Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

інформаційних технологій
(факультет)
Кафедра системного аналізу та управління
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеню магістра

студента Притули Миколи Ігоровича
(ПІБ)
академічної групи 124м–22
(шифр)
спеціальності 124 Системний аналіз
(код і назва спеціальності)
спеціалізації[1] _____
за освітньо-професійною програмою Системний аналіз
(офіційна назва)
на тему «Використання лінгвістичного представлення детермінованих графів у тестуванні програмного забезпечення»

(назва за наказом ректора)

Керівники	Прізвище, ініціали	Оцінка за шкалою		Підпис
		рейтинговою	інституційною	
кваліфікаційної роботи	Д.т.н., проф. Слєсарєв В. В			
розділів:	2			
Інформаційно-аналітичний розділ	Д.т.н., проф. Слєсарєв В. В			
Спеціальний розділ	Д.т.н., проф. Слєсарєв В. В			

Рецензент				

Нормоконтролер	к.ф.-м.н., доц. Хом'як Т.В			

Дніпро
2023

ЗАТВЕРДЖЕНО:
завідувач кафедри
системного аналізу та управління
(повна назва)
_____ к.т.н., доц. Желдак Т.А.
(підпис) (прізвище, ініціали)

«_____»_____ 2023 року

ЗАВДАННЯ
на кваліфікаційну роботу
ступеня магістра

студенту Притулі М.І. **академічної групи** 124м–22
(прізвище та ініціали) (шифр)

Спеціальності 124 Системний аналіз

на тему «Використання лінгвістичного представлення детермінованих графів у тестуванні програмного забезпечення»

затверджену наказом ректора НТУ «Дніпровська політехніка» від 09.10.2023 р. №1227-с

Розділ	Зміст	Термін виконання
1. Інформаційно-аналітичний	Огляд теорії представлення математичних об'єктів. Детерміновані та сильно-детерміновані графи. Представлення детермінованих та сильно-детермінованих графів визначальною парою слів. Канонічна визначальна пара для детермінованих графів	
2. Спеціальний	Програмне моделювання алгоритму побудови графа, для якого задана пара слів є визначальною та алгоритму побудови канонічної визначальної пари для заданого детермінованого графа. Застосування розроблених алгоритмів у тестуванні програмного забезпечення на прикладі побудови схеми тестового покриття прототипу web-додатку. Застосування розроблених алгоритмів для зручного зображення карти логістичних маршрутів з метою подальшого їх тестування.	

Завдання видано _____ Слєсарєв В.В.
(підпис керівника) (прізвище, ініціали)

Дата видачі _____

Дата подання до екзаменаційної комісії _____

Прийнято до виконання _____ Притула М. І.
(підпис студента) (прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 66 с., 5 додатків, 29 джерел, 19 рис.

Актуальність обраної теми обумовлена важливістю розвитку різноманітних методів тестування програмного забезпечення та можливістю використання лінгвістичного представлення детермінованих графів для ефективної побудови та оцінки тестового покриття.

Об'єкт досліджень – абстрактна модель тестового покриття програмного забезпечення, яка може бути подана у вигляді детермінованого графа.

Предмет досліджень – методика побудови та оцінки тестового покриття програмного забезпечення із використанням лінгвістичного представлення детермінованих графів.

Мета роботи – дослідження можливості використання лінгвістичного представлення детермінованих графів визначальною парою у тестуванні програмного забезпечення.

Методи дослідження: методи системного аналізу - експериментальне моделювання, тестування - валідація та верифікація моделей, аналогія (знаходження спільних властивостей у структурі детермінованого графа та автомата, які можуть дозволити розповсюдити результати, що знайдені для автоматів на детерміновані графи). Тип методології дослідницької теми – класична.

Поставлені задачі:

1. Розробка, моделювання, тестування, валідація та верифікація алгоритмів, пов'язаних із лінгвістичним представленням детермінованих графів визначальною парою.
2. Розробка методики застосування представлення детермінованих графів визначальною парою для тестування програмного забезпечення, зокрема для уточнення та оцінки тестового покриття.

В інформаційно–аналітичному розділі наведено коротку історію теорії представлення різноманітних математичних об'єктів та структур. Також

наведено основні поняття, що пов’язані з детермінованими графами, та їх використанням на практиці. Крім того наведено інформацію про сучасний стан теорії лінгвістичного представлення детермінованих графів.

У спеціальному розділі наведено програмну реалізацію алгоритму побудови графа, для якого задана пара слів є визначальною та алгоритму побудови канонічної визначальної пари для заданого детермінованого графа. Подано приклад поширення методики побудови тестового покриття програмного забезпечення із застосуванням розроблених алгоритмів на прикладі аналізу прототипу web-додатку.

Наукова новизна отриманих результатів полягає в розробці теорії лінгвістичного представлення детермінованих графів визначальною парою слів в алфавіті їх міток.

Основні конструктивні, технологічні або техніко-експлуатаційні характеристики: модель побудована з використанням мови програмування Python та бібліотеки для дослідження мереж NetworkX. Пари слів для побудови графів задаються стандартними типами даних мови Python - Tuple(str). Вихідна модель графа представлена стандартним поданням файлу з розширенням .dot який може бути візуалізованим графічним модулем graphviz.

Галузь застосування: запропонована методика може бути використана у тестуванні програмного забезпечення, зокрема при уточненні та оцінці побудови тестового покриття.

Економічна ефективність: розроблена теорія а також програмний код є у відкритому доступі, та може розповсюджуватися за умовами ліцензії GNU GPL. Розроблена теорія може використовуватись, наприклад, у процесах тестування моделей, та зменшити витрати коштів на моделювання тестового покриття системи, шляхом автоматичної побудови графа за певними вимогами системи.

Значення роботи для науки і практики: результати, які одержані в роботі, можуть бути застосованими для розв’язання прикладних задач ідентифікації інформаційного середовища.

Прогнозні припущення про розвиток об’єкта дослідження: запропоноване представлення детермінованих графів визначальною парою слів в алфавіті їх міток може бути корисним при розв'язанні багатьох прикладних задач, у тому числі для побудови моделей тестового покриття програмного забезпечення. Проте специфіка тестування програмного забезпечення потребує також використання орієнтованих детермінованих графів, для яких теорія лінгвістичного представлення на даний час ще не розроблена. Тому вважаємо, що корисно було б розповсюдження такого представлення на орієнтовані детерміновані графи.

Ключові слова: Детерміновані графи, визначальна пара слів, програмне моделювання алгоритмів, тестування програмного забезпечення, тестове покриття.

ABSTRACT

Explanatory note: 66 pages, 5 applications, 29 sources, 19 pictures.

The relevance of the chosen topic is due to the importance of developing various software testing methods and the possibility of using the linguistic presentation of deterministic graphs for effective construction and evaluation of test coverage.

The object of research is an abstract model of software test coverage that can be represented as a deterministic graph.

The subject of research is a methodology for building and evaluating software test coverage using the linguistic presentation of deterministic graphs.

The purpose of the work is to investigate the possibility of using the linguistic presentation of deterministic graphs by a deterministic pair in software testing.

Tasks:

1. Development, modeling, testing, validation, and verification of algorithms related to the linguistic presentation of deterministic graphs by a deterministic pair.
2. Development of a methodology for using the presentation of deterministic graphs by a deterministic pair for software testing, in particular, for refining and evaluating test coverage.

The information and analytical section provides a brief history of the theory of presentation of various mathematical objects and structures. The basic concepts related to deterministic graphs and their practical application are also presented. In addition, information on the current state of linguistic presentation of deterministic graphs is presented.

The special section presents a software implementation of the algorithm for constructing a graph for which a given pair of words is a defining pair and the algorithm for constructing a canonical defining pair for a given deterministic graph. An example of the dissemination of the methodology for building test coverage of

software using the developed algorithms is given on the example of analyzing a web application prototype.

The scientific novelty of the results is the development of a theory of linguistic presentation of deterministic graphs by a defining pair of words in the alphabet of their labels.

Main design, technological or technical and operational characteristics: ehe model was built using the Python programming language and the NetworkX network research library. Word pairs for graph construction are specified by standard Python data types - Tuple(str). The output graph model is represented by a standard file representation with the .dot extension that can be visualized by the graphviz graphical module.

Scope: the proposed methodology can be used in software testing, in particular, in refining and evaluating the construction of test coverage models.

Cost-effectiveness: the developed theory and program code are in the public domain and can be distributed under the terms of the GNU GPL license. The developed theory can be used, for example, in model testing processes, and reduce the cost of modeling the test coverage of the system by automatically building a graph according to certain system requirements.

Significance of the work for science and practice: the theoretical results obtained in this work can be used to solve applied problems of information environment identification.

Forecast assumptions about the development of the object of study: the proposed presentation of deterministic graphs by a defining pair of words in the alphabet of their labels can be useful in solving many applied problems, including building models of software test coverage. However, the specifics of software testing also require the use of oriented deterministic graphs, for which the theory of linguistic presentation has not yet been developed. Therefore, we believe that it would be useful to extend such a presentation to oriented deterministic graphs.

Keywords: Deterministic graphs, defining pair of words, software algorithms modeling, software testing, test coverage.

ЗМІСТ

ВСТУП

В наш час світ інформаційних технологій широко розповсюдився майже у всі сфери людської діяльності. Використання надійного та безпечного програмного забезпечення є необхідним для успішного функціонування багатьох організацій і підприємств. Наразі спостерігається постійне зростання обсягів інформації, що підвищує вимоги для програмного забезпечення, яке необхідне для успішної її обробки, зберігання, перетворення тощо. Тому підвищення якості сучасного програмного забезпечення відіграє дуже важливу роль у нашому житті. Одним із напрямків такого підвищення є тестування програмного забезпечення, яке являє собою важливий аспект у розробці та підтримці програм. Шляхом систематичної перевірки та виявлення дефектів можна гарантувати, що програма працює стабільно і без збоїв. Це особливо важливо в критичних сферах, де навіть невелика помилка може призвести до серйозних наслідків. Додатково, тестування знижує ризики та економить ресурси. Раннє виявлення та усунення помилок у процесі розробки набагато ефективніше й дешевше, ніж внесення змін після випуску програми в експлуатацію. З розвитком технологій автоматизації тестування стало можливим проводити велику кількість тестів у найкоротші терміни. Це дає змогу прискорити процес розробки програмного забезпечення та знизити ризики помилок, які пов'язані з людським фактором.

Нажаль, наразі не одержали загального визнання та повсюдного використання існуючі механізми побудови моделей тестового покриття для подальшого створення автоматичних тестів програмного забезпечення та оцінки їх ефективності, якості та повноти автотестів. Тому використання новітніх підходів до створення таких механізмів є дуже важливою задачею, яка може суттєво підвищити якість програмного забезпечення. Тому вважаємо, що наша спроба поширити методологію розробки тестового покриття для подальшої оцінки його повноти з метою побудови архітектурного рішення автоматизованих тестів є актуальною задачею.

Одним із підходів до такого поширення є використання графів в якості інструмента подання концептуальної моделі структури тестового покриття. Серед різноманітних класів графів на нашу думку для цього доцільно використовувати так звані детерміновані графи - графи з розміченими вершинами, у яких в околі кожної вершини всі вершини мають попарно різні мітки. При цьому лінгвістичне представлення детермінованих графів є зручним до використання у багатьох задачах тестування, зокрема, подання моделі тестового покриття, генерації тестових даних, зображення програмних структур тощо. Наразі теорія лінгвістичного представлення таких графів інтенсивно розвивається у відділі теорії керуючих систем Інституту прикладної математики і механіки НАН України. Отже, підсумовуючи вищесказане, вважаємо, що спроба використання лінгвістичного представлення детермінованих графів у тестуванні програмного забезпечення є актуальною задачею.

Метою роботи є дослідження можливості використання лінгвістичного представлення детермінованих графів визначальною парою у тестуванні програмного забезпечення.

З огляду на мету в роботі ставляться такі *завдання*:

1) Розробка, моделювання, тестування, валідація та верифікація алгоритмів, пов’язаних із лінгвістичним представленням детермінованих графів визначальною парою.

2) Розробка методики застосування представлення детермінованих графів визначальною парою для тестування програмного забезпечення, зокрема для уточнення та оцінки тестового покриття.

Предметом є методика побудови та оцінки тестового покриття програмного забезпечення із використанням лінгвістичного представлення детермінованих графів.

Об’єктом абстрактна модель тестового покриття програмного забезпечення, яка може бути подана у вигляді детермінованого графа.

У роботі використовувались різноманітні *методи* дослідження, зокрема, методи системного аналізу – експериментальне моделювання, тестування – валідація та верифікація моделей, аналогія (знаходження спільних властивостей у структурі детермінованого графа та автомата, які можуть дозволити розповсюдити результати, що знайдені для автоматів на детерміновані графи). Тип методології дослідницької теми – класична.

Наукова новизна отриманих результатів полягає в розробці теорії лінгвістичного представлення детермінованих графів визначальною парою слів в алфавіті їх міток.

Основні конструктивні, технологічні або техніко-експлуатаційні характеристики: Модель побудована з використанням мови програмування Python та бібліотеки для дослідження мереж NetworkX. Пари слів для побудови графів задаються стандартними типами даних мови Python - Tuple(str). Вихідна модель графа представлена стандартним поданням файлу з розширенням .dot який може бути візуалізованим графічним модулем graphviz.

Галузь застосування: Запропонована методика може бути використана у тестуванні програмного забезпечення, зокрема при уточненні та оцінці побудови тестового покриття.

Економічна ефективність: Розроблена теорія а також програмний код є у відкритому доступі, та може розповсюджуватися за умовами ліцензії GNU GPL. Розроблена теорія може використовуватись, наприклад, у процесах тестування моделей, та зменшити витрати коштів на моделювання тестового покриття системи, шляхом автоматичної побудови графа за певними вимогами системи.

Значення роботи для науки і практики: Теоретичні результати, які одержані в роботі, можуть бути застосованими для розв’язання прикладних задач ідентифікації інформаційного середовища.

Прогнозні припущення про розвиток об’єкта дослідження: Запропоноване представлення детермінованих графів визначальною парою

слів в алфавіті їх міток може бути корисним при розв'язанні багатьох прикладних задач, у тому числі для побудови моделей тестового покриття програмного забезпечення. Проте специфіка тестування програмного забезпечення потребує також використання орієнтованих детермінованих графів, для яких теорія лінгвістичного представлення на даний час ще не розроблена. Тому вважаємо, що корисно було б розповсюдження такого представлення на орієнтовані детерміновані графи.

1 ІНФОРМАЦІЙНО-АНАЛІТИЧНИЙ РОЗДІЛ

1.1 Історія виникнення теорії представлення. Представлення груп, напівгруп та скінченних автоматів

Однією з центральних проблем у сучасній математиці є скінченне завдання як скінченних, так і нескінченних об’єктів. Завдання групи за допомогою твірних елементів та визначальних співвідношень вперше було зроблено у 1882 році В. фон Діком [1]. Цей метод опису групи полягає у виокремленні деяких елементів групи (твірних) та наведенні співвідношень між твірними елементами. Таке завдання групи також називають її представленням або генетичним кодом.

Наведемо короткий опис цього методу. Нехай підмножина S групи G породжує її, тобто кожний елемент групи може бути записаний словом в алфавіті з елементів з S і зворотних до них. За такого кодування конкатенація слів відповідає множенню елементів групи, а отже, теоретично вся групова структура задається інформацією про те, які пари таких слів представляють один і той самий елемент групи G. Такі пари називаються визначальними співвідношеннями. Деякі співвідношення можна вивести з інших, наприклад, якщо $ab = ba$ і $b = c$, то $ac = ca$. Метод завдання групи твірними та визначальними співвідношеннями полягає в тому, щоб вказати (за можливістю невеликий) список R визначальних співвідношень, якого, з урахуванням заздалегідь обумовлених правил виведення, вистачить для зберігання повної інформації про групу. У цьому разі пишуть $G \cong <S|R>$.

Цей метод опису груп ефективніший, ніж, наприклад, таблиці Келі. Так, використання таблиць Келі неможливе для нескінченних груп і недоцільне навіть для скінченних груп великого порядку. Наприклад, таблиця Келі циклічної групи порядку n складається з n^2 елементів, але ця група допускає цілком коротке завдання: $<a \mid a^n = 1>$, яке означає, що будь-який її елемент

можна записати як ступінь елемента a, і при цьому n – найменший такий ступінь, що a^n є нейтральним елементом.

Із таким способом завдання груп та напівгруп пов’язані фундаментальні проблеми, які були сформульовані М. Деном для груп у 1912 році та А. Туе для напівгруп у 1914 році [2, 3]. Сформулюємо дві з цих проблем у формулюваннях А. Туе (тобто, для напівгруп).

1) *Проблема рівності слів*. Нехай напівгрупа S задана системою визначальних співвідношень R. Кожному визначальному співвідношенню $a = b$ поставимо у відповідність елементарні перетворення: $uav \to ubv$ та $ubv \to uav$ для довільних елементів u та v. Тут позначення $p \to q$ означає, що слово q одержується зі слова p за допомогою одного елементарного перетворення. Два слова m та n називаються еквівалентними в S, якщо або $m = n$, або існує ланцюг елементарних перетворень вигляду $m = m_0 \to m_1 \to \ldots \to m_k = n$. Треба знайти алгоритм, який дозволяє для довільної напівгрупи S та довільних двох її елементів (слів) визначити, чи є вони еквівалентними.

2) Проблема ізоморфізму. Нехай напівгрупа S_1 задана системою визначальних співвідношень R_1, а напівгрупа S_2 – системою визначальних співвідношень R_2. Треба знайти алгоритм, який за R_1 та R_2 визначає, чи є ізоморфними S_1 та S_2.

Під алгоритмічною проблемою розуміють задачу побудови єдиного алгоритму для розв’язання заданої масової задачі, тобто нескінченної серії однотипних питань, що залежать від деяких параметрів. У разі, коли шуканий алгоритм неможливий, кажуть, що ця алгоритмічна проблема нерозв’язувана. Це означає, що не можна вказати єдиний метод розв’язання всіх задач цієї серії. Можливість розв'язання кожної з них своїм шляхом при цьому не виключається. У 1947 році незалежно А.А.Марков і Е.Пост [3] довели нерозв’язність проблеми рівності слів для напівгруп, а в 1955 році незалежно П.С.Новіков і У.Бун [1] довели нерозв'язність проблеми рівності слів для груп.

Найпростіші (за загальною довжиною) приклади напівгруп з нерозв'язною проблемою рівності слів були побудовані в 1956 році

Г.С.Цейтіним і Д. Скоттом [2]. Обидва ці завдання містили 7 визначальних співвідношень. Проблема ізоморфізму є найважчою з проблем Туе і Дена [4]. А.А.Марковим у 1951 році показано, що проблема ізоморфізму для напівгруп у загальному випадку є нерозв'язною [5].

Згадані вище результати призвели до створення потужної комбінаторної теорії груп та інших одноосновних алгебр, що інтенсивно розвивається.

У 1961 році Ю.І. Соркін у роботі [6] сформулював проблеми, аналогічні проблемам Туе і Дена для автоматів як для багатоосновних алгебр. У цій роботі розглядаються детерміновані, всюдивизначені в загальному випадку не ініціальні автомати без виходу як двохосновні алгебри. Системою твірних M часткового автомата $A = (A, X, \delta_A, A')$ називають будь-яку таку підмножину $M \subseteq A$, що для кожного стану $a \in A$ виконується або $a \in M$, або $mx_1 \dots x_k = a$ для деяких $m \in M$ та $x_1, \dots, x_k \in X$. Слова в автоматі $A = (A, X, \delta_A, M)$ задаються у вигляді a (слово рангу 0) або $ax_1 \dots x_k$ (слово рангу k), де $a \in M$ та $x_1, \dots, x_k \in X, k > 0$. Ранг слова p позначається через $r(p)$. Визначальним співвідношенням називається довільна пара (p,q) слов у вільному автоматі з довільною системою твірних, а системою визначальних співвідношень – деяка множина ρ визначальних співвідношень. За допомогою системи визначальних співвідношень індуктивно визначається бінарне відношення $\approx$ умовної рівності слів. Позначимо $(p, q) \in \rho_k$, якщо $p \approx q(k)$ та $\overline{\rho} = \bigcup_{i=1}^{\infty} \rho_i$. Ю.І. Соркін показав, що $\bar{\rho}$ однозначно задає автомат $A[\rho]$, тобто $\overline{\rho}$ є лінгвістичним представленням автомату $A[\rho]$.

Ю.І. Соркін сформулював алгоритмічні проблеми для довільного автомата A, заданого скінченними системами утворювальних A' і визначальних співвідношень ρ:

1. Проблема тотожності. Вказати алгоритм, що дає змогу для двох довільних слів в автоматі A з'ясувати, чи є вони умовно рівними, чи ні.

2. Проблема ізоморфізму. Вказати алгоритм, що дає змогу встановити, чи буде автомат A ізоморфним автомату B, заданому скінченними системами твірних B' і визначальних співвідношень τ.

Ю.І.Соркін дав позитивне і конструктивне розв'язання обох проблем. Для цього вводиться поняття замкненої системи визначальних співвідношень і алгоритм переходу від довільної системи визначальних співвідношень до замкненої. Далі, для кожного слова з автомата A визначається його канонічне подання. Було доведено, що два слова умовно рівні тоді й тільки тоді, коли рівні їх канонічні подання.

Також було запропоновано розв'язання проблеми ізоморфізму. Для цього вводяться так звані невидалені стани, що утворюють базу автомата. Було показано, що у скінченно-визначеного автомата база скінченна і два скінченно-визначені автомати ізоморфні тоді й тільки тоді, коли ізоморфні їх бази. При цьому перевірка двох скінченних автоматів на ізоморфність є теоретично розв’язною задачею (наприклад, або шляхом повного перебору, або деякою модифікацією повного перебору), але ця задача є обчислювально складною [7, 8].

Надалі ідеї Ю.І. Соркіна узагальнюються і використовуються для дослідження проблем теорії експериментів із скінченними автоматами та лабіринтами. Задачі з цього напрямку активно розглядалися у відділі теорії керуючих систем Інституту прикладної математики і механіки НАН України І.С. Грунським та його учнями. У [9] запропоновано метод побудови економної системи визначальних співвідношень для автомата. У роботі [10] розглядається проблема контролю графів скінченним автоматом. Розв'язується така задача: дано всюдивизначений або частковий орграф G і клас Γ усіх таких графів. Потрібно побудувати скінченний автомат-агент, який переміщується за деяким графом H з класу Γ і за скінченний час видає інформацію чи є H ізоморфним G, чи ні. Задача розв'язується побудовою системи пар слів графа G, за допомогою якої здійснюється контроль графа.

У роботах [11, 12] розглядається опис групового автомата визначальною парою. Кожному всюдивизначеному, ініціально-зв'язному, скінченному автомату $A = (A, X, \delta_A, a_0)$ з n станами та m вхідними символами ставиться у відповідність опорна напівгрупа перестановок $G(A) = \{g_1, \dots, g_m\}$ зі звичайною операцією множення перестановок, де $g_i = \begin{pmatrix} a_0 & \dots & a_{n-1} \\ a_0 x_i & \dots & a_{n-1} x_i \end{pmatrix}$. Автомат *A* називається груповим, якщо його опорна напівгрупа *G(A)* є групою.

У роботі [12] описано алгоритми побудови визначальної пари за заданим автоматом і знаходження за визначальною парою групового автомата, описуваного нею. У роботі [11] описано алгоритм побудови множини визначальних слів опорної групи групового автомата і поліпшено алгоритм побудови визначальної пари групового автомата. Ці алгоритми використовують для побудови контрольного експерименту з груповими автоматами.

Вищевказані роботи показують, що завдання скінченних автоматів за допомогою різних варіантів визначальних пар слів становлять інтерес із погляду вивчення поведінки автоматів і застосування цих варіантів у теорії експериментів з автоматами. При цьому актуальним і важливим є як дослідження внутрішньої структури визначальних співвідношень автомата в сенсі [6], так і задачі побудови оптимальних (неперевідних, мінімальних) систем визначальних співвідношень. Ці проблеми було розглянуто І.С. Грунським та О.С. Сенченком у роботах [13, 14, 15, 16]. Далі докладніше це розглянемо.

У роботах [13, 14] розглядаються детерміновані, ініціально-зв’язні всюдивизначені автомати без виходу$A = (A, X, \delta_A, a_0)$, де A – множина станів, X – алфавіт вхідних символів, δ_A: $A \times X \to A$– функція переходів і a_0 – початковий стан. Через X^* позначимо множину всіх слів в алфавіті X. Автомат A породжує на X^* праву конгруенцію ρ_A за правилом: $(p, q) \in \rho_A$, якщо $\delta_A(a_0, p)$, $\delta_A(a_0, q)$ визначені і $a_0 p = a_0 q$. Нехай ρ – деяке бінарне відношення на X^*. Це відношення назвемо системою визначальних

співвідношень для A, якщо $[\rho] = \rho_A$, де $[\rho]$ – правоконгруентне замикання ρ, тобто найменьша права конгруенція, що містить ρ. У цьому випадку ρ є лінгвістичним представленням автомату A.

У цих роботах вперше і повністю розв'язано задачу характеризації скінченного бінарного відношення $\rho \subseteq X^* \times X^*$ для фіксованого повністю визначеного автомата A, а саме, задачу визначення, чи є ρ системою визначальних співвідношень для A. Для цього введено спеціальну канонічну систему визначальних співвідношень κ_A. Показано, що вона є мінімальною в класі всіх систем визначальних співвідношень для A. Розв'язок зазначеної вище задачі полягає в спеціальному зведенні ρ до κ_A. Наведемо алгоритм побудови канонічної системи визначальних співвідношень.

Нехай $X = \{x_1, \dots, x_m\}$ і $x_1 \le x_2 \le \dots \le x_m$ – довільно зафіксований лінійний порядок P на X. На X^* вводиться лінійний порядок $\preccurlyeq$:

1) $x_i \preccurlyeq x_i$;

2) якщо $d(p) < d(q)$, то $p \preccurlyeq q$, де $d(p)$ – довжина слова p;

3) $x_1' \dots x_s' = p \preccurlyeq q = x_1'' \dots x_s''$, якщо $x_k' \preccurlyeq x_k''$ для деякого $k \le s$, в той час як $x_k' = x_1'', \dots, x_{k-1}' = x_{k-1}''$.

Нехай $A = \{a_0, \dots, a_{n-1}\}$ і v_i – найкоротше за введеним порядком $\preccurlyeq$ слово, для якого $a_0 v_i = a_i$. Покладемо $V_A = \{v_0, \dots, v_{n-1}\}$, ця множина називається найкоротшим базисом досяжності автомата A. Визначимо скінченне бінарне відношення κ_A на множині $V_A \times V_A \cdot X$ за таким правилом: нехай κ_A спочатку порожнє. Для кожних $v_i, v_j \in V_A$ і $x \in X$, якщо $a_0 v_i x = a_0 v_j$ і $v_i x \neq v_j$, то пара $(v_j, v_i x)$ міститься в κ_A. Доведено, що κ_A є системою визначальних співвідношень для автомата A.

Також знайдено метричні характеристики системи визначальних співвідношень ρ для автомата A, у якого потужність вхідного алфавіту дорівнює m, а кількість станів n. Цій системі визначальних співвідношень поставимо у відповідність дві характеристики. Через $|\rho|$ позначимо число пар слів у ρ, а через $S(\rho)$ позначимо $\sum(d(p) + d(q))$, де підсумовування

відбувається за всіма $(p, q) \in \rho$. Доведено, що значення цих характеристик для довільної канонічної системи визначальних співвідношень κ_A автомата A задовольняють співвідношенням:

а) $|\kappa_A| = (m-1)n + 1$;

б) $S(\kappa_A) \geq \left(\frac{m^{l+1}-1}{m-1} - n\right) l + \left(m^l + n - \frac{m^{l+1}-1}{m-1}\right) m(l+1)$,

де $l = \lceil \log_m n(m-1) + 1 \rceil - 1$, причому ця оцінка є досяжною для всіх $n, m > 1$;

в) $S(\kappa_A) \leq n(m-1)(n-1) + m(2n-1)$, причому ця оцінка є досяжною для всіх $n \geq 2,\ m \geq 1$.

