	
3	Огляд тематики роботи, актуальність роботи, формулювання мети на задач дослідження	27.04.22 – 03.05.22	
4	Опис та обґрунтування основних ідей роботи	04.05.22 – 07.05.22	
5	Розробка алгоритмів	08.05.22 – 14.05.22	
6	Програмна реалізація, аналіз результатів дослідження, формулювання висновків та тенденцій	15.05.22 – 31.05.22	
7	Оформлення пояснювальної записки	01.06.22 – 07.06.22	
8	Нормоконтроль пояснювальної записки та додаткових матеріалів	08.06.22 – 12.06.22	
9	Рецензування роботи	13.06.22 – 15.06.22	
10	Захист роботи	24.06.22	

Студентка _____ Юлія ТИХОНОВА
(підпис) (прізвище та ініціали)

Керівник роботи  Володимир СВЯТНИЙ
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Тихонова Ю. Г. Засоби топологічного аналізу MIMD-симулятора мережевого динамічного об’єкту з зосередженими параметрами / Випускна кваліфікаційна робота на здобуття освітнього ступеня «бакалавр» за спеціальністю 123 Комп’ютерна інженерія. – ДВНЗ ДонНТУ, Луцьк, 2022.

Мета даної роботи полягає в розробці, реалізації та експериментальних дослідженнях засобів топологічного аналізу, що забезпечать активну комп’ютерну підтримку побудови паралельних симуляторів МДОЗП.

В кваліфікаційній роботі вирішено такі задачі: топологічний аналіз тестового МДОЗП вручну (проведено початкове кодування графу, побудовано дерево та антидерево, відносно яких було структуровані матриці інциденцій, незалежних контурів, параметрів та векторів), генерування топологічних характеристик та їх подальше структурування відносно дерева й антидерева за допомогою діючого ПЗ, його імплементація та експериментальне дослідження, тестування на прикладі тестового графу.

Практичним результатом кваліфікаційної роботи є: діюче ПЗ, за допомогою якого можна зробити рутинну роботу побудови топологічних характеристик за користувача (розробника).

Ключові слова:

МЕРЕЖЕВИЙ ДИНАМІЧНИЙ ОБ’ЄКТ З ЗОСЕРЕДЖЕНИМИ ПАРАМЕТРАМИ (МДОЗП), ПАРАЛЕЛЬНЕ МОДЕЛЮВАННЯ, MIMD-СИСТЕМА, SIMULATION-МОДЕЛЬ, MIMD-СИМУЛЯТОР, ТОПОЛОГІЧНІ ХАРАКТЕРИСТИКИ МДОЗП.

ANNOTATION

Tykhonova Y. G. Means of topological analysis of MIMD-simulator of network dynamic object with concentrated parameters / Graduate qualification work for bachelor's degree in 123 Computer Engineering of DonNTU, Lutsk, 2022.

The purpose of this work is to develop, implement and conduct experimental research of means of topological analysis that will provide active computer support for the construction of parallel NDOCP simulators.

The following tasks were solved in the graduate qualification work: topological analysis of test NDOCP manually (initial coding of the graph, constructing matrices of incidents, independent loops, parameters and vectors which were based on tree and antitree), generation of topological characteristics and their subsequent structuring for tree and antitree using working software, its implementation and experimental research, testing on the example of a test graph.

The practical result of the qualification work is: the working software, with which you can do routine work to construct topological characteristics for the user (developer).

Keywords:

NETWORK DYNAMIC OBJECT WITH CONCENTRATED PARAMETERS (NDOCP), PARALLEL MODELING, MIMD-SYSTEM, SIMULATION-MODEL, MIMD-SIMULATOR, TOPOLOGICAL CHARACTERISTICS OF NDOCP

ЗМІСТ

ВСТУП 9
1 ПРОБЛЕМА ПАРАЛЕЛЬНОГО МОДЕЛЮВАННЯ МЕРЕЖЕВОГО ДИНАМІЧНОГО ОБ'ЄКТУ З ЗОСЕРЕДЖЕНИМИ ПАРАМЕТРАМИ (МДОЗП). МАТЕМАТИЧНА МОДЕЛЬ МДОЗП 11
1.1 Постановка задачі паралельного моделювання. Етапи розробки паралельного симулятора 11
1.2 Засоби комп'ютерної підтримки розробки паралельного симулятора МДОЗП 16
1.3 Задачі розробок і досліджень 17
2 ТОПОЛОГІЧНІ ХАРАКТЕРИСТИКИ МДОЗП 18
2.1 Граф МДОЗП та його початкове кодування, рівняння вузлів і гілок 18
2.2 Дерево й антидерево графа, матриці інциденцій і незалежних контурів 21
2.3 Модель та Simulation-модель МДОЗП у векторно-матричній формі [3] 24
2.4 Дискретна Simulation-модель МДОЗП і блок-схема MIMD-симулятора 27
2.5 Висновки 29
3 АЛГОРИТМИ ГЕНЕРУВАННЯ ТОПОЛОГІЧНИХ ХАРАКТЕРИСТИК 30
3.1 Діалогова підтримка побудови таблиці кодування графа МДОЗП 30
3.2 Алгоритми генерування дерева й антидерева 32
3.3 Алгоритм генерування матриці інциденцій 36
3.4 Алгоритм генерування матриці незалежних контурів 40
3.5 Формування матриць параметрів і векторів 43
3.6 Висновки 44
4 ПРОГРАМНА ІМПЛЕМЕНТАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ 46
4.1 Структура паралельного симулятора 46

4.2 Програми реалізації розроблених алгоритмів 52
4.3 Експериментальні дослідження 60
4.4 Висновки 62
ВИСНОВКИ 63
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 64
ДОДАТОК А – Заява на затвердження теми і керівника 68
ДОДАТОК Б – Блок-схеми алгоритмів 69
Б.1 Алгоритм генерування дерева й антидерева 69
Б.2 Алгоритм генерування матриці інциденцій 70
Б.3 Алгоритм генерування матриці незалежних контурів 71
ДОДАТОК В – Діаграми класів 73
ДОДАТОК Г – Лістинг ПЗ 75
Г.1 Файли заголовків 75
Г.2 Вихідні файли 78
ДОДАТОК Д – Перелік зауважень нормоконтролера 91

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ

ВКМ	Віртуальна комунікаційна мережа
ВПМ	Віртуальна паралельна модель
ВПС	Віртуальна паралельна система
ВПСМ	Віртуальна паралельна симмодель
ДМО	Динамічний мережевий об’єкт
ДР	Диференціальне рівняння
ДСЗП	Динамічні системи із зосередженими параметрами
ДСРП	Динамічні системи з розподіленими параметрами
ЗРЧ	Задачі реального часу
МДОЗП	Мережевий динамічний об’єкт з зосередженими параметрами
ПЗ	Програмне забезпечення
ПМС	Паралельне моделююче середовище
РКМ	Реальна комунікаційна мережа
РПМС	Розподілене ПМС
САПР	Система автоматизованого проектування
САУП	Системи автоматизованого управління провітрюванням
СДС	Складні динамічні системи
ССА	Структура системи автоматизації
СТЕ	Симуляційний тестовий експеримент
ТС	Технологічна схема
ЦА	Цільова архітектура
ЦПС	Цільова паралельна система
ШВМ	Шахтні вентиляційні мережі
MPI	Massage Passing Interface

ВСТУП

Актуальність теми. Мережеві динамічні об’єкти з зосередженими параметрами (МДОЗП) широко представлені в різних ґалузях техніки й технологій, їх математичне моделювання є актуальною проблемою. Для того, щоб перевірити правильність моделювання, використовуються різні методи, які можуть все більше гарантувати надійність проєктованих систем, так як вони весь час удосконалюються із розвитком комп’ютерних технологій. До таких методів та засобів належать паралельні обчислювальні системи, за допомогою яких проводиться моделювання складних динамічних систем (СДС) [1]. Такі системи як МДОЗП, шахтні вентиляційні системи (ШВМ), системи автоматизованого управління провітрюванням (САУП) тощо, начисляють до СДС, проєктування та розробка яких виконується із участі комп’ютерної техніки та їх подальшої підтримки під час моделювання. На кафедрі КІ ведуться розробки й дослідження паралельних методів і засобів моделювання СДС і зокрема МДОЗП [2-6]. Досвід показує, що для побудови ефективних паралельних симуляторів потрібні апаратно-програмні засоби топологічного аналізу графів МДОЗП, формування топологічних характеристик, генерування систем рівнянь та дискретних Simulation-моделей, розробці яких не приділялось до останнього часу достатньої уваги.

Метою даної кваліфікаційної роботи є розробка, реалізація та експериментальні дослідження засобів топологічного аналізу, що забезпечать активну комп’ютерну підтримку побудови паралельних симуляторів МДОЗП.

Новими рішеннями, запропонованими в роботі, є:

- Подальший розвиток методики побудови MIMD-симулятора МДОЗП, модель якого описується системою диференціально-алгебраїчних рівнянь;
- Визначення топологічних характеристик і алгоритми їх формування на

основі первинного кодування графа МДОЗП;

– Підхід до побудови паралельного симулятора МДОЗП.

Практичне значення результатів роботи:

– Запропоновані засоби можуть використовуватись в практиці проєктування систем автоматизації МДОЗП та в навчальному процесі кафедри КІ;

– Підхід до побудови паралельного симулятора є основою для формулювання технічних завдань на архітектурно релевантне паралельне програмування MIMD-процесів в рамках тематики ParSimTech-центру ДонНТУ і співробітництва кафедри КІ з надпродуктивним обчислювальним центром (HLRS) Штутгартського університету.

1 ПРОБЛЕМА ПАРАЛЕЛЬНОГО МОДЕЛЮВАННЯ МЕРЕЖЕВОГО ДИНАМІЧНОГО ОБ'ЄКТУ З ЗОСЕРЕДЖЕНИМИ ПАРАМЕТРАМИ (МДОЗП). МАТЕМАТИЧНА МОДЕЛЬ МДОЗП

1.1 Постановка задачі паралельного моделювання. Етапи розробки паралельного симулятора

Задача паралельного моделювання полягає в тому, щоб розробити паралельні методи та алгоритми для вирішувачів СДС-рівнянь. Такими рівняннями називають рівняння, які використовуються для опису поведінки СДС [1]. Зазвичай прикладом таких рівнянь є диференціальні рівняння.

СДС безпосередньо складаються з топологічного та математичного опису. Загалом це називається формальним описом математичної моделі СДС, на основі якої записується Simulation-модель, яка у свою чергу має форму, придатну для обчислення числовими методами. До таких методів належать методи Рунге-Кутта, Адамса Башфорта, Ейлера, різні блокові методи тощо.

Simulation-модель подається до симулятора, який проводить її моделювання, що називається також модельним експериментом. Підходи до побудови такого симулятора і будуть розглядатись в рамках даної роботи. При чому будуть освітлені підходи побудови саме паралельного симулятора. Це пов'язано з тим, що на сьогодні широко поширені напрямки розробок паралельного моделювання (Parallel Simulation Technology, Parallele Simulationstechnik), яке зокрема базується на GRID-технологіях [6].

Отже, можна сформувати деякі терміни відносно даного напрямку [2]:

- Математична модель – це формальний опис об'єкта досліджень, що містить топологічну частину та частину рівнянь;
- Симмодель (Simulation-Modell) – модель у формі, придатній до застосування засобів чисельного розв'язання рівнянь;
- Симулятор (Simulator) – апаратно-програмна реалізація симмоделі;

– Модельний експеримент або моделювання (Simulation) – дослідження СДС із застосуванням симуляторів.

Засоби моделювання поділяються на фізичні та математичні моделі. Останні у свою чергу поділяються на ті, які базуються на SISD-компоненті (в рамках даної роботи опускається), і ті, які базуються на MIMD-компоненті (є домінуючою в розподілених паралельних моделюючих середовищах (РПМС)).

До засобів моделювання, які використовують MIMD-компоненту, відносяться мова програмування C, Message Passing Interface (MPI), OpenMP та ін. паралельні мови моделювання.

Під час проєктування та розробки паралельних симуляторів потрібно дотримуватись основних вимог до методів та засобів моделювання:

– Врахування топологічних та математичних складових СДС;
– Дружність та запобігливість до користувача;
– Діалогова підтримка користувача на етапах моделювання СДС (мінімізація обсягів рутинної роботи на етапі первинного кодування графу; аналіз топологічних характеристик та автоматичне генерування матриць і параметрів, їх відображення тощо);
– Здатність моделі вирішувати задачі реального часу (ЗРЧ, Real Time, Echtzeit);
– Наявність засобів для побудови тренажерів (Trainingssimulatoren);
– Інтеграція з галузевими системами автоматизованого проектування (САПР);
– Об’єктно-орієнтованість реалізації моделюючого ПЗ.

Головною задачею паралельного моделювання є побудова РПМС, згідно із концепцією якого розвиваються MIMD-паралельне моделююче ПЗ для динамічних систем із зосередженими параметрами (ДСЗП), а також їх теорія та методика побудови [7, 8].

На рис. 1.1 представлено послідовність розробок моделювання від первісного структурного аналізу СДС та постановки задач досліджень і моделювання до приведення до форми симмоделі.

Рисунок 1.1 – Етапи побудови СДС-моделей та приведення до форми симмоделей

Як видно з рисунку, попри наявність різних СДС областей, існує лише декілька засобів подання топологій: за допомогою технологічних схем (ТС), структурних схем автоматизації (ССА), динамічних мережевих об’єктів (ДМО), графів, а також за допомогою комбінацій таких засобів [2].

До видів СДС окрім ДСЗП належать динамічні системи з розподіленими параметрами (ДСРП [9], в даній роботі такі СДС не будуть розглядатись), які отримуються за допомогою апроксимації ДСЗП.

В рамках цієї роботи СДСЗП буде описана за допомогою ДМО, що представлений у вигляді графу. Задача роботи полягає в тому, щоб провести автоматичне кодування графу, на основі якого зробити аналіз топологічних характеристик.

На сьогодні забезпечено комп'ютерну підтримку всіх етапів побудови симмоделей СДС з врахуванням специфіки представлених на рис. 1.1 трьох видів топологій.

Першим етапом розробки паралельного симулятора, використовуючи Simulation-модель, є аналіз підходів до розпаралелювання, при чому слід зауважити про мінімізацію гранулярності MIMD-процесів.

Структуровані MIMD-процеси з мінімальною гранулярністю, які вирішують симмодель, називається абстрактною алгоритмічною структурою, і вона ж є віртуальною паралельною моделлю першого рівня (ВПМ-1). Такі ВПМ-1 вирішують віртуальні (ВПС) та цільові (ЦПС) паралельні системи, які базуються на MIMD-принципі і мають паралельні процеси з локальною пам'яттю, об'єднані за допомогою віртуальної (ВКМ) та реальної (РКМ) комунікаційних мереж [6].

Для віртуальної ДМО-моделі будується модель із зосередженими/розподіленими параметрами, після чого визначається розподілення задачі по процесам. За допомогою обраного чисельного метода симмодель перетворюється на дискретну симмодель, яка вирішується паралельним алгоритмом.

Обчислення та розпаралелювання відбувається із врахуванням затрат на міжпроцесорний обмін та балансу завантаженості процесів. Отримані результати візуалізуються, використовуючи інтерфейс користувача.

На рис. 1.2 показано етапи розробок, що необхідні для побудови паралельних симуляторів, а також їх подальше застосування [2, 6].

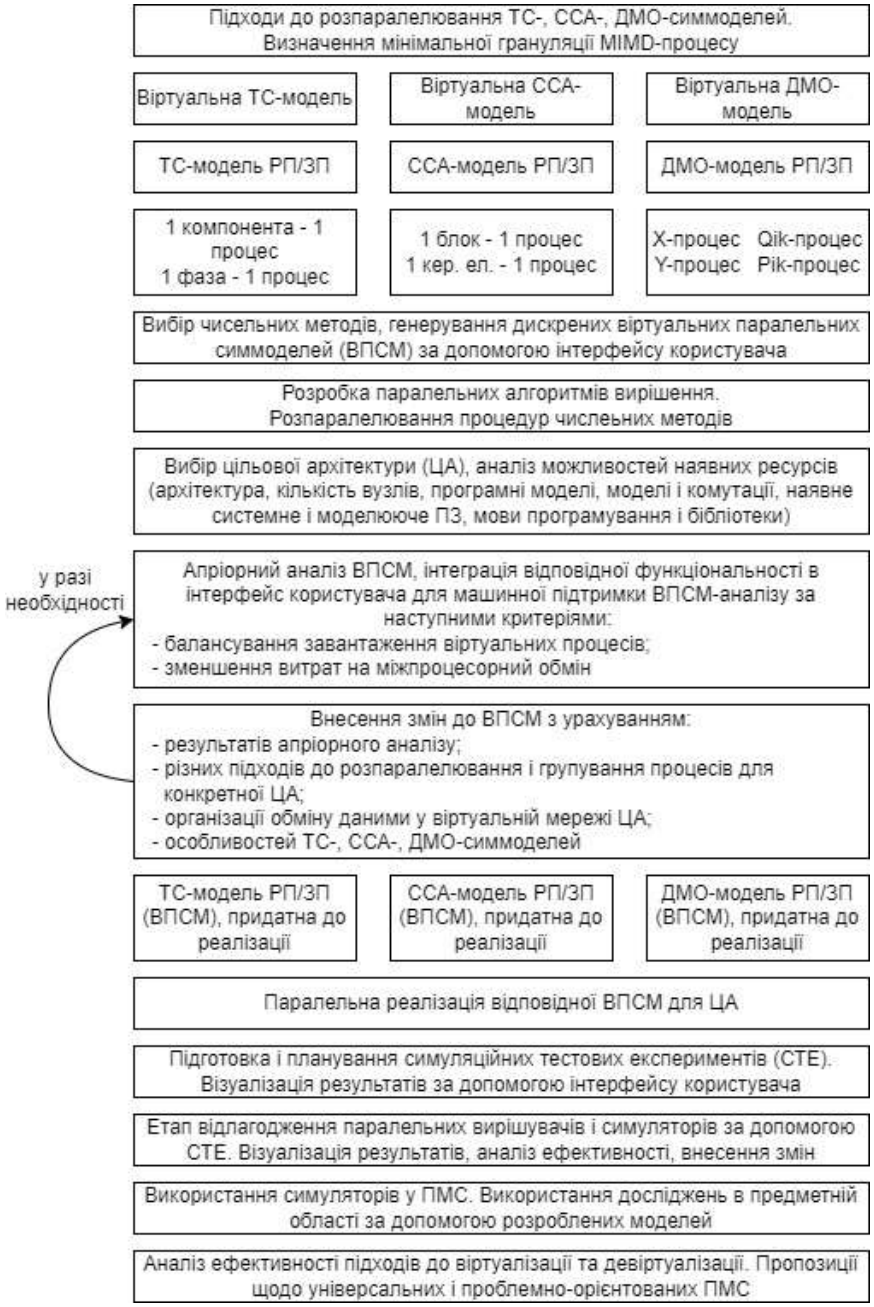


Рисунок 1.2 – Етапи розробки та застосування паралельних симуляторів

На передостанніх етапах проводиться девіртуалізація ВПМ, що означає таке перетворення, яке дало б можливість вирішення задачі на декількох різних ЦПС (або одній). Для цього потрібно визначити можливі рівні розпаралелювання задачі згідно із наявними ресурсами ЦПС. В результаті будуть отримані можливі співвідношення між процесами паралельної

програми і наявними ЦПС, розгляд та аналіз отриманих співвідношень, вибір варіантів імплементації [2, 6].

1.2 Засоби комп’ютерної підтримки розробки паралельного симулятора МДОЗП

РПМС називається таке середовище, яке містить в собі всі засоби, потрібні для функціонування паралельних апаратних засобів, системного та моделюючого ПЗ, виконуючи при цьому сформульовані в п.п. 1.1 вимоги. РПМС також дає можливість для реалізації та застосування паралельних симуляторів для СДСЗП, СДСРП (рис. 1.3).

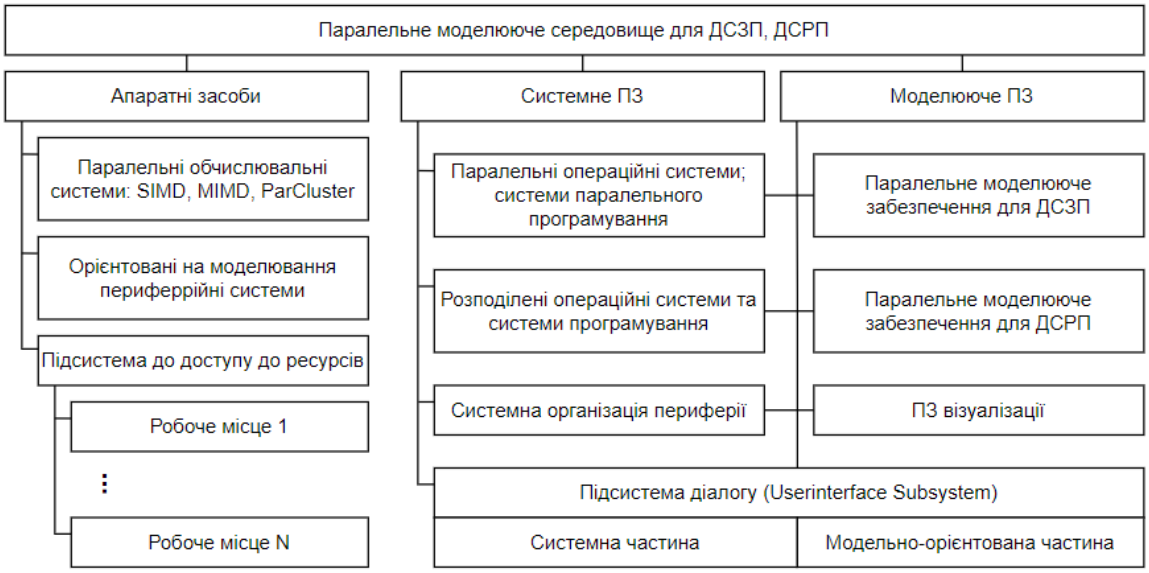


Рисунок 1.3 – Концептуальна структура РПМС

До апаратних засобів належать паралельні обчислювальні системи, орієнтовані на моделювання периферійні системи, підсистема до доступу до ресурсів, яка у свою чергу складається з N робочих місць.

До системного ПЗ належать паралельні операційні системи і системи паралельного програмування, розділені операційні системи та системи програмування, системна організація периферії.

