УДК 519.17+004.4
DOI https://doi.org/10.26661/2413-6549-2022-1-09

ОГЛЯД ЗАСТОСУВАНЬ ТЕОРІЇ ГРАФІВ У РОЗРОБЦІ ТА ОБСЛУГОВУВАННІ ПРОГРАМНИХ СИСТЕМ

Москалик Д. О.
апспірант кафедри інженерії програмного забезпечення
Державний університет «Житомирська політехніка»
вул. Чуднівська, 103, Житомир, Україна
orcid.org/0000-0002-4421-9325
d.moskalyk@sana-commerce.com

Антонюк Д. С.
кандидат педагогічних наук, доцент,
доцент кафедри інженерії програмного забезпечення
Державний університет «Житомирська політехніка»
вул. Чуднівська, 103, Житомир, Україна
orcid.org/0000-0001-7496-3553
dmitry_antonyuk@yahoo.com

Вакалюк Т. А.
доктор педагогічних наук, професор,
професор кафедри інженерії програмного забезпечення
Державний університет «Житомирська політехніка»
вул. Чуднівська, 103, Житомир, Україна
orcid.org/0000-0001-6825-4697
tetianavakaliuk@gmail.com

Огінський Є. В.
аспірант кафедри інженерії програмного забезпечення
Державний університет «Житомирська політехніка»
вул. Чуднівська, 103, Житомир, Україна
orcid.org/0000-0002-7777-8449
oginsky2@gmail.com

Ковалевський В. В.
аспірант кафедри інженерії програмного забезпечення
Державний університет «Житомирська політехніка»
вул. Чуднівська, 103, Житомир, Україна
orcid.org/0000-0001-7144-1899
s.kovalevsky@sana-commerce.com

Ключові слова: *граф, теорія графів, розробка програмних систем, обслуговування програмних систем, застосування теорії графів.*

Теорія графів здобула неабияку популярність у великій кількості сфер застосування завдяки своїй універсальності, в тому числі зручності моделювання та наочності візуалізації. Загалом, у вигляді графу можна представити довільну модель об'єктів реального світу та зв'язків між ними, що дозволяє застосовувати до нього будь-які відомі алгоритми та піддавати іншим видам подальшого аналізу.

Теоретичні основи графів також широко використовуються у сфері дослідження операцій. Наприклад, завдання комівояжера, пошук найменшого кістякового дерева у зваженому графі, знаходження найкоротшого шляху між двома вершинами та інші. Метою цієї роботи

є саме аналіз практичного застосування теорії графів як безпосередньо в розробці програмного забезпечення, так і для його розробки, а також у разі подальшого обслуговування та підтримки програмних систем.
Окрім використання теоретичних основ графів в інших галузях науки, теорія графів також знайшла широке застосування у вирішенні багатьох практичних задач у великому спектрі життєдіяльності людини, як-то у хімії, соціології, логістиці, біології, економіці, техніці та ін.
Так само і в розробці програмного забезпечення теорія графів посіла своє почесне місце. За її допомогою вирішуються задачі багатьох різних видів, як у процесі розробки програм, так і для розвитку самої програмної інженерії.
У результаті проведеної роботи з аналізу можливостей практичного застосування теорії графів для розробки та обслуговування програмних систем було виявлено той факт, що на більшості етапів циклу розробки програмного забезпечення графи використовуються тільки для моделювання та візуалізації різних видів інформації, проте майже відсутній автоматизований алгоритмічний аналіз отриманих моделей. Це може свідчити про наявність перспективного напряму для пошуку додаткових можливостей застосування графів у процесах розробки програмного забезпечення.