Також доведено, що κ_A є мінімальною системою визначальних співвідношень для автомата A за обома наведеними вище характеристиками.

Далі наведемо розв'язок задачі характеризації скінченного бінарного відношення. Нехай ρ – деяке скінченне бінарне відношення на X^*.

Введемо на ρ такі операції:

1. Видалимо з ρ всі пари типу (p, p).
2. Кожну пару $(p, q) \in \rho$ при $q \leq p$ замінимо парою (q, p).
3. Нехай $(p, q), (t, u) \in \rho$. Якщо $p = uw$ для деякого $w \in X^*$, то пару (p, q) в ρ замінюємо парою (tw, q). Якщо ж $q = uw$, то пару (p, q) в ρ замінюємо парою (p, tw).
4. Видалимо пари, що повторюються, залишаючи тільки по одному екземпляру.

Введемо процедуру редукції відношення ρ, яка складається з таких кроків:

а) виконується операція 1;

б) виконується операція 2;

в) якщо операцію 3 неможливо застосувати до жодної пари, то процедура завершується, інакше після кожного одноразового виконання операції 3 переходимо до кроку (г);

г) виконується операція 4, і переходимо до кроку (а).

Результат редукції позначимо через <ρ>. Показано, що процедура редукції визначає результат однозначно.

Отримано критерій, чи є задане скінченне бінарне відношення ρ системою визначальних співвідношень для заданого автомата A: ρ є системою визначальних співвідношень для A тоді і лише тоді, коли $< \rho > = \kappa_A$

Запропоновано окремий випадок розв'язку проблеми ізоморфизма без побудови автоматів за їх системами визначальних співвідношень: нехай ρ і τ – скінченні системи визначальних співвідношень для автоматів A і B відповідно. Автомати A і B ізоморфні тоді і лише тоді, коли $< \rho > = < \tau >$.

У роботах [15, 16] наведені вище результати розповсюджено на детерміновані ініціально-зв'язні часткові скінченні автомати без виходу. Запропоновано нову конструкцію – визначальну систему $\{\rho, M\}$, в якій бінарне відношення ρ виступає аналогом системи визначальних співвідношень, а множина M визначає множину слів, на яких функція переходів автомата не визначена. Така система є узагальненням системи визначальних співвідношень для всюдивизначеного автомата. Знайдено розв'язки задач характеризації скінченної системи.

Позначимо через $Dom\ \delta_A$ множину всіх таких слів $p \in X^*$, для яких $\delta_A(a_0, p)$ визначена та $Codom\ \delta_A = X^* - Dom\ \delta_A$. Через $Codom_m\ \delta_A$ позначимо множину мінімальних за начальними відрізками слів з $Codom\ \delta_A$. Показано, що система $\{\rho_A, Codom_m\ \delta_A\}$ однозначно визначає автомат A, тобто є його лінгвістичним представленням.

Нехай задана довільна система $\{\rho, M\}$. Нехай W_ρ – множина початкових відрізків усіх слів з $pr_1\rho \cup pr_2\rho$ і $[\rho](M)$ – зріз бінарного відношення $[\rho]$ за M. Систему $\{\rho, M\}$ назвемо визначальною для автомата A, якщо для неї одночасно виконуються наступні умови:

i) $[\rho] = \bar{\rho}_A$;

ii) $W_\rho \subseteq Dom\ \delta_A$;

iii) $[\rho](M) = Codom_m\ \delta_A$.

При цьому, у випадку, коли A повністю визначений, і система $\{\rho, M\}$ є визначальною для A, то $M = \emptyset$, а ρ є системою визначальних співвідношень для A.

Показано значення введення визначальної системи для автомата A: нехай система $\{\rho, M\}$ задовольняє умовам (i)–(iii) для деякого автомата A. Тоді вона однозначно визначає автомат A, причому $Dom\ \delta_A = X^* \setminus [\rho](M) \cdot X^*$.

По аналогії із канонічною системою визначальних співвідношень для повністю визначеного автомата, запропоновано канонічну визначальну систему $\{\kappa_A, K_A\}$: нехай κ_A і K_A спочатку порожні. Для кожних $v_i, v_j \in V_A$ і $x \in X$, якщо $a_0 v_i x = a_0 v_j$ і $v_i x \neq v_j$, то пара $(v_j, v_i x)$ міститься в κ_A. Якщо ж $\delta_A(a_0 v_i, x)$ не визначена, то слово $v_i x$ міститься в K_A. Доведено, що система $\{\kappa_A, K_A\}$ є визначальною системою для автомата A.

Також наведено деякі метричні характеристики системи $\{\rho, M\}$. Через $|\rho, M|$ позначимо суму числа пар в ρ і числа слів в M. Нехай також $S(\rho, M) = \sum(d(p) + d(q) + d(r))$, де підсумовування відбувається за всіх парах $(p, q) \in \rho$ і словах $r \in M$ відповідно. Наведено значення цих характеристик для канонічної визначальної системи автомата A:

а) $|\kappa_A, K_A| = (m-1)n + 1$;

б) $S(\kappa_A, K_A) \geq \left(\frac{m^{l+1}-1}{m-1} - n\right) l + \left(m^l + n - \frac{m^{l+1}-1}{m-1}\right) m(l+1)$,

де $l = \lceil \log_m n(m-1) + 1 \rceil - 1$;

в) $S(\kappa_A, K_A) \leq n(m-1)(n-1) + m(2n-1)$, причому оцінки (б) і (в) є досяжними для всіх m и n. Доведено, що $\{\kappa_A, K_A\}$ є мінімальною визначальною системою для автомата A за обома наведеними вище характеристиками.

Під характеризацією системи $\{\rho, M\}$ у роботах [15, 16] розуміють розв'язок наступних задач:

(1) Визначити, чи існує скінченний частковий автомат, для якого система $\{\rho, M\}$ є визначальною.

(2) Визначити, чи є $\{\rho, M\}$ визначальною системою для фіксованого часткового автомата A.

Ці задачі розв'язані для скінченних систем $\{\rho, M\}$. Задачу (2) розв'язано за допомогою введеної для цієї мети процедури редукції системи, яка є модифікацією процедури редукції бінарного відношення, що розглянута вище.

Система $\{\rho, M\}$ називається правильною тоді і лише тоді, коли існує частковий, можливо нескінченний, автомат A, для якого $\{\rho, M\}$ є визначальною системою. У цьому випадку автомат A позначається $A(\rho, M)$. Охарактеризовано правильні системи. Систему $\{\rho, M\}$ називається сумісною, якщо одночасно виконуються такі умови: $W_\rho \cap [\rho](M) = \emptyset$ і $M \cap [\rho](M) \cdot X^+ = \emptyset$, тобто ρ і M не протирічать одне одному. Доведено, що система $\{\rho, M\}$ є правильною тоді і лише тоді, коли вона є сумісною.

Також знайдено умови для ρ і M, за яких автомат $A(\rho, M)$ є скінченним. Клас всіх правильних систем, що задають скінченні автомати на множині X^* позначено через $Q(X)$. Охарактеризовано клас систем $Q(X)$. Сумісна система $\{\rho, M\}$ називається визначеною, якщо для кожних $x \in X$, $p \in X^* \setminus [\rho](M) \cdot X^*$ існують такі $p' \in [\rho](p)$, $z \in N$ для яких виконується хоч одна з умов:
а) $p' \in W_\rho$; б) $p'x^z \in M$.

Інакше кажучи, в цьому випадку ρ і M доповнюють одне одну так, щоб в автоматі $A(\rho, M)$ не було жодного нескінченного підавтомата-дерева. Доведено, що система $\{\rho, M\}$ належить класу $Q(X)$ тоді і лише тоді, коли вона визначена.

Таким чином одержано повний розв'язок першої задачі характеризації системи $\{\rho, M\}$. Він полягає в тому, що спочатку визначається, чи є система сумісною, потім визначається, чи є система визначеною. Якщо система $\{\rho, M\}$ визначена, то автомат $A(\rho, M)$ скінченний, якщо система $\{\rho, M\}$ не визначена, то автомат $A(\rho, M)$ нескінченний, якщо система $\{\rho, M\}$ несумісна, то автомата $A(\rho, M)$ не існує.

Другу задачу характеризації скінченної системи розв'язано за допомогою процедури редукції системи, що є аналогічною до процедури редукції бінарного відношення, в якій операцію 3 введено наступним чином.

Нехай (p, q), $(t, u) \in \rho, z \in M$. Якщо $p = uw$ для деякого $w \in X^*$, то пару (p, q) в ρ замінюємо парою (tw, q). Якщо $q = uw$, то пару (p, q) в ρ замінюємо парою (p, tw). Якщо ж $z = uw$, то слово z в M замінюємо словом tw.

Результат редукції позначається через $\langle \rho, M \rangle$. Наведено критерій, чи є $\{\rho, M\}$ визначальною системою для заданого скінченного часткового автомата A: Нехай A – деякий скінченний частковий автомат і нехай скінченна система $\{\rho, M\}$, де $\rho \subseteq X^* \times X^*$ і $M \subseteq X^*$, є визначеною. $A(\rho, M) = A$ тоді і лише тоді, коли $\langle \rho, M \rangle = \{\kappa_A, k_A\}$.

Далі у авторів робіт [13, 14, 15, 16] природним шляхом виникло бажання розповсюдити отримані результати на інші об’єкти, що схожі на автомати. На їх погляд було б логічно спробувати розповсюдити результати на так звані детерміновані графи (в тому числі на їх підклас – сильно детерміновані графи) з таких причин:

- ці графи можуть успішно використовуватись в якості моделі багатьох програмних об’єктів та інформаційних процесів;
- такі графи мають певну схожість зі скінченними автоматами без виходу;
- для дослідження структури об’єктів, які можна змоделювати графом з розміченими вершинами (включаючи детерміновані та сильно-детерміновані графи, часто використовують один або декілька мобільних агентів з обмеженою пам'яттю, які поміщають у досліджуваний об’єкт. Такий спосіб представлення є досить зручним для подібного дослідження.

Відомості про детерміновані графи наведемо у наступній главі цієї роботи.

1.2 Основні теоретичні відомості про детерміновані графи та напрямки їх застосування

Наведемо основні поняття та означення, які будуть використовуватися далі в роботі. Розглядаємо неорієнтовані, скінченні, непорожні, зв'язні прості (тобто ті, що не містять петлі та кратні ребера) графи з розміченими вершинами $G = (V, E, X, \xi(V))$, де V – множина вершин графа, E – множина його ребер, $\xi(V): V \to X$ – всюдивизначена функція розмітки вершин графа символами скінченного алфавіта $X = \{x_1, \dots, x_p\}$, значення цієї функції для вершини *v* будемо називати її міткою. Через $E(v)$ позначимо множину вершин, які є суміжними вершині *v*: $v' \in E(v) \leftrightarrow (v', v) \in E$, ступенем вершини *v* будемо називати число $|E(v)|$; вершину ступеня 1 називатимемо висячою.

Розмічений граф *G* називається детермінованим [17] (або Д-графом, або графом із детермінованою розміткою), якщо всі вершини у відкритому околі кожної його вершини мають різні мітки: $\forall v, v_1, v_2 \in V(v_1, v_2 \in E(v) \wedge \xi(v_1) = \xi(v_2)) \longrightarrow v_1 = v_2)$. Д-граф G називається сильно детермінованим [17] (або СД-графом), якщо всі вершини у замкненому околі кожної його вершини мають різні мітки. Змістовно кажучи, у Д-графі можуть існувати дві суміжні вершини з однаковими мітками, а в СД-графі таких вершин не може бути.

Зафіксуємо деяку вершину $v_0 \in V$ Д-графа $G = (V, E, X, \xi(V))$, яку далі будемо називати ініціальною, цю вершину за необхідності ми будемо виділяти у позначенні Д-графа: $G = (V, E, X, \xi(V), v_0)$. Граф $G = (V_1, E_1, X, \xi_1(V_1), v_0)$ називається підграфом графа $H = (V_2, E_2, X, \xi_2(V_2), v_0)$ (H є надграфом G), що позначається через $G \subseteq H$, якщо $V_1 \subseteq V_2$, $E_1 \subseteq E_2$ та $\xi_1(V_1) \subseteq \xi_2(V_2)$. Якщо $G \subseteq H$ та $G \neq H$, то G є власним підграфом H (позначатимемо через $G \subset H$).

Ізоморфізмом Д-графів $G_1 = (V_1, E_1, X, \xi_1(V_1), v')$ і $G_2 = (V_2, E_2, X, \xi_2(V_2), v'')$ з однаковою множиною міток вершин X називається взаємно однозначна відповідність $\varphi: V_1 \leftrightarrow V_2$ між множинами їх вершин, що

зберігає їх ініціальні вершини, відношення суміжності вершин та функцію розмітки вершин: $\varphi: (v') = v'',\ \forall v_1, v_2 \in V_1\ (v', v'') \in E_1 \rightarrow (\varphi(v'), \varphi(v'')) \in E_2$ та $\forall v \in V_1\ \xi_1(v) = \xi_2(\varphi(v))$; ізоморфізм графів G_1 та G_2 позначають $G_1 \cong G_2$.

Зафіксуємо вершину v'_1 Д-графа $G = (V, E, X, \xi(V))$. Оскільки шляху $p = v'_1 \dots v'_k$ однозначно відповідає слово в алфавіті міток $\xi(p) = \xi(v'_1) \dots \xi(v'_k)$, то надалі шлях p розглядатимемо як $\xi(p)$ та позначатимо рівністю $v'_1 \xi(p) = v'_k$. Шлях (слово) $p = x'_1 \dots x'_k$ назвемо припустимим для вершини $v'_1 \in V$, якщо $\xi(v'_1) = x'_1$, та існують такі вершини $v'_2 \dots v'_k \in V$, що $\xi(v'_2) = x'_2, \dots, \xi(v'_k) = x'_k$ і $(v'_1, v'_2), \dots, (v'_{k-1}, v'_k) \in E$.

Слово $p = x'_k \dots x'_1$ будемо позначати через p^{-1}, слова p та p^{-1} називатимемо взаємнозворотними. Слово q назвемо початковим відрізком слова p у випадку, якщо існує таке слово w, що $qw = p$, цей факт позначатимемо через $q \subseteq p$.

Всі вершини заданого Д-графа $G = (V, E, X, \xi(V), v_0)$ можна розбити на два класи за такою процедурою: видалимо з G усі висячі вершини, відмінні від ініціальної, разом із ребрами, що є інцидентними цим вершинам. Цю процедуру будемо повторювати до тих пір, поки це можливо. Одержаний у такий спосіб граф $B(G)$ називається базою графа G, вершини, що входять до $B(G)$, називаються базовими, а ті вершини G, що не входять до $B(G)$, – вільними. Неважко бачити, що база графа будується однозначно за скінченну кількість кроків, база $B(G)$ може складатися з однієї ініціальної вершини (у випадку, коли G є деревом), а у випадку, коли $B(G)$ містить хоча б одну вершину, яка відмінна від ініціальної, то вона повинна містити хоча б один простий цикл. Звідси випливає, що кожна базова вершина, яка є відмінною від ініціальної вершини v_0, суміжна щонайменше двом базовим вершинам.

В наш час розмічені графи активно застосовують для описання та моделювання обчислювальних процесів у програмуванні, робототехніці, верифікації та валідації моделей тощо. Розмічені графи є інформаційним

середовищем для мобільних агентів, переміщення яких по графу можна відобразити послідовностями міток вершин – слів в алфавіті міток. При цьому розмічені графи, у яких мітки знаходяться на ребрах (Labeled Transition System, зважені графи, скінченні автомати тощо) досліджені в значно більшій мірі, ніж графи з розміченими вершинами. Але існує багато обчислювальних процесів у програмуванні, робототехніці, верифікації та валідації моделей, які зручно відображати саме графами з розміченими вершинами.

Для дослідження структури розмічених графів часто використовують один або декілька мобільних агентів з обмеженою пам'яттю, які поміщають на досліджуваний граф. Ці агенти можуть сповіщати спостерігачу та/або іншим агентам певну інформацію щодо локальних околів вершин, на яких вони наразі знаходяться. На основі цієї інформації та/або інструкцій спостерігача агенти можуть переміщатися по ребрах графа. Переміщення агентів визначають послідовності міток вершин, які вони відвідали.

Зважаючи на це, детерміновані графи є дуже зручними для такого дослідження, оскільки, якщо спостерігач має карту графа (множини *V, E, X* та функцію $\xi(V)$) та знає, на яку вершину досліджуваного детермінованого графа був поміщений агент на початку дослідження (тобто, ініціальну вершину), то за цією послідовністю міток може бути однозначно відновлена траєкторія переміщення агента по графу.

У випадку, коли спостерігачу не відома карта досліджуваного графа, переміщення агентів можуть бути організовані в такий спосіб, щоб на основі їх аналізу спостерігач одержав шукану інформацію про структуру графа (наприклад, складення карти графа, пошуку у ньому найкоротших шляхів, порівняння досліджуваного графа з графом-еталоном).

Крім того, існують деякі специфічні напрямки застосування детермінованих графів, вони, наприклад, використовуються при розподілі часових слотів (тайм-слотів) у мережах, які працюють за принципом множинного доступу з часовим поділом [18].

Незважаючи на наведені вище переваги детерміновані графи на даний час досліджені, на наш погляд, дуже слабо. Так, у роботі [17], у якій детерміновані графи вперше було формалізовано, досліджуються мови, що породжуються вершинами детермінованого графа. У цьому контексті вводиться поняття нерозрізненості відношення домінування вершин детермінованого графа. Інших робіт, у яких, наприклад, досліджується структура детермінованих графів, чи розглядаються алгоритми детермінованого розфарбування графів у відкритій літературі нами не знайдено. Однією з причин цього, на наш погляд, є те, що традиційне задання детермінованих графів є не досить зручним для їх дослідження за допомогою агентів, що переміщуються всередині графу. Тому вважаємо, що лінгвістичне подання детермінованих графів, теорія якого зараз активно розробляється у відділі теорії керуючих систем Інституту прикладної математики і механіки НАН України, буде дуже корисним для їх дослідження та може підвищити зацікавленість науковців до їх дослідження. Відомості про лінгвістичне подання детермінованих графів більш детально наводяться у наступній главі цієї роботи.

1.3 Лінгвістичне представлення детермінованих графів

У цій главі наведемо основні результати, пов’язані з лінгвістичним представленням детермінованих графів у термінах, визначених у попередній главі цієї роботи. Зазначимо, що до цього часу не розроблено загальновизнаних понять та апарату цього представлення, проте дослідження з цієї тематики активно проводяться в Інституті прикладної математики і механіки НАН України у відділі теорії керуючих систем.

Вперше результати, присвячені лінгвістичному представленню детермінованих графів опубліковано у роботі [19]. Після того у роботах [18,

20, 21] ці результати уточнюються та доповнюються. Наведемо всі ці результати станом на поточний час.

Далі вважаємо, що всі шляхи в графі починаються з ініціальної вершини. Було розроблено представлення Д-графа $G = (V, E, X, \xi(V), v_0)$, у якого $\xi(v_0) = x'$ парою $\{C, L\}(x')$ скінченних множин слів $C, L \in X^*$, для якої всі слова з C та L є припустимими для v_0, слова множини C описують цикли графа G, а слова L описують його висячі вершини. Під терміном «пара» $\{C, L\}(x')$ розуміємо дві скінченні множини слів C та L, що відповідають умовам:

1) будь-яке слово з множини C починається та закінчується символом x';

2) будь-яке слово з множини L починається на символ x';

3) довжина будь-якого слова з множини C більша ніж 2.

Множини C та L називають компонентами пари $\{C, L\}(x')$. Якщо при цьому будь-яке слово з кожної компоненти пари $\{C, L\}(x')$ не містить послідовності з двох однакових символів, то таку пару називають безповторною.

Було висунуто вимоги, за виконанням яких детермінований граф $G = (V, E, X, \xi(V), v_0)$, у якого алфавіт X співпадає з множиною всіх символів, які присутні в словах з обох компонентів пари $\{C, L\}(x')$, вважається представленням за цією парою:

а) $\xi(v_0) = x'$;

б) всі слова з C та L є припустимими для вершини v_0;

в) кожне слово r з C визначає у G цикл $v_0 r = v_0$;

г) для кожного слова $q \in L$ у G вершина $v_0 q$ є висячею;

д) для кожної висячої вершини $v \in V$, що є відмінною від ініціальної вершини v_0, існує хоча б одне таке слово $q \in L$, що $v_0 q = v$;

е) для будь-якої пари $\{C, L\}(x')$ або не існує представлення, або це представлення визначається однозначно.

При цьому декільком різним парам може відповідати одне й те ж представлення.

Так само, як і у [13] та [14] на X^* вводиться лінійний порядок $\preccurlyeq$, що наведено у першій главі цього розділу роботи. Нехай $G = (V, E, X, \xi(V), v_0)$ – довільний граф з розміченими вершинами. Редукцією G називається процедура, яка перетворює G у детермінований граф $[G]$ за таким алгоритмом (АР):

0) Покладемо $V' = V, E' = E, \xi'(V') = \xi(V)$, позначимо $G' = (V', E', X, \xi'(V'), v_0)$.

1) Для кожної вершини $v \in V'$ знаходимо найменше за $\preccurlyeq$ слово w, що відповідає шляху від v_0 до v. Пов'язуємо v з w.

2) Впорядковуємо вершини графа G' за пов'язаними з ним словами за порядком $\preccurlyeq$. Поточною вершиною v_c призначимо першу за вказаним порядком вершину, а поточною міткою призначимо $v = x_1$.

3) Якщо існують такі різні вершини $v'_1 \dots v'_q$, для яких одночасно виконуються умови $v'_1 \dots v'_q \in E'(v_c)$ та $\xi'(v'_1) = \dots = \xi'(v'_q) = x$, то позначимо $U = \{v'_1 \dots v'_q\}$ та виконуємо таку послідовність дій:

3.1. до множини V' додаємо нову вершину v' та покладемо $\xi'(v') = x$;

3.2. з множини E' вилучаємо ребра (v_i, v_j), де $v_i, v_j \in U$;

3.3 для кожного ребра (v_i, v_j), де $v_i \in U$, до множини E' додаємо ребро (v', v_j);

3.4. з множини E' вилучаємо ребра (v_i, v_j), де $v_i \in U$, з множини V' вилучаємо вершини v, де $v \in U$;

3.5 якщо $v_0 \in U$, то вершину v' перейменовуємо на v_0 та вважаємо далі цю вершину ініціальною;

3.6. вилучаємо повтори ребер, залишаючи по одному екземпляру;

3.7. переходимо на крок 1.

4) Якщо існує $x' \in X$, який є наступним за x за порядком $<$ (тобто, $x \neq x_p$), то покладемо $x = x'$ та переходимо до кроку 3.

5) Якщо існує вершина v, що є наступною для v_c, то покладемо $v_c = v$, $x = x_1$ та переходимо на крок 3, у протилежному випадку алгоритм АР завершує свою роботу та *[G] = G'*.

Неважко бачити, що виконання алгоритму АР закінчується за скінченну кількість кроків, його результат визначається однозначно та цей результат є Д-графом.

Зауважимо, що при виконанні цього алгоритму існує певна неоднозначність з іменами нових вершин (окрім випадку, за яким новостворену вершину v' було перейменовано на v_0 на кроці 3.5), проте, для авторів робот [18, 19, 20, 21] є несуттєвими імена усіх інших вершин (окрім ініціальної) у графі. За цієї обставини, якщо було виконано крок 3 алгоритму АР, то не можна говорити про рівність графа *[G]* якомусь іншому графу, а можемо говорити про ізоморфність *[G]*. Також можна бачити, що *[G] = G* тоді та лише тоді, коли *G* є Д-графом.

Також визначено алгоритм (АП), який за заданою парою $\{C, L\}(x')$ або будує Д-граф $G(\{C, L\}(x'))$, або показує, що за цією парою неможливо побудувати Д-граф, що відповідає умовам (а) – (д).

0) Покладемо, що спочатку граф $G(\{C, L\}(x'))$ складається з однієї вершини v_0 з міткою $\xi(v_0) = x'$.

1) У відповідність кожному слову $p^i = x'x_1^i \dots x_n^i x' \in C$ додаємо до графа вершини $v_1^i, \dots, v_n^i$ з мітками відповідно $x_1^i, \dots, x_n^i$ та ребра $(v_0, v_1^i), (v_1^i, v_2^i), \dots, (v_{n-1}^i, v_n^i), (v_n^i, v_0^i)$. Після кожного такого додавання робимо редукцію одержаного графа.

2) Для кожного слова $p^j = x'x_1^j \dots x_n^j \in L$ виконуємо таку послідовність дій: додаємо до графа вершини $v_1^j, \dots, v_n^j$ з мітками відповідно $x_1^j, \dots, x_n^j$, ребра $(v_0, v_1^j), \dots, (v_{n-1}^i, v_n^i)$ та робимо редукцію одержаного графа.

3) Розглядаємо всі слова множини *L*: якщо існує таке $p \in L$, що вершина $v_0 p$ не є висячою, то вважаємо, що граф $G(\{C, L\}(x'))$ не існує.

4) Для кожної висячої вершини $v \in G(\{C, L\}(x'))$ розглядаємо слова компоненти L: якщо не існує такого $p \in L$, що $v = v_0 p$, то вважаємо, що граф $G(\{C, L\}(x'))$ не існує.

Якщо в результаті виконання цієї процедури за парою $\{C, L\}(x')$ можливо побудувати граф $G(\{C, L\}(x'))$, то таку пару назвемо правильною. У алгоритмі АП перший та другий етапи створюють деякий граф, який, за умови успішної перевірки на етапах 3 та 4 є графом $G(\{C, L\}(x'))$. Якщо хоча б одна з перевірок на етапах 3 та 4 не виконується, то вважаємо, що граф $G(\{C, L\}(x'))$ не існує.

З того, що вершини та ребра графа $G(\{C, L\}(x'))$ будуються виключно за словами з множин C та L, у слів з безповторної пари немає послідовності з двох однакових міток, а, отже на будь-якому кроці побудови графа $G(\{C, L\}(x'))$ у ньому не існує суміжних вершин з однаковими мітками, при цьому виконання процедури редукції також не призведе до появи суміжних вершин з однаковими мітками, випливає наступне твердження.

Теорема 1. Якщо $\{C, L\}(x')$ – правильна безповторна пара, то граф $G(\{C, L\}(x'))$ є сильно-детермінованим.

Неважко бачити, що у випадку, коли пара $\{C, L\}(x')$ є правильною, граф $G(\{C, L\}(x'))$ відповідає вимогам (а) – (д), будь-який власний підраф графа $G(\{C, L\}(x'))$ не відповідає вимозі (б), а будь-який власний надграф графа $G(\{C, L\}(x'))$ відповідає вимогам (а) – (в), проте може не відповідати вимогам (г) та (д). Отже, якщо пара $\{C, L\}(x')$ є правильною, то граф $G(\{C, L\}(x'))$ є найменшим за включенням графом, що відповідає вимогам (а) – (д), тому його вважають *представленням* за парою $\{C, L\}(x')$.

Правильну пару $\{C, L\}(x')$ називають визначальною для Д-графа G, якщо $G(\{C, L\}(x')) \cong G$.