До моделюючого ПЗ належить паралельне моделююче забезпечення для ДСЗП/ДСРП і ПЗ візуалізації.

Підсистема діалогу складається з системної та модельно-орієнтованої частин.

РПМС-концепція полягає в тому, щоб надати необхідну функціональність, за допомогою якої було б можливо розробити паралельні методи та алгоритми для моделюючого ПЗ (Modeling and Simulation Software), яке працює з такими СДС як ДСЗП та ДСРП [9].

1.3 Задачі розробок і досліджень

Для моделювання та дослідження МДОЗП було поставлено наступні задачі.

Задача топологічного аналізу. Спочатку треба виконати детальний аналіз топологічних характеристик мережевого об’єкта, а саме: представити його у вигляді графа, зробити початкове кодування; побудувати дерево та антидерево графа, матриці інциденцій і незалежних контурів; описати вектори потоків і діагональні матриці параметрів.

Задача генерування топологічних характеристик. Ця задача полягає в тому, щоб виконати рутинну ручну роботу користувача (розробника) в автоматичному режимі. Тобто, зробити все те, що потребує задача топологічного аналізу, але за допомогою програми.

Задача програмної імплементації та експериментального дослідження. Ця задача характеризується аналізом програмної реалізації алгоритмів, які будуть результатом виконання попередньої задачі. На цьому етапі буде проведено тестування та аналіз отриманих результатів.

2 ТОПОЛОГІЧНІ ХАРАКТЕРИСТИКИ МДОЗП

2.1 Граф МДОЗП та його початкове кодування, рівняння вузлів і гілок

Для виконання різноманітних дослідницьких та проектних завдань моделі із зосередженими параметрами забезпечують необхідну точність [3]. Мережевий об’єкт (схема розподілу повітряного потоку [10]) можна записати у вигляді графу $G(U,Q)$ з $n = |U|$ вузлами і $m = |Q|$ гілками. Граф тестового мережевого об’єкту з $n = 5$ вузлами та $m = 8$ гілками зображено на рис. 2.1.

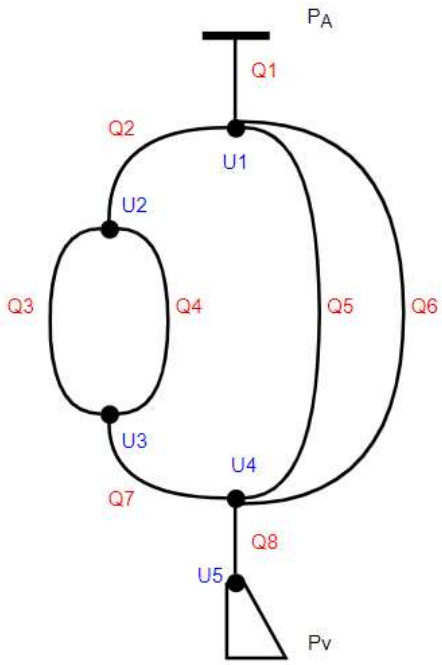


Рисунок 2.1 – Граф МДОЗП

($m =$ 8 гілок Q_1 … Q_8, $n = 5$ вузлів U_1 … U_5)

Кодування графу відбувається за допомогою наступних умовних позначок:

$$Q_j,\ AK_i,\ EK_i, \tag{2.1}$$

де Q_j – гілка графу, $j \in (1, \dots, m)$,

AK_i – початковий вузол гілки, $i \in (1, \dots, n)$,

EK_i – кінцевий вузол гілки, $i \in (1, \dots, n)$.

За допомогою позначок (2.1) можна описати матриці інциденцій A (2.2) та контурів S (2.3) [11-13], які генеруються за допомогою алгоритмів F_A та F_S відповідно.

$$A = F_A(AK_i,\ EK_i,\ Q_j) \tag{2.2}$$

$$S = F_S(AK_i,\ EK_i,\ Q_j) \tag{2.3}$$

Тестовий граф кодується наступною таблицею відносно кожної його гілки (табл. 2.1).

Таблиця 2.1 – Кодування графу МДОЗП

AK_i	EK_i	Q_j	Параметри			
			X_j/Y_j	K_j	R_j	H_j
U_5	U_1	Q_1	X_1	153	0.387	0
U_1	U_2	Q_2	X_2	29	0.0078	0
U_2	U_3	Q_3	Y_1	232.5	1.5	0
U_2	U_3	Q_4	Y_2	273	2.68	0
U_1	U_4	Q_5	Y_3	430	1.93	0
U_1	U_4	Q_6	Y_4	221	1.28	0
U_3	U_4	Q_7	X_3	29	0.42	0
U_4	U_5	Q_8	X_4	125	0.314	4100

В табл. 2.1 позначка X_j/Y_j використовується для опису гілок дерева на антидерева (п.п. 2.2).

Модель МДОЗП складається з $n-1$ рівнянь для вузлів (2.4) та $\gamma = m - (n - 1)$ рівнянь для гілок (2.5) [3].

Кожен i-вузол описується за першим законом Кірхгофа [14-16]:

$$U_i : \sum Q_{in} = \sum Q_{out}, \tag{2.4}$$

де Q_{in} – вхідний потік повітря в вузол U_i,

Q_{out} – вихідний потік повітря з вузла U_i.

Кожна гілка графу Q_j описується наступним диференціальним рівнянням перехідного процесу [3, 17]:

$$K_j \cdot \frac{dQ_j}{dt} + R_j \cdot Q_j|Q_j| = H_j\ , \tag{2.5}$$

де $H_j = P_{AKi} - P_{EKi}$ – різниця між тиском в початковому й кінцевому вузлах гілки, $\left[\frac{\text{Н}}{\text{м}^2}\right]$,

$K_j = \frac{\rho l}{F}$ – коефіцієнт інерційності потоку (ρ – щільність повітря, l – довжина гілки, F – площа поперечного перерізу гілки (повітропроводу)),

R_j – аеродинамічний опір,

Q_j – потік повітря, $\left[\frac{\text{м}^3}{\text{сек}}\right]$.

Граф МДОЗП характеризується наступними векторами та матрицями:

– Вектор потоків повітря:

$$Q = (Q_1\ Q_2 \dots Q_m)^T \tag{2.6}$$

– Вектор тисків вентиляторів, що можуть знаходитись в кожній гілці:

$$H = (H_1\ H_2\ \dots H_m)^T \tag{2.7}$$

Так, в МДОЗП на рис. 2.1 вектор $H = (0\ 0\ \dots H_8)^T$.

– Діагональна матриця параметрів K:

$$K = \begin{pmatrix} K_1 & 0 & 0 \\ 0 & \dots & 0 \\ 0 & 0 & K_m \end{pmatrix} \tag{2.8}$$

– Діагональна матриця параметрів R:

$$R = \begin{pmatrix} R_1 & 0 & 0 \\ 0 & \dots & 0 \\ 0 & 0 & R_m \end{pmatrix} \tag{2.9}$$

Формули (2.6 - 2.9) є базовими і будуть використовуватись в подальшому структуруванні відносно дерева й антидерева.

2.2 Дерево й антидерево графа, матриці інциденцій і незалежних контурів

Дерево – це частина графу, що не містить замкнутих контурів. Антидерево – сукупність гілок, що є базами незалежних контурів [11-12, 18].

Граф МДОЗП (рис. 2.1) треба представити на основі дерева та антидерева, для чого потрібно виділити їх з графу. Такий граф має вектор потоків в гілках дерева (гілки $X_1, X_2, \dots, X_{n-1}$) та антидерева (гілки $Y_1, Y_2, \dots, Y_\gamma$).

На рис. 2.2 представлено граф, в якому показано один з можливих варіантів дерева та антидерева.

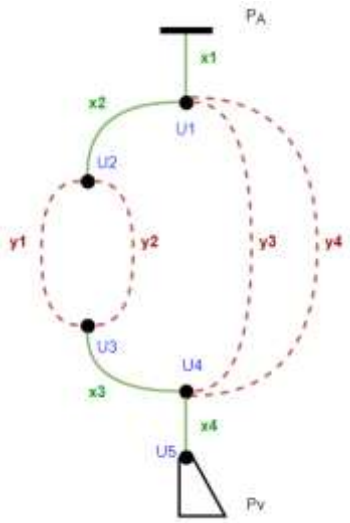


Рисунок 2.2 – Дерево та антидерево графу МДОЗП

Вектори та матриці (2.2 - 2.3), (2.6 - 2.9) можна структурувати відносно гілок дерева й антидерева (2.10 - 2.15).

Вектор потоків:

$$Q = (X, Y)^T = \left(X_1 X_2 \dots X_{n-1} Y_1 Y_2 \dots Y_\gamma\right)^T, \quad (2.10)$$

де $X = (X_1 X_2 \dots X_{n-1})^T$ – вектор потоків в гілках дерева,

$Y = \left(Y_1 Y_2 \dots Y_\gamma\right)^T$ – вектор потоків в гілках антидерева.

Матриці інциденцій A (2.11) та незалежних контурів S (2.12):

$$A = (A_X\ A_Y) \quad (2.11)$$

$$S = (S_X\ S_Y) \quad (2.12)$$

Вектор тисків структурується так:

$$H = (H_X\ H_Y)^T = \left(H_{X_1}\ H_{X_2} \dots H_{X_{n-1}} H_{Y_1}\ H_{Y_2} \dots H_{Y_\gamma}\right)^T \quad (2.13)$$

Діагональна матриця параметрів K:

$$K = \begin{pmatrix} K_X & 0 \\ 0 & K_Y \end{pmatrix} \quad (2.14)$$

Діагональна матриця параметрів R:

$$R = \begin{pmatrix} R_X & 0 \\ 0 & R_Y \end{pmatrix} \quad (2.15)$$

Матриці інцидентності A та незалежних контурів S для тестового графу МДОЗП (рис. 2.1), структуровані відносно дерева та антидерева графа (рис. 2.2), зображено у табл. 2.2 та табл. 2.3 відповідно.

Таблиця 2.2 – Матриця інцидентності A, структурована відносно дерева та антидерева

Гілки / Вузли	X_1	X_2	X_3	X_4	Y_1	Y_2	Y_3	Y_4
U_1	1	-1	0	0	0	0	-1	-1
U_2	0	1	0	0	-1	-1	0	0
U_3	0	0	-1	0	1	1	0	0
U_4	0	0	1	-1	0	0	1	1

Таблиця 2.3 – Матриця контурів S, структурована відносно дерева та антидерева

Гілки / Контури	X_1	X_2	X_3	X_4	Y_1	Y_2	Y_3	Y_4
K_1	1	1	1	1	1	0	0	0
K_2	1	1	1	1	0	1	0	0
K_3	1	0	0	1	0	0	1	0

K_4	1	0	0	1	0	0	0	1

2.3 Модель та Simulation-модель МДОЗП у векторно-матричній формі [3]

Згідно з табл. 2.1 та формулами (2.14) та (2.15) можна сформувати діагональні матриці параметрів K та R з діагоналями:

$$R = (0.387, 0.0078, 0.42, 0.314, 1.5, 2.68, 1.93, 1.28)$$
$$K = (153, 29, 29, 125, 232.5, 273, 430, 221)$$

У векторно-матричній формі модель МДОЗП представляється у вигляді системи рівнянь [2]

$$\begin{cases} A \cdot \bar{Q} = 0, \\ SK\frac{dQ}{dt} + SR \cdot Q|Q| = SH' \end{cases} \quad (2.16)$$

де перше рівняння системи – рівняння вузлів, а друге – рівняння контурів.

Рівняння вузлів.

Першим кроком формування рівняння вузлів є підставлення в це рівняння системи (2.16), визначення (2.11) та (2.10):

$$(A_X\ A_Y) \cdot \begin{pmatrix} X \\ Y \end{pmatrix} = A_X \cdot X + A_Y \cdot Y = 0$$

Другим кроком вважається виведення вектору X. Для цього треба $A_Y \cdot Y$ перенести на іншу сторону. Щоб A_X прибрати від X, рівняння потрібно помножити зліва на A_X^{-1}:

$$A_X \cdot X = -A_Y \cdot Y$$

$A_X^{-1} \cdot A_X \cdot X = -A_X^{-1} \cdot A_Y \cdot Y$, так як $A_X^{-1} \cdot A_X = E$, то:

$E \cdot X = -A_X^{-1} \cdot A_Y \cdot Y$, так як $E \cdot A = A$, то:

$$X = -A_X^{-1} \cdot A_Y \cdot Y.$$

Було введено допоміжну матрицю $W = A_X^{-1} \cdot A_Y$, що визначає зв'язки між потоками в гілках дерева й антидерева. Таким чином, було отримано остаточне рівняння вузлів:

$$X = -W \cdot Y \tag{2.17}$$

Після диференціювання рівняння (2.17) було отриманий наступний вираз:

$$\frac{dX}{dt} = -W \cdot \frac{dY}{dt} \tag{2.18}$$

Рівняння контурів.

В рівняння контурів було підставлені системи (2.16) і визначення з (2.10) та (2.14), а $Q_j|Q_j|$ замінено на Z_j, що є елементами вектору Z $(j = 1, \dots, m)$.

$$S_X \cdot K_X \frac{dX}{dt} + S_Y \cdot K_Y \frac{dY}{dt} = SH - SRZ \tag{2.19}$$

Тепер в рівняння (2.19) треба підставити (2.17):

$$S_X K_X \left(-W \cdot \frac{dY}{dt}\right) + S_Y K_Y \frac{dY}{dt} = SH - SRZ$$

Після винесення за дужки $\frac{dY}{dt}$ з врахуванням правил виконання векторно-матричних операцій [19], було отримано:

$$(S_X K_X(-W) + S_Y K_Y) \cdot \frac{dY}{dt} = SH - SRZ$$

$$(S_Y K_Y - S_X K_X W) \cdot \frac{dY}{dt} = SH - SRZ \quad (2.20)$$

Рівняння (2.20) було помножене зліва на обернену матрицю $(S_Y K_Y - S_X K_X W)^{-1}$ [19]:

$$\frac{dY}{dt} = (S_Y K_Y - S_X K_X W)^{-1} \cdot (SH - SRZ)$$

Нехай $V = (S_Y K_Y - S_X K_X W)^{-1}$, тоді:

$$\frac{dY}{dt} = V \cdot (SH - SRZ)$$

$$\frac{dY}{dt} = V \cdot SH - V \cdot SR \cdot Z \quad (2.21)$$

Нехай $Hu = V \cdot SH$ та $Ru = V \cdot SR$, тоді після підстановок в (2.21) остаточне рівняння контурів буде виглядати так:

$$\frac{dY}{dt} = Hu - Ru \cdot Z \quad (2.22)$$

Отже, Simulation-модель МДОЗП складається з рівнянь (2.17) та (2.22):

$$\begin{cases} X = -W \cdot Y \\ \frac{dY}{dt} = Hu - Ru \cdot Z \end{cases} \quad (2.23)$$

Система (2.23) буде використовуватись для побудови дискретної Simulation-моделі.

2.4 Дискретна Simulation-модель МДОЗП і блок-схема MIMD-симулятора

Якщо обрати певний обчислювальний метод [20], можна отримати з (2.23) дискретну Simulation-модель. Так, за методом Адамса-Башфорта [20-21] вона має вигляд:

$$\begin{cases} Y(i) = \begin{cases} Y(i-1) + hV_1(i-1) & \text{для } i = 1 \\ Y(i-1) + h_1V_1(i-1) - h_2V_1(i-2) & \text{для } i > 1 \end{cases}, \\ X(i) = -WY(i) \end{cases} \quad (2.24)$$

де $V_1 = Hu - RuZ$.

В системі (2.24) використовуються такі параметри як h, h_1 та h_2. В симуляторі вони будуть мати наступні значення: h – заданий автором крок, $h_1 = \frac{3}{2} \cdot h$ та $h_2 = \frac{1}{2} \cdot h$. Параметр $V_1(i)$ розраховується наступним чином: $V_1(i) = TP \cdot H(i) - Ru \cdot Z(i)$, а TP та Ru є матрицями-константами, які передаються до симулятора. H та Z є векторами. В результаті операцій множення матриці на вектор, буде отримано вектор.

Для паралельного розв’язання рівнянь системи МДОЗП згідно з (2.23) та (2.24) було використано наступний підхід: кожен MIMD-процес відповідає за попарне обчислення компонентів векторів X, Y, тому симулятор буде складатися з $\gamma = m - (n-1)$ процесів. Перевагою такого підходу є те, що процеси більш рівномірно завантажені (рис. 2.3).

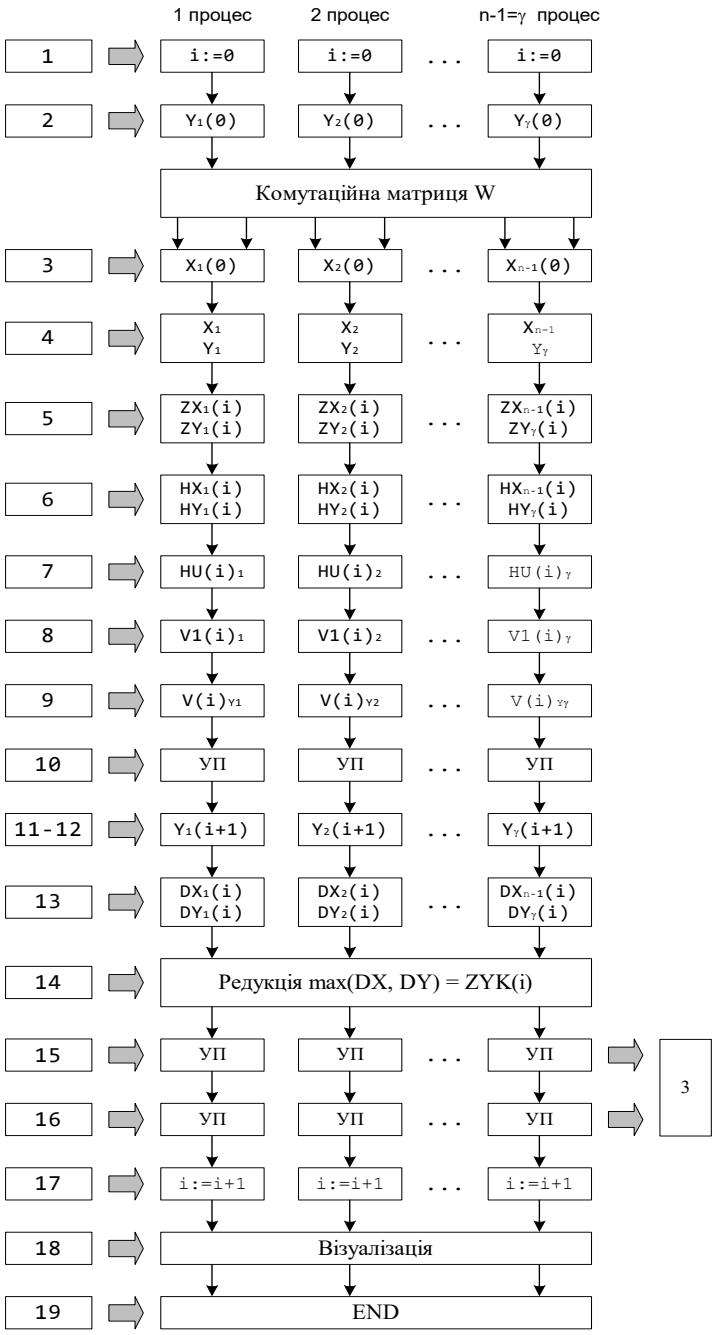


Рисунок 2.3 – Блок-схема паралельної програми MIMD-симулятора

Отже, MIMD-вирішувач повинен складатись з $(X\,Y)_i$-процесорів ($i = 1 \ldots m - (n-1)$), кожен з яких буде обчислювати складові векторів X та Y. Так, для тестового прикладу з $m = 8$ гілками та $n = 5$ вузлами MIMD-вирішувач складається з чотирьох процесів, кожен i-процес обчислює X_i та Y_i.

2.5 Висновки

- Топологічними характеристиками МДОЗП є граф, його таблиця кодування, дерево та антидерево, матриці інциденцій і контурів, вектори потоків, тисків та виразів $Q_j|Q_j|$, а також діагональні матриці параметрів гілок;
- Для тестового МДОЗП побудовано дерево й антидерево, проведено структуризацію топологічних характеристик відносно потоків в гілках дерева та антидерева;
- Топологічні характеристики входять до складу системи диференціально-алгебраїчних рівнянь моделі МДОЗП;
- З огляду на значну розмірність цих характеристик об’єктів реальної промислової складності постає задача комп’ютерної підтримки їх формування та формальних перетворень в процесі побудови паралельного симулятора.

3 АЛГОРИТМИ ГЕНЕРУВАННЯ ТОПОЛОГІЧНИХ ХАРАКТЕРИСТИК

3.1 Діалогова підтримка побудови таблиці кодування графа МДОЗП

Побудова таблиці кодування напряму залежить від того, як було введено граф. Зчитування графу відбувається за допомогою файлу. Тобто, користувач підготовляє файл певної структури, вказує шлях до цього файлу, а програма зчитує дані та перевіряє кожен рядок на відповідність структури. Після чого правильно описані рядки файлу відправляються в програму та проводиться подальша обробка цього графу та генерування первинної таблиці кодування графу.

Кожний рядок файлу повинен мати структуру, що зображена на рис. 3.1.

Рисунок 3.1 – Структура рядку файлу з графом

Кількість елементів в рядку повинна бути рівною шести. Елементи розділяються пробілами.

Перші три елементи повинні бути цілими числами, а решта – числами з плаваючою комою (в файлі запис відбувається із використанням точки, а не коми).

Перший елемент рядку – ідентифікатор гілки (QId). Він потрібен для виведення позначення гілки на екран, для зручного розуміння того, що це за гілка. На рисунку вище цей ідентифікатор дорівнює 0, а це значить, що на екрані вона буде мати позначку Q_0.

Другий та третій елементи рядку – це ідентифікатори початкового (UId_s) та кінцевого (UId_e) вузлів гілки відповідно. В програмі, за

прикладом на рисунку, вузли будуть відображені як: U_1 та U_0 відповідно. Вузли повинні бути різні.

Решта елементів – це параметри гілки: коефіцієнт інерційності потоку (K), аеродинамічний опір (R) [22] та різниця тиску між початковим та кінцевим вузлами відповідно (H).

У випадку, якщо структура рядку файлу не задовольняє умовам, тоді цей рядок не враховується і не додається до графу.