AN OVERVIEW OF USING GRAPH THEORY IN SOFTWARE SYSTEM DEVELOPMENT AND MAINTENANCE

Moskalyk D. O.
Postgraduate Student at the Department of Software Engineering
Zhytomyr Polytechnic State University
Chudnivska str., 103, Zhytomyr, Ukraine
orcid.org/0000-0002-4421-9325
d.moskalyk@sana-commerce.com

Antoniuk D. S.
Candidate of Pedagogical Sciences, Associate Professor,
Associate Professor at the Department of Software Engineering
Zhytomyr Polytechnic State University
Chudnivska str., 103, Zhytomyr, Ukraine
orcid.org/0000-0001-7496-3553
dmitry_antonyuk@yahoo.com

Vakaliuk T. A.
Doctor of Pedagogical Sciences, Professor,
Professor at the Department of Software Engineering
Zhytomyr Polytechnic State University
Chudnivska str., 103, Zhytomyr, Ukraine
orcid.org/0000-0001-6825-4697
tetianavakaliuk@gmail.com

Ohinskyi Ye. V.
Postgraduate Student at the Department of Software Engineering
Zhytomyr Polytechnic State University
Chudnivska str., 103, Zhytomyr, Ukraine
orcid.org/0000-0002-7777-8449
oginsky2@gmail.com

Kovalevskyi V. V.
Postgraduate Student at the Department of Software Engineering
Zhytomyr Polytechnic State University
Chudnivska str., 103, Zhytomyr, Ukraine
orcid.org/0000-0001-7144-1899
s.kovalevsky@sana-commerce.com

Key words: *graph, graph theory, software systems development, software systems maintenance, graph theory applications.*

Graph theory has gained considerable popularity in many areas of application due to its versatility, including ease of modeling and visualization. In general, a graph can represent an arbitrary model of real-world objects and the relationships between them, which allows applying any known algorithms to it and subject it to other types of further analysis.
The theoretical foundations of graphs are also widely used in the field of operations research. For example, the travelling salesman problem, finding the minimum spanning tree of a weighted graph, finding the shortest path between two vertices etc. The purpose of this work is to analyze the practical applications of graph theory, both in the process of software development and for software development in general, as well as in the subsequent maintenance and support of software systems.
In addition to using the theoretical foundations of graphs in other branches of science, graph theory has also found wide application in solving many practical problems in different areas of human life, such as chemistry, sociology, logistics, biology, economics, technology and so on. Similarly, in software development, graph theory has taken its rightful place. Many kinds of problems are solved with its help, both in the process of software engineering and for the development of software engineering itself.
The analysis of the graph theory application opportunities for software development and maintenance has shown that at most stages of the software development cycle, graphs are used only for modeling and visualization of various kinds of information, but there is almost no automated algorithmic analysis of the obtained models. This may indicate a promising direction for discovering additional opportunities of graph theory usage in the software development processes.

Вступ. У сучасному світі теорія графів знаходить своє місце, мабуть, в усіх сферах діяльності людини. Її використовують як в інших галузях науки, так і у вирішенні багатьох прикладних задач. З її допомогою досліджують молекули, атоми, хімічні зв'язки. Вона широко використовується у соціології для вимірювання популярності акторів чи для дослідження механізмів дифузії інновацій. Теорія графів застосовується в біології для відслідковування міграції живих істот, де регіони проживання позначаються вершинами графу, а ребра відображають шляхи міграції чи переміщення між різними регіонами. Ця інформація дуже важлива для дослідження поширення захворювань, паразитів та вивчення впливу міграцій одного виду на життєдіяльність іншого.

Теоретичні основи графів також широко використовуються у сфері дослідження операцій. Наприклад, завдання комівояжера, пошук найменшого кістякового дерева у зваженому графі, знаходження найкоротшого шляху між двома вершинами та інші.

Неможливо не згадати застосування теорії графів у моделюванні транспортних мереж та мереж діяльності. Мережі використовують для розв'язання великої кількості комбінаторних задач. Найбільш поширене та вдале застосування мереж у дослідженні операцій – це планування великих та складних проєктів.

Застосування графів до теорії ігор дозволяє розв'язувати проблеми в машинобудуванні, економіці та військовій справі для пошуку оптимального плану виконання визначених задач у конкурентному середовищі. Для репрезентації скінченої гри використовують орієнтований граф, в якому позиції представлені вершинами, а ребра відображають можливі ходи.