Таким чином, алгоритм АП є частковим відображенням множин пар на множину Д-графів, за яким правильній парі відповідає Д-граф, що будується за алгоритмом АП.

Проілюструємо дію окремих етапів наведеної вище процедури на наступному прикладі.

Приклад 1. Нехай $C = \{12341,142451\}$, $L = \{152125423,14523\}(1)$. На рис. 1.1 а зображено граф, який отримуємо після виконання етапу 1, на рис. 1.1 b зображено граф, який отримуємо після виконання етапу 2. У цьому графі вершини $v_0152125423$ та v_014523 є висячими, тому перевірка на етапі 3 виконується успішно. Для виділеної вершини *v* з міткою 1 немає жодного слова $p \in L$, що $v_0p = v$, тобто перевірка на етапі 4 не виконується, тому граф $\{C, L\}(1)$ не існує.

Зауважимо, що для пари $\{C, L'\}(1)$, де $L' = L \cup \{1521\}$ граф $\{C, L'\}(1)$ існує, він зображений на рис. 1.1 b.

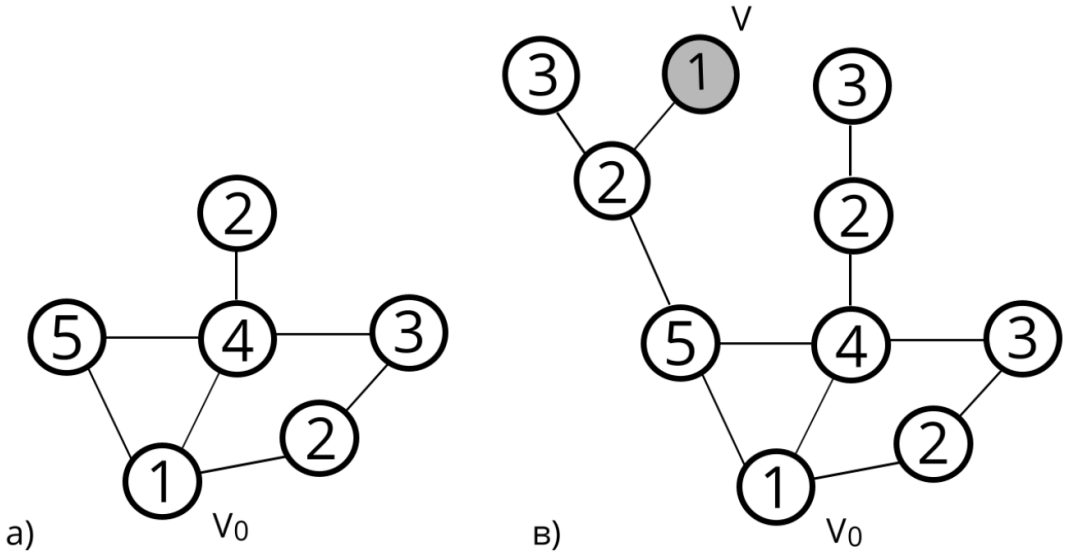


Рис. 1.1. *Ілюстрація процедури побудови Д-графа за заданою парою*

Також визначено особливу канонічну визначальну пару для будь-якого Д-графа. Нехай $G = (V, E, X, \xi(V), v_0)$ – деякий Д-граф, $V = \{v_0, \dots, v_{n-1}\}$ та $\xi(v_0) = x'$.

Визначається допоміжна множина слів в алфавіті *X*. Базисом досяжності $\mathcal{V}_G$ називається така множина слів $\{w_0, \dots, w_{n-1}\}$, де $w_0 = x'$ та для кожної $v_i \neq v_0$ існує таке слово $w_i \in \mathcal{V}_G$, що виконується $v_0w_i = v_i$, та для будь-якого $w \neq w_i$ з $v_0w = v_i$, випливає $w_i \preccurlyeq w$. Кістякове дерево графа *G*, яке

визначається базисом $\mathcal{V}_G$, позначають $T(\mathcal{V}_G)$ або $T(G, v_0)$. Для кожної вершини $v \in V$ через sp_v позначено таке слово з $\mathcal{V}_G$, що $v_0 sp_v = v$.

Визначено алгоритм АК побудови для Д-графа G визначальної пари $\{\Sigma_G, \Lambda_G\}$, яку, завдяки деяким її властивостям, було названо канонічною.

Спочатку покладемо $\Sigma_G = \emptyset$ і $\Lambda_G = \emptyset$. Якщо граф G складається з однієї вершини v_0, то покладемо $\Sigma_G = \emptyset$, $\Lambda_G = \emptyset$ і $\mathcal{V}_G = \{\xi(v_0)\}$.

Нехай граф G містить більше ніж одну вершину. Спочатку до множини Λ_G додаємо усі слова $w \in \mathcal{V}_G$ такі, що вершина $v_0\, w$ є висячою вершиною графа G. Після цього для кожної двійки слів $p, q \in \mathcal{V}_G \backslash \Lambda_G$, якщо жодне з них не є початковим відрізком іншого і $v_0 p q^{-1} = v_0$, то додаємо до множини Σ_G одне з двох слів pq^{-1} або qp^{-1}, яке є меншим за порядком $\preccurlyeq$ (вершини $v_0 p$ та $v_0 q^{-1}$ називають твірними для того слова pq^{-1} або qp^{-1}, що було додане до Σ_G). З алгоритмів АП та АК можна бачити, що пара $\{\Sigma_G, \Lambda_G\}$ є правильною та $G(\{\Sigma_G, \Lambda_G\}) \cong G$, тобто $\{\Sigma_G, \Lambda_G\}$ є визначальною парою для G.

Також наведено деякі властивості канонічної визначальної пари, що безпосередньо випливають з алгоритму АК.

Теорема 2. Нехай $\{\Sigma_G, \Lambda_G\}$ – канонічна визначальна пара Д-графа $G = (V, E, X, \xi(V), v_0)$.

1) Якщо хоча б одна компонента $\{\Sigma_G, \Lambda_G\}$ не є порожньою множиною, то для кожної вершини $v \in V$ існують такі $z_1, z_2, z_3 \in X^*$, що виконується хоча б одне з тверджень:

а) $sp_v\, z_1 \in \Lambda_G$;

б) $sp_v\, z_2 \in \Sigma_G$;

в) $z_3 {sp_v}^{-1} \in \Sigma_G$.

2) Нехай (v_1, v_2) – деяке ребро графа G, $\xi(v_1) = {x_1}'$ та $\xi(v_2) = {x_2}'$. Тоді існують такі $z_1, z_2 \in X^*$, що виконується хоча б одне з тверджень:

а) $sp_{v_1}\, {x_2}' z_1 \in \Sigma_G \cup \Lambda_G$;

б) $sp_{v_2}\, {x_1}' z_2 \in \Sigma_G \cup \Lambda_G$.

3) Якщо $\Sigma_G \neq \emptyset$, то для кожного $\sigma \in \Sigma_G$ існують такі $p, q \in X^*$, що $pq \in \sigma$, та $p \in \mathcal{V}_G$ і $q^{-1} \in \mathcal{V}_G$.

4) Нехай (v_1, v_2) – деяке ребро графа G, яке не належить кістяковому дереву $T(G, v_0)$. Тоді існує єдине слово $\sigma \in \Sigma_G$, що ребро (v_1, v_2) входить до шляху $v_0\sigma$.

5) Якщо $\Sigma_G \neq \emptyset$, то для кожного $\sigma \in \Sigma_G$ виконується нерівність $d(\sigma) \geq 4$.

6) Якщо $\Sigma_G \neq \emptyset$, то для кожного $\sigma \in \Sigma_G$ знайдуться такі $p = p'x_1, q \in X^*$ $(x_1 \in X)$, $x_2, x_3 \in X$, $x_2 \neq x_3$, що $\sigma = p\ x_2\ q\ x_3 p^{-1}$ та $v_0 p = v_0 p\ x_2\ q\ x_3 x_1$.

7) Якщо v_1, v_2 – твірні вершини деякого слова $\sigma \in \Sigma_G$, то $(v_1, v_2) \notin T(G, v_0)$ та $\left| d(sp_{v_1}) - d(sp_{v_1}) \right| \leq 1$.

Змістовно кажучи, перше та друге твердження стверджують, що пара $\{\Sigma_G, \Lambda_G\}$ містить певну інформацію про кожну вершину та ребро графа G, третє твердження встановлює своєрідну мінімальність початкових та кінцевих відрізків слова $\sigma \in \Sigma_G$, четверте твердження вказує, що кожне ребро, яке не належить кістяковому дереву $T(\mathcal{V}_G)$, описується окремим словом з Σ_G, п’яте твердження показує, що будь-яке $\sigma \in \Sigma_G$ містить простий цикл; який, як встановлює шосте твердження, не може бути представлений у вигляді $p\ x\ p^{-1}$ для будь-яких $p \in X^*$, $x \in X$.

Також знайдено деякі метричні властивості компонент канонічної визначальної пари. Нехай далі Д-граф $G = (V, E, X, \xi(V), v_0)$ має n вершин та m ребер. Наведено оцінки потужності (кількості елементів) та об'єму (суми довжин усіх слів) кожної компоненти канонічної визначальної пари (відповідно $|\Sigma_G|$, $|\Lambda_G|$, $||\Sigma_G||$, $||\Lambda_G||$). Далі позначення $\lceil x \rceil$ означає найменше натуральне число, що більше або дорівнює x.

Теорема 3. $|\Sigma_G| = m - n + 1$.

Теорема 4. Якщо $m = n - 1$ (тобто G є деревом), то $1 \leq |\Lambda_G| \leq n - 1$. Якщо $m > n - 1$, то $0 \leq |\Lambda_G| \leq n - \left\lceil \frac{3}{2} + \sqrt{\frac{9}{4} - 2n + 2m} \right\rceil$, причому всі оцінки є досяжними.

Далі наведемо всі знайдені на цей час оцінки об'єму компонент канонічної визначальної пари.

Теорема 5. Нехай G є деревом. Тоді:

1) $||\Sigma_G|| = 0$;

2) $n \leq ||\Lambda_G|| \leq \left\lceil \frac{n^2+2n}{4} \right\rceil$, причому ці оцінки є досяжними.

Теорема 6. Нехай G не є деревом. Тоді $||\Sigma_G|| \geq 4(m - n + 1)$, причому ця оцінка є досяжною.

Тепер перейдемо до розгляду верхньої оцінки об'єму першої компоненти канонічної визначальної пари для графа, який не є деревом. Розглянемо наступний граф F з n вершинами та m ребрами, який, завдяки його певній схожості з квіткою було названо «графом-квіткою», а його структурним елементам було надано відповідні біологічні назви. Далі позначимо $\delta(n, m) = \left\lceil \frac{3}{2} + \sqrt{\frac{9}{4} - 2n + 2m} \right\rceil$.

Всі вершини графа F було розділено на два класи. До першого класу (суцвіття) входять вершини, які є твірними для слів з Σ_G, показано, що суцвіття містить $\delta(n, m)$ вершин. Другий клас (стебло) містить решту $n - \delta(n, m)$ вершин, у випадку $n = \delta(n, m)$ всі вершини графа F входять до суцвіття. Якщо $n > \delta(n, m)$, то стебло являє собою ланцюг, один кінець якого є ініціальною вершиною, а інший кінець з'єднується ребром з однією з вершин суцвіття (цю вершину у суцвітті було названо квітколожем). Інших ребер між стеблом та суцвіттям не існує. Якщо $n = \delta(n, m)$, то квітколожем стає ініціальна вершина. При цьому суцвіття може бути повним графом, або не бути їм. У другому випадку через $\mu(n, m)$ позначено кількість ребер, додавання яких до суцвіття робить його повним графом.

Було знайдено $\mu(n,m)$. Повний граф $K_{\delta(n,m)}$ містить $\frac{\delta(n,m)(\delta(n,m)-1)}{2}$ ребер, стебло містить $n-\delta(n,m)$ ребер, граф F містить m ребер. Таким чином, $\mu(n,m)=\frac{\delta(n,m)(\delta(n,m)-1)}{2}-(m-n+\delta(n,m))$. Якщо $\mu(n,m)>0$, то відсутні ребра між квітколожем та деякими $\mu(n,m)$ іншими вершинами суцвіття, а між будь-якими іншими вершинами, що входять до суцвіття, ребро існує.

Проілюструємо наведені визначення на рис. 1.2.

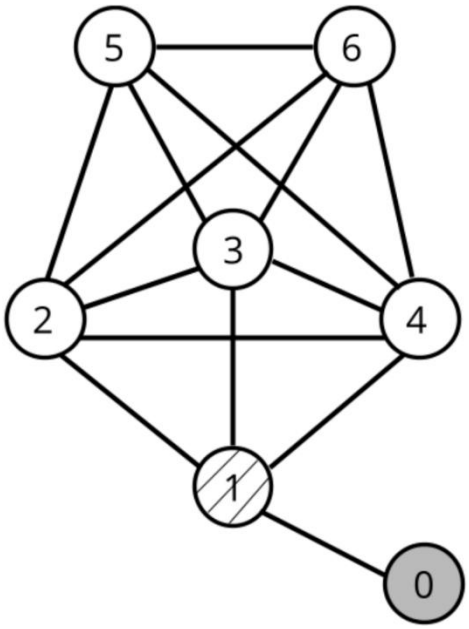


Рис. 1.2. *Граф-квітка, у якого 7 вершин та 14 ребер*

Для «графа-квітки», що зображено на рис. 1.2, $\delta(n,m)=6$, $\mu(n,m)=2$, стебло складається з ініціальної вершини з мікою 0, до суцвіття входять вершини з мітками 1, 2, 3, 4, 5, 6, квітколожем є вершина з міткою 1, у суцвітті відсутні ребра (1, 5) та (1, 6).

Знайдемо $||\Sigma_G||$ для «графа-квітки» F, у якого n вершин та m ребер. Довжина найкоротшого шляху від ініціальної вершини до квітколожа становить $n-\delta(n,m)+1$. Розглянемо два класи вершин у суцвітті. До класу I віднесемо ті вершини суцвіття, що є суміжними вершині-квітколожу, до класу II – ті, що не суміжні вершині-квітколожу (вершина-квітколоже не належить жодному з цих класів). Кількість вершин класу II дорівнює $\mu(n,m)$,

довжина найкоротшого шляху до цих вершин становить $n-\delta(n,m)+3$. Кількість вершин класу I дорівнює $\delta(n,m)-\mu(n,m)-1$, довжина найкоротшого шляху до цих вершин становить $n-\delta(n,m)+2$.

Розглянемо, яким чином утворюються пари твірних вершин у «графі-квітці» F. Неважко бачити, що вершини, які утворюють стебло, та вершина-квітколоже не є твірними для будь-якого слова з Σ_F. Усі можливі пари з вершин класу I є твірними для слів з Σ_F. Кількість таких пар дорівнює $C^2_{\delta(n,m)-\mu(n,m)-1}$, тому сума довжин слів з Σ_F, що утворюються такими вершинами дорівнює

$$\frac{(\delta(n,m)-\mu(n,m)-1)(\delta(n,m)-\mu(n,m)-2)}{2}2(n-\delta(n,m)+2)=$$
$$=(\delta(n,m)-\mu(n,m)-1)(\delta(n,m)-\mu(n,m)-2)(n-\delta(n,m)+2).$$

Усі можливі пари з вершин класу II є твірними для слів з Σ_F. Кількість таких пар дорівнює $C^2_{\mu(n,m)}$, тому сума довжин слів з Σ_F, що утворюються такими вершинами дорівнює

$$\frac{\mu(n,m)(\mu(n,m)-1)}{2}2(n-\delta(n,m)+3)=$$
$$=\mu(n,m)(\mu(n,m)-1)(n-\delta(n,m)+3)$$

Крім того, кожна вершина класу II утворює пару твірних вершин з будь-якою, крім однієї, вершини класу I, тому довжина слів з Σ_F, що утворюються у такий спосіб дорівнює

$$\mu(n,m)(\delta(n,m)-(\mu(n,m)-2))(2n-2\delta(n,m)+5).$$

Отже,

$$||\Sigma_F||=(\delta(n,m)-\mu(n,m)-1)(\delta(n,m)-\mu(n,m)-2)(n-\delta(n,m)+2)+$$
$$+\mu(n,m)(\mu(n,m)-1)(n-\delta(n,m)+3)+$$
$$+\mu(n,m)(\delta(n,m)-(\mu(n,m)-2))(2n-2\delta(n,m)+5).$$

На даний час досліджується знаходження верхньої оцінки об'єму першої компоненти канонічної визначальної пари для будь-якого графа. Найбільша така оцінка, що була знайдена, є у графа-квітки. Наразі розробляються програмні прототипи, які випадковим чином будують Д-граф із заданими

параметрами кількості вершин та ребер і знаходять об’єм об'єму першої компоненти канонічної визначальної пари для такого графа. Проведені тести не заперечили гіпотезу про те, що найбільша шукана оцінка є саме у графа-квітки.

Наведене представлення Д-графів та СД-графів визначальною парою слів може бути корисним при розв'язанні багатьох прикладних задач. При цьому природним чином виникає цілий ряд задач, пов'язаних з цим представленням, серед яких можна виділити наступні:

- знаходження взаємозв'язків між графами G та його редукцією $[G]$;

- розв'язання задачі характеризації пари: для заданого Д-графа та заданої пари визначити, чи є ця пара визначальною для графа без безпосередньої побудови графа за парою;

- знаходження необхідних та достатніх умов для множин C та L, за яких пара $\{C, L\}(x')$ є правильною;

- оптимальний вибір ініціальної вершини, за яким метричні властивості компонент канонічної визначальної пари будуть мінімальними;

- ефективна побудова за $\{\Sigma_G, \Lambda_G\}$ найкоротших за порядком $\preccurlyeq$ шляхів між двома довільними вершинами графа G.

1.4 Висновки до розділу

Отже, в інформаційно-аналітичному розділі розглянуто коротку історію теорії представлення різноманітних математичних об’єктів та структур. У главі 1.1 наведено теорію представлення груп, напівгруп та автоматів, а також окреслено фундаментальні проблеми, які пов’язані з представленням. У главі 1.2 наведено основні поняття та означення, які пов’язані з детермінованими графами, а також виділено напрямки застосування таких графів. У главі 1.3 наведено сучасний стан теорії лінгвістичного представлення детермінованих графів та наведено перспективні напрямки дослідження, пов’язані з лінгвістичного представлення детермінованих графів.

2 СПЕЦІАЛЬНИЙ РОЗДІЛ

2.1 Програмне моделювання алгоритмів

В якості технічної специфікації для програмного моделювання алгоритмів побудови графу та його канонічної визначальної пари визначено роботу [20], зміст якої було розглянуто в інформаційно-аналітичному розділі. Ця робота містить поточні версії понять та алгоритмів за темою лінгвістичного представлення детермінованих графів.

Технічна реалізація виконана з використанням мови програмування Python, та бібліотеки для моделювання мереж NetworkX [22], в якості модуля візуалізації використано бібліотеку Matplotlib [23], а в якості графічного інтерфейсу для демонстрації прикладів, модуль Tkinter (входить до стандартної бібліотеки мови програмування). Крім того слід зазначити, що в окремих випадках, графи можуть бути збережені у форматі файлу .dot або .gml які можуть бути візуалізовані програмним забезпеченням Graphvis [24].

Фактичний прототип реалізує алгоритми в якості окремих підпрограм, які складаються з 3 основних функцій (алгоритмів) описаних в роботі [20], а також допоміжних функцій які забезпечують деякі технічні аспекти, наприклад реалізацію лічільника викликів функції тощо.

Реалізацію модулю з алгоритмами умовно можна розділити на дві частини, перша з яких містить допоміжні функції, друга реалізує основні алгоритми.

Допоміжні функції реалізують додатковий функціонал, який використовується основними алгоритмами на певних етапах їх виконання (рис. 2.1).

```
7    # ========================== HELPERS FUNCTIONS ==============================
8  > def counter(reset=False):
16   ···  return counter.count
17
18
19 > def get_all_leaf_nodes_from_graph(G: nx.Graph) -> dict:
23   ···  return dict(zip(leaf_nodes, node_labels))
24
25
26 > def find_neighbours_with_the_same_labels(nghb: List, lbls: List) -> Dict:
38   ···  return equal_elements
39
40
41 > def walk_by_word(graph: nx.Graph,
56   ···  return current_node
57
58
59 > def check_q_node(graph: nx.Graph,
86   ···  return state
87
88
89 > def word_pair_data_validation(c_tuple: Tuple[str],
108  ···  return True
```

Рис. 2.1. *Нотація допоміжних функцій*

Наведемо функціонал допоміжних функцій, які наведено на рис. 2.1:

1. counter(reset) – реалізація лічильника для генерації унікальних ідентифікаторів вершин графа. В якості аргумента приймає булеве значення, яке відповідає за зведення на нуль значення лічильника.

2. get_all_leaf_nodes_from_graph() – модифікація бібліотечної функції, яка повертає словник висячих вершин графа, з їх мітками. В якості аргумента приймає граф з розміченими вершинами.

3. find_neighbours_with_the_same_labels() – функція використовується алгоритмом редукції, повертає словник з елементами вершин які містять однакові мітки. В якості аргумента приймає список суміжних вершин та список їх міток.

4. walk_by_word() – функція реалізує обхід графа за зазначеним словом, в якості аргумента приймає граф, ідентифікатор початкової вершини, та слово за яким відбувається обхід. Використовується алгоритмом АП в контексті валідації доданих вершин.

5. check_q_node() – функція перевірки валідності вершини, згідно специфікації алгоритму АП. Приймає аргументи: граф, список множини L, ідентифікатор та мітку перевіряємої вершини, початкову вершину графа.

6. word_pair_data_validation() – функція реалізує валідацію початкових даних в алгоритмі АП, перевіряє множину слів на відповідність зазначеним параметрам згідно вимог побудови визначальної пари.

Друга частина модулю функцій містить основні алгоритми АП, АК, АР та функцію метріки визначальної пари. Наведемо опис функціоналу цих функцій та їх нотацію (рис. 2.2):

```
111     # ======================== ALGORITHMS REALIZATION ============================
112   > def ar_nodes(graph: nx.Graph) -> nx.Graph:
134     ... return G_
135
136
137   > def ap_graph(C:Tuple[str], L:Tuple[str], x_='1') -> Union[nx.Graph, str]:
196     ... return G
197
198
199   > def ac_pair(graph: nx.Graph) -> Union[Tuple[List[str], List[str]], int, str]:
239     ... return (sigma_g, lambda_g)
240
241     # ======================== CANONICAL PAIR METRICS ============================
242   > def get_canonical_pair_metrics_from_graph(graph: nx.Graph) -> dict:
270     ... }
```

Рис. 2.2. *Нотація функцій алгоритмів*

1. ar_nodes() – функція реалізує алгоритм редукції АР: виконує розбиття графу та видаляє вершини з однаковими мітками, які суміжні однієї вершині, залишаючи лише одну таку вершину. В якості аргумента приймає об’єкт графу *G*, повертає модифікований об’єкт.

2. ap_graph() – функція реалізує алгоритм побудови графа за визначальною парою слів. В якості аргументів приймає кортеж з двома множинами слів, які описують всі цикли графа та його висячі вершини. В результаті виконання функція або повертає побудований граф у вигляді

об’єкта бібліотеки NetworkX, або повертає рядок з повідомленням, що граф побудувати неможливо.

3. ac_pair() – функція реалізує алгоритм побудови канонічної пари слів з існуючого графу. В якості аргумента функція приймає граф з розміченими вершинами.

4. get_canonical_pair_metrics_from_graph() – функція приймає об’єкт графа та повертає словник з метричними властивостями графа, а також його канонічної пари слів, що задають цей граф.

Технічна реалізація коду виконана з дотриманням вимог специфіки мови програмування Python яка перевірялась спеціальними модулями - статичного та синтаксичного аналізатора MyPy та Pylint. Якість коду, згідно цих перевірок, програмно оцінена як 9.3 / 10 за стандартом PEP8 [25]. Ця оцінка не є максимально можливою завдяки наявності в функціях збільшеної кількості, замість рекомендованої, локальних змінних, а також використання назв змінних у вигляді констант іменованих однією літерою, наприклад G, що є не рентабельним при написанні складних програмних модулів, але ця реалізація обрана свідомо, для відповідності термінам, використаним у роботі, яку обрано в якості специфікації.

Результати оцінки коду подано на рис. 2.3.

```
************* Module alglib
alglib.py:19:34: C0103: Argument name "G" doesn't conform to snake_case naming style (invalid-name)
alglib.py:114:4: C0103: Variable name "G_" doesn't conform to snake_case naming style (invalid-name)
alglib.py:137:13: C0103: Argument name "C" doesn't conform to snake_case naming style (invalid-name)
alglib.py:137:27: C0103: Argument name "L" doesn't conform to snake_case naming style (invalid-name)
alglib.py:137:41: C0103: Argument name "x_" doesn't conform to snake_case naming style (invalid-name)
alglib.py:137:0: R0914: Too many local variables (22/15) (too-many-locals)
alglib.py:142:4: C0103: Variable name "q" doesn't conform to snake_case naming style (invalid-name)
alglib.py:146:4: C0103: Variable name "G" doesn't conform to snake_case naming style (invalid-name)
alglib.py:159:8: C0103: Variable name "G" doesn't conform to snake_case naming style (invalid-name)
alglib.py:161:4: C0103: Variable name "q" doesn't conform to snake_case naming style (invalid-name)
alglib.py:173:8: C0103: Variable name "G" doesn't conform to snake_case naming style (invalid-name)
alglib.py:137:0: R0912: Too many branches (20/12) (too-many-branches)
alglib.py:223:4: C0103: Variable name "ni" doesn't conform to snake_case naming style (invalid-name)

-------------------------------------------------------------------
Your code has been rated at 9.30/10 (previous run: 9.30/10, +0.00)
```

Рис. 2.3. *Метричні оцінки якості коду*

Оцінимо реалізацію модулю з цими алгоритмами на програмну складність та тестованість за допомогою бібліотеки Radon. [26].

Обчислимо цикломатичну складність алгоритмів. Цикломатична складність (метрика МакКейба) – це метрика програмної складності, яка вимірює кількість лінійно незалежних шляхів через програму, вона була запропонована Томасом МакКейбом у 1976 році [27]. Цикломатична складність часто використовується для оцінки складності коду і може бути корисною при тестуванні програмного забезпечення. Вважається, що чим вища цикломатична складність, то складнішим є тестування та аналіз коду, тому рекомендовано реалізовувати програмний код у такий спосіб, щоб він мав низькі значення цієї метрики, а саме був більш зрозумілим і легко піддавався тестуванню.

На рис. 2.4 наведено шкалу складності за метрикою МакКейба та значення цієї метрики для програмного модуля, що розглядається в даній роботі. Всі показники знаходяться на достатньому рівні, чотири з них знаходяться на найкращому рівні, з чого можна зробити висновки, що код піддається тестуванню та підтримці.

CC score	Rank	Risk
1 - 5	A	low - simple block
6 - 10	B	low - well structured and stable block
11 - 20	C	moderate - slightly complex block
21 - 30	D	more than moderate - more complex block
31 - 40	E	high - complex block, alarming
41+	F	very high - error-prone, unstable block

```
F 137:0 ap_graph - C (20)
F 199:0 ac_pair - C (15)
F 59:0 check_q_node - C (13)
F 89:0 word_pair_data_validation - C (12)
F 112:0 ar_nodes - B (8)
F 26:0 find_neighbours_with_the_same_labels - A (5)
F 19:0 get_all_leaf_nodes_from_graph - A (4)
F 41:0 walk_by_word - A (4)
F 242:0 get_canonical_pair_metrics_from_graph - A (4)
```

Рис. 2.4. *Оцінки складності за метрикою МакКейба*

Обчислимо метрики Halstead Complexity (складність Холстеда). Ці метрики складаються з таких параметрів:

1. h1: Довжина програми Холстеда (Загальна кількість операторів та операндів у програмі).
2. h2: Словник Холстеда (Загальна кількість унікальних операторів та операндів у програмі).