Після завантаження графу до програми починається його перевірка на задовільнення умові, яка звучить так: граф повинен мати для кожного вузла хоча б одну вхідну та вихідну гілки. Якщо вхідний граф не задовольняє таку умову, тоді граф не розглядається. Присутність умови виправдана тим, що якщо є хоча б одна гілка, у якої відсутня хоча б одна інцидентна (вхідна або вихідна) гілка, тоді граф не є замкнутий і контуру через цю гілку не може бути.

Якщо ж граф задовольняє умові, тоді починається його подальша обробка та аналіз його топологічних характеристик.

На першому етапі формується таблиця кодування графу. Вона містить структуру, що зображена на рис. 3.2.

QId	*UId_s*	*UId_e*	*XYId*	*K*	*R*	*H*
Q0	U1	U0	X0	153	0.387	0
...	...	...	...	...	...	...

Рисунок 3.2 – Структура таблиці кодування графу

Стовпець з ідентифікатором дерева/антидерева ($XYId$) поки опускається, бо це є наступний алгоритм – алгоритм генерування дерева/антидерева, який буде описано в наступному підрозділі.

Алгоритм генерування таблиці кодування полягає в тому, щоб привести введену в файлі інформацію про граф в структуру, зрозумілу для

користувача. Для випадків, коли в файлі були введені гілки не за зростанням її ідентифікатора, було реалізовано його сортування, для кращого сприйняття інформації.

Таким чином у результаті виконання цього алгоритму буде отримана таблиця кодування графу.

3.2 Алгоритми генерування дерева й антидерева

Щоб реалізувати алгоритм генерування дерева й антидерева, потрібно зрозуміти, що таке орієнтоване дерево.

Орієнтоване дерево – це таке дерево, в якому тільки один вузол не має вхідних гілок (корінь), а решта вузлів – мають тільки одну вхідну гілку (листя дерева) [11-12, 18].

Алгоритм полягає в тому, щоб з вхідного графа зробити дерево, позначаючи при цьому гілки, які не мають бути в графі (антидерево), щоб граф став деревом.

Опис алгоритму. Для першого вузла графу треба позначити всі його вхідні гілки за антидерево. Цей вузол буде коренем дерева. Для решти вузлів потрібно спочатку визначити, чи є вже вхідна гілка, позначена за дерево. Якщо є, тоді решта вхідних гілок позначається за антидерево. Якщо немає – тоді одна із вхідних гілок позначається за дерево, а останні – за антидерево. Блок-схема алгоритму розміщена в додатку Б.1.

Приклад виконання алгоритму вручну. У якості прикладу буде використано тестовий граф, який було задано на рис. 2.1. Він був представлений по-іншому (рис. 3.3).

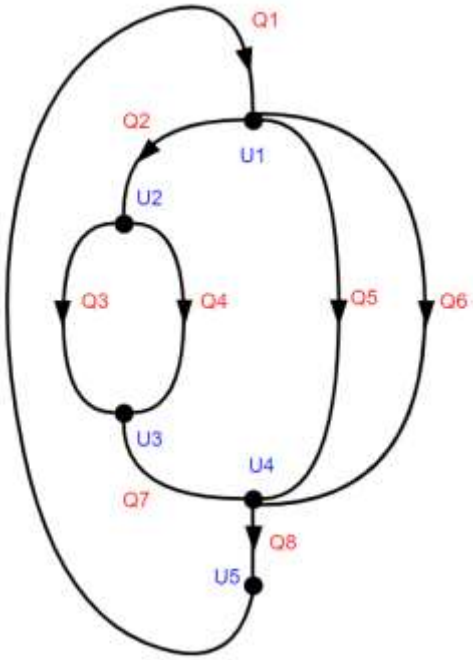


Рисунок 3.3 – Вхідний граф для прикладу ручного виконання алгоритму генерування дерева та антидерева

Крок 1.

Для першого вузла (U_1) всі вхідні гілки (Q_1) позначаються за антидерева ($Q_1 \rightarrow Y_1$) (рис. 3.4).

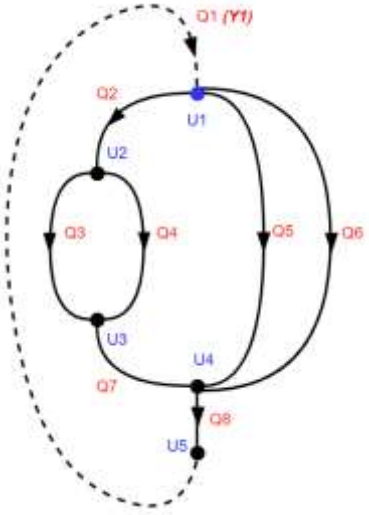


Рисунок 3.4 – Крок 1: вузол U_1

Крок 2.

Вузол U_2 має одну вхідну гілку (Q_2), яка позначається за дерево ($Q_2 \to X_1$) (рис. 3.5).

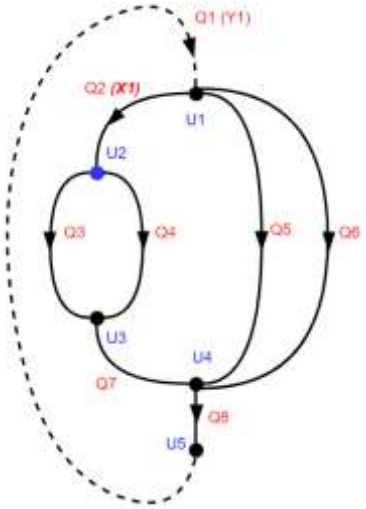


Рисунок 3.5 – Крок 2: вузол U_2

Крок 3.

Вузол U_3 має дві вхідні гілки (Q_3, Q_4), одна з них (наприклад, перша) позначається за дерево, а друга – за антидерево ($Q_3 \to X_2, Q_4 \to Y_2$) (рис. 3.6).

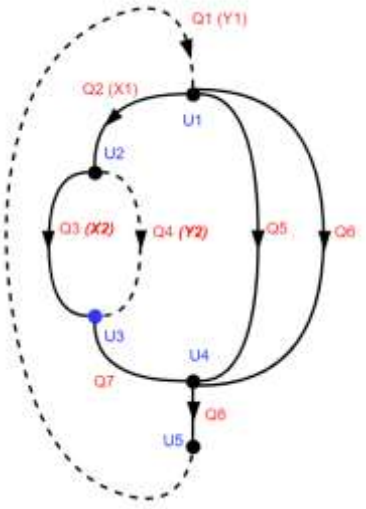


Рисунок 3.6 – Крок 3: вузол U_3

Крок 4.

Вузол U_4 має три вхідні гілки (Q_7, Q_5, Q_6), одна з них (наприклад, перша) позначається за дерево, а решта – за антидерево $(Q_7 \to X_3, Q_5 \to Y_3, Q_6 \to Y_3)$ (рис. 3.7).

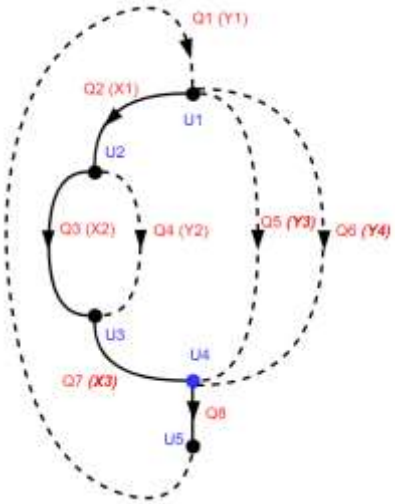


Рисунок 3.7 – Крок 4: вузол U_4

Крок 5.

Вузол U_5 має одну вхідну гілку (Q_8), яка позначається за дерево $(Q_8 \to X_4)$ (рис. 3.8).

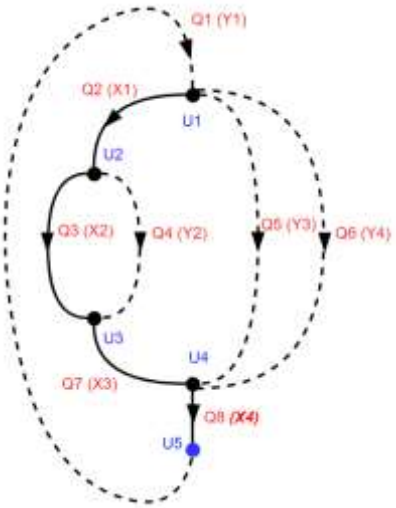


Рисунок 3.8 – Крок 5: вузол U_5

Таким чином в результаті кожна гілка графу позначається деревом або антидеревом, що дає можливість надалі структурувати топологічні характеристики (матриці інциденцій, незалежних контурів, параметрів та вектори) відносно дерева та антидерева.

Такий граф не має замкнутих контурів.

3.3 Алгоритм генерування матриці інциденцій

Матриця інциденцій – це матриця, яка описує взаємозв'язок між гілками та вузлами графа. Вона має структуру, що зображена на рис. 3.9, де n – кількість гілок дерева, m – кількість гілок антидерева, k – кількість вузлів в графі.

Рисунок 3.9 – Матриця інциденцій

Опис алгоритму. Кожний вузол графу має вхідні та вихідні гілки. Рядок матриці інциденцій відповідає вузлу. Для гілки, яка входить в поточний вузол, що розглядається, у відповідній комірці таблиці записується 1. Для гілки, яка виходить з поточного вузла – записується -1. У тому випадку, якщо гілка ніяк не пов'язана з вузлом, в комірці таблиці це позначається нулем. Блок-схема алгоритму розміщена в додатку Б.2.

Приклад виконання алгоритму вручну. У якості прикладу буде використано тестовий граф, який було задано на рис. 2.1. Він був структурований відносно дерева та антидерева (рис. 3.10).

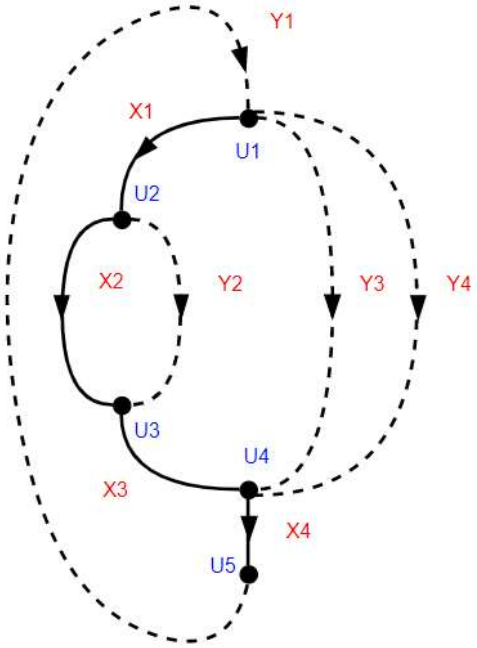
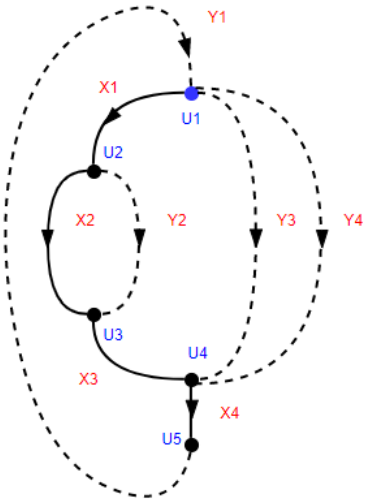


Рисунок 3.10 – Вхідний граф, структурований відносно дерева та антидерева

Крок 1.

Проводиться перегляд вхідних та вихідних гілок для першого вузла (U_1). Вхідна гілка Y_1 в таблиці позначається одиницею, вихідні гілки X_1, Y_3, Y_4 позначаються -1. Решта гілок в графі позначаються нулем (рис. 3.11).

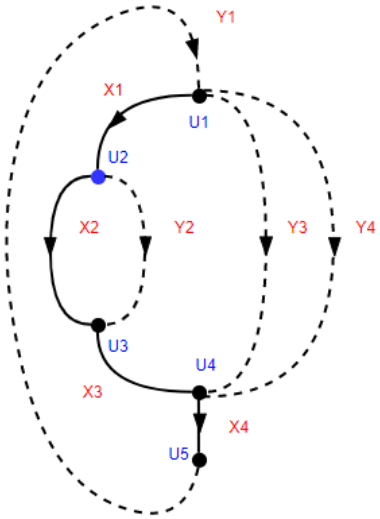


	X1	X2	X3	X4	Y1	Y2	Y3	Y4
U1	-1	0	0	0	1	0	-1	-1
U2	...	...	...	...	...	...	...	...
U3	...	...	...	...	...	...	...	...
U4	...	...	...	...	...	...	...	...
U5	...	...	...	...	...	...	...	...

Рисунок 3.11 – Крок 1: вузол U_1

Крок 2.

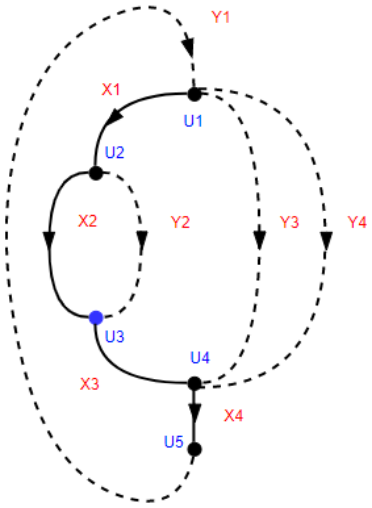
Вузол U_2 має вхідну гілку X_1 (позначається одиницею), вихідні гілки X_2, Y_2 (позначаються -1), решта гілок – позначаються нулем (рис. 3.12).



	X1	X2	X3	X4	Y1	Y2	Y3	Y4
U1	-1	0	0	0	1	0	-1	-1
U2	1	-1	0	0	0	-1	0	0
U3	...	...	...	...	...	...	...	...
U4	...	...	...	...	...	...	...	...
U5	...	...	...	...	...	...	...	...

Рисунок 3.12 – Крок 2: вузол U_2

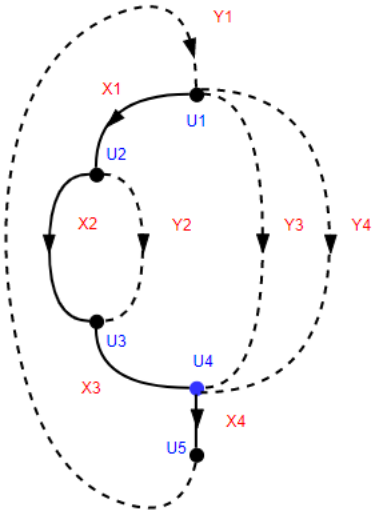
Крок 3. Вузол U_3 має вхідні гілки X_2, Y_2 (позначаються одиницею), вихідну гілку X_3 (позначається -1), решта гілок – позначаються нулем (рис. 3.13).



	X1	X2	X3	X4	Y1	Y2	Y3	Y4
U1	-1	0	0	0	1	0	-1	-1
U2	1	-1	0	0	0	-1	0	0
U3	0	1	-1	0	0	1	0	0
U4	...	...	...	...	...	...	...	...
U5	...	...	...	...	...	...	...	...

Рисунок 3.13 – Крок 3: вузол U_3

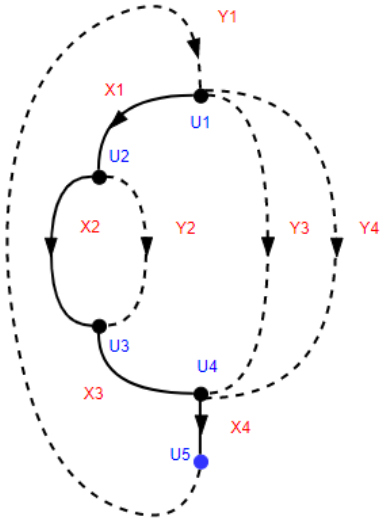
Крок 4. Вузол U_4 має вхідні гілки X_3, Y_3, Y_4 (позначаються одиницею), вихідну гілку X_4 (позначається -1), решта гілок – позначаються нулем (рис. 3.14).



	X1	X2	X3	X4	Y1	Y2	Y3	Y4
U1	-1	0	0	0	1	0	-1	-1
U2	1	-1	0	0	0	-1	0	0
U3	0	1	-1	0	0	1	0	0
U4	0	0	1	-1	0	0	1	1
U5	...	...	...	...	...	...	...	...

Рисунок 3.14 – Крок 4: вузол U_4

Крок 5. Вузол U_5 має вхідну гілку X_4 (позначаються одиницею), вихідну гілку Y_1 (позначається -1), решта гілок – позначаються нулем (рис. 3.15).



	X1	X2	X3	X4	Y1	Y2	Y3	Y4
U1	-1	0	0	0	1	0	-1	-1
U2	1	-1	0	0	0	-1	0	0
U3	0	1	-1	0	0	1	0	0
U4	0	0	1	-1	0	0	1	1
U5	0	0	0	1	-1	0	0	0

Рисунок 3.15 – Крок 5: вузол U_5

Для перевірки отриманої матриці інцидентності можна порахувати суму всіх значень в кожному стовпці. Ця сума повинна дорівнювати нулю. Як видно з рис. 3.15, матриця інциденцій сформована правильно.

3.4 Алгоритм генерування матриці незалежних контурів

Матриця незалежних контурів – це матриця, яка описує взаємозв'язок між контурами та гілками графу, яка має структуру, що зображена на рис. 3.16, де n – кількість гілок дерева, m – кількість гілок антидерева, j – кількість контурів в графі.

Рисунок 3.16 – Структура матриці незалежних контурів

Контур – це замкнутий шлях графу.

Кожний контур складається з гілок. Рядок матриці контурів відповідає визначеному контуру. Для гілки, яка входить в поточний контур, що розглядається, у відповідній комірці таблиці записується 1. Для гілки, яка не входить до контуру – записується 0.

Але перед цим потрібно знайти самі контури графу.

Опис алгоритму. Виконання алгоритму починається з того, що береться перший вузол і визначається скільки вихідних гілок є з цього вузла. Стільки створюється незалежних контурів, в кожен з яких додається початкова гілка. Потім розглядається кожен з цих контурів. Якщо контур має

надалі один шлях, тобто з вузла є тільки одна вихідна гілка, тоді в цей контур гілку треба записати та переходити далі, в іншому випадку – треба скопіювати поточний контур на один менше ніж кількість вихідних гілок. Контур вважається знайденим, якщо початковий та кінцевий вузли контуру співпадають (шлях замкнувся). Блок-схема алгоритму розміщена в додатку Б.3.

Приклад виконання алгоритму вручну. У якості прикладу буде використано тестовий граф, який було структуровано відносно дерева та антидерева (рис. 3.10).

Крок 1.

Спочатку береться перший вузол графу –U_1. З цього вузла є три вихідні гілки: X_1, Y_3, Y_4. Тоді на першому етапі створюються три контури (масиви) із початковими елементами X_1, Y_3, Y_4 відповідно (рис. 3.17).

{ X1, ... }

{ Y3, ... }

{ Y4, ... }

Рисунок 3.17 – Схематичне зображення контурів на першому кроці

Крок 2.

Знаходження першого контуру. З гілки X_1 можна проходити шлях двома способами: через гілку X_2 або через Y_2. Тому, поточний контур, що розглядається, копіюється один раз. В першому випадку до поточного контуру додається гілка X_2, а в другому – Y_2, будуть отримані контури, які зображено на рис. 3.18.

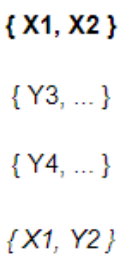


Рисунок 3.18 – Схематичне зображення контурів на другому кроці

Крок 3.

Продовження аналізу першого контуру. Після гілки X_2 є лише один шлях – через гілку X_3, а з неї – через гілку X_4 та Y_1, після чого шлях замикається на вузлі U_1, з якого починався пошук даного контуру (рис. 3.19).

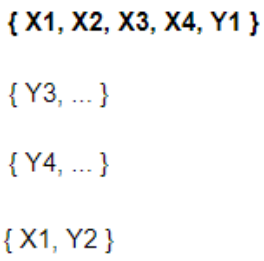


Рисунок 3.19 – Схематичне зображення контурів на третьому кроці

Крок 4.

Аналізується наступний контур у списку (рис. 3.19). Контур має лише один спосіб, починаючи з гілки Y_3: $Y_3 \rightarrow X_4 \rightarrow Y_1$ (рис. 3.20).

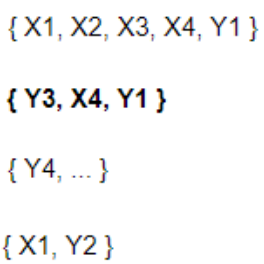


Рисунок 3.20 – Схематичне зображення контурів на четвертому кроці

На кроках 5-6 аналогічним чином знаходяться решта контурів. Отримано список контурів, що зображено на рис. 3.21. Їх кількість рівна кількості гілок антидерева. Кожна гілка антидерева є базою для контура.

K1 = { X1, X2, X3, X4, Y1 }

K2 = { Y3, X4, Y1 }

K3 = { Y4, X4, Y1 }

K4 = { X1, Y2, X3, X4, Y1 }

Рисунок 3.21 – Список всіх контурів графа

Тепер можна заповнити таблицю на рис. 3.16 із врахуванням знайдених контурів. В результаті таблиця буде виглядати так, як на рис. 3.22.

	X1	***X2***	***X3***	***X4***	***Y1***	***Y2***	***Y3***	***Y4***
K1	1	1	1	1	1	0	0	0
K2	0	0	0	1	1	0	1	0
K3	0	0	0	1	1	0	0	1
K4	1	0	1	1	1	1	0	0

Рисунок 3.22 – Матриця незалежних контурів графа, структурована відносно дерева та антидерева

3.5 Формування матриць параметрів і векторів

Параметри розподіляються по гілкам. Тобто, кожна гілка має коефіцієнт інерційності потоку (K), аеродинамічний опір (R) та вектор тисків вентиляторів (H). Такі дані користувач повинен ввести під час передачі кожної гілки графу до програми.

В результаті для всього графу формуються діагональні матриці K та R і вектор H. Вони мають однакову структуру (рис. 3.23).