Мета цієї роботи – здійснити аналіз практичного застосування теорії графів у розробці програмного забезпечення у разі подальшого обслуговування та підтримки програмних систем.

Результати. Найпершим, найочевиднішим та найширшим варіантом застосування графів є різноманітні структури даних. Насамперед ідеться

про сам граф як окрему структуру даних. За допомогою графу дані організовуються таким чином, щоб відображати об'єкти разом з відповідними зв'язками між ними. Частіше за все, графи використовуються для ефективного зберігання даних в оперативній пам'яті, але існують також і спеціалізовані системи для зберігання великих об'ємів даних, представлених у вигляді графу.

Одним з типів таких систем є ієрархічні бази даних, такі як, наприклад, "Neo4j" чи "Amazon Neptune", які орієнтовані на збереження об'єктів разом з їх властивостями у вигляді вершин графу та зв'язків між ними, представлених ребрами. Графові бази даних підтримують спеціальні мови запитів, що дозволяють оперувати даними в графі, в тому числі виконувати загальновідомі алгоритми над графами, наприклад, пошук найкоротшого шляху тощо.

Одним з яскравих прикладів моделей даних, які найзручніше зберігати саме в графовій базі даних, є модель соціальної мережі, наприклад, "Facebook", оскільки саме зв'язки людей з іншими людьми, спільнотами, їх вподобання відіграють ключову роль в ефективному функціонуванні соціальної мережі.

Ще однією дуже поширеною структурою даних, яка за своєю суттю також являє собою граф, є дерево. На практиці використовують досить багато різних видів дерев, відштовхуючись від конкретних вимог. Серед найпоширеніших структур даних можна відзначити пошукові дерева та їх варіації, пріоритетні дерева, префіксні дерева та інші. Деревоподібні структури даних переважно використовуються для зберігання ієрархічних даних та для оптимізації швидкодії у разі імплементації різноманітних алгоритмів.

Найпоширенішими прикладами застосування дерев є файлові системи, ієрархії класів у більшості об'єктно-орієнтованих мов програмування, абстрактні синтаксичні дерева, об'єктні моделі XML та HTML документів тощо. На рисунку 1 зображено приклад бінарного пошукового дерева, де в кожному лівому піддереві містяться тільки вершини з меншим ключем, ніж у предка, а в правому – тільки з більшим ключем.

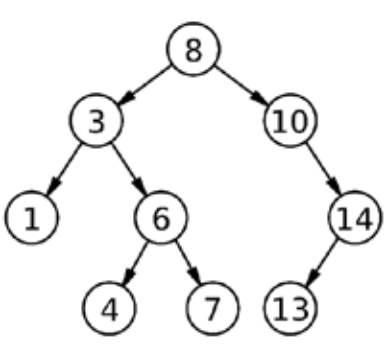

Рис. 1. Приклад бінарного пошукового дерева

Однією з найширших сфер використання графів є саме моделювання об'єктів реального світу для їх подальшого аналізу. Графи, на відміну від списків чи таблиць, є дуже зручною формою візуалізації об'єктів та їх зв'язків, що може зумовлювати таку високу популярність їх використання в моделюванні.

Найпершим прикладом можна навести представлення скінченого автомату у вигляді орієнтованого графу, де вершинами графу відображаються різні стани скінченного автомату, а ребрами – переходи з одного стану в інший. Оскільки скінченні автомати дуже широко використовуються в розробці програмного забезпечення, візуалізація за допомогою графу допомагає швидше зрозуміти механізм роботи конкретного скінченного автомату та дозволяє без зайвих зусиль перевірити автомат на коректність станів та переходів. На рисунку 2 наведений приклад представлення скінченного автомату у вигляді графу.