3. N1: Приблизна довжина програми (Оцінка розміру програми після впровадження).

4. N2: Приблизний словник програми (Оцінка кількості унікальних операторів та операндів після впровадження).

5. vocabulary: Загальний словник (Сума унікальних операторів та операндів).

6. length: Загальна довжина програми (Сума операторів та операндів).

7. calculated_length: Розрахована довжина на основі метрик Холстеда.

8. volume: Об'єм програми (Міра розміру програми).

9. difficulty: Складність програми (Міра складності розуміння програми).

10. effort: Зусилля для розробки програми (Міра кількості зусиль, необхідних для розробки та підтримки програми).

11. time: Приблизний час для розробки програми.

12. bugs: Приблизна кількість помилок в програмі.

Ці метрики надають інформацію щодо складності та розміру коду в файлі. Нижчі значення для difficulty, effort та bugs взагалі вважаються кращими, оскільки вони вказують на простіший і, можливо, більш здатний для підтримки код. Зауважимо, що ці метрики є евристичними вимірюваннями і слід враховувати їх у контексті конкретного коду та вимог проекту. На рис. 2.5 наведено значення параметрів цих метрік для алгоритмів АП (ap_graph) та АК (ac_pair).

```
ap_graph:
    h1: 8
    h2: 22
    N1: 19
    N2: 35
    vocabulary: 30
    length: 54
    calculated_length: 122.10749561002054
    volume: 264.97209216286
    difficulty: 6.363636363636363
    effort: 1686.1860410363818
    time: 93.67700227979898
    bugs: 0.08832403072095334
```

```
ac_pair:
    h1: 8
    h2: 20
    N1: 16
    N2: 27
    vocabulary: 28
    length: 43
    calculated_length: 110.43856189774725
    volume: 206.71626164847697
    difficulty: 5.4
    effort: 1116.2678129017756
    time: 62.01487849454309
    bugs: 0.06890542054949232
```

Рис. 2.5. *Оцінки складності за метриками Холстеда*

Всі параметри метрик Холстеда для алгоритма AP (ar_nodes) дорівнюють нулю, це вказує на те, що ця функція відносно проста і має мінімальний обсяг роботи та складність. З отриманих результатів можна бачити, що наведені алгоритми мають рівень вище середнього для їх розуміння, та високий коефіцієнт необхідних зусиль для їх реалізації, але при цьому мають гарний показник приблизної кількості помилок який прямує до нуля.

Тестування зазначених алгоритмів проводилося у мануальному режимі, згідно розроблених прикладів даних, які зазначено у роботі [20], обраної в якості специфікації. Для демонстрації валідації алгоритму побудови графу, який має працювати в множині алфавіту міток (символів юнікоду), в якості тестового прикладу було обрано дані з [20]. У випадку негативного тесту в якості визначальної пари зазначено набір цифр, та свідомо внесена помилка в нотацію цієї пари. Очікуваний результат - повідомлення про помилку виконання, та повідомлення – “граф не існує”. В якості позитивного сценарію обрано аналогічну порядкову послідовність літер англійського алфавіту з коректною нотацією. Наведемо приклади тестових даних (рис 2.6) та результат виконання негативного (рис 2.7) та позитивного тесту (рис 2.8).

```
# TEST DATA
# ================================================
testC1 = ("153521", "152431")
testL1 = ("1342531", "123241", "13412", "1523")
# 1 2 3 4 5
# a b c d e
testC1a = ("aeceba", "aebdca")
testL1a = ("acdbeca", "abcbdab", "acdab", "aebc")
```

Рис. 2.6. *Тестові дані*

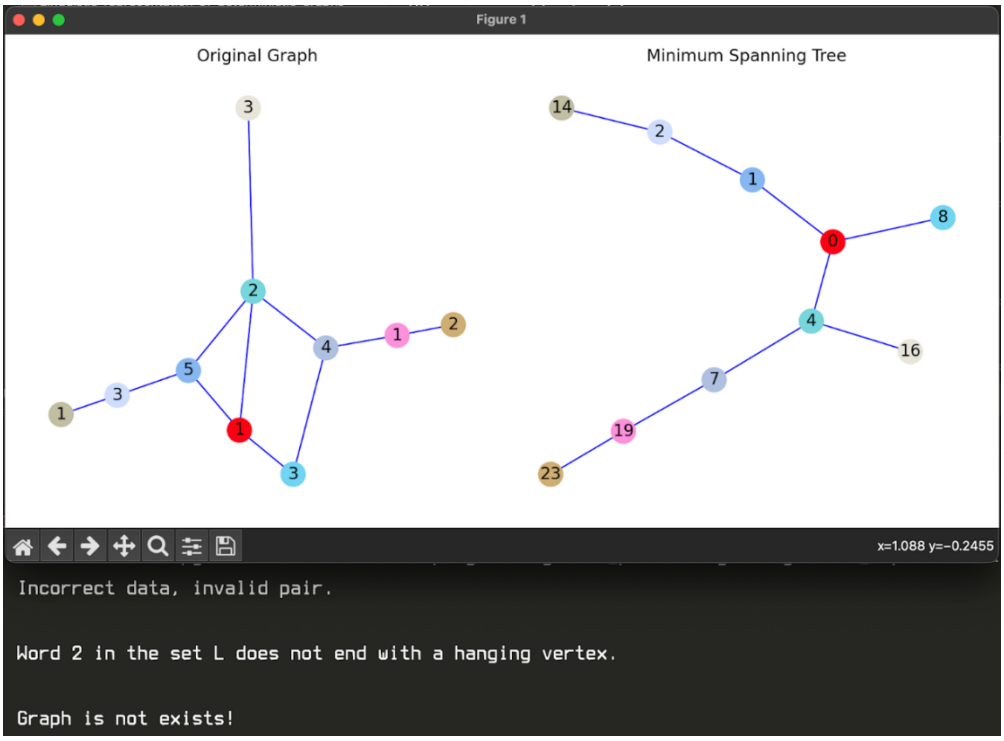


Рис. 2.7. *Негативний тест*

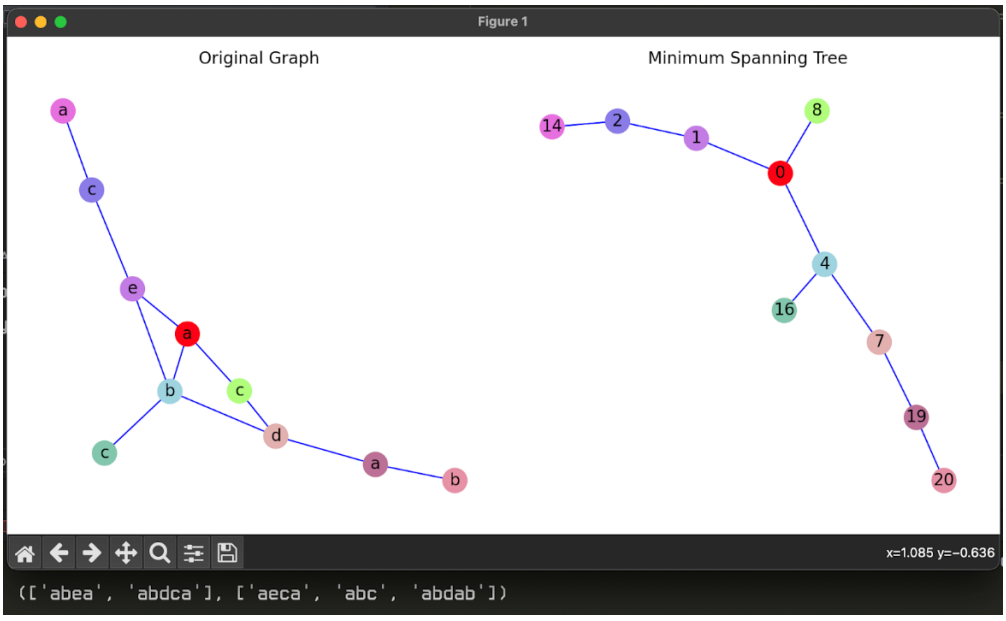


Рис. 2.8. *Позитивний тест*

Валідація алгоритму отримання метричних значень канонічної визначальної пари частково проводилася за допомогою алгоритмів які генерують складні графи, такі графи в процесі розробки та аналізу було названо “квітками” за їх характерну схожість. Алгоритм будує граф з 50 вершинами, за заданими параметрами. Розробка та верифікація алгоритму та пов’язаної з ним теоретичної частини зараз знаходиться в процесі розробці, та представляє собою подальший напрямок досліджень та розвитку роботи.

Приклад згенерованих тестових даних подано на рис. 2.9

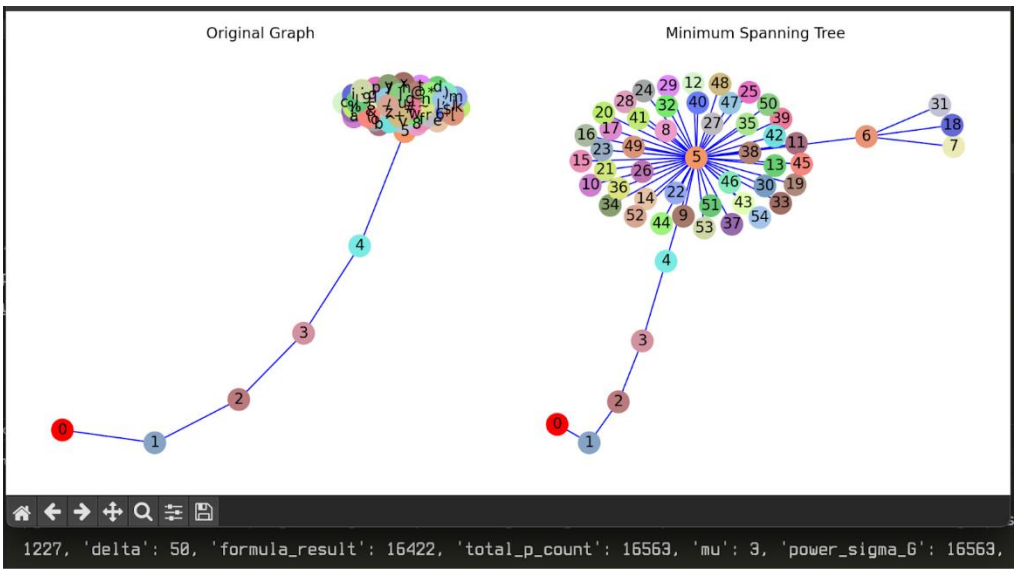


Рис. 2.9. *Приклад тестових даних функції метрики*

Для зручності опрацювання результатів обчислень алгоритмів було розроблено прототипи графічних інтерфейсів, desktop та web-інтерфейс, який наразі знаходиться на стадії доведення та тестування. Desktop версія містить прості елементи введення початкової вершини та множини слів визначальної пари, а також вікно з результатами отриманих метрик канонічної визначальної пари (рис. 2.10).

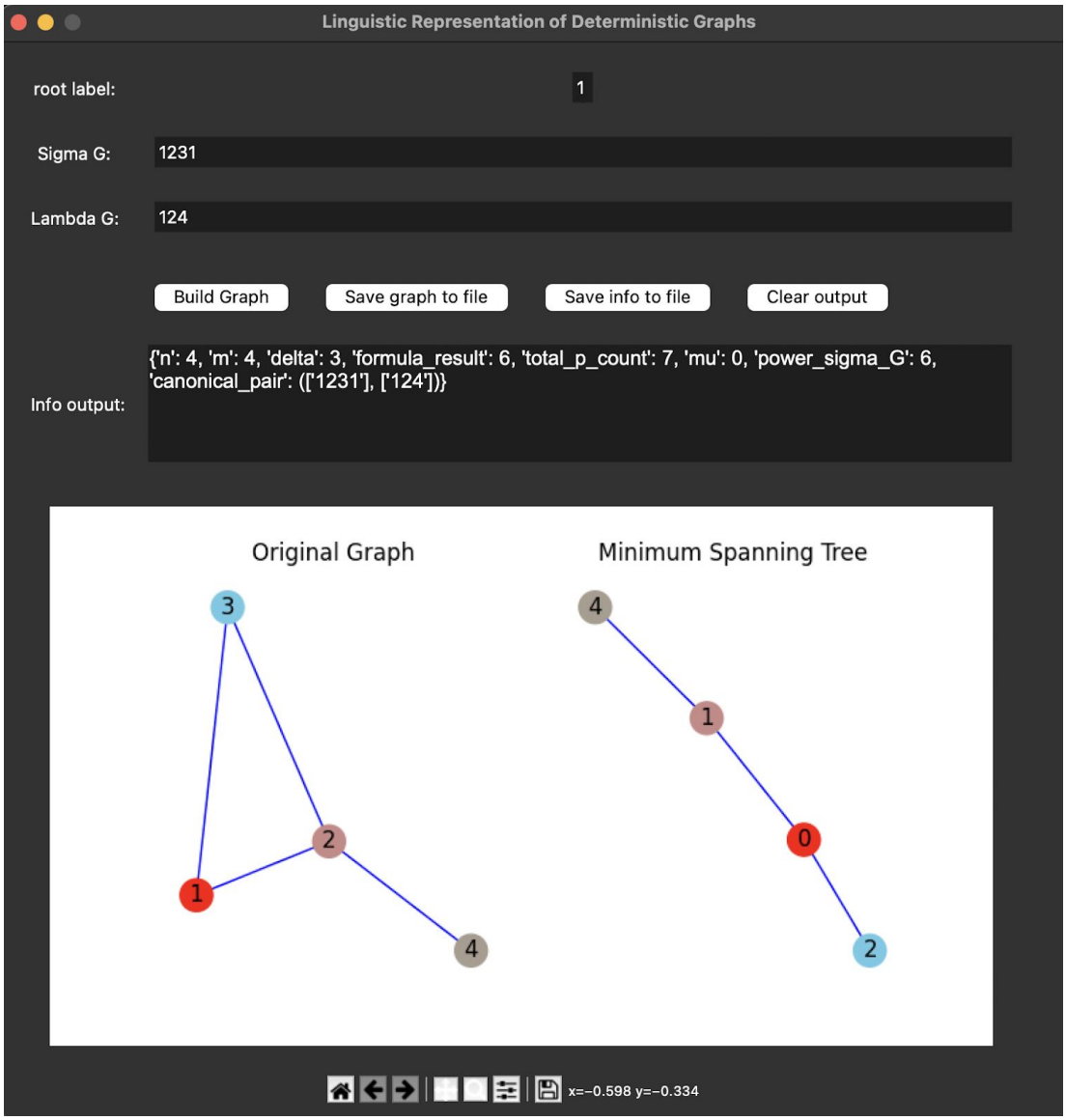


Рис. 2.10. *Desktop interface для роботи із зазначеними алгоритмами*

Слід зазначити, що розробка інтерфейсів не є цільовою задачею роботи, тому вважаємо, що прототип інтерфейсу може бути реалізований за будь-якою сучасною технологією представлення графів. В даному випадку використовується бібліотека matplotlib зі стандартного пакету інструментів візуалізації мови Python. За цієї причини програмний код інтерфейсів в роботі не наводиться, проте ці прототипи можна знайти у [28].

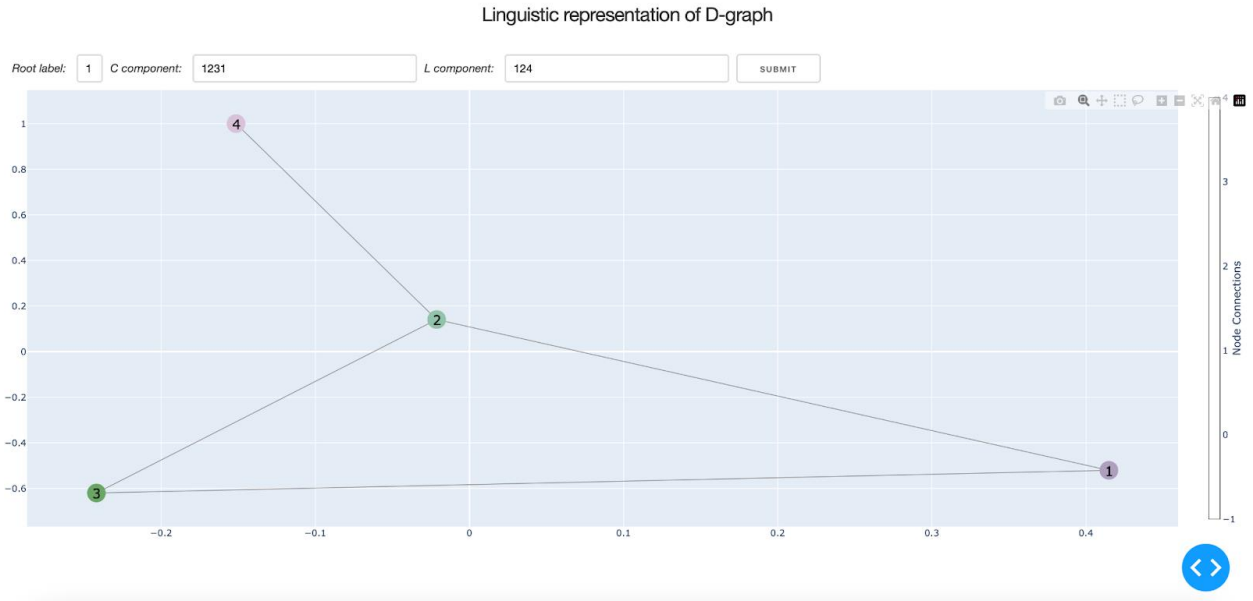


Рис. 2.11. *Прототип web-інтерфейсу*

Крім того, графи можуть бути збережені у файлах з розширенням .dot або .gml для їх подальшого представлення у вигляді зображень за допомогою програмного забезпечення graphviz. Нотація графа в такому вигляді фактично задає граф у класичному виді - за допомогою словника (рис. 2.12).

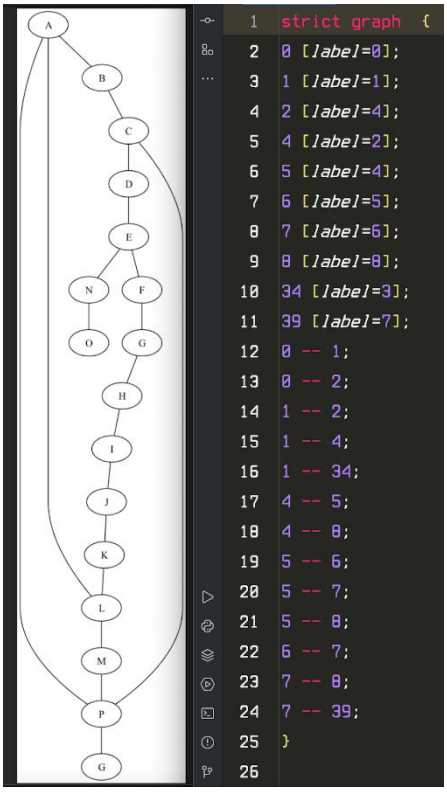


Рис. 2.12. *Задання графа у файлі з розширенням .dot*

Зображення графа у вигляді файлу з розширенням .dot або .gml дозволяє ефективно використовувати побудований граф у різних сучасних програмних додатках та середовищах програмування.

2.2 Застосування розроблених алгоритмів у тестуванні програмного забезпечення на прикладі побудови схеми тестового покриття прототипу web-додатку

На сьогоднішній день при тестуванні програмного забезпечення складною проблемою залишається питання якісного планування процесів валідації та оцінки якості побудови тестового покриття для програм з високим рівнем складності. Існує багато методик планування тестової активності, але єдиний, стандартний підхід відсутній, при цьому ймовірність того, що такий підхід взагалі буде колись розроблено та широко застосовано, є низькою. У зв’язку з цим, вважається доречним провести огляд деяких існуючих методик побудови тестового покриття та доповнити їх механізмом побудови моделі покриття, що заснована на лінгвістичному представленні детермінованих графів. Завдяки цієї моделі може бути полегшено процес якісної її оцінки та стати більш зрозумілим подальший напрямок дій, направлених на побудову набору автоматизованих тестів. Також передбачається можливість отримання кількісної оцінки автоматизованих тестів які відповідають вимогам до тестування певного програмного забезпечення.

Розглянемо деякі підходи до тестування програмного забезпечення, які з нашої точки зору можуть бути розширені, за допомогою розробленої теорії лінгвістичного представлення детермінованих графів.

Методика покриття за сценаріями використання (Use Case Coverage), дає змогу перевірити, як програма працює в конкретних ситуаціях. Підхід до реалізації цієї методики полягає у створенні тестів, орієнтованих на

конкретні сценарії використання програми або системи. Ця методика дає змогу переконатися, що програма працює коректно в реальних ситуаціях, з якими можуть зустрітися користувачі. Перелік можливих дій цієї методики включає:

1. *Ідентифікація сценаріїв використання*. Спочатку необхідно ідентифікувати основні сценарії, які користувачі виконуватимуть за допомогою програми. Ці сценарії часто описуються у вигляді сценаріїв використання або use cases.

2. *Розробка тестових випадків*. Для кожного сценарію використання створюються тестові випадки. Ці випадки мають містити послідовність кроків, які користувач повинен виконати, щоб досягти певної мети. Важливо врахувати різні аспекти: введення даних, навігація, очікуваний висновок тощо.

3. *Розгляд різних варіантів виконання*. Для кожного сценарію необхідно розглянути різні варіанти виконання. Це може включати в себе різні комбінації вхідних даних, умови та альтернативні шляхи.

4. *Покриття всіх основних функціональних блоків*. Необхідно переконатися, що кожен основний функціональний блок програми тестується в рамках сценарію використання.

5. *Обробка виняткових ситуацій*. Створення тестів, які перевіряють обробку помилок або виняткових ситуацій. Таке створення повинно враховувати коректну поведінку програми при введенні некоректних даних, помилкових варіантів введення тощо.

6. *Перевірка взаємодії із зовнішніми системами*. Якщо програма взаємодіє із зовнішніми системами (наприклад, базою даних, API тощо), треба переконатися, що тести покривають ці взаємодії.

7. *Автоматизація тестування*. Після розробки тестових випадків вони можуть бути автоматизовані, щоб забезпечити повторюваність і зручність тестування.

8. *Виконання тестів*. Після автоматизації тести можуть бути запущені, а результати будуть аналізуватися для виявлення будь-яких помилок або неполадок.

Методика покриття за сценаріями використання дає змогу зосереджуватися на реальних сценаріях, які відтворюють користувачі, що робить цей підхід важливим для забезпечення коректності та надійності програми.

Методика покриття тестів за вікнами (Window Testing): Цей підхід орієнтований на тестування користувацького інтерфейсу програми, такого як вікна, діалогові вікна, форми тощо. Метою є перевірка, чи всі елементи інтерфейсу функціонують коректно, та чи може користувач взаємодіяти з ними відповідно до очікувань. Опис цієї методики включає кроки:

1. *Ідентифікація елементів інтерфейсу*. Необхідно ідентифікувати всі елементи користувацького інтерфейсу, з якими користувач взаємодіятиме. Це можуть бути вікна, кнопки, поля введення, чекбокси, списки тощо.

2. *Розробка тестових випадків*. Для кожного елемента інтерфейсу розробляються тестові випадки. Ці випадки описують послідовність дій, які користувач має виконати з цим елементом.

3. *Врахування різних станів елементів інтерфейсу*. Кожен елемент інтерфейсу має тестуватися в різних станах, таких як активний, неактивний, ввімкнений, вимкнений тощо.

4. *Тестування взаємодії елементів інтерфейсу*. На цьому кроці відбувається перевірка взаємодії між різними елементами інтерфейсу, наприклад, необхідно переконатися, що при натисканні на кнопку виконується правильна дія.

5. *Тестування введення даних*. Елементи інтерфейсу, що приймають участь у введенні даних, мають приймати коректні дані та реагувати на можливу некоректність введення даних.

6. *Тестування навігації*. Перевірка навігації по додатку з використанням елементів інтерфейсу, таких як меню, вкладки тощо.

7. *Тестування адаптивності*. Інтерфейс має коректно відображатися і працювати на різних пристроях і роздільній здатності екранів.

8. *Тестування спеціальних сценаріїв*. Включає в себе перевірку додаткових сценаріїв, таких як обробка помилок, переходи між вікнами та інші специфічні випадки.

9. *Автоматизація тестування* (за можливості). Після розробки тестових випадків, вони можуть бути автоматизовані для підвищення ефективності та повторюваності тестування.

Ця методика особливо важлива для програм із графічним інтерфейсом, таких як додатки з GUI (графічним користувацьким інтерфейсом), де коректна робота елементів інтерфейсу відіграє вирішальну роль.

Наступні приклади ідеї розширення зазначених методик, запропонованою теорією лінгвістичного представлення детермінованих графів, будуть демонструватися на прикладі графічного інтерфейсу web-додатку - прототипу інтернет магазина, з використанням безкоштовної CMS (системи управління контентом) OpenCart.

Теорія графів може бути дуже корисною під час розроблення тестового покриття, особливо для програм зі складною логікою або великою кількістю взаємозв'язків між компонентами. Ось кілька способів застосування теорії графів у розробці тестового покриття:

1. *Побудова графа шляхів виконання (Control Flow Graph)*. Полягає у побудові діаграми, яка являє собою візуальне представлення всіх шляхів виконання в програмі. Вершини графа представляють блоки коду, а ребра показують переходи між ними. Аналізуючи граф можна виявити основні шляхи виконання, що допоможе в розробці тестів, які покривають різні гілки та умови.

2. *Ідентифікація критичних шляхів (Critical Paths)*. Тести, спрямовані на покриття критичних шляхів, можуть бути особливо корисними, оскільки вони охоплюють найбільш важливі частини програми.

3. *Визначення шляхів, що потребують уваги (Paths of Interest)*. Іноді існують специфічні шляхи виконання, які становлять особливий інтерес (наприклад, важливі для бізнес-логіки або потенційно проблемні). Використовуючи теорію графів, можна ідентифікувати ці шляхи і написати тести, що перевіряють їх коректність.

4. *Покриття взаємозв'язків (Edge Coverage)*. Граф може допомогти візуалізувати зв'язки між різними компонентами програми. Це дає змогу оцінити, наскільки тести покривають усі можливі взаємозв'язки.

5. *Тестування шляху повернення (Backward Testing)*. Використовуючи теорію графів, можна виявити шляхи, які ведуть до повернення до попередніх станів програми. Це важливо для перевірки правильної роботи механізмів відкату або відновлення.

6. *Розбиття графа на модулі (Modularization of Graphs)*. Якщо програма складається з різних модулів, то кожен модуль може бути представлений окремим підграфом. Це дає змогу розробляти тести для кожного модуля окремо, забезпечуючи більш глибокий рівень тестування.

Використання теорії графів допомагає структурувати та візуалізувати логіку програми, що полегшує аналіз шляхів виконання та дає змогу розробити ефективне тестове покриття.

Теорія лінгвістичного представлення детермінованих графів може бути використана під час тестування конкретного програмного забезпечення. Виходячи з факту наявності документації або списку технічних вимог до нього, можна визначити список станів, вікон програми, шляхів виконання, URL адрес (роутів) тощо, що підлягають тестуванню. При цьому вважаємо, що головне вікно або стан програми мають бути однозначно визначеними та включеними в процес тестування. Далі вважаємо, що кожному вікну програми чи її стану, що підлягає тестуванню, буде однозначно відповідати деяка вершина детермінованого графа, а переходу між вікнами чи станами – ребро між відповідними вершинами графа. Головному вікну або стану програми буде відповідати ініціальна вершина графа.