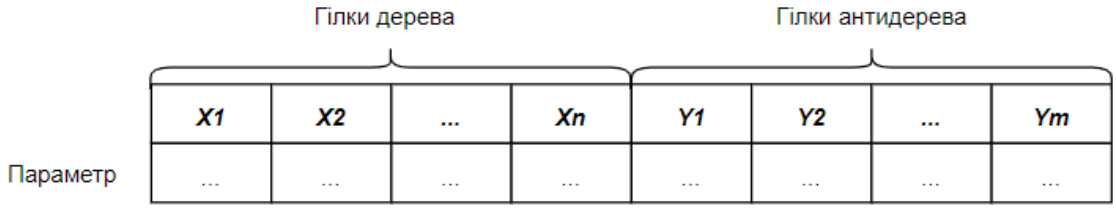


Рисунок 3.23 – Структура параметрів графа

Для діагональних матриць в структуру записується тільки діагональ, а вектор H транспонується та записується саме транспонованим.

У якості прикладу буде використовуватись параметри графу, що описані в табл. 2.1. В результаті буде отримана наступні параметри, структуровані відносно дерева та антидерева, що зображені на рис. 3.24.

K =

X1	X2	X3	X4	Y1	Y2	Y3	Y4
29	232.5	430	125	153	273	221	29

R =

X1	X2	X3	X4	Y1	Y2	Y3	Y4
0.0078	1.5	1.93	0.314	0.387	2.68	1.28	0.42

H^T =

X1	X2	X3	X4	Y1	Y2	Y3	Y4
0	0	0	4100	0	0	0	0

Рисунок 3.24 – Параметри, структуровані відносно дерева та антидерева

3.6 Висновки

Отже, було розглянуто алгоритми генерування топологічних характеристик: алгоритм побудови таблиці кодування графа МДОЗП,

генерування дерева та антидерева, генерування матриці інциденцій, матриці незалежних контурів та алгоритм формування матриць параметрів і векторів.

Для кожного алгоритму було приведено теоретичні визначення об’єкту, з яким відбувається робота та наведено його короткий опис.

Було проведено ручне покрокове виконання алгоритмів генерування дерева й антидерева, матриці інциденцій та незалежних контурів, на прикладі тестового МДОЗП.

Розроблено структуру кожної матриці та дано опис кожного елементу структури.

Розглянуто принцип введення графа до програми, його умови та структуру кожного рядку файлу, яка відповідає за його окрему гілку.

4 ПРОГРАМНА ІМПЛЕМЕНТАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ

4.1 Структура паралельного симулятора

Симулятор має наступну структуру:

- частина введення параметрів, які потрібні для обчислень і вводяться за допомогою файлів;
- частина обміну даними між процесами;
- частина обчислень, в якій використовуються введені параметри для вирішення системи (2.23);
- частина виведення результату, в якій кожен i-процес виводить свою пару (X_i Y_i).

Параметри.

Симулятор потребує такі параметри:

- матриця W;
- матриця TP;
- матриця Ru;
- вектор Y (початкова умова);
- вектор H.

Параметр W – це допоміжна матриця, описана в другому розділі. Вона розраховується за формулою:

$$W = A_X^{-1} \cdot A_Y,$$

де A_X^{-1} – обернена матриця інциденцій в гілках дерева,

A_Y – матриця інциденцій в гілках антидерева.

Матриця інциденцій (A) буде рівна значенню для тестового прикладу (рис. 3.15). З матриці А можна виділити A_X та A_Y (4.1).

$$A_X = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 \end{pmatrix} \; A_Y = \begin{pmatrix} 1 & 0 & -1 & -1 \\ 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix} \tag{4.1}$$

За допомогою інструменту Wolfram Alpha [23], буде розраховано параметр W (рис. 4.1).

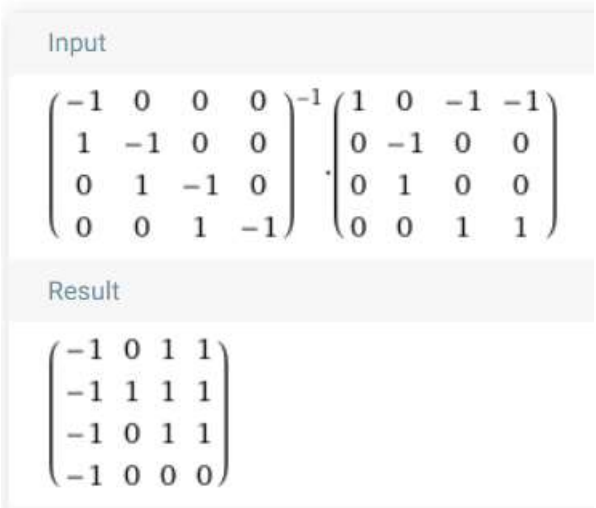


Рисунок 4.1 – Параметр W

Також можна знайти цей параметр аналітично, використовуючи формулу:

$$X = -W \cdot Y \tag{4.2}$$

Для цього потрібно розписати (4.2) наступним чином:

$$\begin{pmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{pmatrix} = -\begin{pmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix} \cdot \begin{pmatrix} Y_1 \\ Y_2 \\ Y_3 \\ Y_4 \end{pmatrix}$$

Тепер всі $X_1 - X_4$ треба розписати відносно $Y_1 - Y_4$, використовуючи рівняння вузлів (2.4) для заданого графа:

$$\begin{pmatrix} Y_1 - 0 - Y_3 - Y_4 \\ Y_1 - Y_2 - Y_3 - Y_4 \\ Y_1 - 0 - Y_3 - Y_4 \\ Y_1 - 0 - 0 - 0 \end{pmatrix} = -\begin{pmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix} \cdot \begin{pmatrix} Y_1 \\ Y_2 \\ Y_3 \\ Y_4 \end{pmatrix}$$

Залишилось підставити значення в матрицю так, щоб рівняння було правильним:

$$\begin{pmatrix} Y_1 - 0 - Y_3 - Y_4 \\ Y_1 - Y_2 - Y_3 - Y_4 \\ Y_1 - 0 - Y_3 - Y_4 \\ Y_1 - 0 - 0 - 0 \end{pmatrix} = -\begin{pmatrix} -1 & 0 & 1 & 1 \\ -1 & 1 & 1 & 1 \\ -1 & 0 & 1 & 1 \\ -1 & 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} Y_1 \\ Y_2 \\ Y_3 \\ Y_4 \end{pmatrix}$$

Отже, аналітичним та численним способами було отримано параметр W:

$$W = \begin{pmatrix} -1 & 0 & 1 & 1 \\ -1 & 1 & 1 & 1 \\ -1 & 0 & 1 & 1 \\ -1 & 0 & 0 & 0 \end{pmatrix} \tag{4.3}$$

Зауваження: параметр W передається до симулятору заздалегідь помножений на мінус.

Параметр TP розраховується за формулою:

$$TP = (S_Y K_Y - S_X K_X W)^{-1} \cdot S,$$

де S_X та S_Y – матриці незалежних контурів в гілках дерева та антидерева відповідно;

K_X та K_Y – діагональні матриці коефіцієнтів інерційності потоку повітря в гілках дерева та антидерева відповідно;

W – допоміжна матриця (4.3);

S – матриця незалежних контурів.

За допомогою розрахованих матриць в п.п 3.4 – 3.5 буде знайдено покрокове рішення матриці TP. На рис. 4.2 – 4.5.

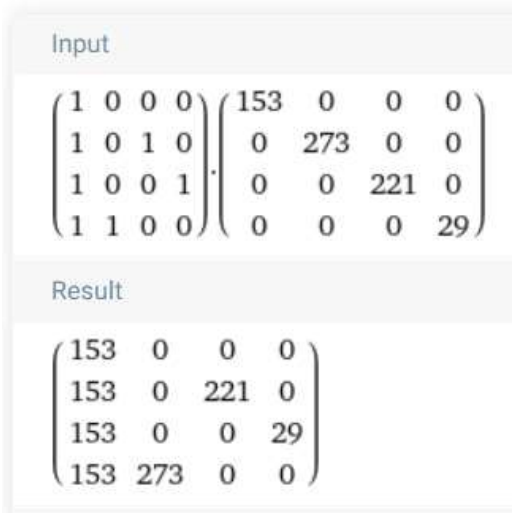


Рисунок 4.2 – Матриця $S_Y K_Y$

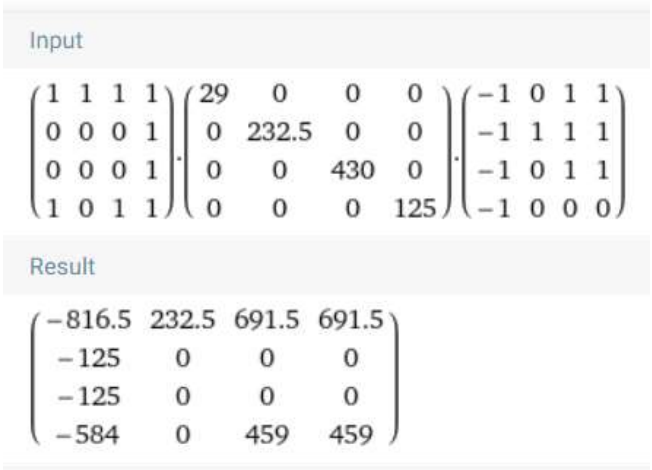


Рисунок 4.3 – Матриця $S_X K_X W$

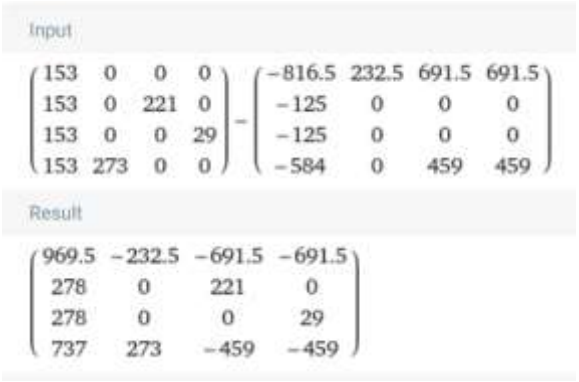


Рисунок 4.4 – Матриця $S_Y K_Y - S_X K_X W$

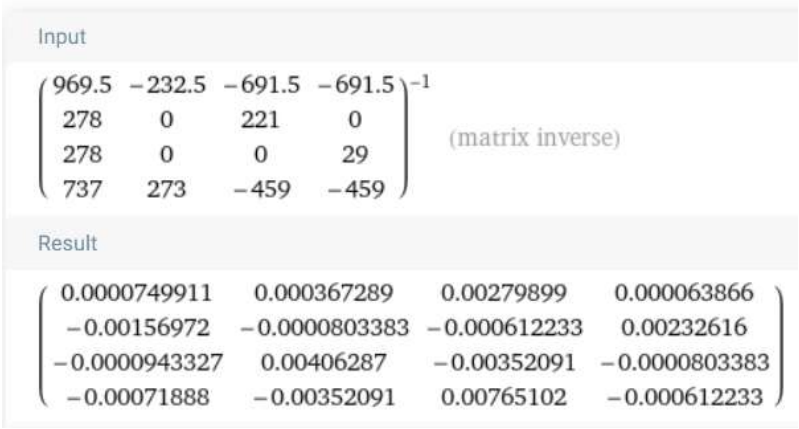


Рисунок 4.5 – Матриця $(S_Y K_Y - S_X K_X W)^{-1}$

Значення матриці TP зображено на рис. 4.6.

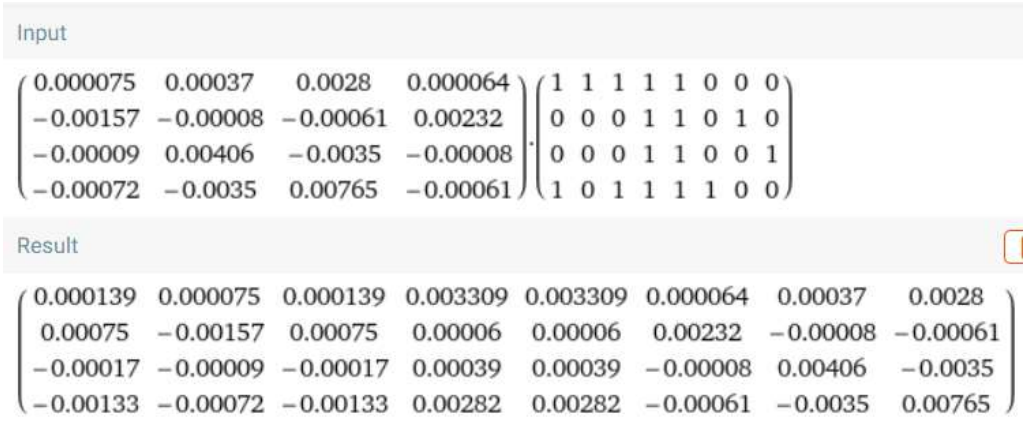


Рисунок 4.6 – Матриця TP

Параметр Ru знаходиться за формулою:

$$Ru = TP \cdot R = (S_Y K_Y - S_X K_X W)^{-1} \cdot S \cdot R,$$

де R – діагональна матриця коефіцієнтів інерційності потоку повітря в гілках дерева та антидерева відповідно.

Значення матриці Ru зображено на рис. 4.7.

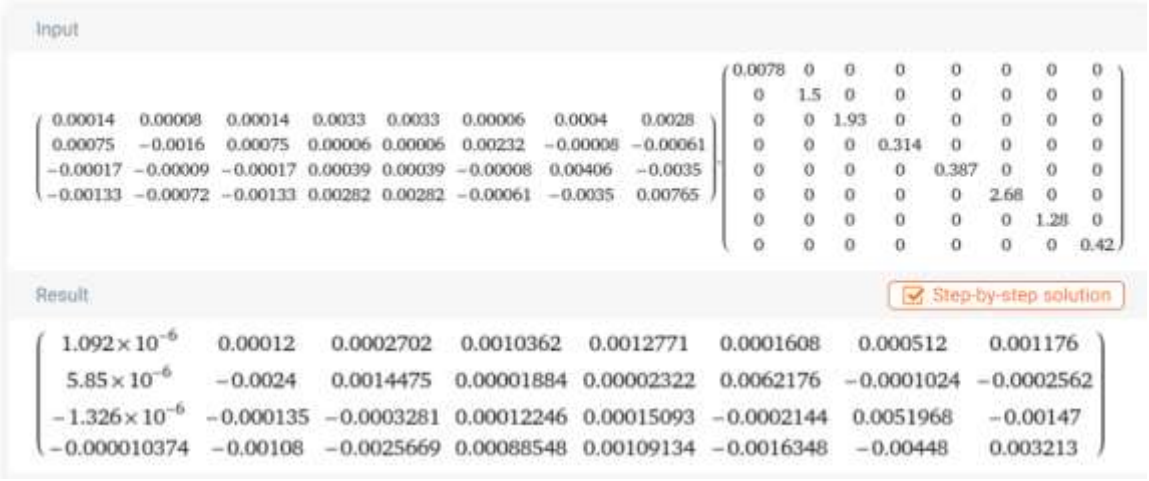


Рисунок 4.7 – Матриця Ru

Вектор H зображено на рис. 3.24.

Обмін даними.

Обмін даними між процесами паралельного симулятора відбувається із використанням функцій MPI [24-28], такими як MPI_Bcast (розсилка одних і тих же даних з головного процесу всім іншим) та MPI_Gather (збирання даних із всіх процесів в головному). Синхронізація процесів виконується за допомогою функції MPI_Barrier.

Обчислення.

Обчислення починаються з того, що задаються початкові умови:

$$Y(0) = Y_1, X(0) = -W \cdot Y_1$$

Вектор стаціонарних потоків $Q_{\text{стац}}$ при цьому рівний:

$$Q_{\text{стац}} = (71.65\ 28.92\ 14.04\ 14.88\ 19.20\ 23.65\ 28.92\ 71.65\ )^T$$

Вирішення відбуваються, поки не буде досягнута вказана точність δ:

$$\Delta Q_i = |Q(i) - Q(i-1)| \leq \delta$$

Результати кожен процес записує в окремий файл.

4.2 Програми реалізації розроблених алгоритмів

Було створено ПЗ із графічним інтерфейсом, яке генерує топологічні характеристики заданого графа. При першому запуску додатку можна бачити вікно (рис. 4.8).

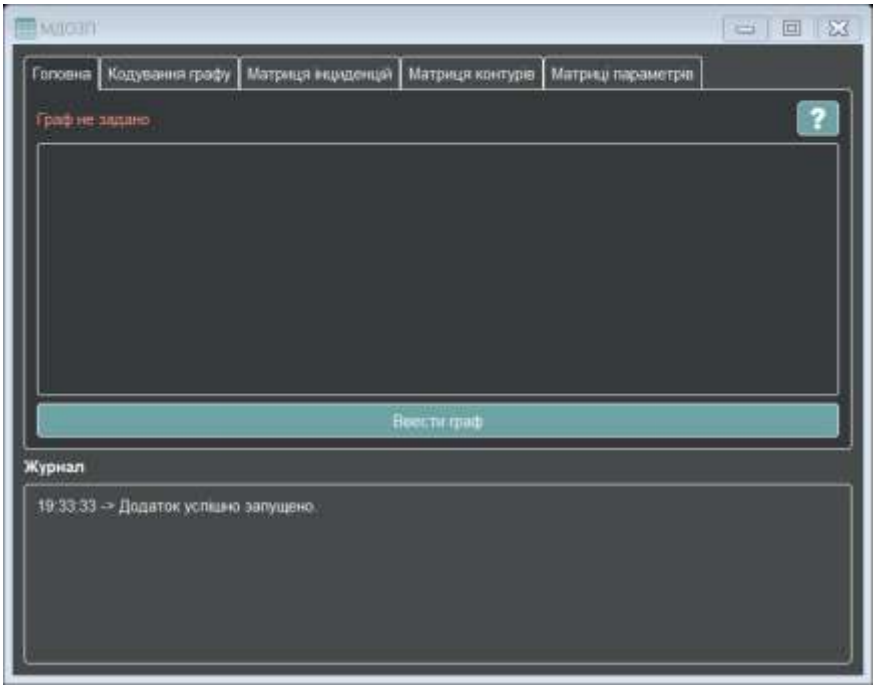


Рисунок 4.8 – Вікно програми після першого запуску

ПЗ готове для того, щоб граф було введено. Після натискання на кнопку довідки («?») можна побачити, якими є умови для вхідного графу (рис. 4.9).

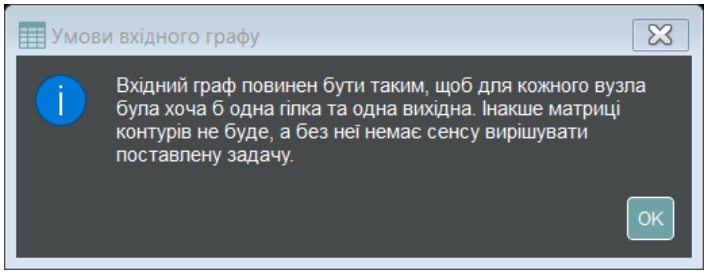


Рисунок 4.9 – Вікно довідки щодо умов вхідного графу

Решта вкладок зображені на рис. 4.10 – 4.13.

Рисунок 4.10 – Вкладка «Кодування графу»

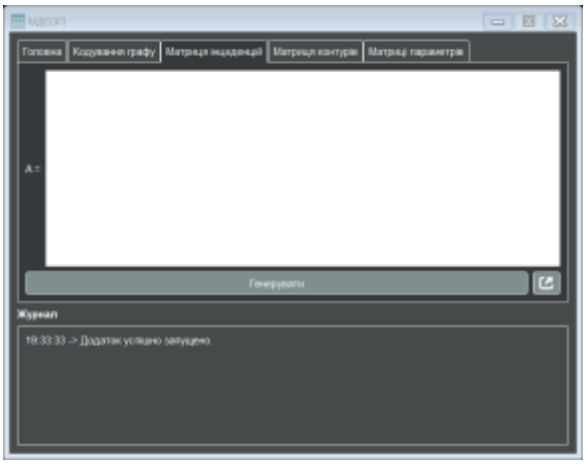


Рисунок 4.11 – Вкладка «Матриця інциденцій»

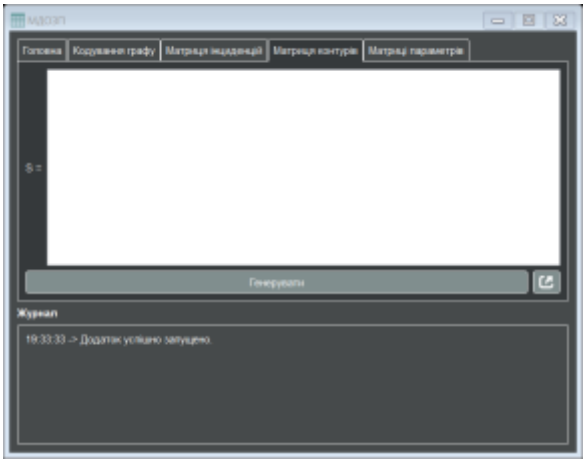


Рисунок 4.12 – Вкладка «Матриця контурів»

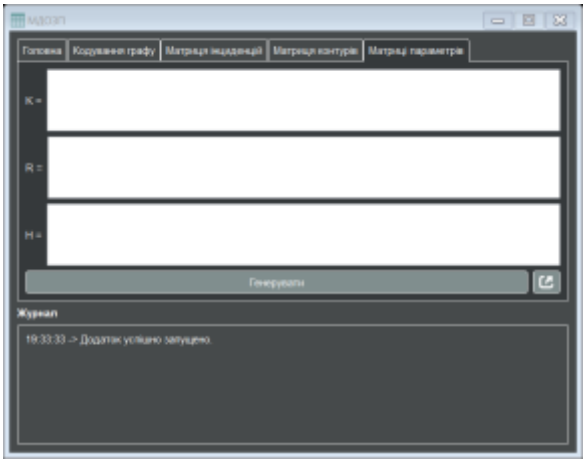


Рисунок 4.13 – Вкладка «Матриці параметрів»

Як видно, всі вкладки пусті, коли граф не задано.

Після натискання на кнопку «Ввести граф» можна бачити наступний діалог, що зображено на рис. 4.14.

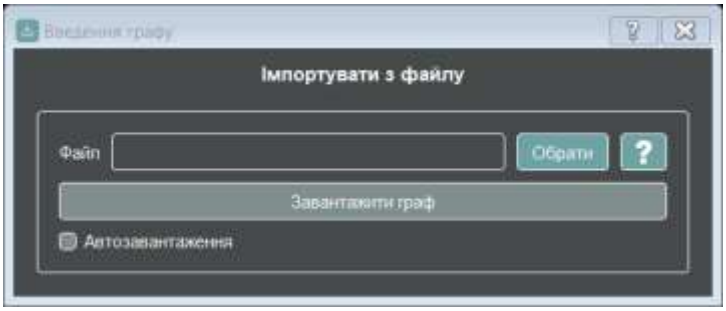


Рисунок 4.14 – Діалогове вікно «Введення графу»

Кнопка довідки («?») видасть на екран інформацію щодо структури файлу з графом (рис. 4.15).