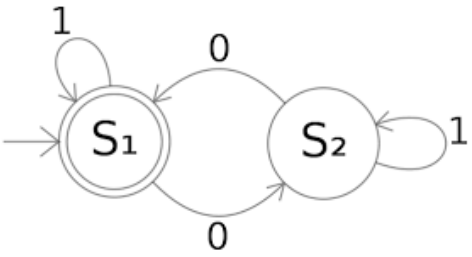

Рис. 2. Скінченний автомат у вигляді графу

Моделювання за допомогою графів застосовується не тільки під час розробки програмного забезпечення, але і на більш ранніх стадіях, як-от у разі аналізу функціональних вимог майбутньої програмної системи.

На етапі збору та аналізу функціональних вимог будують діаграму потоків даних, яка візуалізує процеси перетворення даних, сховища даних, зовнішніх стосовно системи сутності, та потоки даних між ними. Потоки даних подають у вигляді ребер графу, які сполучають інші типи елементів діаграми між собою, представлених вершинами графу [3, с. 27].

Абстрактний приклад діаграми потоків даних зображено на рисунку 3.

Проєктування програмної системи також не обходиться без моделювання її структури і зазвичай це також моделюється у вигляді графу, оскільки програмне забезпечення середнього розміру та більше має досить складну структуру, яку дуже важко зобразити за допомогою простіших графічних форм.

На поточний момент одним з найпоширеніших способів візуалізації архітектури програмної системи є UML-діаграми, які поділяють на структурні, поведінкові та діаграми взаємодії, але нині

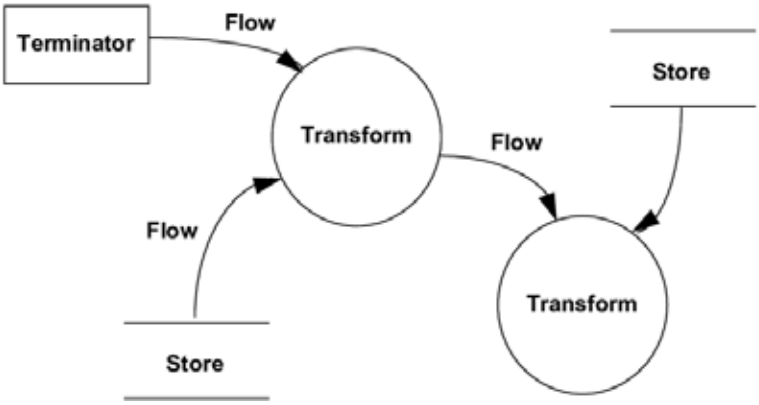

Рис. 3. Абстрактна діаграма потоків даних

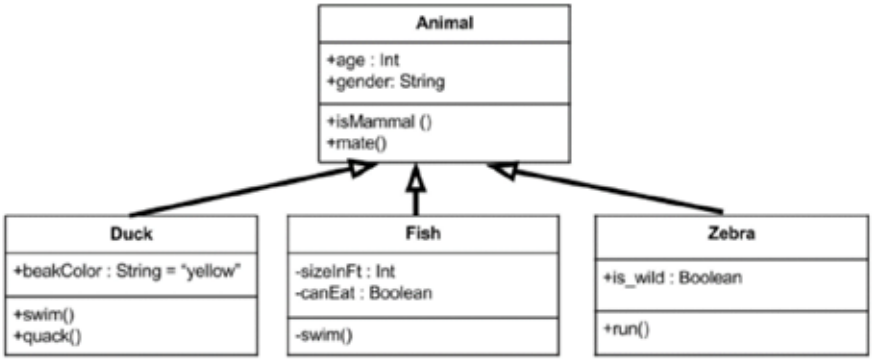

Рис. 4. UML-діаграма класів

набирає все більшої популярності «модель С4» для візуалізації архітектури програмного продукту [4].

Як і в більшості UML-діаграм, так і в «моделі С4» усі діаграми являють собою орієнтовані або неорієнтовані графи з додатковим інформаційним навантаженням, що означає, що вони також піддаються тим самим методам аналізу графів. На рисунку 4 зображений приклад UML-діаграми класів.