Ґрунтуючись на цій інформації можна скласти наступний перелік дій:

1. Нехай $X = \{x_1, \dots, x_n\}$ – деяка скінченна множина (алфавіт) міток, потужність якої не меньша, ніж кількість вікон чи станів, що підлягають тестуванню. Кожному вікну (стану), яке тестується, присвоїти певну унікальну мітку з алфавіту X. Через x' позначимо мітку, що відповідає головному вікну або стану програми.

2. Скласти наскрізні тестові сценарії (end-to-end test cases) [29], що однозначно передбачають переходи між станами або вікнами. Кожен такий сценарій буде словом із набору міток. Ці сценарії мають бути однозначно розділені на дві групи: а) циклічні сценарії; б) сценарії, що ведуть до помилки або унеможливлюють перехід куди-небудь крім переходу до попереднього вікна або стану. При цьому для кожного випадку помилки або неможливості переходу куди-небудь крім переходу до попереднього вікна або стану повинен бути сценарій, який закінчується у цьому вікні чи стані.

3. Позначимо через C множину сценаріїв з групи (а), а через L – множину сценаріїв з групи (б).

4. За допомогою алгоритму АП побудувати граф $G = G(\{C, L\}(x'))$. У цьому графі кожен сценарій групи (б) відповідає деякому шляху, що веде від ініціальної вершини графа G до деякої його висячої вершини.

За умови коректної побудови множин C та L граф G існує та визначається однозначно, оскільки кроки (3) та (4) алгоритму АП, які призводять до неможливості існування графа $G(\{C, L\}(x'))$, не можуть виконуватися за умови переходу кожного сценарію групи (б) саме так, як було визначено. Якщо ж виявилося, що за сформованними на кроках (2) та (3) поточного алгоритму множинами C та L неможливо побудувати граф $G\big(\{C, L\}(x')\big)$, то свідчить про те, що множини C та L складені некоректно, та треба правильність їх складання.

Побудований граф являє собою графічне зображення карти тестового покриття програмного забезпечення. Воно може використовуватись для подальшого аналізу та оцінки необхідної кількості автоматизованих тестів.

Набір таких тестів може бути побудований як для кожної вершини графу, так і для переходів між вершинами у вигляді end-to-end тестів з використанням, популярної на сьогоднішній день, архітектури проектування Page-object. Для цієї архітектури згідно з отриманою моделлю графа, може бути також побудована діаграма класів з використанням нотацій мови моделювання UML.

Алгоритм АК перетворює набір сценаріїв (у вигляді слів, які складаються з набору міток вершин) на оптимальний набір переходів без повторень. Цей підхід забезпечує оптимізацію представлення тестових сценаріїв у вигляді звичайного тексту, з якого не тільки може бути отримана інформація про тестовий сценарій але й може бути побудований граф, який повністю відповідає вимогам тих сценаріїв, які були створені тестувальником до оптимізації їх представлення.

Підсумовуючи зазначену інформацію можна вивести основну мету запропонованих алгоритмів:

Мета алгоритму АП в зазначеній послідовності дій – побудова моделі (карти) тестового покриття програмного забезпечення, що тестується, на основі користувацьких сценаріїв переходів між вікнами (станами), що позначені унікальними мітками кожного вікна (стану програмного забезпечення, заздалегідь відомих із вимог або реалізації) у вигляді графа.

Мета алгоритму АК – оптимізація обсягу (кількості сценаріїв) та їх представлення у вигляді оптимізованої текстової послідовності. В результаті аналізу цієї оптимізованої послідовності також можна прийняти рішення про зменшення або модифікацію існуючих сценаріїв.

Наведена теорія лінгвістичного представлення детермінованих графів може бути використана для розширення зазначених вище методів тестування - Use Case Coverage, Window Testing на певних етапах їх виконання.

Наведемо приклад використання зазначених алгоритмів при побудові тестового покриття для web-додатку, прототипу інтернет магазину OpenCart:

Побудуємо перелік вікон програми (див. Додаток Д) та призначимо їм відповідні мітки, а також складемо приклади тестових сценаріїв:

0 – головна сторінка;

1 – сторінка авторизації користувача;

2 – авторизація успішна (сторінка налаштувань аккаунту користувача);

3 – авторизація неуспішна (валідаційне повідомлення про помилку);

4 – сторінка перегляду товарів;

5 – продукт додано в корзину (відображення наповнення корзини);

6 – сторінка, яка відображає місткість корзини;

7 – сторінка створення платежу;

8 – домашня сторінка, користувача авторизовано.

Вважаємо, що побудова всіх тестових сценаріїв починається з головної сторінки сайту. Стан 0.

Складемо циклічні сценарії користувача:

I. Тестування навігаційної панелі, можливість переходів між сторінками неавторизованого користувача.

1. Перейти на сторінку перегляду товарів. (4)
2. Перейти на сторінку авторизації користувача. (1)
3. Повернутися на головну сторінку сайту. (0)

Отриманий сценарій: 0410 (рис. 2.13).

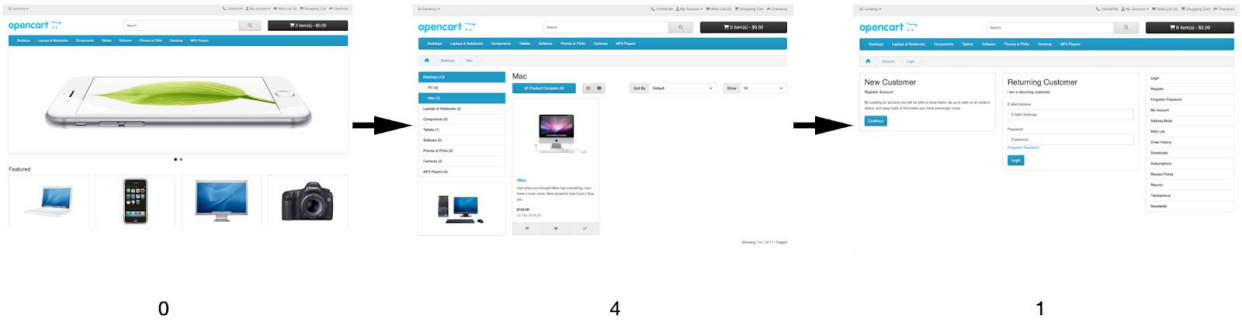


Рис. 2.13. *Приклад переходів у тестовому сценарію 0410*

II. End-to-end тести, авторизація користувача, додавання товарів в корзину, продовження вибору товарів.

1. Перейти на сторінку авторизації користувача. (1)
2. Авторизація успішна. (2)

3. Перейти на сторінку перегляду товарів. (4)

4. Додати товар в корзину. (5)

5. Перейти на сторінку яка відображає місткість корзини. (6)

6. Повернутися до перегляду товарів (4)

Отриманий сценарій: 0124564210.

Слід зазначити, що в цьому тесті можливе тестування більшої кількості сценаріїв, які не приймаються до уваги в цілях краткості та зрозумілості описання методики. Очевидно, що описаний сценарій може повертати нас до деавторизації користувача, та повернення до стартової сторінки. Також в цьому сценарії можливі варіанти переходу до домашньої сторінки сайту авторизованим користувачем, що надає додатковий функціонал сторінки, стан якої описано міткою (8). Тому зазначимо додаткові можливі сценарії, які наводяться в якості прикладу даної роботи у вигляді послідовностей міток: 0124568210, 01245684210.

Отже, маємо першу компоненту *C* (Sigma G) яка складається зі слів: 0410, 0124564210, 0124568210, 01245684210.

Складемо сценарії користувача які ведуть в висячі вершини:

I. Негативний сценарій авторизації.

1. Перейти на сторінку авторизації користувача. (1)

2. Ввести невалідні дані та отримати вікно валідаційного повідомлення про помилку авторизації. (3)

Отриманий сценарій: 013 (рис. 2.14).

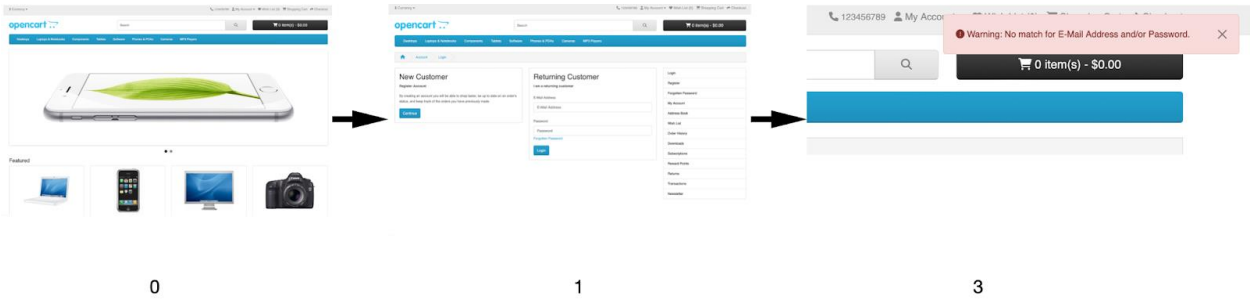


Рис. 2.14. *Приклад переходів у тестовому сценарію 013*

II. end-to-end сценарій покупки з оплатою.

1. Виконати послідовність переходів по міткам 12456

2. Перейти до сторінки оформлення замовлення. (7)

Отриманий сценарій 0124567, або, якщо продукти заздалегідь додані до корзини, маємо 012467, або 0124867, або 012867.

Отже маємо другу компоненту *L* (Lambda G) яка складається зі слів: 013, 0124567, 012467, 0124867, 012867.

В результаті складання тестових сценаріїв отримуємо визначальну пару слів, за якою можна побудувати граф, а саме:

Sigma G: 0410, 0124564210, 0124568210, 01245684210.

Lambda G: 013, 0124567, 012467, 0124867, 012867.

Неважко бачити, що деякі сценарії можуть повторюватись, фактично в процесі складання визначальної пари присутній людський фактор. Тому, застосувавши алгоритм побудови канонічної визначальної пари АК до отриманих даних, можна перетворити дані на більш простий, оптимізований вигляд:

Sigma G: ['0140', '01248210', '0124564210', '012468210']

Lambda G: ['013', '012467']

За таким виглядом даних можна побудувати аналогічний граф, який буде отримано з не оптимізованої послідовності (рис. 2.15).

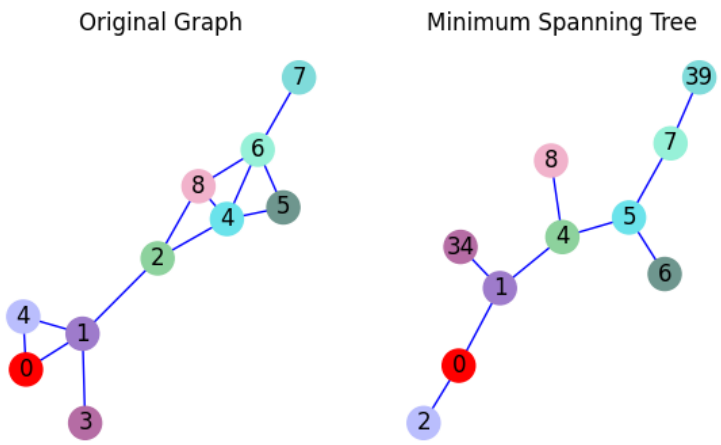


Рис. 2.15. *Граф, який побудовано за допомогою визначальної пари*

Крім того отриманий граф може бути представлений у вигляді списку, та dot нотації, для його подальшого використання та представлення у іншому програмному забезпеченні (рис. 2.15).

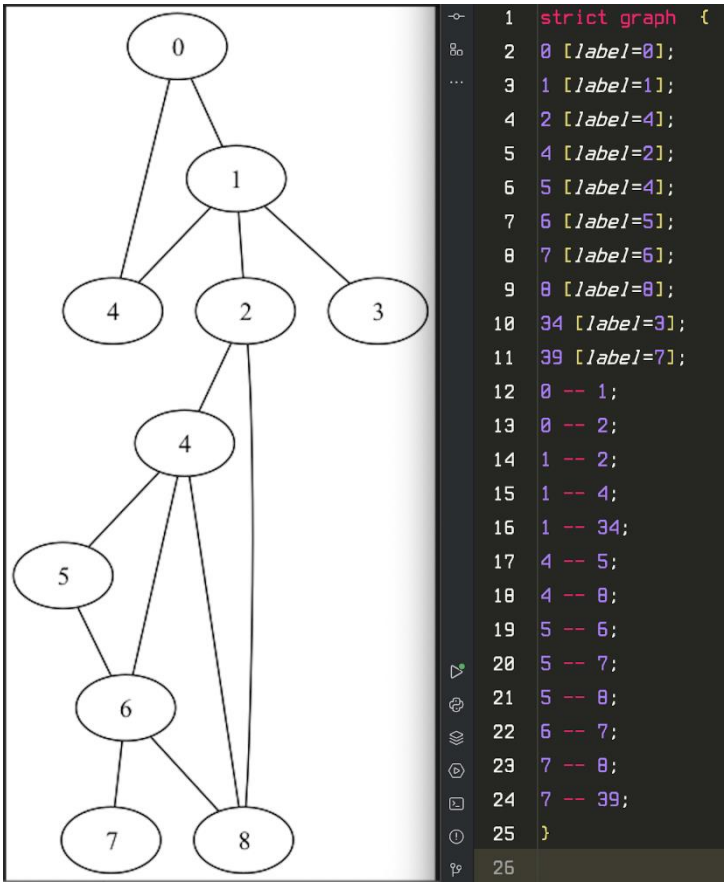


Рис. 2.16. *Зображення одержаного графу у вигляді dot нотації*

Отримана карта графа представляє собою граф шляхів виконання (Control flow graph), який може використовуватись для оцінки та моделювання тестового покриття. За допомогою його можна планувати архітектуру автоматизованих тестів, які будуть відтворюватися для зазначеного програмного забезпечення.

Неважко бачити, що кожна вершина графу може являти собою клас програмного коду, який буде описувати функціональність сторінки або певного стану. Діаграму взаємодії класів можна побудувати за допомогою UML нотації. Архітектуру цих класів, їх порядок створення, взаємодію, тощо заздалегідь можна оцінити з отриманого графу.

Наприклад, умовну імплементацію сценарію 013 можна представити у вигляді діаграми класів UML, що демонструють елемент архітектурного патерну Page Object, де класи сторінок застосунку наслідуються від базового класу сторінки, в якому зазначені службові методи, які можуть використовувати всі екземпляри класів, наприклад, пошук елементів тощо.

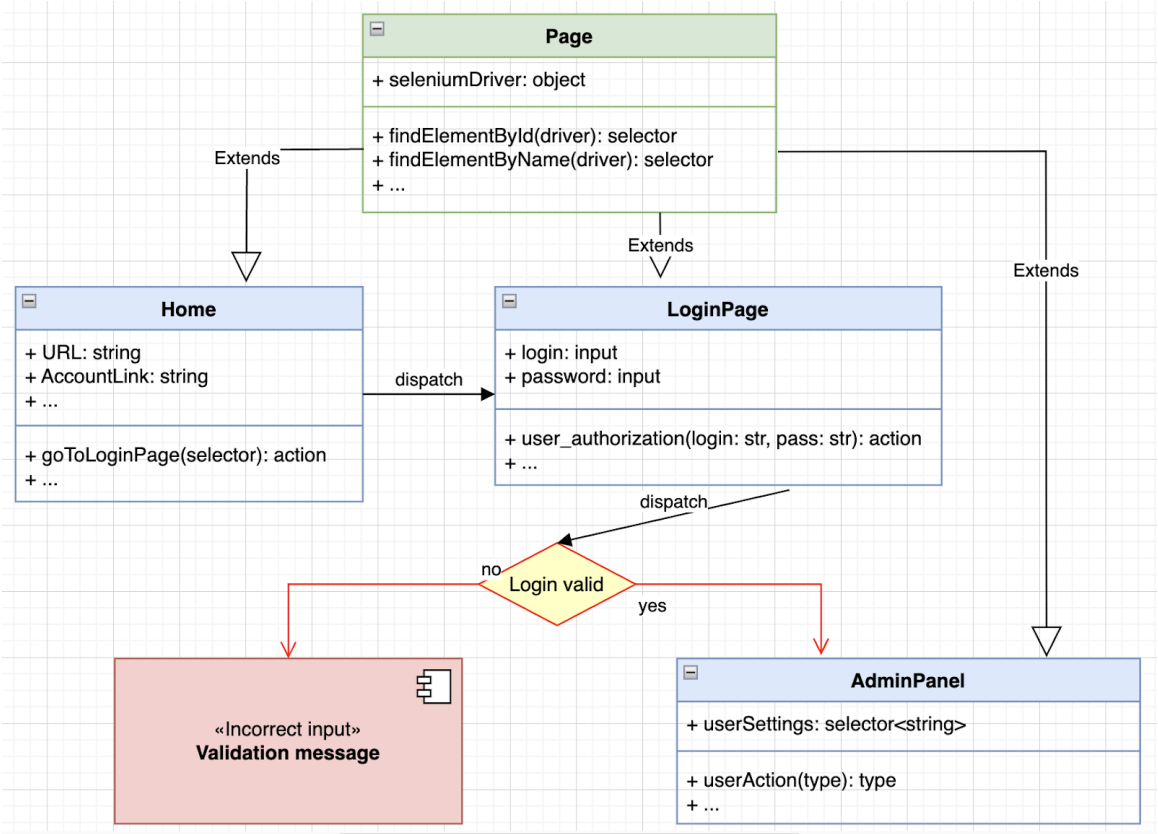


Рис. 2.17. *Приклад імплементації діаграми класів архітектурного патерну Page Object для можливого сценарія тестування 013*

На рис 2.17 об’єкти сторінок, які розмічені мітками ‘0’, ‘1’, ‘2’ позначені синім кольором, стан відображаючий валідаційне повідомлення з міткою ‘3’ - червоним. Таким чином за допомогою теорії лінгвістичного представлення детермінованих графів, а також при чіткому визначенні алгоритмів побудови графу АП та алгоритму побудови канонічної визначальної пари АК, теорія тестування програмного забезпечення та теорія використання графів як основної концептуальної моделі тестування може бути в певній мірі поширена.

2.3 Висновки до розділу

В даному розділі були змодельовані алгоритми лінгвістичного представлення детермінованих графів, наведена деяка програмна метрика цих алгоритмів. Запропоновано варіант прикладу застосування цих алгоритмів для побудови карти тестового покриття, яке може бути корисним при плануванні розробки автоматизованих тестів для програмного забезпечення. Слід зазначити, що на даний час розроблені алгоритми мають певні програмні особливості, такі як можливість опрацювання тільки неорієнтованих графів, та містять специфічні перевірки, які в деяких випадках викликають зупинку роботи алгоритма з позначкою “граф не існує”. Такі особливості можуть бути змінені в подальшому розвитку теорії побудови тестового покриття за допомогою цих алгоритмів. Наприклад, можливо, більш актуальною буде розробка аналогічних алгоритмів які будуть опрацьовувати орієнтовані графи та дозволяти будувати об'єкти без певних обмежень.

ВИСНОВКИ

Робота присвячена дослідженню можливості застосування лінгвістичного представлення детермінованих графів для використання у процесах тестування програмного забезпечення, зокрема у процесах побудови та оцінки тестового покриття. У роботі наведено коротку історію теорії представлення різноманітних математичних об’єктів та структур, виділено основні поняття, що пов’язані з детермінованими графами, їх використанням на практиці, інформацію про сучасний стан теорії лінгвістичного представлення детермінованих графів. Також у роботі наведено програмну реалізацію прототипів алгоритму побудови графа, для якого задана пара слів є визначальною та алгоритму побудови канонічної визначальної пари для заданого детермінованого графа та подано приклад поширення методики побудови тестового покриття програмного забезпечення із застосуванням розроблених алгоритмів на прикладі аналізу прототипу web-додатку.

Таким чином, в роботі були поставлені та виконані наступні задачі:

- Змодельовано алгоритми, що використовуються у лінгвістичному представленні детермінованих графів визначальною парою слів.
- Наведено значення деяких тестових метрик цих алгоритмів.
- Розглянуто підхід до застосування запропонованих алгоритмів у процесі побудови, оптимізації та оцінки тестового покриття для програмного забезпечення.

Слід зауважити, що деякі аспекти зазначених в цій роботі прикладів можуть уточнюватися та модифікуватися. Наведені алгоритми та рекомендації використання цієї методики є насамперед варіативними інструментами аналізу тестового покриття, яке може бути побудовано для програмного забезпечення. Наведені аспекти методики побудови тестового покриття є спробою поширення сучасної теорії тестування, завдяки впровадженню нових інструментальних засобів у вигляді алгоритмів лінгвістичного представлення детермінованих графів.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Chandler B., Magnus W. The History of Combinatorial Group Theory: A Case Study in the History of Ideas, Studies in the History of Mathematics and Physical Sciences (1st ed.). New York: Springer, 1982. – 234 p.

2. Адян С.И., Дурнев В.Г. Алгоритмические проблемы для групп и полугрупп // Успехи математических наук – 2000. – т.55, вып 2. – с. 3-94.

3. Lallement G. Semigroups and combinatorial applications. Pure and Applied Mathematics Series. New York: Wiley, 1979. – 376 p.

4. Magnus W., Karrass A., Solitar D. Combinatorial Group Theory. New York: John Wiley-Interscience, 1966. – 456 p.

5. Markov A.A. The theory of Algorithms // Journal of Symbolic Logik – 1953. – Vol. 18, Is. 4. – p. 340-341.

6. Соркин Ю.И. Теория определяющих соотношений для автоматов // Проблемы кибернетики – 1961. – вып.9. – с. 45-69.

7. Капитонова Ю.В. Об изоморфизме абстрактных автоматов.I // Кибернетика – 1965. – №3. – с. 16-19.

8. Капитонова Ю.В. Об изоморфизме абстрактных автоматов.II // Кибернетика – 1965. – №5. – с. 10-13.

9. Грунский И.С. Определяющие соотношения для автоматов // 3 Алгебраическая конференция в Украине (Сумы, 2-8 июня 2001г.) – Сумы: Изд-во Сум. пед. ун-та, 2001. – с. 159-160.

10. Олейник Р.И. О контроле частичных графов с использованием определяющих соотношений // Вісник Донецького університету, серія А: Природничі науки – 2001. – Вип.1 – с. 289-296.

11. Козловский В.А., Толмачевская Л.А. Построение множества определяющих слов опорной группы конечного группового автомата // Вісник Донецького університету, серія А: Природничі науки – 2001. – Вип.2 – с. 357-363.

12. Толмачевская Л.А. Циклические представления групповых автоматов // Труды института прикладной математики и механики НАН Украины – 2000. – т. 5. – с. 145-154.

13. Грунский И.С., Сенченко А.С. Каноническая система определяющих соотношений для автоматов // Труды ИПММ НАН Украины – 2002. – т. 7. – с. 58-63.

14. Grunskii I.S., Senchenko. A.S. Properties of systems of defining relations for automata // Discrete Mathematics and Applications – 2004. – Vol. 14, Is. 6. – p. 593–601.

15. Сенченко А.С. Определяющая система для частичных автоматов // Вісник Донецького університету, серія А: Природничі науки – 2003. – Вип.1 – с. 345-350.

16. Сенченко А.С. Свойства определяющих систем для частичных автоматов // Труды института прикладной математики и механики НАН Украины – 2003. – т. 8. – с. 111-119.

17. Grunskii I.S., Mikhaylova I.A., Sapunov S.V. Domination on the vertices of labeled graphs // Algebra and Discrete Mathematics – 2012. – Vol. 15, Is. 2. – p. 174-184.

18. Сапунов С.В., Сенченко О.С., Середа О.А. Метричні властивості канонічної визначальної пари для детермінованих графів // Праці ІПММ НАН України – 2020. – том 34. – с. 134-145.

19. Сапунов С.В., Сенченко А.С. Лингвистическое представление графов с помеченными вершинами // Доповіді НАН України – 2019. – № 11. – с. 29-34.

20. Сенченко О.С., Притула М.І., Середа О.А. Представлення детермінованих графів визначальною парою слів // Праці ІПММ НАН України – 2022. – том 36, №2. – с. 104-116.

21. Сенченко О.С., Середа О.А., Притула М.І. Визначальна пара для детермінованих графів // Наук.-практ. конф. «Актуальні проблеми теорії керуючих систем у комп’ютерних науках АПТКС’2021»: праці конф.,

Слов'янськ, 21-24 грудня 2021 р. – Слов'янськ: вид. Б.І. Маторіна, – 2021. – с. 112 – 120.

22. NetworkX is a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks. URL: https://networkx.org/documentation/stable/release/release_3.2.1.html (дата звернення 11.11.23р.).

23. Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations. URL:https://matplotlib.org/stable/index.html (дата звернення 11.11.23р.).

24. Graphviz is open source graph visualization software. URL: https://graphviz.org/download/ (дата звернення 11.11.23р.).

25. PEP 8 – Style Guide for Python Code. URL: https://peps.python.org/pep-0008 (дата звернення 11.11.23р.).

26. Radon is a Python tool which computes various code metrics. URL: https://radon.readthedocs.io/en/latest/ (дата звернення 11.11.23р.).

27. McCabe T.J. A Complexity Measure // IEEE Transactions on Software Engineering – 1976. – Vol. SE-2, No. 4 – p. 308-320.

28. This repository presents the implementation of algorithms for the linguistic representation of deterministic graphs, according to scientific publications in the collection of scientific works of the Institute of Applied Mathematics and Mechanics of the National Academy of Sciences of Ukraine. URL: https://github.com/NickIT87/IAMM_proceedings:(дата звернення 11.11.23р.).

29. Kaner C., Falk J., Hung Q. Nguyen. Testing Computer Software, 2nd Edition. New York: John Wiley & Sons, 1999 – p. 496

ДОДАТКИ

Додаток А

Відомість матеріалів кваліфікаційної роботи

№ з/п	Позначення	Назва	Кількість	Примітки
1				
2		Документація		
3				
4	САУ.КР.23.14.ПЗ	Пояснювальна записка	66	Формат А4
5				
6	САУ.КР.23.14.ДМ	Демонстраційні матеріали		Презентація на CD-R
7				
8	САУ.КР.23.14.КР	Копія роботи	1	Диск CD-R
9				
10				
11				
12				
13				
14				
15				
16				
17				
18				

					САУ.КР.23.14.ДА.ПЗ.
Змін.	Аркуш	№ докум.	Підпис	Дата	

Розроб.	Притула М.І.			Матеріали кваліфікаційної роботи	Літ.	Аркуш	Аркушів
К. Розд.	Слєсарєв В.В.						
Керівн.	Слєсарєв В.В.				НТУ «ДП» 124; 124м-22		
Н.контр.	Хомяк Т.В.						
Зав. каф.	Желдак Т.А.						

Додаток Б

Відгук
на кваліфікаційну роботу магістра
Студента групи *124м-22 Притули Миколи Ігоровича*
(група) (ПІБ)
Спеціальності 124 Системний аналіз

Тема кваліфікаційної роботи магістра: *«Використання лінгвістичного представлення детермінованих графів у тестуванні програмного забезпечення».*

Обсяг кваліфікаційної роботи магістра: 66с., 19 рис., 5 додатків, 29 джерел.

Мета кваліфікаційної роботи магістра: дослідження можливості використання лінгвістичного представлення детермінованих графів визначальною парою у тестуванні програмного забезпечення.

Актуальність теми обумовлена важливістю розвитку різноманітних методів тестування програмного забезпечення та можливістю використання лінгвістичного представлення детермінованих графів для ефективної побудови та оцінки тестового покриття.