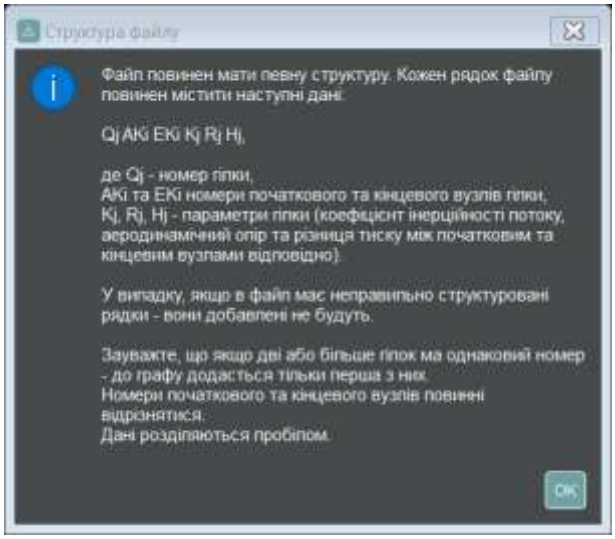
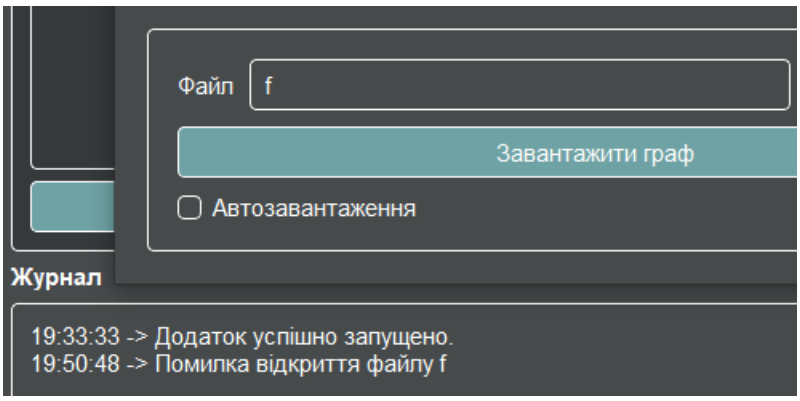


Рисунок 4.15 – Довідка щодо структури файлу з графом

Якщо натиснути на кнопку «Обрати», тоді відкриється провідник з файлами в корені диску та буде чекати на вибір файлу з графом. Після чого треба натиснути на кнопку «Завантажити граф». Якщо файл не буде задано, або ім’я файлу буде задано неправильне, у додатку в журналі роздрукується про це повідомлення (рис. 4.16а). У випадку успішного зчитування всього файлу, або деяких рядків (тобто будуть рядки, які були не правильно структуровані), тоді в журналі буде про це повідомлення відповідно. Якщо зчитаний граф не підходить умовам (рис. 4.16б) або підходить умовам (рис. 4.16в), можна бачити в статусі графа про це.

а)



б)

в)

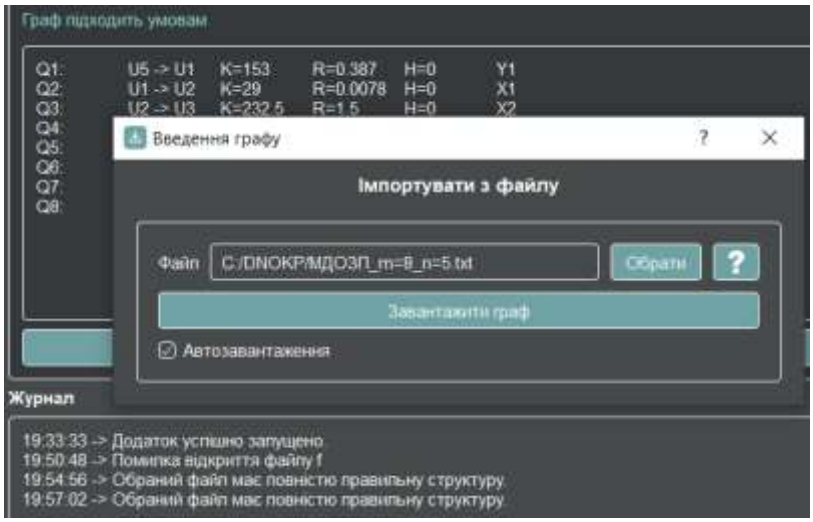


Рисунок 4.16 – Фрагменти додатку із завантаженням графу: а) такого файлу немає; б) файл має граф, що не підходить умовам; в) файл немає помилок структури, граф підходить умовам

Функція автозавантаження при кожному наступному запуску додатку буде завантажувати граф із раніше вказаного файлу.

Після завантаження графу окрім статусу можна побачити його схематичний опис (рис. 4.17). У якості вхідного графу було обрано тестову модель МДОЗП.

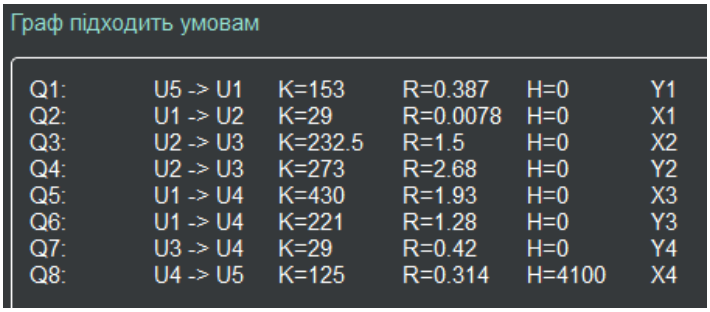
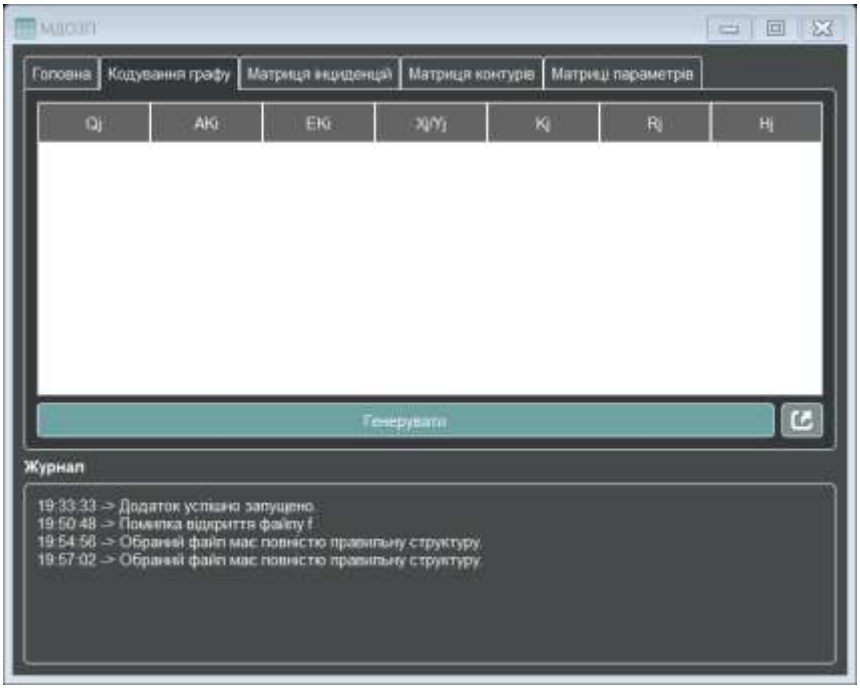


Рисунок 4.17 – Схематичний опис заданого графу

Коли статус графа «Граф підходить умовам», алгоритми, які знаходяться на інших вкладках, тепер доступні.

Для генерування таблиці кодування, треба перейти на вкладку «Кодування графу» (рис. 4.18а) і натиснути на кнопку «Генерувати», після чого таблиця кодування буде заповнена (рис. 4.18б).

а)



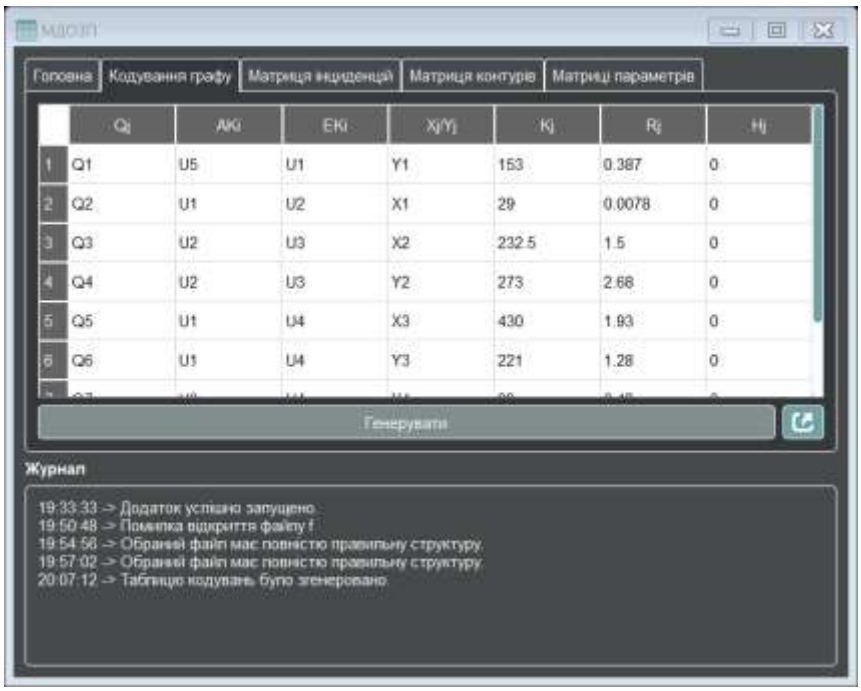
б) 

Рисунок 4.18 – Таблиця кодування: а) до, б) після генерування

Також є можливість зберегти таблицю кодування у файл. Для цього потрібно натиснути на стрілку праворуч від кнопки «Генерувати». Після чого з’явиться діалогове вікно з провідником де потрібно вказати директорію, де буде створено файл. В результаті буде отримано файл зі змістом, що зображено на рис. 4.19.

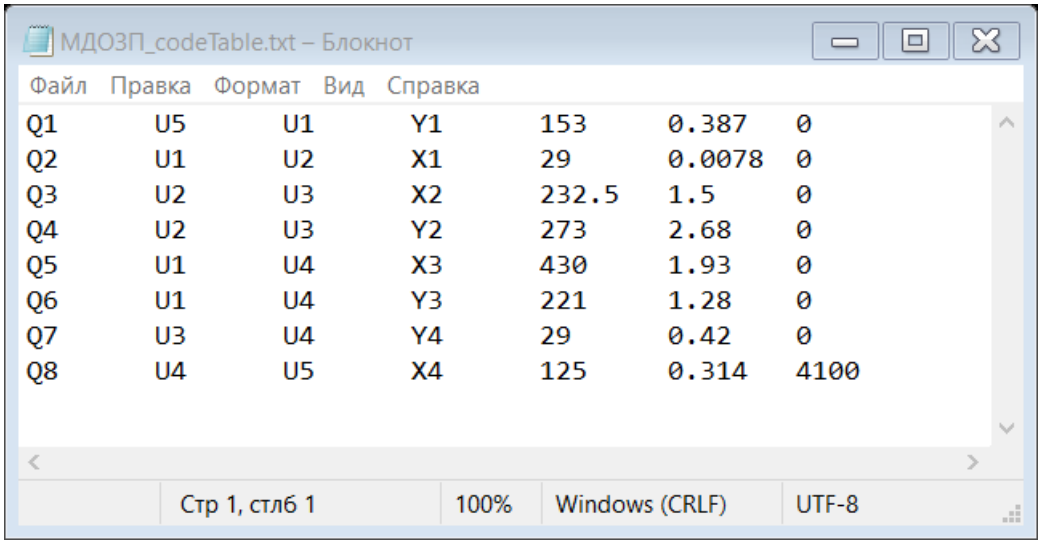


Рисунок 4.19 – Файл із таблицею кодування графу

Аналогічним чином структуровані решта вкладок: «Матриця інциденцій» (рис. 4.20), «Матриця контурів» (рис. 4.21) та «Матриці параметрів» (рис. 4.22).

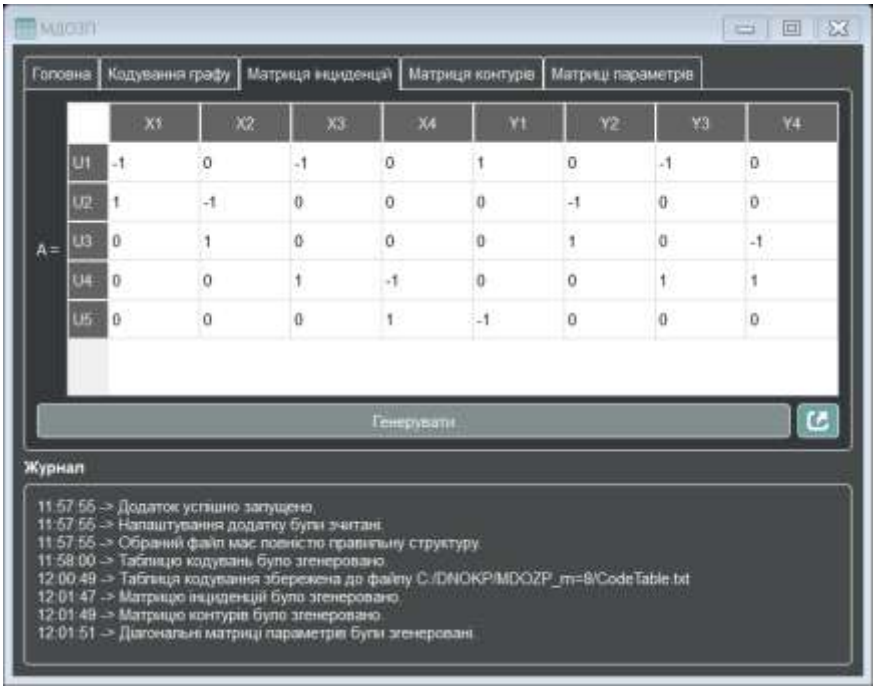


Рисунок 4.20 – Згенерована матриця інциденцій

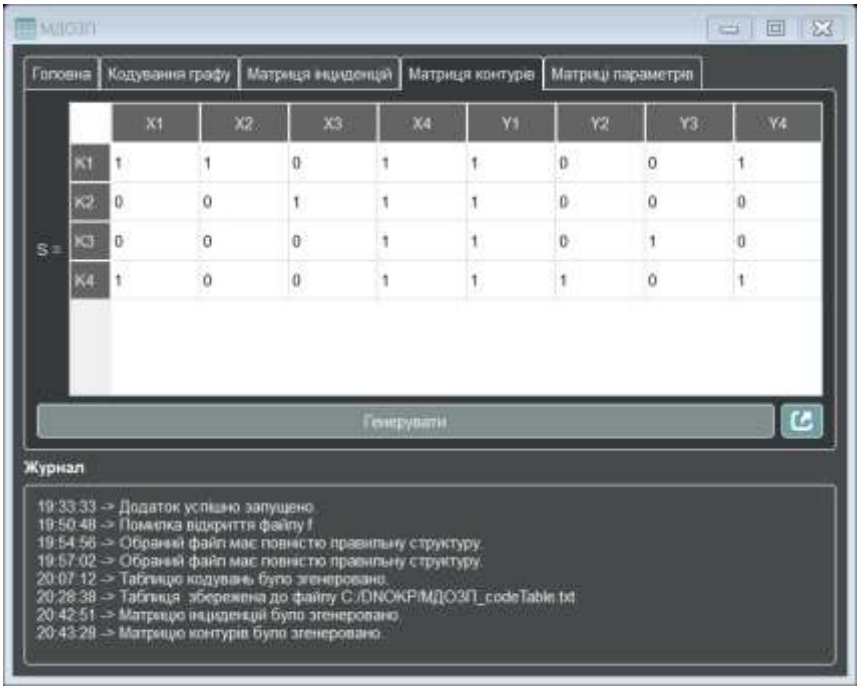


Рисунок 4.21 – Згенерована матриця контурів

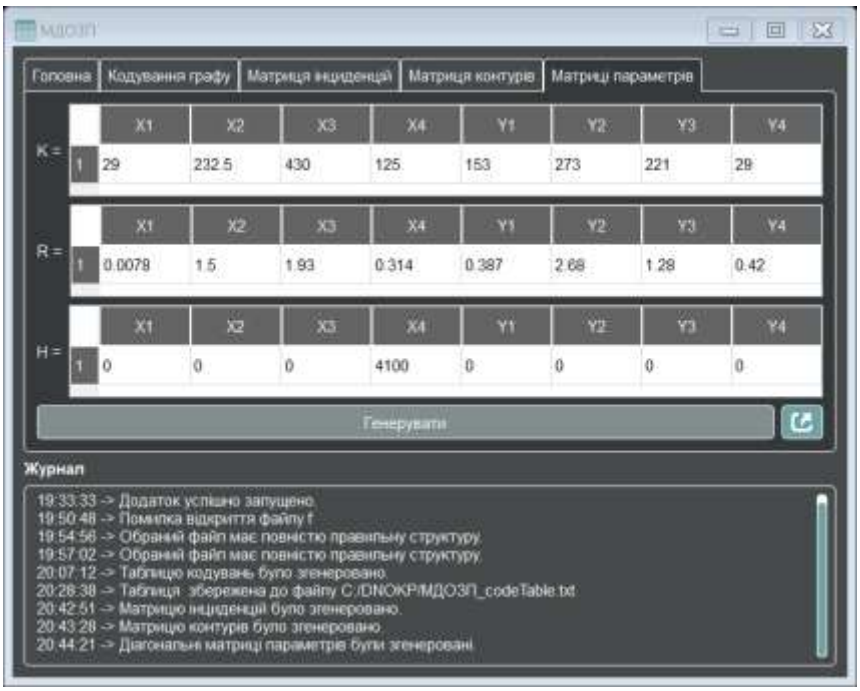


Рисунок 4.22 – Згенеровані матриці параметрів та векторів

4.3 Експериментальні дослідження

Застосувавши отримані топологічні характеристики можна підготувати потрібні параметри для паралельного симулятора. На основі отриманих файлів можна побудувати графіки потоків відносно вузлів $U_1 - U_4$ (рис. 4.23-4.26).

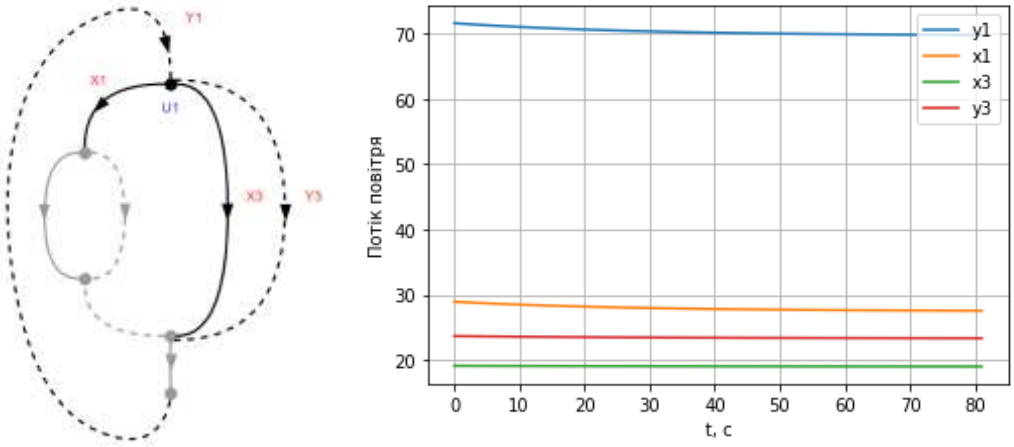


Рисунок 4.23 – Потоки повітря відносно вузла U_1

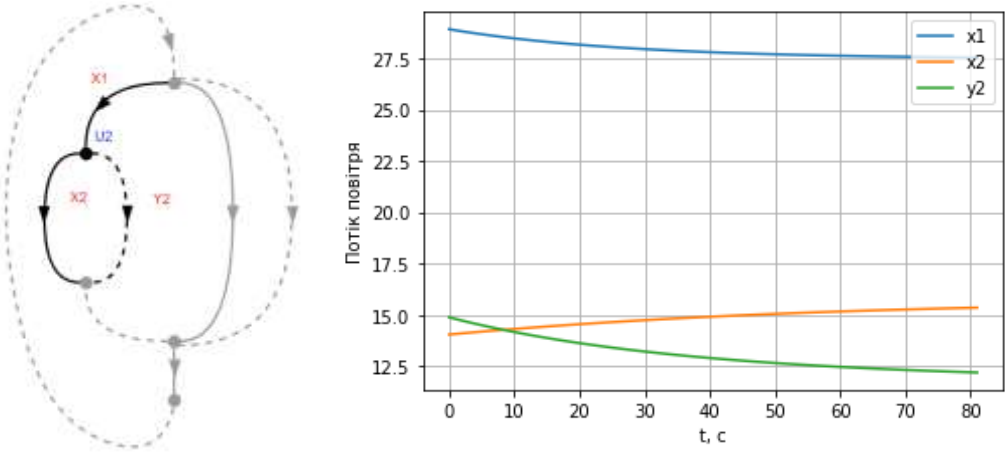


Рисунок 4.24 – Потоки повітря відносно вузла U_2

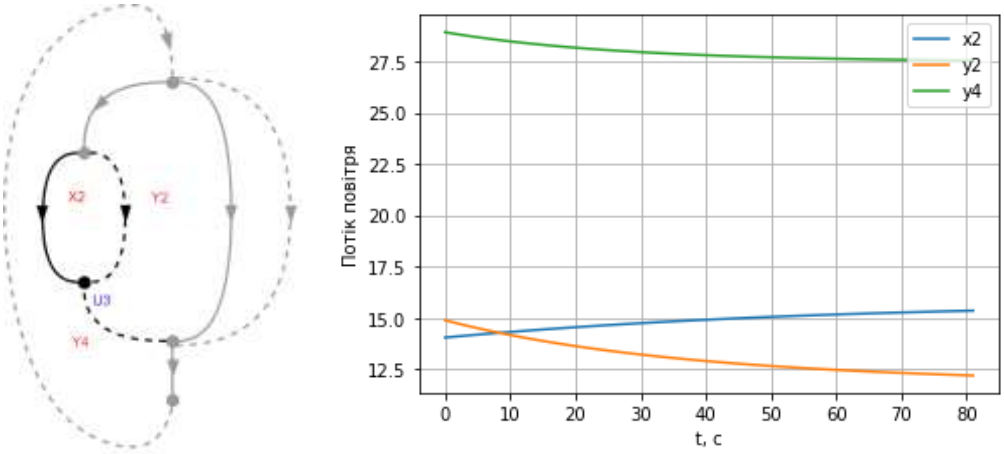


Рисунок 4.25 – Потоки повітря відносно вузла U_3

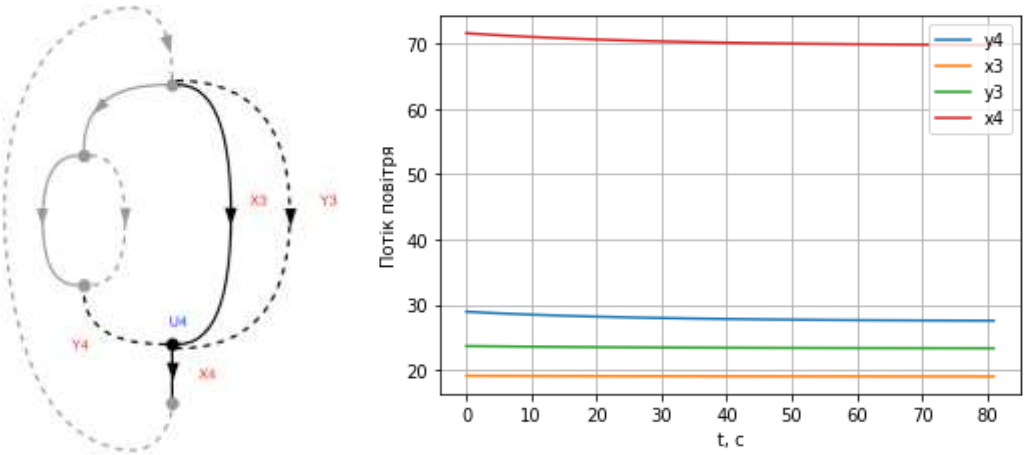


Рисунок 4.26 – Потоки повітря відносно вузла U_4

Правдивість даних графіків легко довести, використовуючи перший закон Кігхгофа [14-16].