Не можна також не згадати, що графи широко використовуються для моделювання комп'ютерних мереж. Такий підхід також дуже стає в нагоді як у разі проєктування розподіленого програмного комплексу, так і у разі його розгортання, особливо в мережі хмарних ресурсів, оскільки топологія внутрішньої мережі розподіленої системи зазвичай має дуже значний вплив на його швидкодію загалом.

У 1976 році Дж.П. Хейз запропонував підхід до моделювання відмово-стійких програмних систем. Для цього використовуються два графи. Перший – граф програмної системи, в якому вершинами представлені всі програмні та апаратні обчислювальні засоби, які можуть у будь-який момент вийти з ладу, а ребрами виступають комунікаційні зв'язки між ними. Другий – це граф алгоритму, чиї вершини відображають обчислювальні засоби, необхідні для виконання такого алгоритму, а ребра – необхідні зв'язки між ними. Для того щоб система могла виконати такий алгоритм, граф алгоритму повинен бути ізоморфним до породженого підграфу системи. Видаляючи з початкового графу системи вершини та їх суміжні ребра, які відображають ті обчислювальні засоби, що можуть вийти з ладу одночасно, можна робити висновки чи такий алгоритм все ще може бути виконаний поточною програмною системою шляхом повторної перевірки умови ізоморфізму графу алгоритму до породженого підграфу програмної системи [5].

Ще одним корисним прикладом моделювання в розробці програмного забезпечення є створення так званого «вебграфу» – орієнтованого графу багатосторінкового вебдодатку, де вершинами графу виступають окремі вебсторінки, а ребрами – можливі переходи (посилання чи переадресації) з однієї сторінки на іншу. На рисунку 5 зображено приклад графу вебблогу.

Така модель даних корисна насамперед для ідентифікації недосяжних сторінок, але також широко використовується для пошуку довжин шляхів з початкової сторінки до цільової задля покращення користувацького досвіду.

Рис. 5. Граф переходів між сторінками вебблогу

Звичайно ж, першим та найочевиднішим застосуванням є завдання розфарбовування суміжних країн на політичній карті світу [3, с. 368], але це є далеко не єдиний приклад використання. Такий підхід також застосовується у разі розфарбовування дротів або доріжок на електронній схемі [3, с. 375], для виділення радіочастот під суміжні радіостанції, в яких перетинаються зони трансляції [3, с. 353; 6], а також для відокремленого зберігання хімічних сполук, які можуть спалахнути у разі їх змішування [3, с. 354]. На рисунку 6 зображено приклад графу накладання радіочастот.

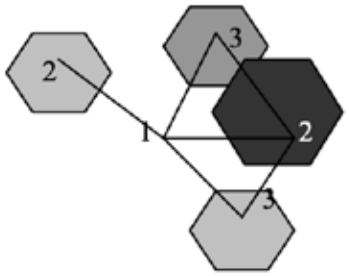

Рис. 6. Граф накладання радіочастот

Та у сфері інформаційних технологій також є досить прикладів застосування проблеми розфарбовування графів. Найбільш широкою сферою використання проблеми розфарбовування графів є вирішення конфліктів у разі складання розкладу екзаменів, планування виконання роботи тощо.

В операційних системах розфарбовування графу застосовують для планування роботи процесору, де різні задачі представлені вершинами графу, а ребра з'єднують ті з них, які не повинні виконуватись одночасно.

У деяких обчислювальних системах є обмежена кількість процесорних регістрів спеціального призначення, які дозволяють пришвидшити виконання деяких алгоритмічних операцій порівняно з використанням звичайної пам'яті. Програмні змінні, які використовуються найчастіше, можуть бути позначені як ті, що потрібно зберігати в даних регістрах, проте, якщо кількість змінних буде перевищувати кількість доступних спеціальних регістрів, то час, витрачений на переміщення значень змінних зі звичайної пам'яті в регістри та назад, буде повністю нівелювати користь від приросту швидкодії під час використання цих регістрів.

Для вирішення такої проблеми всі змінні представляються у вигляді вершин графу, а ребрами з'єднуються ті з них, що можуть бути активними одночасно. В такому випадку хроматичне число графу процедури буде дорівнювати кількості необхідних спеціальних регістрів, що дає змогу уникати надмірного копіювання змінних [3, с. 354].