Тема кваліфікаційної роботи магістра безпосередньо пов’язана з об’єктом діяльності магістра спеціальності 124 Системний аналіз, оскільки включає все необхідні етапи системного дослідження, а саме: аналіз об’єкта дослідження, побудову відповідних моделей і алгоритмів і впровадження їх у практичній діяльності.

Виконані в кваліфікаційній роботі магістра завдання **відповідають** вимогам до професійної діяльності фахівця освітньо-кваліфікаційного рівня магістра.

Практичне значення результатів кваліфікаційної роботи магістра полягає в тому, що результати, які одержані в роботі, можуть бути застосованими для розв’язання прикладних задач ідентифікації інформаційного середовища.

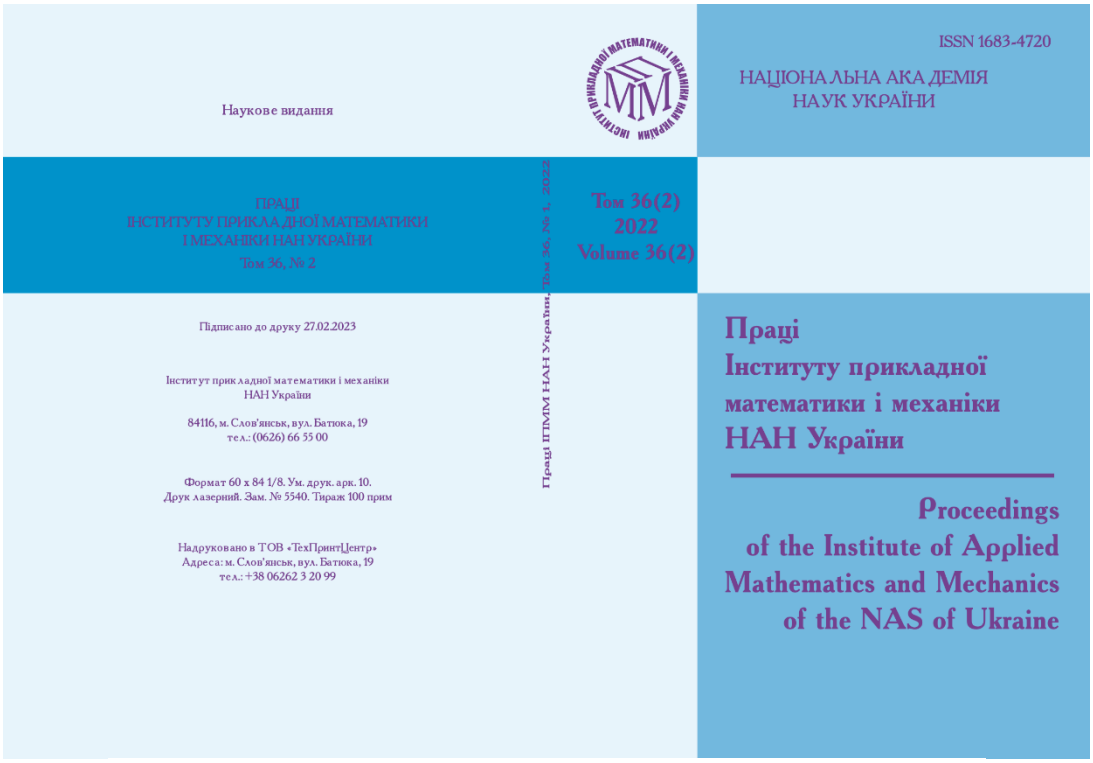
Оформлення пояснювальної записки та демонстраційного матеріалу до неї виконано згідно з вимогами. Роботу виконано самостійно, відповідно до завдання та у повному обсязі.

Кваліфікаційна робота магістра в цілому заслуговує на оцінку відмінно добре, а її автор заслуговує на присвоєння кваліфікації «магістр з системного аналізу».

Керівник кваліфікаційної роботи магістра,
Д.т.н., проф. _____ Слєсарєв В.В.

Додаток В

Наукова праця, опублікована автором цієї кваліфікаційної роботи



НАЦІОНАЛЬНА АКАДЕМІЯ НАУК УКРАЇНИ

Том 36, № 2
Слов'янськ, 2022
Заснований у 1997 р.

ПРАЦІ ІНСТИТУТУ ПРИКЛАДНОЇ МАТЕМАТИКИ І МЕХАНІКИ НАН УКРАЇНИ

ЗМІСТ

ISSN 1683-4720 **Праці ІПММ НАН України. 2022. Том 36, № 2**

УДК 519.7
DOI: 10.32782/1683-4720-2022-36-09

©2022. **О.С. Сенченко, М.І. Притула, О.А. Середа**

ПРЕДСТАВЛЕННЯ ДЕТЕРМІНОВАНИХ ГРАФІВ ВИЗНАЧАЛЬНОЮ ПАРОЮ СЛІВ

У роботі запропоновано представлення детермінованих графів множинами слів у алфавіті міток їх вершин. В наш час графи є концептуальним інструментом аналітики, розробкита тестування різноманітного програмного забезпечення. Серед усіх графів виділяють підклас розмічених графів, елементи яких мають мітки з попередньо визначеного алфавіту. Такі графи активно застосовують для описання та моделювання обчислювальних процесів у програмуванні, робототехніці, веріфікації та валідації моделей тощо. Розмічені графи є інформаційним середовищем для мобільних агентів, переміщення яких по графу можна відобразити послідовностями міток вершин – слів в алфавіті міток. Детермінованими називаються графи з розміченими вершинами, у яких в околі кожної вершини всі інші вершини мають попарно різні мітки. Для таких графів у випадку, коли відомі його карта графа (тобто множини вершин і ребер та функція розмітки) та ініціальна вершина, з якої агенти починають свої переміщення, існує однозначна відповідність між послідовністю міток вершин, які відвідує агент, і траєкторією переміщень цього агента на графі. У випадку, коли зовнішньому спостерігачу невідома карта досліджуваного детермінованого графа, переміщення агентів можуть бути організовані в такий спосіб, щоб на основі їх аналізу спостерігач одержав шукану інформацію щодо структури графа (наприклад, карту графа, найкоротші шляхи між вершинами, порівняння досліджуваного графа з графом-еталоном). Такий аналіз може суттєво спростити лінгвістичне представлення детермінованого графа – співставлення графу однієї чи декількох скінченних множин слів у алфавіті міток вершин графа, за якими можна відновити граф. У даній роботі запропоновано представлення детермінованих графів визначальною парою множин слів, перша компонента якої описує цикли графа, а друга – його висячі вершини. Це представлення є аналогом системи визначальних співвідношень для повністю визначених автоматів. Наведено алгоритм, який за довільною парою множин або будує детермінований граф, для якого ця пара є визначальною, або сповіщує, що це зробити неможливо. Також наведено алгоритм побудови канонічної визначальної пари для детермінованого графа та знайдено чисельні оцінки цієї пари для детермінованих графів із відомою кількістю вершин та ребер. Окреслено подальші напрямки дослідження за цією тематикою. Результати дозволять використовувати нові методи та алгоритми для розв'язання задач анализу графів з розміченими вершинами.

MSC: 68R10.

Ключові слова: *детерміновані графи, представлення, визначальна пара.*

1. Вступ.

В наш час графи є концептуальним інструментом аналітики, розробки та тестування різноманітного програмного забезпечення. Серед усіх графів виділяють підклас розмічених графів, тобто графів, у яких вершини та/або ребра мають деякі мітки з попередньо визначеного алфавіту. За допомогою таких графів природньо моделюються різноманітні процеси в сучасних інформаційних системах. При цьому розмічені графи, у яких мітки знаходяться на ребрах (LTS, зважені графи, скінченні автомати тощо) досліджені в значно більшій мірі, ніж графи з розміче-

ними вершинами. Але існує багато обчислювальних процесів у програмуванні [1], робототехніці [2], верифікації та валідації моделей [3], які зручно відображати саме графами з розміченими вершинами.

У роботі [4] запропоновано представлення скінченних, всюди визначених автоматів без виходів системами визначальних співвідношень – множинами пар слів спеціального вигляду, що задають еквівалентні траєкторії в автоматі. За допомогою такого представлення було розв'язано задачу характеризації – визначення, чи є дана множина пар слів системою визначальних співвідношень для заданого автомата без безпосередньої побудови самого автомата. Природним чином постає задача розповсюдження такого представлення на інші об'єкти дискретної математики, зокрема на графи з розміченими вершинами, принаймні на деякий підклас таких графів. На наш погляд, найбільш зручними для такого розповсюдження є детерміновані графи – клас графів з розміченими вершинами, у яких в околі кожної вершини всі вершини мають попарно різні мітки. Вперше такі графи було формалізовано в роботі [5]. Доцільність такого розповсюдження випливає з природнього взаємозв'язку між детермінованими графами та мовами в алфавіті міток вершин (для детермінованих графів траєкторія руху однозначно відновлюється за послідовністю міток вершин). Цей взаємозв'язок дозволяє застосувати методи теорії автоматів до аналізу розмічених графів.

В даній роботі узагальнено всі одержані нами результати за цією темою та окреслено напрямки подальших досліджень, пов'язаних з представленням детермінованих графів визначальною парою слів.

2. Основні означення.

У роботі розглядаються неорієнтовані, скінченні, непорожні, зв'язні, звичайні (такі, що не містять петлі та кратні ребра) графи з розміченими вершинами $G = (V, E, X, \xi(V))$, де V – множина вершин графа, E – множина його ребер, $\xi(V) : V \to X$ – всюди визначена функція розмітки вершин графа символами скінченного алфавіту $X = \{x_1, \ldots, x_p\}$. Далі такі графи будемо скороченно називати терміном «розмічений граф». Значення функції розмітки для вершини v будемо називати її міткою. Множину всіх слів скінченної довжини в алфавіті X (у тому числі порожнє слово ε) будемо позначати через X^*. Нехай $p = x'_1 \ldots x'_k$ $(x'_i \in X)$, тоді довжину слова p будемо позначати через $d(p)$. Через $E(v)$ позначатимемо множину вершин, що є суміжними вершині v: $v' \in E(v) \leftrightarrow (v, v') \in E$.

У залежності від додаткових умов, що накладаються на функцію розмітки вершин, з множини всіх розмічених графів виділяють окремі підкласи. У цій роботі розглядаються так звані детерміновані графи, у яких для будь-яких вершин $v, v', v'' \in V$ з $(v, v'), (v, v'') \in E$ та $\xi(v') = \xi(v'')$ випливає рівність $v' = v''$ [5]. Змістовно кажучи, наведене означає, що у детермінованих графів будь-які дві вершини, що одночасно суміжні деякій вершині, мають різні мітки. Зауважимо, що на відміну від так званих сильнодетермінованих графів [5], у детермінованих графів можуть існувати суміжні вершини з однаковими мітками.

Зафіксуємо деяку вершину $v_0 \in V$ Д-графа $G = (V, E, X, \xi(V))$, яку далі будемо називати ініціальною. У цій роботі ми будемо розглядати саме детерміновані

графи з ініціальною вершиною та коротко їх називати «Д-графами», цю вершину за необхідності ми будемо виділяти у позначенні Д-графа: $G = (V, E, X, \xi(V), v_0)$.

Ізоморфізмом Д-графів $G_1 = (V_1, E_1, X, \xi_1(V_1), v')$ і $G_2 = (V_2, E_2, X, \xi_2(V_2), v'')$ з однаковою множиною міток вершин X називаємо взаємно однозначну відповідність $\varphi : V_1 \leftrightarrow V_2$ між множинами їх вершин, що зберігає їх ініціальні вершини, відношення суміжності вершин та функцію розмітки вершин: $\varphi(v') = v''$, $\forall v', v'' \in V_1 \quad ((v', v'') \in E_1 \leftrightarrow (\varphi(v'), \varphi(v'')) \in E_2)$ та $\forall v \in V_1 \quad \xi_1(v) = \xi_2(\varphi(v))$; ізоморфізм графів G_1 та G_2 будемо позначати $G_1 \cong G_2$ або $G_1 \cong_\varphi G_2$. Зафіксуємо вершину v'_1 Д-графа $G = (V, E, X, \xi(V))$. Оскільки шляху $p = v'_1 \dots v'_k$ однозначно відповідає слово в алфавіті міток $\xi(p) = \xi(v'_1)\,\xi(v'_2) \dots \xi(v'_k)$ [5], то надалі шлях p розглядатимемо як $\xi(p)$ та позначатимо рівністю $v'_1 p = v'_k$.

Шлях $p = x'_1 x'_2 \dots x'_k$ назвемо припустимим для вершини $v'_1 \in V$, якщо $\xi(v'_1) = x'_1$, та існують такі вершини $v'_2, \dots, v'_k \in V$, що $\xi(v'_2) = x'_2, \dots, \xi(v'_k) = x'_k$, і $(v'_1, v'_2), \dots, (v'_{k-1}, v'_k) \in E$. Слово $x'_k \dots x'_1$ будемо називати реверсом слова $p = x'_1 \dots x'_k$ та позначати його через p^{-1}, слова p та p^{-1} назвемо взаємнозворотними. Слово p, для якого виконується рівність $vp = v$, назвемо циклічним для вершини v. Слово $q = x''_1 \dots x''_m$ назвемо початковим відрізком слова $p = x'_1 \dots x'_k$ у випадку, якщо $m \le k$ та виконуються рівності $x''_1 = x'_1, \dots, x''_m = x'_m$, цей факт позначатимемо через $q \subseteq p$.

Всі вершини заданого Д-графа $G = (V, E, X, \xi(V), v_0)$ можна розбити на два класи за такою процедурою: видалимо з G усі висячі вершини, відмінні від ініціальної, разом із ребрами, що є інцидентними цим вершинам. Цю процедуру будемо повторювати до тих пір, поки це можливо. Одержаний у такий спосіб граф $B(G)$ назвемо базою графа G, вершини, що входять до $B(G)$, назвемо базовими, а ті вершини G, що не входять до $B(G)$, – вільними. Неважко бачити, що база графа будується однозначно за скінченну кількість кроків, база $B(G)$ може складатися з однієї ініціальної вершини (у випадку, коли G є деревом), а у випадку, коли $B(G)$ містить хоча б одну вершину, яка відмінна від ініціальної, то вона повинна містити хоча б один простий цикл. Звідси випливає, що кожна базова вершина, яка є відмінною від ініціальної вершини v_0, суміжна щонайменше двом базовим вершинам. Вільну вершину v назвемо прямим предком вільної вершини v', якщо для будь-якого шляху p', такого, що $v_0 p' = v'$, існує такий початковий відрізок $p \subseteq p'$, що $v_0 p = v$, цей факт позначатимемо через $v \preceq v'$. Звернемо увагу, що для будь-якої вільної вершини v виконується $v \preceq v$.

3. Лінгвістичне представлення Д-графів.

Для дослідження структури розмічених графів часто використовують один або декілька мобільних агентів з обмеженою пам'яттю, які поміщають на досліджуваний граф [6]. Ці агенти можуть сповіщати спостерігачу та/або іншим агентам певну інформацію щодо локальних околів вершин, на яких вони наразі знаходяться. На основі цієї інформації та/або інструкцій спостерігача агенти можуть переміщатися по ребрах графа. Переміщення агентів визначають послідовності міток вершин, які вони відвідали. Зважаючи на це, Д-графи є дуже зручними для такого дослідження, оскільки, якщо спостерігач має карту графа (множини V, E, X та

функцію $\xi(V)$) та знає, на яку вершину досліджуваного Д-графа був поміщений агент на початку дослідження (тобто, ініціальну вершину), то за цією послідовністю міток може бути однозначно відновлена траєкторія переміщення агента по графу. У випадку, коли спостерігачу не відома карта досліджуваного графа, переміщення агентів можуть бути організовані в такий спосіб, щоб на основі їх аналізу спостерігач одержав шукану інформацію про структуру графа (наприклад, складення карти графа, пошуку у ньому найкоротших шляхів, порівняння досліджуваного графа з графом-еталоном).

Такий аналіз може суттєво спростити лінгвістичне представлення Д-графа, під яким ми розуміємо співставленню графу однієї чи декілька скінченних множин слів у алфавіті міток вершин графа, що описують можливі (та/або неможливі) траєкторії руху в графі. Для такого представлення треба визначитися з кількістю таких множин слів (бажано, щоб така кількість була якомога меншою) та роллю, яку відіграє кожна з цих множин слів у графі (які структурні елементи графа описує кожна множина). При цьому, ці множини треба обрати у такий спосіб, щоб можна було винайти універсальний алгоритм, який кожному набору цих множин або однозначно співставляє детермінований граф, або сповіщає, що це зробити неможливо. Також треба винайти алгоритм переходу від будь-якого Д-графа до сказаного вище набору множин.

Далі вважаємо, що всі шляхи в графі починаються з ініціальної вершини. Будемо робити представлення Д-графа $G = (V, E, X, \xi(V), v_0)$, у якого $\xi(v_0) = (x')$ парою $\{C, L\}(x')$ скінченних множин слів $C, L \in X^*$, для якої всі слова з C та L є припустимими для v_0, слова множини C описують цикли графа $G = (V, E, X, \xi(V), v_0)$, а слова L описують його висячі вершини. Далі в роботі під терміном «пара» $\{C, L\}(x')$ розуміємо саме такі дві скінченні множини C та L, що всі слова множини C починаються та закінчуються символом x', а всі слова множини L починаються з цього символа. При цьому множини C та L будемо називати компонентами пари $\{C, L\}(x')$.

Нехай $G = (V, E, X, \xi(V), v_0)$ – деякий Д-граф. Зафіксуємо на X лінійний порядок $<$: $x_1 < x_2 < \dots < x_p$. Визначимо на X^* лінійний порядок $\preceq$:

1) для будь-якого $x_i \in X$, де $1 \le i \le p$, покладемо $x_i \preceq x_i$;

2) для будь-яких $p, q \in X^*$, якщо $d(p) < d(q)$, то $p \preceq q$;

3) для будь-яких $p, q \in X^*$, якщо $p = x'_1 \dots x'_s$, $q = x''_1 \dots x''_s$, $x'_1 = x''_1$, $\dots$, $x'_{k-1} = x''_{k-1}$ та $x'_k < x''_k$, то $p \preceq q$.

Неважко бачити, що порядок $\preceq$ на множині X^* визначається лінійним порядком $<$ на множині X.

Нехай $G = (V, E, X, \xi(V), v_0)$ – довільний граф з розміченими вершинами. Редукцією G назвемо процедуру, яка перетворює G у Д-граф $[G]$ за таким алгоритмом (АР):

0) Покладемо $V' = V, E' = E, \xi'(V') = \xi(V)$, позначимо $G' = (V', E', X, \xi'(V'), v_0)$.

1) Для кожної вершини $v \in V'$ знаходимо найменше за $\preceq$ слово w, що відповідає шляху від v_0 до v. Пов'язуємо v з w.

2) Впорядковуємо вершини графа G' за пов'язаними з ним словами за порядком

$\preceq$. Поточною вершиною v_c призначимо першу за вказаним порядком вершину, а поточною міткою призначимо $x = x_1$.

3) Якщо існують такі різні вершини $v'_1, \ldots, v'_q$, для яких одночасно виконуються умови $v'_1, \ldots, v'_q \in E'(v_c)$ та $\xi'(v'_1) = \ldots = \xi'(v'_q) = x$, то позначимо $U = \{v'_1, \ldots, v'_q\}$ та виконуємо таку послідовність дій:

3.1. до множини V' додаємо нову вершину v' та покладемо $\xi'(v') = x$;

3.2. з множини E' вилучаємо ребра (v_i, v_j), де $v_i, v_j \in U$;

3.3 для кожного ребра (v_i, v_j), де $v_i \in U$, до множини E' додаємо ребро (v', v_j);

3.4. з множини E' вилучаємо ребра (v_i, v_j), де $v_i \in U$, з множини V' вилучаємо вершини v, де $v \in U$;

3.5 якщо $v_0 \in U$, то вершину v' перейменовуємо на v_0 та вважаємо далі цю вершину ініціальною;

3.6. вилучаємо повтори ребер, залишаючи по одному екземпляру;

3.7. переходимо на крок 1.

4) Якщо існує $x' \in X$, який є наступним за x за порядком $<$ (тобто, $x \neq x_p$), то покладемо $x = x'$ та переходимо до кроку 3.

5) Якщо існує вершина v, що є наступною для v_c, то покладемо $v_c = v$, $x = x_1$ та переходимо на крок 3, у протилежному випадку алгоритм АР завершує свою роботу та $[G] = G'$.

Неважко бачити, що виконання алгоритму АР закінчується за скінченну кількість кроків, його результат визначається однозначно та цей результат є Д-графом.

Зауваження. При виконанні цього алгоритму існує певна неоднозначність з іменами нових вершин (окрім випадку, за яким новостворену вершину v' було перейменовано на v_0 на кроці 3.5), проте, для нас є несуттєвими імена усіх інших вершин (окрім ініціальної) у графі. За цієї обставини, якщо було виконано крок 3 алгоритму АР, ми не можемо говорити про рівність графа $[G]$ якомусь іншому графу, а можемо говорити про ізоморфність $[G]$. Також неважко бачити, що $[G] = G$ тоді та лише тоді, коли G є Д-графом.

Далі під терміном «пара» $\{C, L\}(x')$ розуміємо дві множини C та L, елементи яких – слова в алфавіті X – задовольняють таким умовам:

а) кожне слово множини C розпочинається та закінчується символом x';

б) довжина кожного слова множини C більша 2;

в) кожне слово множини L розпочинається символом x'.

Визначимо алгоритм (АП), який за заданою парою $\{C, L\}(x')$ або будує Д-граф $G(\{C, L\}(x'))$, або показує, що за цією парою Д-граф побудувати неможливо.

0) Покладемо, що спочатку граф $G(\{C, L\}(x'))$ складається з однієї вершини v_0 з міткою $\xi(v_0) = x'$.

1) У відповідність кожному слову $p^i = x' x^i_1 \ldots x^i_n x' \in C$ додаємо до графа вершини $v^i_1, \ldots, v^i_n$ з мітками відповідно $x^i_1, \ldots, x^i_n$ та ребра $(v_0, v^i_1), (v^i_1, v^i_2), \ldots, (v^i_{n-1}, v^i_n)$, (v^i_n, v_0). Після кожного такого додавання робимо редукцію одержаного графа.

2) У графі, який ми одержали після виконання кроку 1, можуть існувати висячі вершини. Кожну таку вершину v, що є відмінною від ініціальної вершини v_0,

поміщаємо для подальшого аналізу у множину Q, яка на початку була порожня.

3) Для кожного слова $p^j = x'x_1^j \dots x_n^j \in L$ виконуємо таку послідовність дій: додаємо вершини $v_1^j, \dots, v_n^j$ з мітками відповідно $x_1^j, \dots, x_n^j$, ребра $(v_0, v_1^j), \dots, (v_{n-1}^j, v_n^j)$ та робимо редукцію одержаного графа. Якщо в результаті редукції вершина v_0p_j не є висячею, то вважаємо, що граф $G(\{C, L\}(x'))$ не існує, і процедура завершується. Якщо ж в результаті редукції з'явилась висяча вершина v, відмінна від вершини v_0p_j, то цю вершину поміщаємо для подальшого аналізу у множину Q. Після цього переходимо до наступного слова компоненти L.

4) Для кожної вершини $v \in Q$ розглядаємо всі слова компоненти L: якщо не існує такого $p \in L$, що вершина v є прямим предком вершини v_0p, то вважаємо, що граф $G(\{C, L\}(x'))$ не існує.

5) Розглядаємо всі слова множини L: якщо існує таке $p \in L$, що вершина v_0p не є висячою, то вважаємо, що граф $G(\{C, L\}(x'))$ не існує.

Якщо в результаті виконання цієї процедури за парою $\{C, L\}(x')$ можливо побудувати граф $G(\{C, L\}(x'))$, то таку пару назвемо правильною. Правильну пару $\{C, L\}(x')$ назвемо *визначальною* для Д-графа G, якщо $G(\{C, L\}(x')) \cong G$. Зауважемо, що таке завдання Д-графа визначальною парою в деякій мірі (етапами 1 та 3) є аналогічним представленню автоматів системою визначальних співвідношень, запропонованому у [4].

Доцільність введення другого, четвертого та п'ятого кроків до алгоритму АП обумовлено намаганням чітко розділити функції компонентів пари: передбачається, що слова компоненти C описують базові вершини графа, а слова компоненти L – його вільні вершини, причому кожне слово $p \in L$ описує висячу вершину v_0p, а для кожної вільної висячої вершини v потрібне існувати хоча б одне таке слово $p \in L$, що $v_0p = v$ (зауважимо, що у випадку, коли вершина v_0 є висячею, то її не потрібно описувати у компоненті L). У випадку, коли слова множини C задають деякі вільні вершини графа (тобто після виконання кроку 2 множина Q не є порожньою), ці вільні вершини обов'язково повинні бути описаними словами компоненти L (див. крок 4), а якщо такого опису нема, то вважаємо, що за парою $\{C, L\}(x')$ граф $G(\{C, L\}(x'))$ побудувати неможливо.

Розглянемо дію окремих етапів наведеної вище процедури на такому прикладі.

Приклад 1. Нехай $C = \{153521, 152431\}$, $L = \{1342531, 123241, 13412, 1523\}(1)$. На рис. 1 а зображено граф, який одержуємо після виконання етапів 1 та 2. Вершина v', що виділена, є висячею та відмінною від ініціальної вершини v_0, тому цю вершину поміщаємо до множини Q.

На рис. 1 б зображено граф, який одержуємо після виконання етапу 3. Після розгляду слова 123241 вершина v'', що виділена, є висячею та відмінною від вершини v_0, тому цю вершину поміщаємо до множини Q.

Далі робимо аналіз вершин v' та v'', що входять до множини Q. Оскільки вершина v' є прямим предком вершини u', $1342531 \in L$ та $v_01342531 = u'$, а також вершина v'' є прямим предком самої себе, $1523 \in L$ та $v_01523 = v''$ (рис. 1 в), то переходимо до виконання етапу 5.

Оскільки вершина v (рис. 1 г) не є висячою, $v_0 123241 = v$ та $123241 \in L$, то за умовою етапу 5 граф $G(\{C, L\}(1))$ не існує.

Зауважимо, що для пари $\{C, L'\}(1)$, де $L' = (L \setminus \{123241\}) \bigcup \{1232412\}$ граф $G(\{C, L'\}(1))$ існує, він зображений на рис. 1 г.

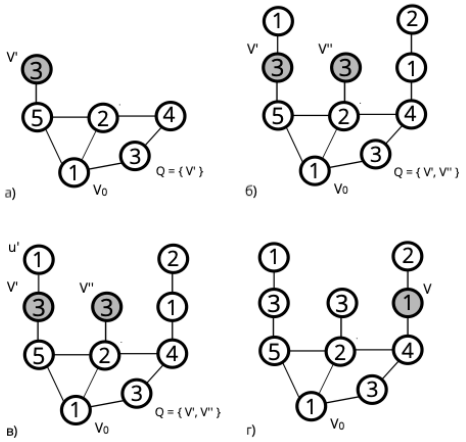


Рис. 1. Ілюстрація процедури побудови Д-графа за заданою парою

4. Канонічна визначальна пара.

Нехай $G = (V, E, X, \xi(V), v_0)$ – деякий Д-граф, та $V = \{v_0, \ldots, v_{n-1}\}$. Визначимо допоміжну множину слів в алфавіті X. Базисом досяжності $\mathcal{V}_G$ назвемо таку множину слів $\{w_1, w_2, \ldots, w_n\}$, де для кожної вершини v_i існує таке слово $w_i \in \mathcal{V}_G$, що виконується $v_0 w_i = v_i$, та для будь-якого $w \neq w_i$ з $v_0 w = v_i$, випливає $w_i \preceq w$. Кістякове дерево графа G, яке визначається базисом $\mathcal{V}_G$, позначимо $T(\mathcal{V}_G)$ або $T(G, v_0)$. Для кожної вершини $v \in V$ через sp_v позначимо таке слово з $\mathcal{V}_G$, що $v_0 sp_v = v$.