4.4 Висновки

Отже, було розглянуто структуру паралельного симулятора, який складається з частини введення параметрів за допомогою файлів; обміну даними між процесами, що реалізується за допомогою колективних операцій MPI; обчислень, в яких диференціальне рівняння першого порядку вирішується за допомогою метода Адамса-Башфорта та частини виведення результатів кожного процесу в окремий файл, на основі яких відбувається візуалізація результатів і їх аналіз на правильність.

Наведено опис та покроковий розрахунок кожного параметру, який потрібен симулятору для початку розрахунків.

Приведені скріншоти програми та опис її функціональних елементів, з яких вона складається. На прикладі моделі тестового МДОЗП було побудовано за допомогою додатку топологічні характеристики, які в подальшому були використані для розрахунків в паралельному симуляторі і за допомогою візуалізації результатів – перевірені на правильність.

ВИСНОВКИ

Під час виконання даної кваліфікаційної роботи було спроектовано та реалізовано ПЗ як засіб топологічного аналізу, який забезпечує активну комп'ютерну підтримку побудови паралельних симуляторів МДОЗП. Для цього було розглянуто теоретичний аналіз алгоритмів генерування матриць інциденцій, незалежних контурів, тощо та виділені основні етапи розрахунків, які були використані в програмній реалізації цих алгоритмів.

Дане ПЗ є об'єктно-орієнтованим. Воно було розроблено за допомогою інструменту Qt [29-30] і написане використовуючи мову програмування C++. ПЗ забезпечене автозавантаженням графу з раніше вказаного файлу, що дозволяє не вводити кожного разу при запуску додатку один і той же файл.

Вхідний граф перевіряється і якщо він підходить умовам, тоді можна безперешкодно продовжувати роботу з ним. Для перегляду структури графа використовується його текстовий опис.

Робота з графом після його завантаження полягає в генеруванні його топологічних характеристик, кожну з яких можна завантажити в окремий файл.

На основі вихідних результатів ПЗ було зроблено аналіз розрахунків, які проводилися вручну та за допомогою програмних алгоритмів. В результаті порівняння було зроблено висновок, що мету даної роботи було досягнуто.

Подальша робота полягає в тому, щоб розглянути ДСРП, що було опущено в рамках даної роботи. Спроектоване ПЗ може бути використане для топологічного аналізу графа мережевого динамічного об'єкту з розподіленими параметрами, але для цього можливості ПЗ треба розширити.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Gao Z., Kong D., Gao C. Modeling and control of complex dynamic systems: applied mathematical aspects. *Journal of applied mathematics*. 2012. Vol. 2012. P. 1–5. URL: https://doi.org/10.1155/2012/869792 (дата звернення: 05.06.2022).
2. Svyatnyy V.A., Kushnarenko V.G., Resch M. Problematik der parallelen Simulationstechnik. *Наукові праці ДонНТУ*. Сер. Інформатика, кібернетика та обчислювальна техніка. 2016. №2 (23). С. 5-20.
3. Святный В.А. Моделирование аэрогазодинамических процессов и разработка систем управления проветриванием угольных шахт: дисс. докт. техн. наук. ДПИ, Донецк, 1985. 407 с.
4. Масюк А.Л. Інтерактивні засоби паралельного моделюючого середовища, орієнтованого на мережеві динамічні об’єкти: дис. к. т. н. ДонНТУ, Покровськ, 2018. 168 с.
5. Svyatnyy V.A., Kushnarenko V.G., Miroshkin O.M. Parallele Modellierung und Simulation der luftdynamischen Prozesse in den Grubenbewetterungsnetzen. *Вісті Донецького гірничого інституту*. 2018. №1 (42). С. 116-129.
6. Святний В.А., О.В. Молдованова, А.М. Чут. Стан розробок та перспективи інтеграції паралельних моделюючих середовищ з Grid-технологіями. *Наукові праці Донецького національного технічного університету*. Сер. Інформатика, кібернетика та обчислювальна техніка. 2008. Вип. 9. С. 103-110. URL: http://nbuv.gov.ua/UJRN/Npdntu_inf_2008_9_17 (дата звернення 01.06.2022).
7. Bertoni G., Penati M. E. Lumped parameter dynamic models for large space structures with flexible and rigid parts. *IFAC proceedings volumes*. 1983.

Vol. 16, no. 11. P. 125-131. URL: https://doi.org/10.1016/s1474-6670(17)62194-3 (дата звернення: 05.06.2022).

8. Incerti G., Petrogalli C., Solazzi L. Lumped parameter models for numerical simulation of the dynamic response of hoisting appliances. *International Science Index (WASET) conference*, Venice, Italy, 9-10 November 2015, Vol. 17, no. 11, Part II. 2015.
9. Liu Y. On model reduction of distributed parameter models : Licentiate thesis, monograph. 2002. 158 p. URL: http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-1541 (дата звернення: 05.06.2022).
10. Голинько В.И., Лебедев Я.Я., Литвиненко А.А. Аэрология горных предприятий: учеб. пособие. НГУ. Донецк, 2015. С. 125-148, 195-203.
11. Wilson R.J. Introduction to graph theory. London : Longman, 1975. 180 p.
12. Bondy J. A., Murty U. S. R. Graph theory with applications. London : Macmillan Education UK, 1976. 270 p. URL: https://doi.org/10.1007/978-1-349-03521-2 (дата звернення: 31.05.2022).
13. Contributors to Wikimedia projects. Incidence matrix - Wikipedia. *Wikipedia, the free encyclopedia*. URL: https://en.wikipedia.org/wiki/Incidence_matrix (дата звернення: 01.06.2022).
14. Reese T.M. Kirchhoff graphs : text. 2018. 347 p. URL: https://digitalcommons.wpi.edu/etd-dissertations/72 (дата звернення: 31.05.2022).
15. Choi J., Kim J. Kirchhoff's circuit law applications to graph simplification in search problems. *2020 international conference on information and communication technology convergence (ICTC)*, Jeju Island, Korea (South), 21–23 October 2020. 2020. URL: https://doi.org/10.1109/ictc49870.2020.9289222 (дата звернення: 31.05.2022).

16. Contributors to Wikimedia projects. Kirchhoff's circuit laws - Wikipedia. *Wikipedia, the free encyclopedia*. URL: https://en.wikipedia.org/wiki/Kirchhoff's_circuit_laws (дата звернення: 31.05.2022).
17. Дерець О.Л. Конспект лекцій з дисципліни «Спеціальні питання математичного опису і моделювання динаміки складних систем»: для студ. спеціальностей 7.05070108, 8.05070108 «Енергетичний менеджмент», 7.05070202, 8.05070202 «Електричні системи і комплекси транспортних засобів» та 7.05070204, 8.05070204 «Електромеханічні системи автоматизації і електропривод» ден. та заоч. форм навчання / ДДТУ. Дніпродзержинськ, 2011. С. 6-14.
18. Contributors to Wikimedia projects. Tree (graph theory) - Wikipedia. *Wikipedia, the free encyclopedia*. URL: https://en.wikipedia.org/wiki/Tree_(graph_theory) (дата звернення: 31.05.2022).
19. Golub G.H. Matrix computations. 3rd ed. Baltimore : Johns H Kirchhoff Graphs opkins University Press, 1996. 694 p.
20. Aceto L., Trigiante D. The stability problem for linear multistep methods: old and new results. *Journal of computational and applied mathematics*. 2007. Vol. 210, no. 1-2. P. 2–12. URL: https://doi.org/10.1016/j.cam.2006.10.052 (дата звернення: 01.06.2022).
21. Adams-Bashforth and Adams-Moulton methods - Wikiversity. *Wikiversity*. URL: https://en.wikiversity.org/wiki/Adams-Bashforth_and_Adams-Moulton_methods (дата звернення: 31.05.2022).
22. Панасюк А.В., Левицький В.Г., Іськов С.С. Аерологія гірничих підприємств: методичні вказівки до виконання практичних робіт. Житомир: ЖДТУ, 2013. С. 5-10.
23. Wolfram|Alpha: making the world’s knowledge computable. *Wolfram|Alpha: Computational Intelligence*. URL: https://www.wolframalpha.com/ (дата звернення: 31.05.2022).

24. Contributors to Wikimedia projects. Message passing interface - wikipedia. *Wikipedia, the free encyclopedia*. URL: https://en.wikipedia.org/wiki/Message_Passing_Interface (дата звернення: 01.06.2022).
25. Бройнль Т. Паралельне програмування: Початковий курс: навч. посібник / пер. з нім. В.А. Святного. К.: Вища шк., 1997. 358 с.
26. Хьюз К., Хьюз Т. Параллельное и распределенное программирование с использованием C++ / пер. с англ. Изд.: Вильямс, 2004. 672 с.
27. MPI documentation | RookieHPC. *RookieHPC*. URL: https://www.rookiehpc.com/mpi/docs/index.php (дата звернення: 01.06.2022).
28. Эндрюс Г.Р. Основы многопоточного, параллельного и распределенного программирования. М.: Издательский дом "Вильямс", 2003. 512 с.
29. Contributors to Wikimedia projects. Qt (software) - Wikipedia. *Wikipedia, the free encyclopedia*. URL: https://en.wikipedia.org/wiki/Qt_(software) (дата звернення: 31.05.2022).
30. Qt documentation | home. *Qt Documentation | Home*. URL: https://doc.qt.io/ (дата звернення: 01.06.2022).

ДОДАТОК А – Заява на затвердження теми і керівника

Завідувачу кафедри
комп’ютерної інженерії
Ковальову Сергію Олександровичу
студентки 4 курсу, групи КІ-18,
спеціальності 123 Комп’ютерна інженерія
Тихонової Юлії Григорівни

ЗАЯВА

Прошу призначити керівником моєї роботи бакалавра д.т.н., проф. Святного Володимира Андрійовича

(вчена ступінь, вчене звання, посада, П.І.Б викладача)

і закріпити за мною наступну тему: «Засоби топологічного аналізу MIMD-симулятора мережевого динамічного об’єкту з зосередженими параметрами»

(найменування теми)

«___» _____ 2022 р. _____

(підпис студента)

ПОГОДЖЕНО:

Науковий керівник _____ Святний В. А.

(підпис) (прізвище та ініціали)

ДОДАТОК Б – Блок-схеми алгоритмів

Б.1 Алгоритм генерування дерева й антидерева

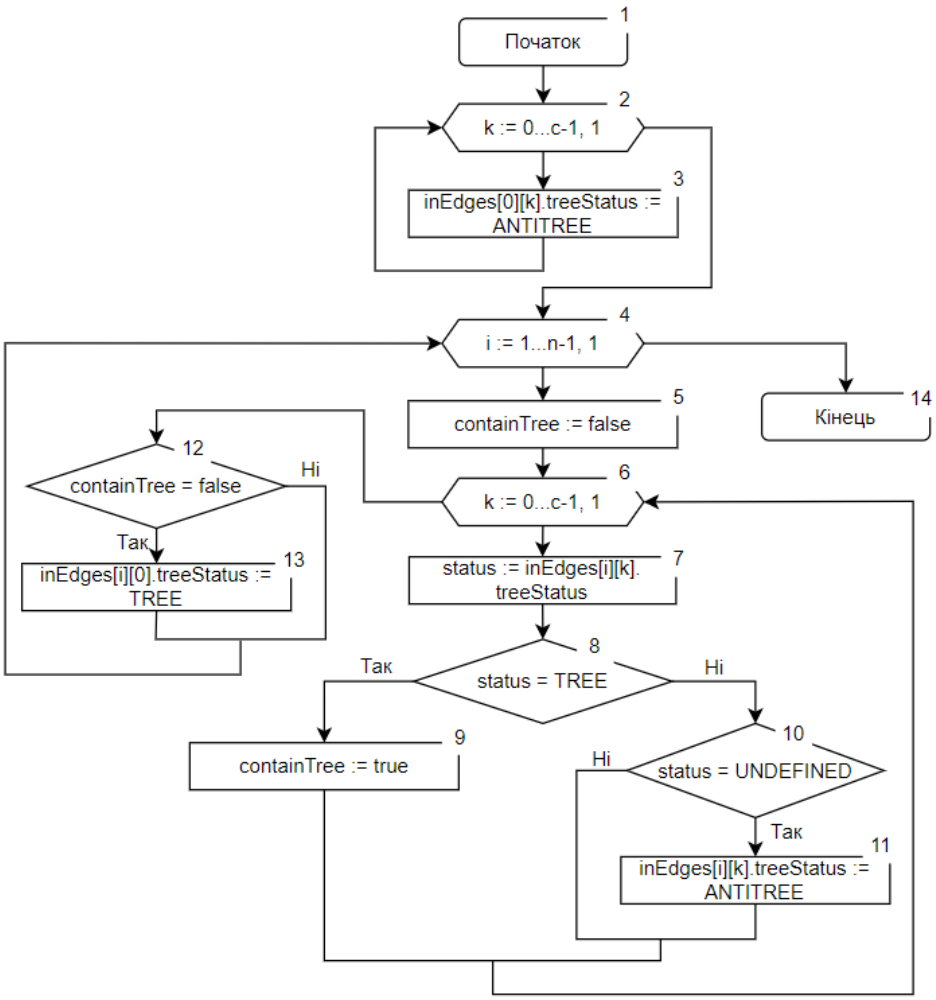


Рисунок Б.1.1 – Блок-схема алгоритму генерування дерева й антидерева

Блоки 2-3. Всі вхідні гілки першого вузла позначаються антидеревами.

Блоки 4-11. Прохід по всі вузлах графу окрім першого. Для кожної k-вхідної гілки i-вузла проводиться перевірка статусу (7-8). Якщо гілка вже позначена деревом (8-9) – вона пропускається та всі останні вхідні непозначені гілки позначаються антидеревом (8, 10-11).

Блоки 12-13. Якщо при проході по вхідним гілкам вузла не було знайдено гілку, позначену деревом, тоді вибирається перша із вхідних гілок вузла та позначається деревом.

Б.2 Алгоритм генерування матриці інциденцій

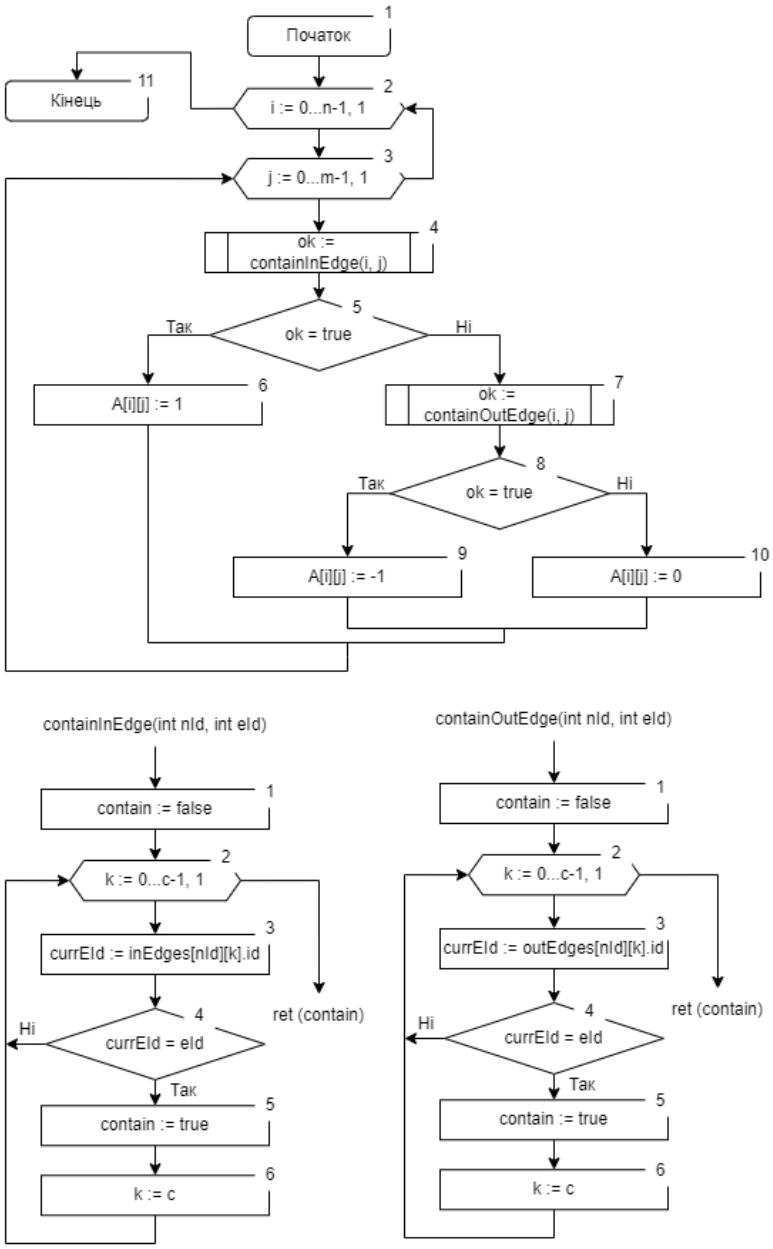


Рисунок Б.2.1 – Блок-схема алгоритму генерування матриці інциденцій

Основна блок-схема (рис. Б.2.1, перша схема).

Блоки 2-3. Прохід в головному циклі (2) відбувається по вузлам, а у внутрішньому циклі (3) – по гілкам.

Блоки 4-10. Викликається процедура (4), яка повертає в змінну ok відповідь, чи є в даному i-вузлу j-вхідна гілка (j-вихідна гілка для процедури

7). Якщо дана j-гілка є вхідною для i-вузлу, тоді в матрицю записується 1 (6), якщо вихідною – тоді -1 (9). В іншому випадку – 0 (10).