У матеріалах курсу лекцій з теорії графів Массачусетського технологічного інституту 2010 року професор Том Лейтон наводив приклад застосування проблеми розфарбовування графів для оновлення програмного забезпечення на серверах компанії "Akamai" [7, с. 146].

Френк Томсон Лейтон – генеральний виконавчий директор та співзасновник американської компанії "Akamai Technologies", яка пропонує послуги мережі доставки контенту, кібербезпеки та розподілених обчислень. Платформа "Akamai Intelligent Edge Platform" є нині однією з найбільших платформ розподілених обчислень у світі [8].

Проблема, яка постала перед компанією, в той момент полягала в такому. У глобальній мережі компанії на той момент використовувались більше ніж 75000 серверів, на кожному з яких працювало їх власне програмне забезпечення різних видів, серед яких – і системи маршрутизації трафіку, і системи зберігання даних тощо. Поставлене завдання полягало в необхідності оновлення програмного забезпечення на всіх серверах компанії без зупинки постачання послуг кінцевим споживачам.

Для оновлення програмного забезпечення на одному сервері такий сервер потрібно було відключати від загальної мережі, а сам процес оновлення тривав близько однієї години. У разі неупорядкованого паралельного оновлення серверів є ризик відмови всієї системи, якщо всі екземпляри програм одного типу призначення будуть вимкнені одночасно. У разі оновлення усіх серверів по черзі процес повного оновлення тривав би близько 8,5 року.

Для вирішення цього завдання вся мережа серверів була представлена у вигляді неорієнтованого графу, де вершинами були сервери, а ребра з'єднували попарно сервери, на яких виконувались задачі однакового типу. Далі граф розфарбовувався в найменшу необхідну кількість кольорів, що гарантувало можливість одночасного оновлення всіх серверів одного кольору без ризику відмови всієї системи. Таким чином, у компа-

нії “Akamai” змогли оновити всі 75000 серверів усього за 8 «хвиль» оновлень.

Іншою категорією задач, які вимагають вирішення в комп’ютерних системах, є пошук шляхів у графах. Як уже згадувалось раніше, теорія графів дуже часто використовується для моделювання комп’ютерних мереж. Основною ціллю моделювання мережі є, звичайно, відображення її складників, структури та топології, але також така модель є дуже зручною для подальшого її аналізу, наприклад, для пошуку найкоротшого шляху для маршрутизації трафіку між двома вузлами чи для знаходження максимального потоку трафіку в поточній мережі.

Задача оптимізації потоків даних є актуальною не тільки для організації комп’ютерних мереж, але також і для їх подальшого використання. Для прикладу, топологія розміщення компонентів розподіленої програмної системи в різних мережах буде мати дуже суттєвий вплив на кінцеву швидкодію такої системи загалом. У такому випадку аналізу підлягають як довжини шляхів комунікації між різними компонентами, так і інтенсивність та об’єм цих комунікацій, що своєю чергою вимагає аналізу потоків та пропускних здібностей каналів зв’язку між окремими віддаленими складниками програмної системи.

Ще одним наочним прикладом необхідності аналізу шляхів у графі є реалізація так званих «збирачів сміття» у деяких віртуальних машинах для виконання проміжного коду з автоматичними механізмами керування пам’яттю. Для прикладу, однією з таких віртуальних машин є платформа для розробки додатків “.NET” від компанії “Microsoft”. Тоді як віртуальна машина виявляє факт нестачі керованої пам’яті для подальшого виділення під нові дані, запускається процес «збирання сміття», який аналізує, які раніше виділені об’єкти більше не потрібні і можуть бути видалені для звільнення зайнятої ними області пам’яті. Для виконання поставленої задачі «збирач сміття» аналізує граф посилань між всіма наявними об’єктами, починаючи з кореневих об’єктів програми, та видаляє ті вершини-об’єкти, які є недосяжними, тобто до яких не існує шляху в графі від початкових вершин.