Опишемо алгоритм (АК) побудови для Д-графа G визначальної пари $\{\Sigma_G, \Lambda_G\}$, яку, завдяки деяким її властивостям, ми назвали канонічною.

Спочатку покладемо $\Sigma_G = \emptyset$ і $\Lambda_G = \emptyset$. Якщо граф G складається з однієї вершини v_0, то покладемо $\Sigma_G = \emptyset$, $\Lambda_G = \emptyset$ і $\mathcal{V}_G = \{\xi(v_0)\}$.

Нехай граф G містить більше ніж одну вершину. Спочатку до множини Λ_G додаємо усі слова $w \in \mathcal{V}_G$ такі, що вершина $v_0 w$ є висячою вершиною графа G. Після цього для кожної двійки слів $p, q \in \mathcal{V}_G \setminus \Lambda_G$, якщо жодне з них не є початковим відрізком іншого і $v_0 p q^{-1} = v_0$, то додаємо до множини Σ_G одне з двох слів pq^{-1} або qp^{-1}, яке є меншим за порядком $\preceq$ (далі вершини $v_0 p$ та $v_0 q^{-1}$ називатимемо твірними для того слова pq^{-1} або qp^{-1}, що було додане до Σ_G). Після цього прибираємо усі повтори слів у Σ_G, залишаючи по одному екземпляру.

Опишемо деякі властивості канонічної визначальної пари, що безпосередньо випливають з алгоритму АП.

Теорема 1. *Нехай $\{\Sigma_G, \Lambda_G\}$ – канонічна визначальна пара Д-графа $G = (V, E, X, \xi(V), v_0)$.*

1) Якщо хоча б одна компонента $\{\Sigma_G, \Lambda_G\}$ не є порожньою множиною, то

для кожної вершини $v \in V$ існують такі $z_1, z_2, z_3 \in X^*$, що виконується хоча б одне з тверджень: а) $sp_v z_1 \in \Lambda_G$; б) $sp_v z_2 \in \Sigma_G$; в) $z_3 (sp_v)^{-1} \in \Sigma_G$.

2) Нехай (v_1, v_2) – деяке ребро графа G, $\xi(v_1) = x_1'$ та $\xi(v_2) = x_2'$. Тоді існують такі $z_1, z_2 \in X^*$, що виконується хоча б одне з тверджень: а) $sp_{v_1} x_2' z_1 \in \Sigma_G \bigcup \Lambda_G$; б) $sp_{v_2} x_1' z_2 \in \Sigma_G \bigcup \Lambda_G$.

3) Якщо $\Sigma_G \neq \emptyset$, то для кожного $\sigma \in \Sigma_G$ існують такі $p, q \in X^*$, що $pq = \sigma$, та $p \in \mathcal{V}_G$ і $q^{-1} \in \mathcal{V}_G$.

4) Нехай (v_1, v_2) – деяке ребро графа G, яке не належить кістяковому дереву $T(G, v_0)$. Тоді існує едине слово $\sigma \in \Sigma_G$, що ребро (v_1, v_2) входить до шляху $v_0 \sigma$.

5) Якщо $\Sigma_G \neq \emptyset$, то для кожного $\sigma \in \Sigma_G$ виконується нерівність $d(\sigma) \geq 4$.

6) Якщо $\Sigma_G \neq \emptyset$, то для кожного $\sigma \in \Sigma_G$ знайдуться такі $p = p' x_1, q \in X^* (x_1 \in X)$, $x_2, x_3 \in X$, $x_2 \neq x_3$, що $\sigma = p x_2 q x_3 p^{-1}$ та $v_0 p = v_0 p x_2 q x_3 x_1$.

7) Якщо v_1, v_2 – твірні вершини деякого слова $\sigma \in \Sigma_G$, то $(v_1, v_2) \notin T(G, v_0)$ та $\mid d(sp_{v_1}) - d(sp_{v_2}) \mid \leq 1$.

Змістовно кажучи, перше та друге твердження стверджують, що пара $\{\Sigma_G, \Lambda_G\}$ містить певну інформацію про кожну вершину та ребро графа G, третє твердження встановлює своєрідну мінімальність початкових та кінцевих відрізків слова $\sigma \in \Sigma_G$, четверте твердження вказує, що кожне ребро, яке не належить кістяковому дереву $T(\mathcal{V}_G)$, описується окремим словом з Σ_G, п'яте твердження показує, що σ містить простий цикл; який, як встановлює шосте твердження, не може бути представлений у вигляді pxp^{-1} для будь-яких $p \in X^*$, $x \in X$.

З алгоритмів АП та АК можна бачити, що пара $\{\Sigma_G, \Lambda_G\}$ є правильною (для кожного слова $\lambda \in \Lambda_G$ вершина $v_0 \lambda$ є висячою, а множина Q є порожньою). Також справедлива

Теорема 2. $G(\{\Sigma_G, \Lambda_G\}) \cong G$.

Для доведення цієї теореми була досліджена покрокова побудова графа $G(\{\Sigma_G, \Lambda_G\})$ за словами множин Σ_G та Λ_G. Для цього покладемо $\Sigma_G = \{\sigma_1, \ldots, \sigma_k\}$ та $\Lambda_G = \{\lambda_1, \ldots, \lambda_q\}$. Введемо сімейство множин M_i $(0 \leq i \leq k+q)$ у такий спосіб: $M_0 = \varnothing$, $M_1 = \{\sigma_1\}$, $M_2 = \{\sigma_1, \sigma_2\}$, …, $M_k = \{\sigma_1, \ldots, \sigma_k\} = \Sigma_G$, $M_{k+1} = \{\sigma_1, \ldots, \sigma_k, \lambda_1\}$, …, $M_{k+q} = \{\sigma_1, \ldots, \sigma_k, \lambda_1, \ldots, \lambda_q\} = \Sigma_G \bigcup \Lambda_G$. За словами цього сімейства множин вводяться два сімейства графів: G_i та H_i. Графи сімейства G_i є підграфами графа G, що визначаються вершинами та ребрами слів з відповідної множини M_i, а графи сімейства H_i побудовані за словами з відповідної множини M_i за допомогою алгоритму АП. За цим алгоритмом $H_{k+q} = G(\{\Sigma_G, \Lambda_G\})$, а з пршого та другого тверджень теореми 1 випливає рівність $G_{k+q} = G$. Індукцією за i $(0 \leq i \leq k+q)$ показано, що $H_i \cong G_i$. Доведення є досить великим за обсягом, тому в цій роботі ми його не наводимо.

5. Метричні властивості компонент канонічної визначальної пари.

Наведемо всі знайдені нами чисельні оцінки компонент канонічної визначальної пари. Нехай далі Д-граф $G = (V, E, X, \xi(V), v_0)$ містить n вершин та m ребер $(n - 1 \leq m \leq \frac{n \cdot (n-1)}{2})$. Далі позначення $\lceil x \rceil$ означає найменьше натуральне число, що не меньше за x. У [7] було знайдено та доведено точне значення потужності

першої компоненти та досяжні оцінки потужності другої компоненти канонічної визначальної пари.

Теорема 3.
1) $|\Sigma_G| = m - n + 1$.

2) Якщо $m = n - 1$ *(тобто* G *є деревом), то* $1 \leq |\Lambda_G| \leq n - 1$. *Якщо ж* $m > n - 1$, *то* $0 \leq |\Lambda_G| \leq n - \lceil \frac{3}{2} + \sqrt{\frac{9}{4} - 2 \cdot n + 2 \cdot m} \rceil$, *причому всі оцінки є досяжними.*

Також було знайдено досяжні оцінки $\|\Sigma_G\|$ значення сумарної довжини усіх слів (об'єм) першої компоненти визначальної пари Д-графа G. Якщо G є деревом, то $\Sigma_G = \emptyset$, тому у цьому випадку $\|\Sigma_G\| = 0$. Для випадку, коли G не є деревом, справедлива

Теорема 4. *Якщо* $m > n - 1$, *то*
$4(m - n + 1) \leq \|\Sigma_G\| \leq 4(m - n + 1) \left(n - \lceil \frac{3}{2} + \sqrt{\frac{9}{4} - 2 \cdot n + 2 \cdot m} \rceil + 2 \right)$, *причому ці оцінки є досяжними.*

Доведення. Нижня оцінка безпосередньо випливає з першого твердження теореми 3 та п'ятого твердження теореми 1. Ця оцінка є досяжною для графа, всі вершини якого суміжні ініціальній вершині. У такого графа Σ_G складається з $m - n + 1$ слів довжини 4.

Доведемо верхню оцінку. За першим твердженням теореми 3, $|\Sigma_G| = m - n + 1$, покладемо $p = m - n + 1$. За третім твердженням теореми 1, кожне слово $\sigma \in \Sigma_G$ може бути представленим парою твірних вершин u, w, тому $d(\sigma) = d(sp_u) + d(sp_w)$. Нехай V' – множина вершин графа G, що є твірними для хоча б одного слова з Σ_G. Таким чином, знаходження верхньої оцінки $\|\Sigma_G\|$ перетворюється у знаходження максимально можливих за довжиною значень шляхів $sp_v (v \in V')$. Зважаючи на сьоме твердження теореми 1 та той факт, що збільшення кількості вершин у V' при фіксованій кількості вершин у G, призводить до зменшення максимально можливої довжини їх $sp_v (v \in V')$, найбільше значення $\|\Sigma_G\|$ можливо у графі, у якому потужність V' є мінімально можливою для утворення p слів у Σ_G, всі вершини з V' суміжні деякій фіксованій вершині v', та іншим вершинам з V' та не суміжні жодній вершині з $V \setminus (V' \bigcup \{v'\})$. У такому графі всі прості цикли є трикутниками, одна вершина якого є v', а дві інші вершини належать множині V', $d(sp_v) = d(sp_{v'}) + 1$ $(v \in V')$ та $\|\Sigma_G\| = 2p(d(sp_{v'} + 1))$; найбільш можливе значення $\|\Sigma_G\|$ буде у графа, у якого значення $y = d(sp_{v'})$ буде максимально можливим.

Для знаходження максимально можливого значення y опишемо структуру графа G' з n вершинами та m ребрами, для якого кількість висячих вершин, що не належать базі $B(G')$, буде максимальною. Нехай у графі G' t вершин належать його базі. Неважко бачити, що максимальна кількість вільних висячих вершин у G' дорівнює $n - t$, це можливо у випадку, при якому всі вільні вершини графа є висячими (як приклад, всі вільні вершини такого графа є суміжними ініціальній вершині), мінімально можливе значення t далі будемо позначати через $\delta(m, n)$. Для знаходження $\delta(m, n)$ використаємо той очевидний факт,

що найбільша кількість простих циклів та найбільша потужність першої компоненти канонічної визначальної пари для графа G з n вершинами є у повного графа K_n. Тому $\delta(m,n)$ дорівнює такому мінімальному значенню z, що число p не переважає потужність $|\Sigma_{K_z}|$. Оскільки у K_z кількість ребер дорівнює $\frac{z\cdot(z-1)}{2}$, то $|\Sigma_{K_z}| = \frac{z\cdot(z-1)}{2} - z + 1 = (\frac{z}{2}-1)\cdot(z-1)$. Іншими словами, значенням $\delta(m,n)$ буде таке мінімальне натуральне число z, що задовільняє нерівності $m-n+1 \leq (\frac{z}{2}-1)\cdot(z-1)$. З урахуванням натуральності m, n та z шукане значення $\delta(m,n) = \lceil \frac{3}{2} + \sqrt{\frac{9}{4} - 2\cdot n + 2\cdot m} \rceil$. Оскільки $\delta(m,n) = |V'|+1$, та існують $n-\delta(m,n)$ вершин, що не належать множині $V' \bigcup \{v'\}$, то максимально можливе значення y буде $n - \delta(m,n) + 1$, тому $\|\Sigma_G\| \leq 4(m-n+1)\left(n - \lceil \frac{3}{2} + \sqrt{\frac{9}{4} - 2\cdot n + 2\cdot m} \rceil + 2\right)$. □

Зауваження. Зауважимо, що наведені формули для верхньої та нижньої оцінок цієї теореми справедливі також для випадку $m = n-1$, за яким вони дорівнюють 0, що збігається зі значенням об'єму першої компоненти канонічної визначальної пари для дерева, яке було знайдено вище.

Знайдено оцінки $\|\Lambda_G\|$ для випадку, коли граф G є деревом.

Теорема 5. *Нехай Д-граф $G = (V, E, X, \xi(V), v_0)$ є деревом та містить n вершин. Тоді $n \leq \|\Lambda_G\| \leq \lceil \frac{n^2+2n}{4} \rceil$, причому ці оцінки є досяжними.*

Доведення. Оскільки G є деревом, то $\Sigma_G = \emptyset$. Тоді, з урахуванням другого твердження теореми 1, для кожного ребра (v_1, v_2) графа G повинно існувати таке слово $\lambda \in \Lambda_G$, що (v_1, v_2) входить до шляху $v_0\lambda$. Оскільки шлях з k ребер складається з $k+1$ символів алфавіта X, то найменш можлива сумарна кількість символів у Λ_G, яка необхідна для входження у слова з Λ_G всіх $n-1$ ребер графа G, становить $n-1+1$, що доводить нижню оцінку.

Доведемо верхню оцінку. Нехай граф G має q висячих вершин, які відмінні від вершини v_0, та, відповідно $|\Lambda_G| = q$ ($q = \overline{1, \ldots, n-1}$). Неважко бачити, що максимально можливе значення $\|\Lambda_G\|$ буде у графа, у якого всі висячі вершини, відмінні від v_0 суміжні деякій фіксованій вершині v', та значення $d(sp_{v'})$ буде максимально можливим, тобто $d(sp_{v'}) = n - q$. Тоді довжина кожного слова з Λ_G дорівнює $n-q+1$, отже, $\|\Lambda_G\| \leq q(n-q+1)$. Максимальне значення $\|\Lambda_G\|$ буде у випадку $q = \frac{n+1}{2}$. З урахуванням натуральності числа q, для непарних n $q = \frac{n+1}{2}$, а для парних n, $q = \frac{n}{2}$, або $q = \frac{n}{2} + 1$. Об'єднуючи ці випадки, одержемо $\|\Lambda_G\| \leq \lceil \frac{n^2+2n}{4} \rceil$. □

6. Подальші напрямки дослідження.

Наведене лінгвістичне представлення Д-графа визначальною парою слів може бути корисним при розв'язанні багатьох прикладних задач. При цьому природним чином виникає цілий ряд задач, пов'язаних з цим лінгвістичним представленням, серед яких автори виділяють наступні:

- знаходження взаємозв'язків між графами G та $[G]$;

- розв'язання задачі характеризації пари: для заданого Д-графа та заданої пари визначити, чи є ця пара визначальною для графа без безпосередньої побудови

графа за парою;

- знаходження необхідних та достатніх умов для множин C та L, за яких пара $\{C, L\}$ є правильною;

- оптимальний вибір ініціальної вершини, за яким метричні властивості компонент канонічної визначальної пари будуть мінімальними;

- ефективна побудова за $\{\Sigma_G, \Lambda_G\}$ найкоротших за порядком $\preceq$ шляхів між двома довільними вершинами графа G;

- застосування наведених у цій роботі понять та алгоритмів для сильнодетермінованих [5] графів.

7. Висновки.

У роботі запропоновано лінгвістичне представлення детермінованого графа визначальною парою слів. Наведено алгоритм, який за довільною парою множин або будує Д-граф, для якого ця пара є визначальною, або сповіщує, що це зробити неможливо. Також наведено алгоритм побудови канонічної визначальної пари для Д-графа. Знайдено чисельні оцінки цієї пари для Д-графів із відомою кількістю вершин та ребер – потужність першої компоненти пари, мінімальні й максимальні досяжні оцінки потужності другої компоненти пари та об'єму її першої компоненти. Також знайдено мінімальні й максимальні досяжні оцінки об'єму другої компоненти канонічної визначальної пари для випадку, коли граф є деревом. Окреслено подальші напрямки дослідження за цією тематикою. Результати дозволять використовувати нові методи та алгоритми для розв'язання задач анализу графів з розміченими вершинами.

Цитована література

1. *Okhotin A.* Graph-Walking Automata: From Whence They Come, and Whither They are Bound. In: Hospodár M., Jirásková G. (eds) Implementation and Application of Automata. CIAA 2019. Lecture Notes in Computer Science, 2019, vol 11601. Springer, Cham.
2. *Dudek, G., Jenkin, M.* Computational Principles of Mobile Robotics, 2nd ed. Cambridge: Cambridge Univ. Press, 2010.
3. *Baier C., Katoen J.-P.* Principle of Model Checking. The MIT Press, 2008. 984 p.
4. *Grunskii I.S., Senchenko A.S.* Properties of systems of defining relations for automata // Discrete Mathematics and Applications. – 2004. – Vol. 14, Iss. 6. – P. 593–601.
5. *Grunskii I., Mikhaylova I., Sapunov S.* Domination on the vertices of labeled graphs // Algebra and Discrete Mathematics. – 2012. – Vol. 14, Iss. 2. – P. 174–184.
6. *Stepkin A.V.* Using a Collective of Agents for Exploration of Undirected Graphs // Cybernetics and System Analysis. – 2015. – Vol. 51, Iss. 2. – P. 223–233.
7. *Сапунов С.В., Сенченко О.С., Середа О.А.* Метричні властивості канонічної визначальної пари для детермінованих графів // Праці Інституту прикладної математики і механіки НАН України. – 2020. – Т. 34. – С. 134–145.

References

1. Okhotin A. (2019) Graph-Walking Automata: From Whence They Come, and Whither They are Bound. In: Hospodár M., Jirásková G. (eds) Implementation and Application of Automata. CIAA 2019. Lecture Notes in Computer Science, vol 11601. Springer, Cham. https://doi.org/10.1007/978-3-030-23679-3_2

Додаток Г

Програмний модуль

```
""" IAMM - Graphs - ASP Work """
from typing import Tuple, List, Union, Dict
from math import ceil, sqrt
import re
import networkx as nx # type: ignore

# ========================== HELPERS FUNCTIONS ==============================
def counter(reset=False):
    """helper for AP alg. generate IDs for nodes"""
    if reset:
        counter.count = 0
    elif not hasattr(counter, 'count'):
        counter.count = 1
    else:
        counter.count += 1
    return counter.count


def get_all_leaf_nodes_from_graph(G: nx.Graph) -> dict:
    """helper for AP alg. for checking leafs nodes"""
    leaf_nodes = [node for node, degree in G.degree() if degree == 1]
    node_labels = [G.nodes[id]['label'] for id in leaf_nodes]
    return dict(zip(leaf_nodes, node_labels))


def find_neighbours_with_the_same_labels(nghb: List, lbls: List) -> Dict:
    """ helper for AR reduction algorithm """
    element_positions: Dict = {}
    for index, element in enumerate(lbls):
        if element in element_positions:
            element_positions[element].append(nghb[index])
        else:
            element_positions[element] = [nghb[index]]
    equal_elements: Dict = {
        element: positions for element,
        positions in element_positions.items() if len(positions) > 1
    }
    return equal_elements


def walk_by_word(graph: nx.Graph, word: str, root_node: int) -> int:
    """ get last node id by label (word) path in graph """
    current_node = root_node
    for symbol in word[1:]:
        neighbors = list(graph.neighbors(current_node))
        labels = [graph.nodes[id]['label'] for id in neighbors]
        try:
            next_node = neighbors[labels.index(symbol)]
        except Exception as exc:
            raise ValueError(
                f"{exc} Invalid data. No symbol as label. Graph is not exists!"
            ) from exc
        current_node = next_node
    return current_node
```

```
def check_q_node(graph: nx.Graph, l_set: Tuple[str], vq_id: int, vq_label: str, root: int) -> bool:
    """ helper function, check node is valid by step 4 in AP alg """
    state = False
    if vq_id == root:
        state = True
    if graph.degree(vq_id) > 1:
        for word_l in l_set:
            if vq_label in word_l and word_l[-1] != vq_label:
                f_pattern = word_l[:word_l.rfind(vq_label) + 1]
                l_pattern = word_l[word_l.rfind(vq_label) + 1:]
                checked_label = f_pattern[-2]
                if vq_id == walk_by_word(graph, f_pattern, root):
                    if checked_label != l_pattern[0]:
                        state = True
    elif graph.degree(vq_id) == 1:
        for word_l in l_set:
            if vq_label in word_l and word_l[-1] == vq_label:
                if vq_id == walk_by_word(graph, word_l, root):
                    state = True
    else:
        raise ValueError(
            f"Id: {vq_id} Lbl:{vq_label} not in graph. Graph is not exists!"
        )
    return state


def word_pair_data_validation(c_tuple: Tuple[str], l_tuple: Tuple[str], root: str) -> bool:
    """ rules for using a pair of words in an algorithm """
    pattern = r"(.)\1"
    if not c_tuple and not l_tuple:
        return False
    if c_tuple:
        for c_word in c_tuple:
            if c_word[0] != c_word[-1] and c_word[0] != root:
                return False
            if bool(re.search(pattern, c_word)):
                return False
    if l_tuple:
        for l_word in l_tuple:
            if l_word[0] != root:
                return False
            if bool(re.search(pattern, l_word)):
                return False
    return True


# ======================== ALGORITHMS REALIZATION ============================
def ar_nodes(graph: nx.Graph) -> nx.Graph:
    """" reduction algorithm AR """"
    G_ = graph.copy()
    trigger = True
    while trigger:
        trigger = False
        for node in G_.nodes:
            neighbors = list(G_.neighbors(node))
            labels = [G_.nodes[id]['label'] for id in neighbors]
            equals_labels = find_neighbours_with_the_same_labels(neighbors, labels)
            if equals_labels:
                for neighbours_ids in equals_labels.values():
                    not_changeable_node = min(neighbours_ids)
                    neighbours_ids.remove(not_changeable_node)
                    for vertex in neighbours_ids:
```

```
                nghb_of_del_node = list(G_.neighbors(vertex))
                nghb_of_del_node.remove(node)
                G_.add_edges_from((v_node, not_changeable_node)
                                  for v_node in nghb_of_del_node)
                G_.remove_node(vertex)
            trigger = True
            break
    return G_


def ap_graph(C:Tuple[str], L:Tuple[str], x_='1') -> Union[nx.Graph, str]:
    """ build graph on pair of words, algorithm AP """
    # ================================ STEP 0 ====================================
    if not word_pair_data_validation(C, L, x_):
        raise ValueError("Incorrect data. Graph is not exists!")
    q: Dict = {}
    trash: Dict = {}
    root = 0
    check_leaf_node = None
    G = nx.Graph()
    G.add_node(root, label=x_)
    # ======================= STEP 1 Construct the graph for C ============================
    for c_word in C:
        for index, label in enumerate(c_word[1:-1], start=1):
            custom_id = counter()
            G.add_node(custom_id, label=label)
            if index == 1:
                G.add_edge(root, custom_id)
            else:
                G.add_edge(custom_id - 1, custom_id)
            if index == len(c_word) - 2:
                G.add_edge(custom_id, root)
        G = ar_nodes(G)
    # ====================== STEP 2 Get all leaf nodes from the graph ========================
    q = get_all_leaf_nodes_from_graph(G)
    # ====================== STEP 3 Add nodes for L and check if the graph is valid ==============
    for l_word in L:
        for index, label in enumerate(l_word[1:], start=1):
            node_id = counter()
            G.add_node(node_id, label=label)
            if index == 1:
                G.add_edge(root, node_id)
            else:
                G.add_edge(node_id - 1, node_id)
            if index == len(l_word) - 1:
                check_leaf_node = node_id
        G = ar_nodes(G)
        if G.has_node(check_leaf_node):
            if G.degree(check_leaf_node) != 1:
                raise ValueError("Incorrect data. Graph is not exists!")
        all_leafs = get_all_leaf_nodes_from_graph(G)
        for key_id, val_label in all_leafs.items():
            if val_label != l_word[-1] \
                    and key_id not in q and key_id not in trash:
                q[key_id] = val_label
            else:
                trash[key_id] = val_label
    # ========================= STEP 4 Check if the graph is valid ========================
    for vq_id, vq_label in q.items():
        if not check_q_node(G, L, vq_id, vq_label, root):
            print(f"Incorrect data ID: {vq_id}, LBL: {vq_label}. Graph is not exists!")
    # ==================== STEP 5 Check if each word in L ends with a hanging vertex ============
```

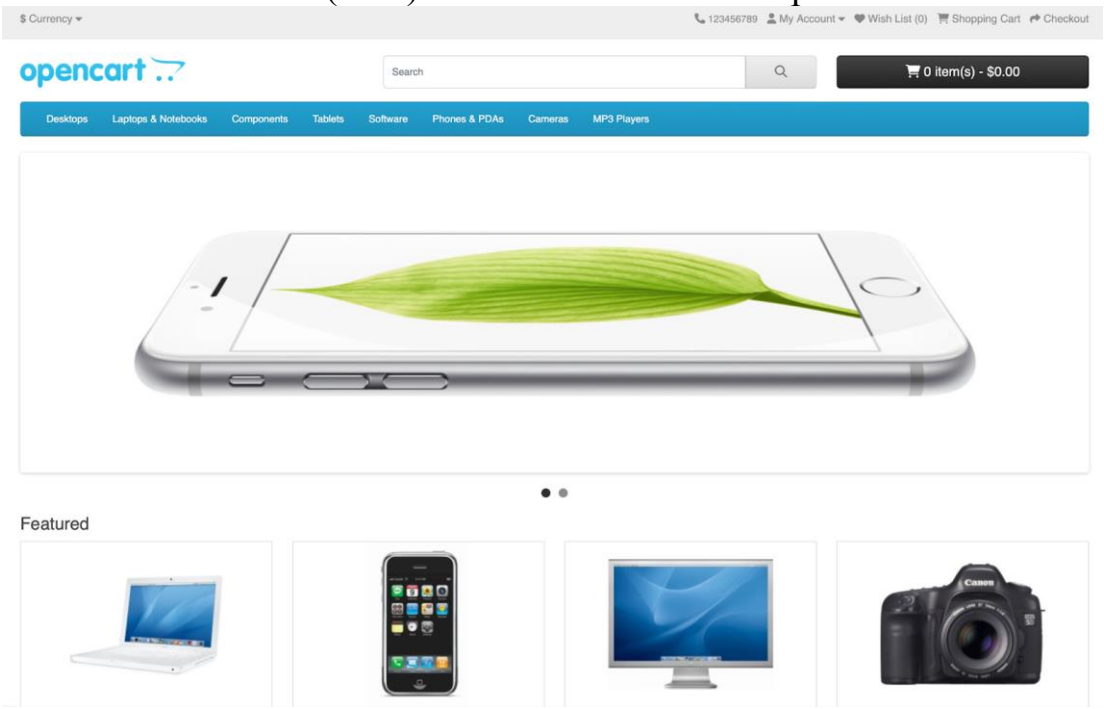
```
    for p_word_index, p_word in enumerate(L):
        checked_node = walk_by_word(G, p_word, root)
        if G.degree(checked_node) != 1:
            print(f"""Incorrect data, invalid pair.
            \nWord {p_word_index + 1} in the set L does not end with a hanging vertex.
            \nGraph is not exists!""")

    return G


def ac_pair(graph: nx.Graph) -> Union[Tuple[List[str], List[str]], int, str]:
    """ get canonical pair of words, algorithm AK """
    # ================================ base checks ==================================
    if len(graph.nodes) == 0:
        return 0
    if len(graph.nodes) == 1:
        try:
            return graph.nodes[0]['label']
        except KeyError:
            return list(graph.nodes)[0]
    # ============================== local definitions ================================
    root = list(graph.nodes)[0]
    sigma_g: List[str] = []
    lambda_g: List[str] = []
    reachability_basis: Dict[str, List[int]] = {}
    # ================== Find reachability basis in the graph and fill lambda_g ================
    ms_tree = nx.minimum_spanning_tree(graph)
    for node in ms_tree.nodes:
        node_path_id = nx.shortest_path(ms_tree, source=root, target=node)
        node_path_labels = [ms_tree.nodes[id]['label'] for id in node_path_id]
        reachability_basis[''.join(node_path_labels)] = node_path_id
        if graph.degree(node) == 1 and node != root:
            lambda_g.append(''.join(node_path_labels))
    # ================ create ni var as reachibility basis list without lambda_g values =============
    ni = [w for w in reachability_basis if w not in lambda_g]
    ni.pop(root)
    # ======================== Find cycles by ni and fill sigma_g ============================
    for index, p_path in enumerate(ni):
        for q_path in ni[index+1:]:
            if p_path not in q_path[:len(p_path)]:
                if not graph.has_edge(reachability_basis[p_path][-1],
                                      reachability_basis[q_path][-1]):
                    continue
                pqr = p_path + q_path[::-1]
                qpr = q_path + p_path[::-1]
                if pqr < qpr:
                    sigma_g.append(pqr)
                else:
                    sigma_g.append(qpr)
    return (sigma_g, lambda_g)

# ======================== CANONICAL PAIR METRICS ============================
def get_canonical_pair_metrics_from_graph(graph: nx.Graph) -> dict:
    """ find canonical pair metrics by graph values"""
    canonical_pair: Union[Tuple[List[str], List[str]], int, str] = ac_pair(graph)
    total_pair_count = 0
    if isinstance(canonical_pair, tuple):
        for c_word in canonical_pair[0]:
            total_pair_count += len(c_word)
        for l_word in canonical_pair[1]:
            total_pair_count += len(l_word)
    else:
```

```
    raise ValueError("Incorrect data. Graph is not exists!")
n_nodes = graph.number_of_nodes()
m_edges = graph.number_of_edges()
delta = ceil(3/2 + sqrt(9/4 - 2 * n_nodes + 2 * m_edges))
result = 2 * (m_edges - n_nodes + 1) * (n_nodes - delta + 2)
mu_mn = ceil(delta*(delta-1)/2-(m_edges-n_nodes+delta))
power_of_sigma_g = ceil((delta-mu_mn -1)*(delta-mu_mn-2)*(n_nodes-delta+2)+\
                mu_mn*(mu_mn-1)*(n_nodes-delta+3)+\
                mu_mn*(delta-mu_mn-2)*(2*n_nodes-2*delta+5))
return {
    "n": n_nodes,
    "m": m_edges,
    "delta": delta,
    "formula_result": result,
    "total_p_count": total_pair_count,
    "mu": mu_mn,
    "power_sigma_G": power_of_sigma_g,
    "canonical_pair": canonical_pair
}
```