Б.3 Алгоритм генерування матриці незалежних контурів

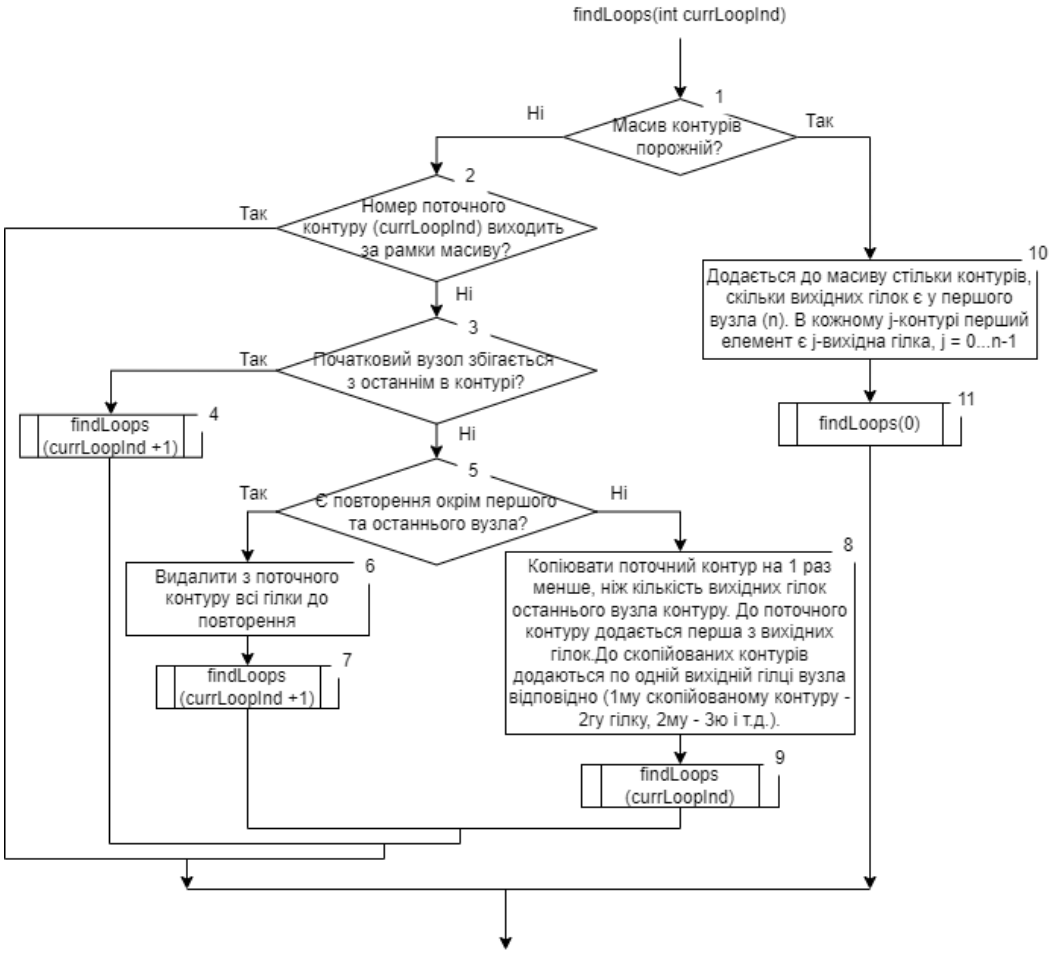


Рисунок Б.3.1 – Процедура знаходження контуру графа

Рисунок Б.3.2 – Блок-схема алгоритму пошуку контурів

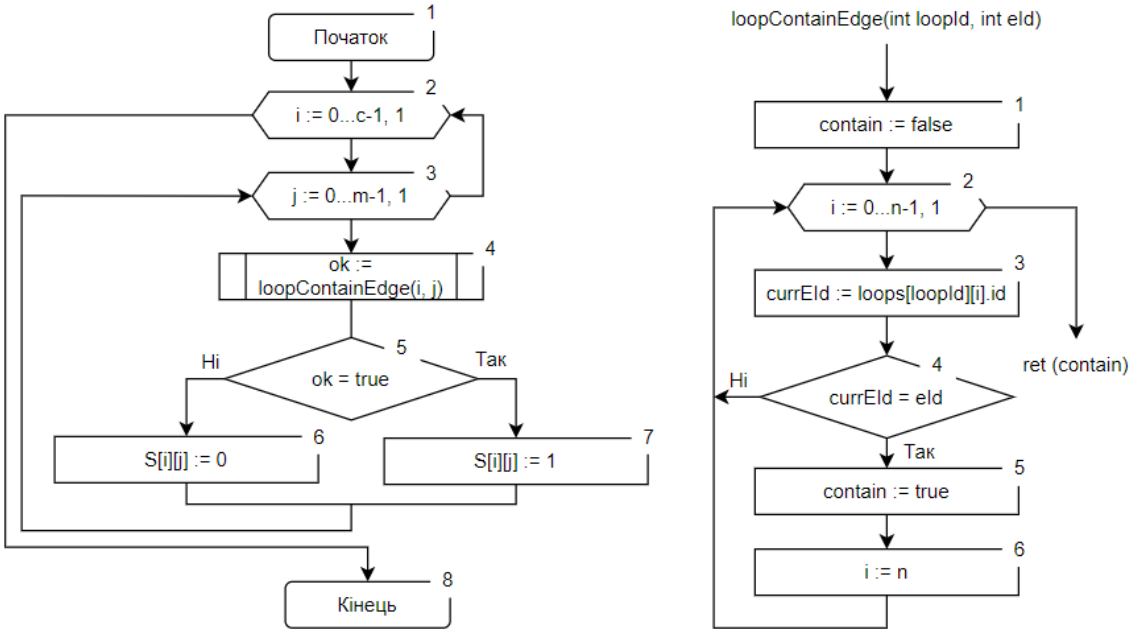


Рисунок Б.3.3 – Блок-схема алгоритму генерації матриці незалежних контурів графа

Основна блок-схема (рис. Б.3.3, схема ліворуч).

Блоки 2-4. Цикл проводиться по масиву контурів (2), а для кожного i-контуру перевіряється, чи належить j-гілка графу цьому контурі.

Блоки 5-7. Якщо j-гілка належить поточному i-контуру, тоді у матрицю контурів записується 1, в іншому випадку -1.

ДОДАТОК В – Діаграми класів

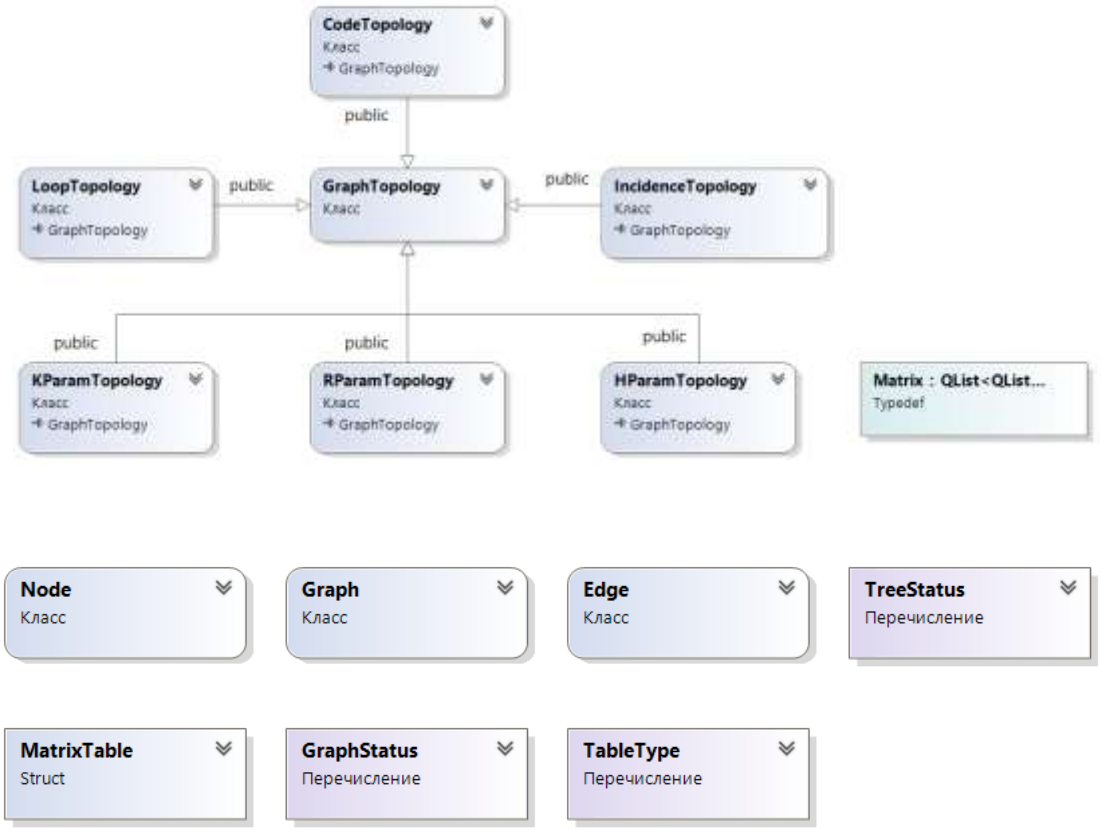
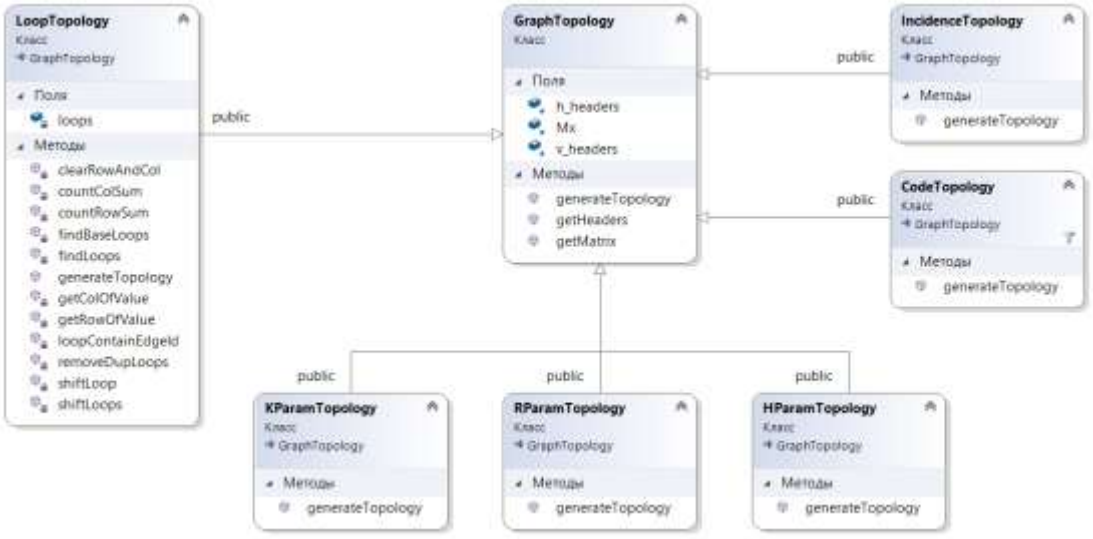
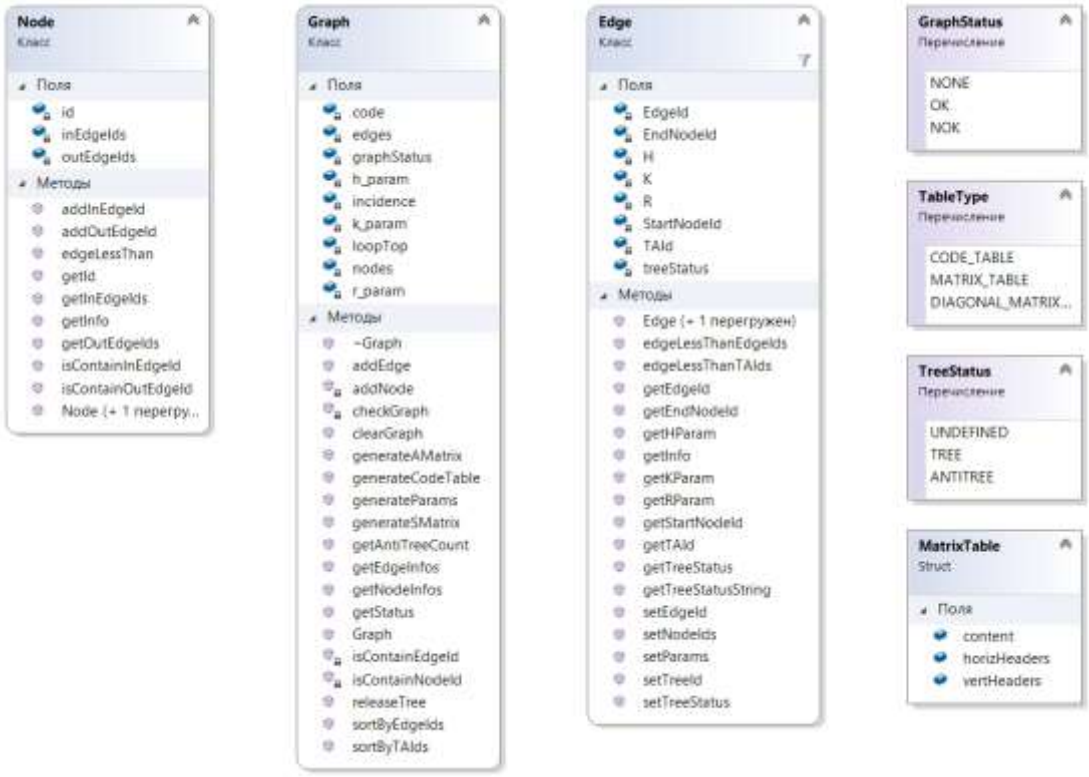


Рисунок В.1 – Загальна схема класів ПЗ



а)



б)

Рисунок В.2 – Розширена схема: а) класів топологій, б) допоміжних класів

ДОДАТОК Г – Лістинг ПЗ

Г.1 Файли заголовків

```
// Гілка
class Edge
{
private:
    // Ідентифікатори гілки
    int EdgeId = 0;
    int TAId = 0;   //Tree- / AntitreeId
    TreeStatus treeStatus = UNDEFINED;

    // Початкова та кінцева вузли
    int StartNodeId = 0;
    int EndNodeId = 0;

    // Параметри гілки
    float K = 0.0;
    float R = 0.0;
    float H = 0.0;

public:
    Edge() : Edge(0, 0, 0, 0.0, 0.0, 0.0) {}
    Edge(int edgeid, int s_nodeid, int e_nodeid, float k, float r, float h) :
         EdgeId(edgeid), StartNodeId(s_nodeid), EndNodeId(e_nodeid), K(k), R(r), H(h) {}

    // Сеттери
    void setEdgeId(int edgeId) { this->EdgeId = edgeId; }
    void setTreeId(int treeId) { this->TAId = treeId; }
    void setTreeStatus(TreeStatus treeStatus) { this->treeStatus = treeStatus; }

    void setNodeIds(int s_nodeid, int e_nodeid) { this->StartNodeId = s_nodeid; this->EndNodeId =
e_nodeid; }
    void setParams(float K, float R, float H) { this->K = K; this->R = R; this->H = H; }

     // Геттери
    int getEdgeId() { return this->EdgeId; }
    int getTAId() { return this->TAId; }
    TreeStatus getTreeStatus() { return treeStatus; }
    QString getTreeStatusString();

    int getStartNodeId() { return this->StartNodeId; }
    int getEndNodeId() { return this->EndNodeId; }

    float getKParam() { return this->K; }
    float getRParam() { return this->R; }
    float getHParam() { return this->H; }

    // Отримати інформацію про гілку
    QString getInfo();

    // Сортування
    bool static edgeLessThanEdgeIds(Edge *e1, Edge *e2);     // по гілкам
    bool static edgeLessThanTAIds(Edge *e1, Edge *e2);       // по деревам/антидеревам
};

enum TreeStatus {
    UNDEFINED,  // дерево/антидерево не задано
    TREE,       // дерево
    ANTITREE    // антидерево
                                                    };


// Граф
class Graph
{
public:
    Graph();
    ~Graph();
```

```
    int getAntiTreeCount();

    // Додати гілку до графа
    bool addEdge(Edge* e);

    // Отримати інформацію про гілки та вузли графу
    QString getEdgeInfos();
    QString getNodeInfos();

    // Отримати інформацію щодо статусу графа
    GraphStatus getStatus() { return graphStatus; }

    // Очистити граф
    void clearGraph();

    // Алгоритми генерації
    bool generateCodeTable(MatrixTable &codeTable);       // Таблиця кодування
    bool generateAMatrix(MatrixTable &AMatrixTable);      // Матриця інциденцій
    bool generateSMatrix(MatrixTable &SMatrixTable);      // Матриця незалежних контурів
    bool generateParams(MatrixTable &k_paramTable,
                        MatrixTable &r_paramTable,
                        MatrixTable &h_paramTable);       // Матриці параметрів та вектори
    bool releaseTree();                                   // Дерева/антидерева

    // Сортування
    void sortByEdgeIds();   // по гілкам
    void sortByTAIds();     // по деревам/антидеревам

private:
    // Визначення наявності в графі конкретної гілки/вузла
    bool isContainEdgeId(int edgeId);
    bool isContainNodeId(int nodeId);

    // Додати вузол до графу
    void addNode(Node* n);

    // Перевірка графу на задовільнення умовам
    GraphStatus checkGraph();

private:
    // Списки гілок та вузлів
    QList<Edge*> edges;
    QList<Node*> nodes;

    // Статус графа
    GraphStatus graphStatus = NONE;

    // Топологічні характеристики графа
    CodeTopology code;
    IncidenceTopology incidence;
    LoopTopology loopTop;
    KParamTopology k_param;
    RParamTopology r_param;
    HParamTopology h_param;
        };


// Таблиця для матриць інциденцій та контурів
struct MatrixTable {
    QStringList horizHeaders;
    QStringList vertHeaders;
    QList<QString> content;
};

// Типи таблиць
enum TableType {
    CODE_TABLE,
    MATRIX_TABLE,
    DIAGONAL_MATRIX_TABLE
};

// Статус графа
enum GraphStatus {
    NONE,    // граф не заданий
    OK,      // граф підходить умовам
    NOK      // граф не підходить умовам
};

// Вузол
```

```
class Node
{
private:
    // Ідентифікатор вузла, список його вхідних
    // та вихідних гілок
    int id = 0;
    QList<int> inEdgeIds = QList<int>();
    QList<int> outEdgeIds = QList<int>();

public:
    Node() {}
    Node(int id) { this->id = id; }

    int getId() { return this->id; }
    const QList<int> &getOutEdgeIds() const { return outEdgeIds; }
    const QList<int> &getInEdgeIds() const { return inEdgeIds; }

    void addInEdgeId(int id);
    void addOutEdgeId(int id);

    bool isContainInEdgeId(int inId);
    bool isContainOutEdgeId(int inId);

    QString getInfo();

    // Сортувати за ідентифікатором
    bool static edgeLessThan(Node *n1, Node *n2) { return n1->getId() < n2->getId(); }
        };


class GraphUnit
{
public:
    // Отримати кількість гілок дерева
    static int getAntiTreeCount(const QList<Edge*> &edges)
    {
        int count = 0;
        for(Edge *e : qAsConst(edges))
            if(e->getTreeStatus() == 2)     //ANTITREE
                count++;
        return count;
    }

    // Отримати об'єкти гілок/вузла по ідентифікатору
    static Edge* getEdgeById(int edgeId, const QList<Edge*> &edges)
    {
        for(Edge* e : qAsConst(edges))
            if(e->getEdgeId() == edgeId)
                return e;

        return NULL;
    }

    static Node* getNodeById(int nodeId, const QList<Node*> &nodes)
    {
        for(Node* n : qAsConst(nodes))
            if(n->getId() == nodeId)
                return n;

        return NULL;
    }
        };


class GraphTopology
{
public:
    void getMatrix(Matrix &Mx);
    void getHeaders(QStringList &h_headers, QStringList &v_headers);

    virtual void generateTopology(const QList<Edge*> &edges, const QList<Node*> &nodes) = 0;

protected:
    Matrix Mx;
    QStringList h_headers;
    QStringList v_headers;
        };


class CodeTopology : public GraphTopology
```

```
{
public:
    void generateTopology(const QList<Edge*> &edges, const QList<Node*> &nodes);
        };


class HParamTopology : public GraphTopology
{
public:
    void generateTopology(const QList<Edge*> &edges, const QList<Node*> &nodes);
        };


class IncidenceTopology : public GraphTopology
{
public:
    void generateTopology(const QList<Edge*> &edges, const QList<Node*> &nodes);
        };


class KParamTopology : public GraphTopology
{
public:
    void generateTopology(const QList<Edge*> &edges, const QList<Node*> &nodes);
        };


class LoopTopology : public GraphTopology
{
public:
    void generateTopology(const QList<Edge*> &edges, const QList<Node*> &nodes);

private:
    void findLoops(int currLoopInd, const QList<Edge*> &edges, const QList<Node *> &nodes);
// пошук контурів
    void shiftLoop(int loopInd, int n);                                 // зсув контуру на n позицій
ліворуч
    void shiftLoops();                                                  // зсув контурів
    void removeDupLoops();                                              // перевірка на те, щоб
контури не повторювались
    void findBaseLoops(const QList<Edge*> &edges);                      // пошук базових yCount
контурів

    bool loopContainEdgeId(int loopId, int edgeId);                     // перевірка на наявність у
вказаному контуру гілок

    int getRowOfValue(const QList<QList<int>> &Mx, int value, int col);       // отримання номеру
рядку за стовпцем та значенням матриці
    int getColOfValue(const QList<QList<int>> &Mx, int value, int row);       // отримання номеру
стовпця за рядком та значенням матриці
    void clearRowAndCol(QList<QList<int>> &Mx, int row, int col);       // заміна всіх елементів
заданого рядку та стовпця нулями
    void countColSum(const QList<QList<int>> &Mx, QList<int> &counts);        // сума елементів в
стовпцю
    void countRowSum(const QList<QList<int>> &Mx, QList<int> &counts);        // сума елементів в
рядку
private:
    // Список контурів графа
    QList<QList<Edge*>> loops;
        };


class RParamTopology : public GraphTopology
{
public:
    void generateTopology(const QList<Edge*> &edges, const QList<Node*> &nodes);
        };
```