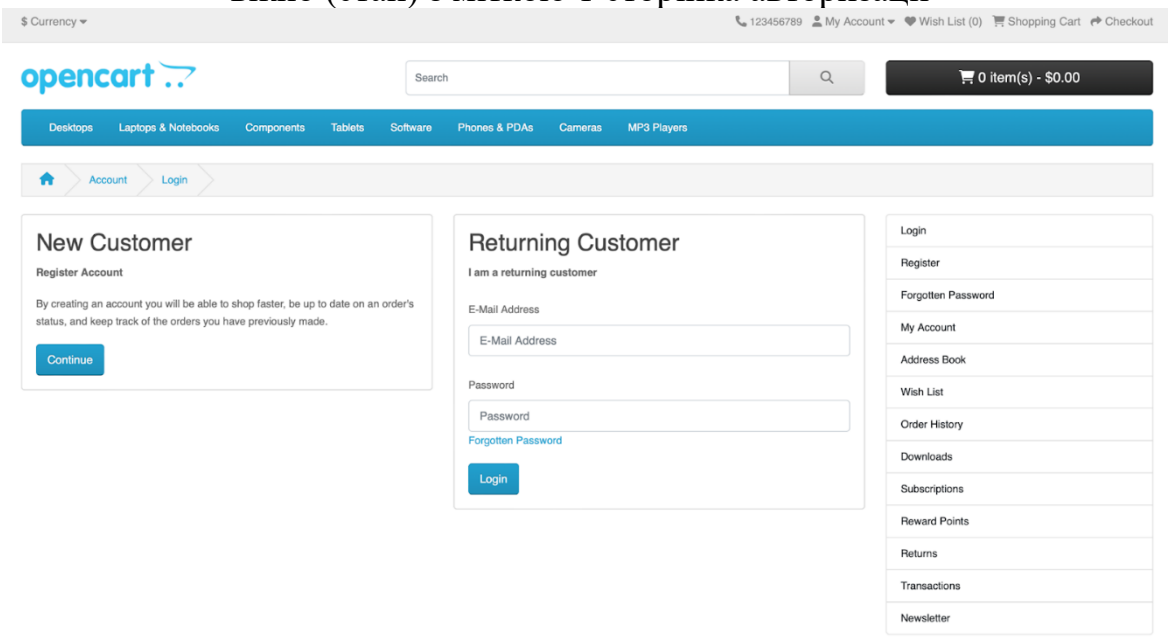
Додаток Д

Перелік вікон web-додатку з відповідними мітками

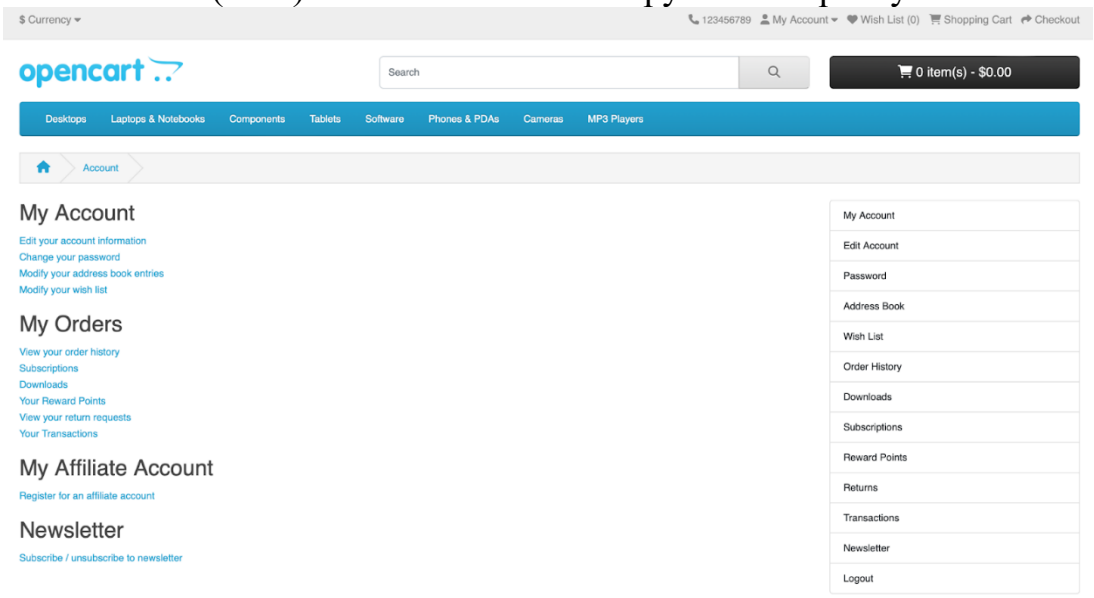
вікно (стан) з міткою 0 головна сторінка



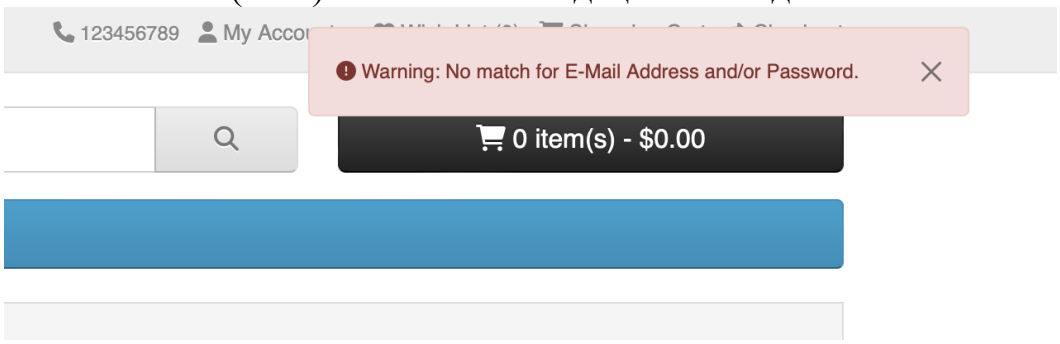
вікно (стан) з міткою 1 сторінка авторизації



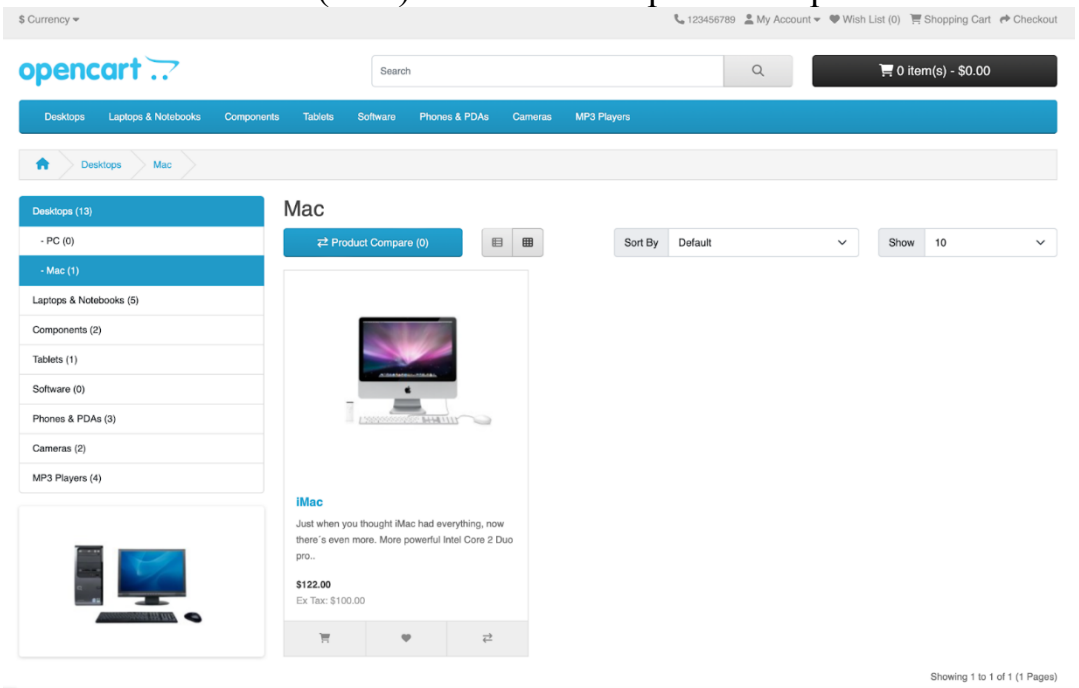
вікно (стан) з міткою 2 панель керування користувача



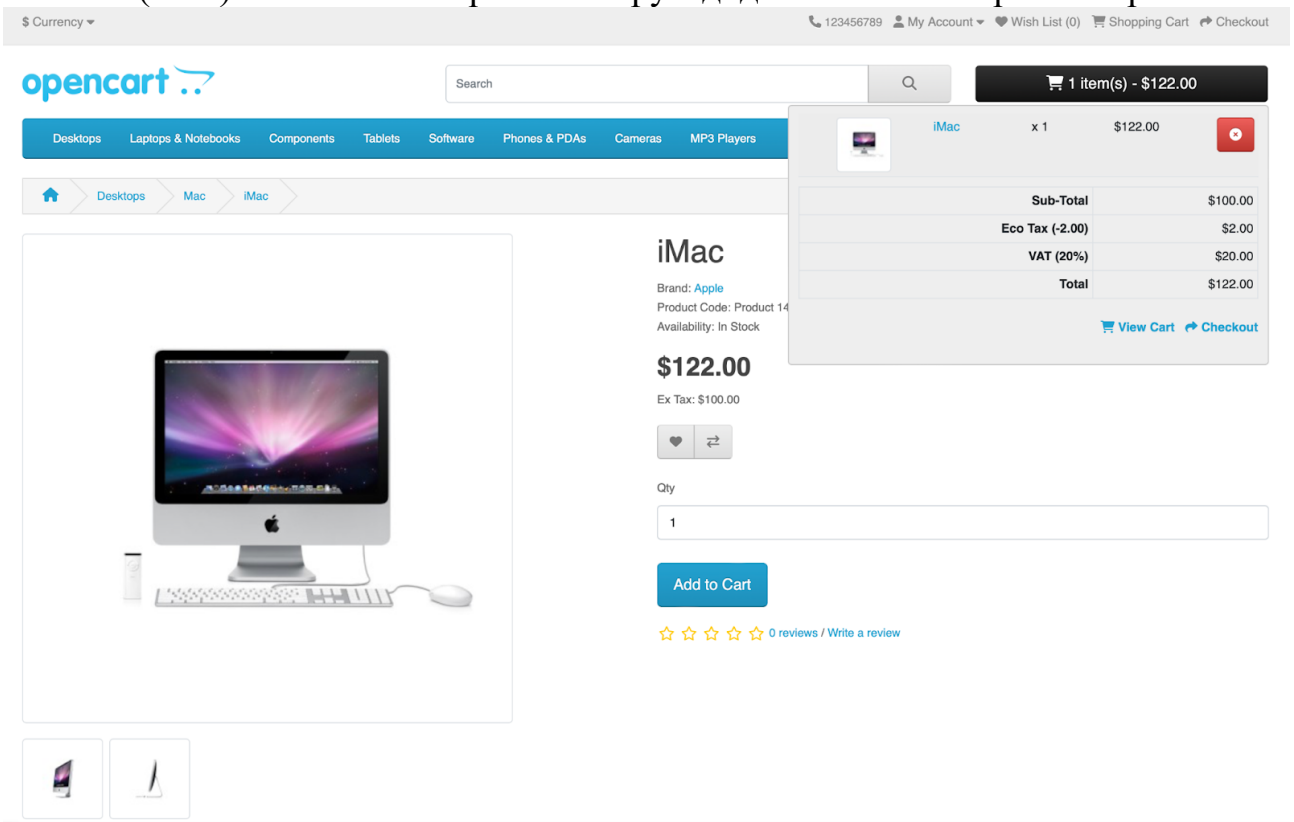
вікно (стан) з міткою 3 валідаційне повідомлення



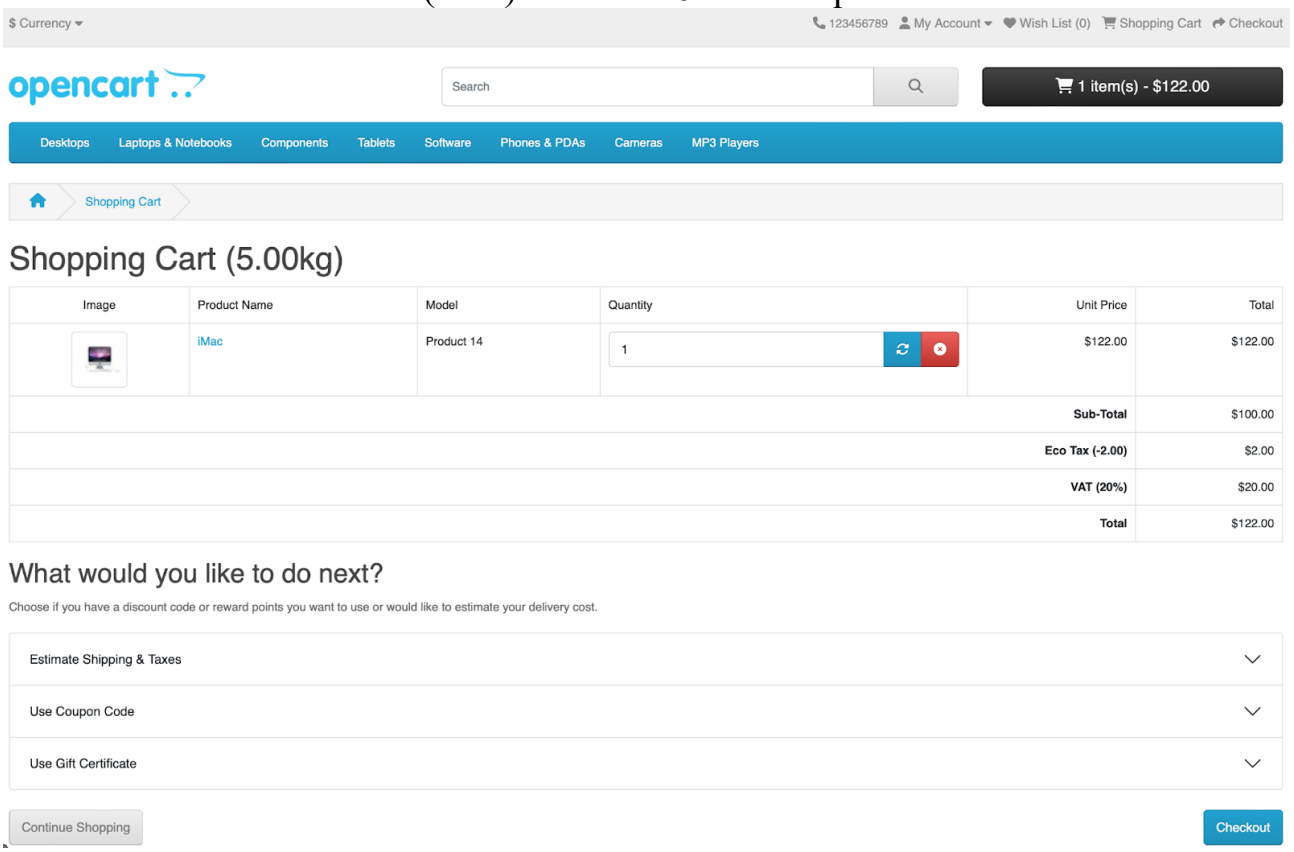
вікно (стан) з міткою 4 сторінка товарів



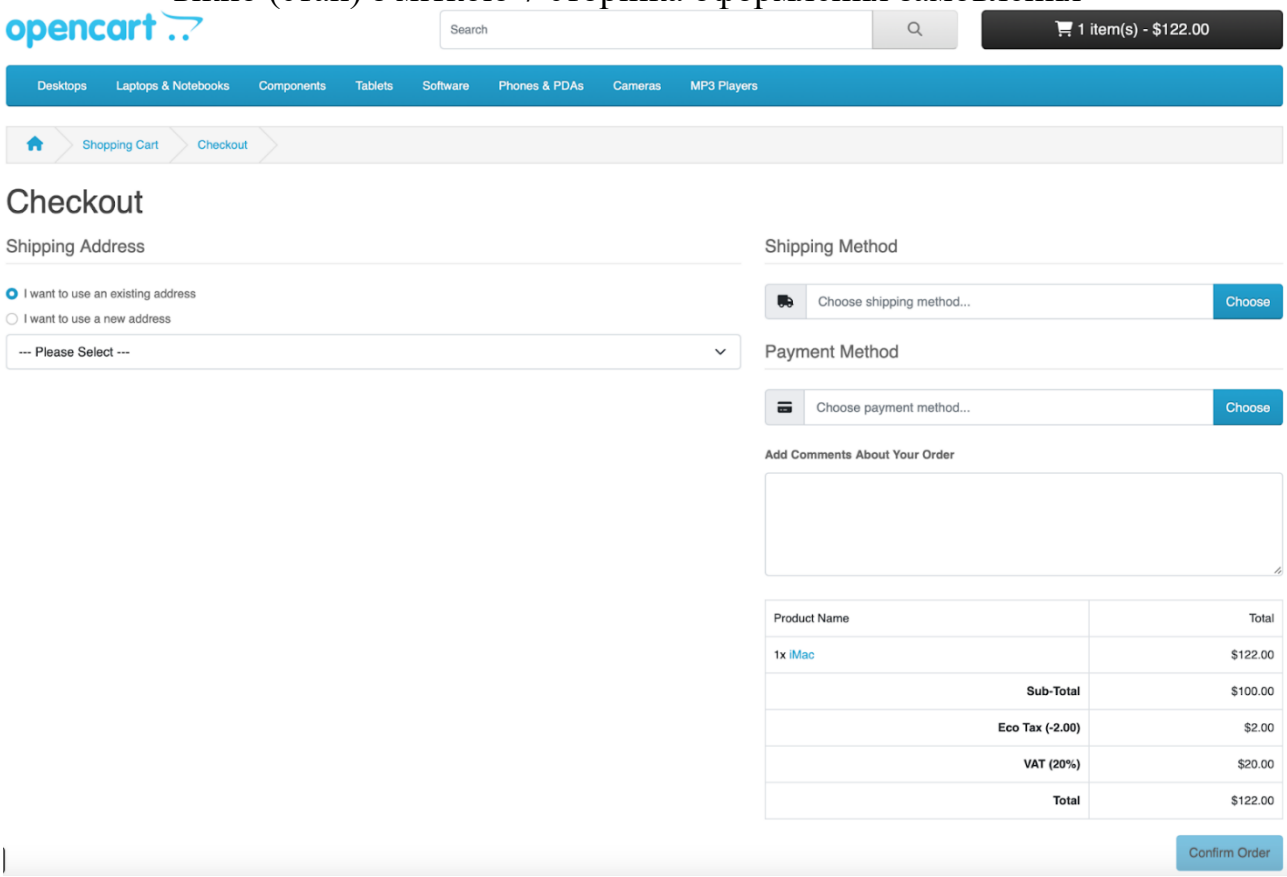
вікно (стан) з міткою 5 сторінка товару з доданим екземпляром в корзині



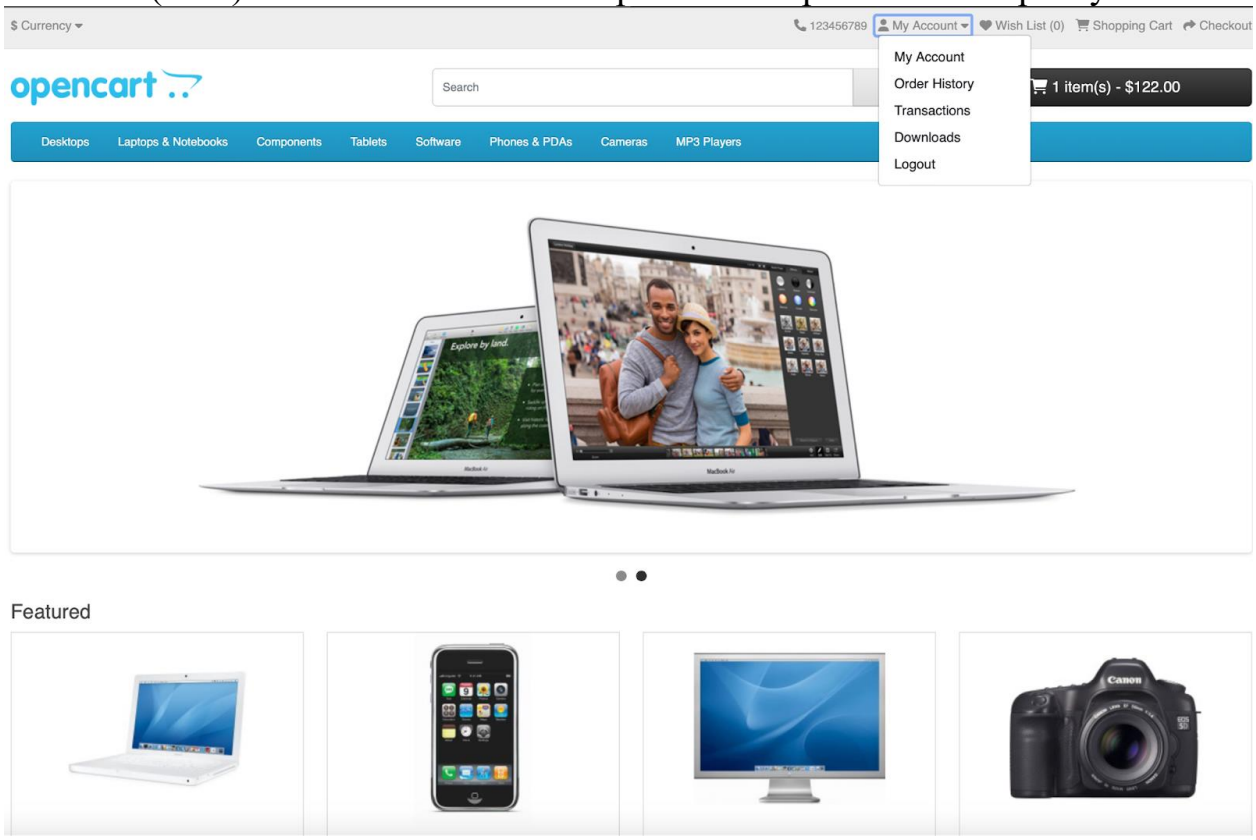
вікно (стан) з міткою 6 вікно корзини



вікно (стан) з міткою 7 сторінка оформлення замовлення



вікно (стан) з міткою 8 головна сторінка з авторизованим користувачем



Рецензія
на кваліфікаційну роботу магістра
Студента групи *124м-22 Притули Миколи Ігоровича*
Спеціальності 124 Системний аналіз

Тема кваліфікаційної роботи: *«Використання лінгвістичного представлення детермінованих графів у тестуванні програмного забезпечення».*

Обсяг кваліфікаційної роботи магістра: 66с., 19 рис., 5 додатків, 29 джерел.

Висновок про відповідність роботи завданню та освітньо-професійній програмі спеціальності – кваліфікаційна робота відповідає вимогам до професійної діяльності фахівця освітньо-кваліфікаційного рівня магістра спеціальності 6.040303 Системний аналіз. .

Зміст пояснювальної записки відповідає темі кваліфікаційної роботи.

Загальна характеристика кваліфікаційної роботи, ступінь використання нормативно-методичної літератури та передового досвіду. Кваліфікаційна робота містить два розділи. В інформаційно-аналітичному розділі наведено коротку історію теорії представлення різноманітних математичних об'єктів та структур, основні поняття та означення, які пов'язані з детермінованими графами, та виділено напрямки застосування таких графів і сучасний стан теорії лінгвістичного представлення детермінованих графів із перспективними напрямками дослідження. У спеціальному розділі наведено програмну реалізацію прототипів алгоритму побудови графа та алгоритму побудови канонічної визначальної пари для детермінованого графа. Подано приклад поширення методики побудови тестового покриття програмного забезпечення із застосуванням розроблених алгоритмів на прикладі аналізу прототипу web-додатку.

Позитивні сторони **кваліфікаційної роботи**: використовуються новітні наукові результати, над якими на даний час ведеться робота по їх вдосконаленню, розроблено програмний модуль, який реалізує алгоритми, що подані в інформаційно-аналітичному розділі роботи, запропоновано оригінальну методику, яка може розширити та доповнити інші сучасні методи побудови тестового покриття програмного забезпечення.

Основні недоліки кваліфікаційної роботи: теоретичні результати отримані нещодавно та не впроваджені у сучасний інженерний стандарт, при цьому не доведено універсальність запропонованого методу моделювання тестового покриття, результат варто було б порівняти з іншими методами.

Кваліфікаційна робота магістра в цілому заслуговує оцінки: _____, а її автор Притула М.І. заслуговує присвоєння кваліфікації "магістр з системного аналізу".

Рецензент,