Г.2 Вихідні файли

```
String Edge::getTreeStatusString()
{
    QString string = "-X/Y-";   //UNDEFINED
```

```
    if(treeStatus == 1)         // TREE
        string = "X";
    else if(treeStatus == 2)    //ANTITREE
        string = "Y";

    return string;
}

QString Edge::getInfo()
{
    return QString("Q%1:\tU%2 ->
U%3\tK=%4\tR=%5\tH=%6\t%7%8").arg(EdgeId).arg(StartNodeId).arg(EndNodeId).arg(K).arg(R).arg(H)
            .arg(getTreeStatusString()).arg(TAId);
}

bool Edge::edgeLessThanEdgeIds(Edge *e1, Edge *e2)
{
    return e1->getEdgeId() < e2->getEdgeId();
}

bool Edge::edgeLessThanTAIds(Edge *e1, Edge *e2)
{
    /*
     * Якщо гілки мають однаковий статус: обидві дерева або антидерева,
     * сортування відбувається по їх ідентифікатору
     * Якщо перша гілка дерево, а друга - антидерево,
     * меншим з них є дерево. Інакше - антидерево.
    */

    bool result;

    if((e1->getTreeStatus() == 1 && e2->getTreeStatus() == 1) ||
       (e1->getTreeStatus() == 2 && e2->getTreeStatus() == 2))
        result = (e1->getTAId() < e2->getTAId());
    else
        result = (e1->getTreeStatus() == 1 && e2->getTreeStatus() == 2);

    return result;
}


Graph::Graph()
{
    edges = QList<Edge*>();
    nodes = QList<Node*>();
}

Graph::~Graph()
{
    for(Edge* e : qAsConst(edges))
        delete e;

    for(Node* n : qAsConst(nodes))
        delete n;
}

int Graph::getAntiTreeCount()
{
    return GraphUnit::getAntiTreeCount(edges);
}

bool Graph::addEdge(Edge* e)
{
    int res = false;

    // Отримали ідентифікатор гілки
    int currEdgeId = e->getEdgeId();

    // та його початкового та кінцевого вузлів
    int currSNodeId = e->getStartNodeId();
    int currENodeId = e->getEndNodeId();

    Node *currSNode, *currENode;

    // Якщо такого вузла в графі немає - створити новий,
    // інакше - дістати об'єкт з графу
    if(!isContainNodeId(currSNodeId))
        currSNode = new Node(currSNodeId);
    else
        currSNode = GraphUnit::getNodeById(currSNodeId, nodes);
```

```
    if(!isContainNodeId(currENodeId))
        currENode = new Node(currENodeId);
    else
        currENode = GraphUnit::getNodeById(currENodeId, nodes);

    // Якщо такої гілки ще немає в графі
    if(!isContainEdgeId(currEdgeId))
    {
        // Додати гілку в граф
        edges.append(e);

        // Для початкового вузла додати поточну гілку як вихідну,
        // а для кінцевого - як вхідну
        currSNode->addOutEdgeId(currEdgeId);
        currENode->addInEdgeId(currEdgeId);

        addNode(currSNode);
        addNode(currENode);

        // Перевірити граф на відповідність умовам
        graphStatus = checkGraph();

        res = true;
    }

    return res;
}

QString Graph::getEdgeInfos()
{
    QStringList info;
    for(Edge* e : qAsConst(edges))
        info.append(e->getInfo());
    return info.join("\n");
}

QString Graph::getNodeInfos()
{
    QStringList info;
    for(Node* n : qAsConst(nodes))
        info.append(n->getInfo());
    return info.join("\n");
}

void Graph::clearGraph()
{
    graphStatus = NONE;     // Граф не задано

    edges.clear();
    nodes.clear();
}

bool Graph::generateCodeTable(MatrixTable &codeTable)
{
    if(edges.count() == 0)
        return false;

    // Сортувати за гілками
    sortByEdgeIds();

    Matrix codeMatrix;
    code.generateTopology(edges, nodes);
    code.getMatrix(codeMatrix);
    code.getHeaders(codeTable.horizHeaders, codeTable.vertHeaders);

    // Заповнення codeTable
    for(int i = 0; i < codeMatrix.count(); i++)
        codeTable.content << codeMatrix[i];

    return true;
}

bool Graph::generateAMatrix(MatrixTable &AMatrixTable)
{
    if(edges.count() == 0)
        return false;

    // Сортувати за деревами/антидеревами
    sortByTAIds();
```

```
    Matrix A;
    incidence.generateTopology(edges, nodes);
    incidence.getMatrix(A);
    incidence.getHeaders(AMatrixTable.horizHeaders, AMatrixTable.vertHeaders);

    // Заповнення AMatrixTable
    for(int i = 0; i < A.count(); i++)
        AMatrixTable.content << A[i];

    return true;
}

bool Graph::generateSMatrix(MatrixTable &SMatrixTable)
{
    /*
     * По горизонталі - дерево/антидерево
     * По вертикалі - контури
     * Якщо гілка входить в контур позначається 1, інакше 0
    */

    if(edges.count() == 0)
        return false;

    // Сортувати за деревами/антидеревами
    sortByTAIds();

    Matrix S;
    loopTop.generateTopology(edges, nodes);
    loopTop.getMatrix(S);
    loopTop.getHeaders(SMatrixTable.horizHeaders, SMatrixTable.vertHeaders);

    // Заповнення SMatrixTable
    for(int i = 0; i < S.count(); i++)
        SMatrixTable.content << S[i];

    return true;
}

bool Graph::generateParams(MatrixTable &k_paramTable, MatrixTable &r_paramTable, MatrixTable
&h_paramTable)
{
    /*
     * По горизонталі - дерево/антидерево
    */

    if(edges.count() == 0)
        return false;

    // Сортувати за деревами/антидеревами
    sortByTAIds();

    Matrix K;
    k_param.generateTopology(edges, nodes);
    k_param.getMatrix(K);
    k_param.getHeaders(k_paramTable.horizHeaders, k_paramTable.vertHeaders);

    Matrix R;
    r_param.generateTopology(edges, nodes);
    r_param.getMatrix(R);
    r_param.getHeaders(r_paramTable.horizHeaders, r_paramTable.vertHeaders);

    Matrix H;
    h_param.generateTopology(edges, nodes);
    h_param.getMatrix(H);
    h_param.getHeaders(h_paramTable.horizHeaders, h_paramTable.vertHeaders);


    // Заповнення k_paramTable
    for(int i = 0; i < K.count(); i++)
        k_paramTable.content << K[i];

    // Заповнення r_paramTable
    for(int i = 0; i < R.count(); i++)
        r_paramTable.content << R[i];

    // Заповнення h_paramTable
    for(int i = 0; i < H.count(); i++)
        h_paramTable.content << H[i];
```

```
    return true;
}

bool Graph::isContainEdgeId(int edgeId)
{
    int isContain = false;

    for(Edge* e : qAsConst(edges))
    {
        if(e->getEdgeId() == edgeId)
        {
            isContain = true;
            break;
        }
    }

    return isContain;
}

bool Graph::isContainNodeId(int nodeId)
{
    int isContain = false;

    for(Node* n : qAsConst(nodes))
    {
        if(n->getId() == nodeId)
        {
            isContain = true;
            break;
        }
    }

    return isContain;
}

void Graph::addNode(Node* n)
{
    if(!isContainNodeId(n->getId()))
        nodes.append(n);
}

GraphStatus Graph::checkGraph()
{
    for(Node *n : qAsConst(nodes))
        if(n->getOutEdgeIds().count() == 0 || n->getInEdgeIds().count() == 0)
            return NOK;

    return OK;
}

void Graph::sortByTAIds()
{
    std::sort(edges.begin(), edges.end(), Edge::edgeLessThanTAIds);
    std::sort(nodes.begin(), nodes.end(), Node::edgeLessThan);
}

void Graph::sortByEdgeIds()
{
    std::sort(edges.begin(), edges.end(), Edge::edgeLessThanEdgeIds);
    std::sort(nodes.begin(), nodes.end(), Node::edgeLessThan);
}

bool Graph::releaseTree()
{
    /*
     * Для позначення дерева/антидерева, для першого записаного в масиві вузла
     * позначаємо всі вхідні гілки за антиерева - робимо перший вузол корнем дерева.
     * Для решти вузлів якщо немає вхідної гілки, яка вже позначена за
     * дерево, тоді позначаємо першу з них за дерево, а останні за антидерево.
     * Якщо у вузла є гілка, яка позначена за дерево, тоді останні позначаємо за
     * антидерево.
     * В кінці призначаємо деревам та антидеревам свої ідентифікатори
    */

    if(nodes.count() == 0)
        return false;

    // Для першого вузла, позначимо всі його вхідні гілки за антидерева
    for(int inId : nodes[0]->getInEdgeIds())
    {
```

```
        Edge *e = GraphUnit::getEdgeById(inId, edges);
        e->setTreeStatus(ANTITREE);
    }

    for(int i = 1; i < nodes.count(); i++)
    {
        bool containTree = false;

        for(int inId : nodes[i]->getInEdgeIds())
        {
            Edge *e = GraphUnit::getEdgeById(inId, edges);

            //Якщо вхідна гілка - дерево
            if(e->getTreeStatus() == 1)          //TREE
                containTree = true;
            else if(e->getTreeStatus() == 0)     //UNDEFINED
                e->setTreeStatus(ANTITREE);
        }

        if(!containTree)
            GraphUnit::getEdgeById(nodes[i]->getInEdgeIds()[0], edges)->setTreeStatus(TREE);
    }

    int treeIndex = 1;
    int antiIndex = 1;

    for(Edge *e : qAsConst(edges))
    {
        if(e->getTreeStatus() == 1)
            e->setTreeId(treeIndex++);
        else
            e->setTreeId(antiIndex++);
    }

    return true;
        }


void Node::addInEdgeId(int id)
{
    if(!isContainInEdgeId(id))
        inEdgeIds.append(id);
}

void Node::addOutEdgeId(int id)
{
    if(!isContainOutEdgeId(id))
        outEdgeIds.append(id);
}

bool Node::isContainInEdgeId(int inId)
{
    bool isContain = false;

    for(int id : qAsConst(inEdgeIds))
    {
        if(id == inId)
        {
            isContain = true;
            break;
        }
    }

    return isContain;
}

bool Node::isContainOutEdgeId(int inId)
{
    bool isContain = false;

    for(int id : qAsConst(outEdgeIds))
    {
        if(id == inId)
        {
            isContain = true;
            break;
        }
    }

    return isContain;
```

```
}

QString Node::getInfo()
{
    QString info;

    QStringList inEdges, outEdges;

    for(int id : qAsConst(inEdgeIds))
        inEdges.append(QString("Q%1").arg(id));

    for(int id : qAsConst(outEdgeIds))
        outEdges.append(QString("Q%1").arg(id));

    info += QString("Вхідні гілки: %1\n").arg(inEdges.join(", "));
    info += QString("Вихідні гілки: %1").arg(outEdges.join(", "));

    return info;
       }


void CodeTopology::generateTopology(const QList<Edge *> &edges, const QList<Node *> &nodes)
{
    Mx.clear();
    v_headers.clear();
    h_headers.clear();

    h_headers << "Qj" << "AKi" << "EKi" << "Xj/Yj" << "Kj" << "Rj" << "Hj";

    for(int i = 0; i < edges.count(); i++)
    {
        v_headers << QString("%1").arg(i + 1);

        Mx.append(QList<QString>());

        Mx[i].append(QString("Q%1").arg(edges[i]->getEdgeId()));
        Mx[i].append(QString("U%1").arg(edges[i]->getStartNodeId()));
        Mx[i].append(QString("U%1").arg(edges[i]->getEndNodeId()));
        Mx[i].append(QString("%1%2").arg(edges[i]->getTreeStatusString()).arg(edges[i]-
>getTAId()));
        Mx[i].append(QString("%1").arg(edges[i]->getKParam()));
        Mx[i].append(QString("%1").arg(edges[i]->getRParam()));
        Mx[i].append(QString("%1").arg(edges[i]->getHParam()));
    }
       }


void GraphTopology::getMatrix(Matrix &Mx)
{
    Mx = Matrix(this->Mx);
}

void GraphTopology::getHeaders(QStringList &h_headers, QStringList &v_headers)
{
    h_headers = QStringList(this->h_headers);
    v_headers = QStringList(this->v_headers);
       }


void HParamTopology::generateTopology(const QList<Edge *> &edges, const QList<Node *> &nodes)
{
    Mx.clear();
    v_headers.clear();
    h_headers.clear();

    v_headers << "1";

    Mx.append(QList<QString>());

    for(int j = 0; j < edges.count(); j++)
    {
        h_headers << QString("%1%2").arg(edges[j]->getTreeStatusString()).arg(edges[j]-
>getTAId());

        Mx[0].append(QString("%1").arg(edges[j]->getHParam()));
    }
       }


void IncidenceTopology::generateTopology(const QList<Edge *> &edges, const QList<Node *> &nodes)
```

```
{
    /*
     * По горизонталі - дерево/антидерево
     * По вертикалі - вузли
     * Якщо гілка входить в вузол позначається 1, інакше -1
     * Якщо гілка ніяк не зв'язана з вузлом 0
    */

    Mx.clear();
    v_headers.clear();
    h_headers.clear();

    for(int i = 0; i < nodes.count(); i++)
    {
        Mx.append(QList<QString>());
        v_headers << QString("U%1").arg(nodes[i]->getId());

        for(int j = 0; j < edges.count(); j++)
        {
            if(i == 0)
                h_headers << QString("%1%2").arg(edges[j]->getTreeStatusString()).arg(edges[j]-
>getTAId());

            if(nodes[i]->isContainInEdgeId(edges[j]->getEdgeId()))
                Mx[i].append("1");
            else if(nodes[i]->isContainOutEdgeId(edges[j]->getEdgeId()))
                Mx[i].append("-1");
            else
                Mx[i].append("0");
        }
    }
       }


void KParamTopology::generateTopology(const QList<Edge *> &edges, const QList<Node *> &nodes)
{
    Mx.clear();
    v_headers.clear();
    h_headers.clear();

    v_headers << "1";

    Mx.append(QList<QString>());

    for(int j = 0; j < edges.count(); j++)
    {
        h_headers << QString("%1%2").arg(edges[j]->getTreeStatusString()).arg(edges[j]-
>getTAId());

        Mx[0].append(QString("%1").arg(edges[j]->getKParam()));
    }
       }


void LoopTopology::generateTopology(const QList<Edge *> &edges, const QList<Node *> &nodes)
{
    Mx.clear();
    v_headers.clear();
    h_headers.clear();

    loops.clear();

    // Пошук контурів та перевірка на його неповторення в графі
    findLoops(0, edges, nodes);

    shiftLoops();

    removeDupLoops();

    if(!(GraphUnit::getAntiTreeCount(edges) == loops.count()))
        findBaseLoops(edges);

    for(int i = 0; i < loops.count(); i++)
    {
        Mx.append(QList<QString>());
        v_headers << QString("K%1").arg(i + 1);

        for(int j = 0; j < edges.count(); j++)
        {
            if(i == 0)
```

```
                h_headers << QString("%1%2").arg(edges[j]->getTreeStatusString()).arg(edges[j]-
>getTAId());

            if(loopContainEdgeId(i, edges[j]->getEdgeId()))
                Mx[i].append("1");
            else
                Mx[i].append("0");
        }
    }
}

void LoopTopology::findLoops(int currLoopInd, const QList<Edge *> &edges, const QList<Node *>
&nodes)
{
    // Якщо масив не порожній
    if(loops.count() != 0)
    {
        // Якщо поточний індекс контура більше ніж кількість контурів
        if (currLoopInd >= loops.count())
            return;

        int countEdges = loops[currLoopInd].count();
        int startNodeId = loops[currLoopInd][0]->getStartNodeId();
        int lastNodeId = loops[currLoopInd][countEdges - 1]->getEndNodeId();

        // Якщо кінцевий та початковий номери вузлів однакові - контур знайдено
        // Переходимо до наступного контуру та завершуємо з поточним роботу
        if(startNodeId == lastNodeId)
        {
            findLoops(currLoopInd + 1, edges, nodes);
            return;
        }

        // Останній вузол поточного контуру та його кількість вихідних гілок
        Node *n = GraphUnit::getNodeById(lastNodeId, nodes);
        int outCount = n->getOutEdgeIds().count();

        // Порівнюємо всі вузли контуру з останнім
        for (int i = 0; i < countEdges - 1; i++)
        {
            // Якщо є повторення вузлів окрім першого та останнього вузла контуру
            if (loops[currLoopInd][i]->getStartNodeId() == n->getId())
            {
                // Тоді шукаємо в контурі гілку, з якої відбувалось повторення
                for (int j = 0; j < countEdges; j++)
                {
                    if (loops[currLoopInd][j]->getEndNodeId() == n->getId())
                    {
                        // Видаляємо з поточного контуру всі останні гілки до повторення
                        // Отримаємо новий контур
                        for (int k = 0; k < j + 1; k++)
                            loops[currLoopInd].pop_front();

                        // Переходимо до наступного контуру
                        // Завершення роботи з поточним контуром
                        findLoops(currLoopInd + 1, edges, nodes);
                        return;
                    }
                }
            }
        }

        // Внутрішніх повторень гілок немає
        // Робимо копії поточної гілки на 1 менше ніж кількість її вихідних гілок
        for(int x = 0; x < outCount - 1; x++)
        {
            // Копіювання поточного контуру
            // Додавання до скопійованого контуру вихідної гілки
            loops.append(loops[currLoopInd]);
            loops[loops.count() - 1].append(GraphUnit::getEdgeById(n->getOutEdgeIds()[x+1],
edges));
        }

        //Додавання до поточного контуру першу з вихідних гілок поточного вузла
        //Продовження пошуку поточного контуру
        loops[currLoopInd].append(GraphUnit::getEdgeById(n->getOutEdgeIds()[0], edges));
        findLoops(currLoopInd, edges, nodes);

    }
    else    //Якщо масив порожній
```

```
    {
        Node *firstNode = nodes[0];
        int outCount = firstNode->getOutEdgeIds().count();

        //Додаємо стільки контурів, скільки вихідних гілок з першого вузла
        for(int x = 0; x < outCount; x++)
        {
            QList<Edge*> l;
            l.append(GraphUnit::getEdgeById(firstNode->getOutEdgeIds()[x], edges));
            loops.append(l);
        }

        //Починаємо пошук з нульового (першого) контуру
        findLoops(0, edges, nodes);
    }
}

void LoopTopology::removeDupLoops()
{
    /*
     * Порівняння контурів кожний з кожним, якщо було знайдено повторення -
     * помічаємо цей контур на видалення
     * В кінці, коли всі порівняння були зроблені,
     * видаляємо контури, номери яких розміщені в масиві delLoopIds
    */

    int i = 0;
    while(i < loops.count())
    {
        int j = i + 1;
        while(j < loops.count())
        {
            if(loops[i].count() != loops[j].count())
            {
                j++;
                continue;
            }

            bool equal = true;
            for(int k = 0; k < loops[i].count(); k++)
            {
                if(loops[i][k]->getEdgeId() != loops[j][k]->getEdgeId())
                {
                    equal = false;
                    break;
                }
            }

            if(equal)
                loops.removeAt(j);
            else
                j++;
        }
        i++;
    }
}

void LoopTopology::findBaseLoops(const QList<Edge *> &edges)
{
    // Об'єкти антидерев
    QList<Edge*> unbased;
    for(Edge *e : qAsConst(edges))
        if(e->getTreeStatus() == 2)      // Антидерево
            unbased.append(e);

    int yCount = unbased.count();

    // Матриця Sy
    QList<QList<int>> Sy;
    for(int i = 0; i < loops.count(); i++)
    {
        Sy.append(QList<int>());
        for(int j = 0; j < yCount; j++)
            Sy[i].append(loopContainEdgeId(i, unbased[j]->getEdgeId()) ? 1 : 0);
    }

    // Контури, які треба зберегти
    QList<QList<Edge*>> saveLoops;

    // Суми стовпців та рядків
```

```
    QList<int> colSum, rowSum;
    countColSum(Sy, colSum);
    countRowSum(Sy, rowSum);

    while(saveLoops.count() != yCount)
    {
        int minColSum = *std::min_element(colSum.begin(), colSum.end());
        int minRowSum = *std::min_element(rowSum.begin(), rowSum.end());
        int minSum = (minColSum < minRowSum) ? minColSum : minRowSum;

        for(int i = 0; i < colSum.count(); i++)
        {
            if(colSum[i] == minSum)
            {
                int loopInd = getRowOfValue(Sy, 1, i);
                saveLoops.append(loops[loopInd]);

                clearRowAndCol(Sy, loopInd, i);
                countColSum(Sy, colSum);
                countRowSum(Sy, rowSum);
            }
        }

        for(int i = 0; i < rowSum.count(); i++)
        {
            if(rowSum[i] == minSum)
            {
                int colInd = getColOfValue(Sy, 1, i);
                saveLoops.append(loops[i]);

                clearRowAndCol(Sy, i, colInd);
                countColSum(Sy, colSum);
                countRowSum(Sy, rowSum);
            }
        }
    }

    loops = QList<QList<Edge*>>(saveLoops);
}

void LoopTopology::shiftLoop(int loopInd, int n)
{
    std::rotate(loops[loopInd].begin(), loops[loopInd].begin() + n, loops[loopInd].end());
}

void LoopTopology::shiftLoops()
{
    /*
     * Для порівняння контурів, знаходимо його мінімальний елемент та
     * зміщуємо ліворуч на стільки позицій, щоб він став на перше місце
    */

    for(int i = 0; i < loops.count(); i++)
    {
        int minEdgeId = loops[i][0]->getEdgeId();
        int n = 0;
        for(int j = 1; j < loops[i].count(); j++)
        {
            if(loops[i][j]->getEdgeId() < minEdgeId)
            {
                minEdgeId = loops[i][j]->getEdgeId();
                n = j;
            }
        }
        shiftLoop(i, n);
    }
}

bool LoopTopology::loopContainEdgeId(int loopId, int edgeId)
{
    bool isContain = false;

    for(int i = 0; i < loops[loopId].count(); i++)
    {
        if(loops[loopId][i]->getEdgeId() == edgeId)
        {
            isContain = true;
            break;
        }
    }
```

```
    return isContain;
}

int LoopTopology::getRowOfValue(const QList<QList<int>> &Mx, int value, int col)
{
    for(int i = 0; i < Mx.count(); i++)
        if(Mx[i][col] == value)
            return i;
    return -1;
}

int LoopTopology::getColOfValue(const QList<QList<int>> &Mx, int value, int row)
{
    for(int j = 0; j < Mx[row].count(); j++)
        if(Mx[row][j] == value)
            return j;
    return -1;
}

void LoopTopology::clearRowAndCol(QList<QList<int>> &Mx, int row, int col)
{
    for(int i = 0; i < Mx.count(); i++)
        for(int j = 0; j < Mx[i].count(); j++)
            if(i == row || j == col)
                Mx[i][j] = 0;
}

void LoopTopology::countColSum(const QList<QList<int>> &Mx, QList<int> &counts)
{
    counts.clear();
    // Прохід по матриці по стовпцям
    for(int i = 0; i < Mx[0].count(); i++)
    {
        int currCount = 0;
        for(int j = 0; j < Mx.count(); j++)
            currCount += Mx[j][i];

        // Позначаємо максимальним неможливим числом
        // для прибрання мінімального нуля
        if(currCount == 0)
            currCount = Mx.count() + 1;

        counts.append(currCount);
    }
}

void LoopTopology::countRowSum(const QList<QList<int> > &Mx, QList<int> &counts)
{
    counts.clear();
    // Прохід по матриці по рядкам
    for(int i = 0; i < Mx.count(); i++)
    {
        int currCount = 0;
        for(int j = 0; j < Mx[i].count(); j++)
            currCount += Mx[i][j];

        // Позначаємо максимальним неможливим числом
        // для прибрання мінімального нуля
        if(currCount == 0)
            currCount = Mx[0].count() + 1;

        counts.append(currCount);
    }
       }


void RParamTopology::generateTopology(const QList<Edge *> &edges, const QList<Node *> &nodes)
{
    Mx.clear();
    v_headers.clear();
    h_headers.clear();

    v_headers << "1";

    Mx.append(QList<QString>());

    for(int j = 0; j < edges.count(); j++)
    {
```

```
        h_headers << QString("%1%2").arg(edges[j]->getTreeStatusString()).arg(edges[j]-
>getTAId());

        Mx[0].append(QString("%1").arg(edges[j]->getRParam()));
    }
       }
```

ДОДАТОК Д – Перелік зауважень нормоконтролера

Сторінка	Позначення	Зміст зауваження
11-12	перелік	Переліки оформлені не за вимогами
23	таблиця	Оформлення таблиці не відповідає вимогам