У сучасному світі існує дуже багато засобів, які допомагають швидше та краще розробляти програмне забезпечення більш високої якості. Зазвичай більшість таких засобів вбудовані в інтегровані середовища розробки програмного забезпечення або прив’язані до конкретної мови програмування або середовища виконання програм, але інколи трапляються і окремі програмні комплекси, що створені для тієї ж мети. Однією з основних функцій таких засобів є допомога в написанні коректного, зручного для читання та якісного коду.

Всі ці засоби та системи об’єднує одна спільна риса – найкращі з них оперують вихідним кодом, представленим у вигляді графів. Вони використовують як синтаксичне абстрактне дерево коду, так і семантичне, а також надбудовують ще і додаткові довільні графи для подальшого більш високорівневого аналізу.

Прикладами задач, які допомагають вирішити такі автоматизовані засоби, можна назвати пошук конструкцій коду, які ніколи не використовуються, посилання на неіснуючі змінні чи класи, знаходження конфліктів імен змінних у спільній області видимості, виявлення недосяжних інструкцій у тілі функції та загублених необхідних розгалужень коду та багато чого іншого [9].

Можливості для розширення аналізу вихідного коду на базі графів ще далеко не вичерпані і мають досить великий потенціал. Основним обмеженням можливостей для такого аналізу є те, яку інформацію можна включити в такий граф, а яку ні. Саме тому мови програмування зі статичною типізацією (наприклад, C#, F#, Java, TypeScript) набагато краще піддаються аналізу, аніж мови з динамічною типізацією (наприклад, JavaScript, Ruby, Python).

Задача пошуку найменшого вершинного покриття графу також знайшла своє застосування у сфері програмної інженерії, а саме в розробці систем кіберзахисту в реальному часі. У такому випадку алгоритм пошуку найменшого вершинного покриття графу використовується для симуляції поширення скритних вірусів-хробаків у великих комп’ютерних мережах та розробки оптимальної стратегії захисту мережі від вірусів у реальному часі. Проведена симуляція у великій інтернет-подібній віртуальній мережі показала, що топологія маршрутизації має дуже великий вплив на швидкість поширення шкідливого програмного забезпечення в мережі. Важливість вивчення механізму поширення вірусу-хробака полягає в необхідності перешкоджання його поширенню в режимі реального часу. Головною ідеєю тут є пошук найменшого вершинного покриття графу, вершинами якого є сервери маршрутизації, а ребрами – зв’язки між серверами. У разі знаходження оптимального механізму поширення вірусу можна побудувати ефективну стратегію захисту мережі в реальному часі [10].

Висновки. Теорія графів здобула неабияку популярність у великій кількості сфер застосування завдяки своїй універсальності, в тому числі зручності моделювання та наочності візуалізації. Загалом, у вигляді графу можна представити довільну модель об’єктів реального світу та зв’язків між ними, що дозволяє застосовувати до нього будь-які відомі алгоритми та піддавати іншим видам подальшого аналізу.

Окрім використання теоретичних основ графів в інших галузях науки, теорія графів також знайшла широке застосування у вирішенні багатьох практичних завдань у великому спектрі життєдіяльності людини, як-то у хімії, соціології, логістиці, біології, економіці, техніці та ін. Так само і в розробці програмного забезпечення теорія графів посіла своє почесне місце. За її допомогою вирішуються задачі багатьох різних видів як у процесі розробки програм, так і для розвитку самої програмної інженерії.

У результаті проведеної роботи з аналізу можливостей практичного застосування теорії графів для розробки та обслуговування програмних систем було виявлено той факт, що на більшості етапів циклу розробки програмного забезпечення графи використовуються тільки для моделювання та візуалізації різних видів інформації, проте майже відсутній автоматизований алгоритмічний аналіз отриманих моделей. Це може свідчити про наявність перспективного напряму для пошуку додаткових можливостей застосування графів у процесах розробки програмного забезпечення.

Беручи до уваги те, що деякі задачі, пов'язані з теорією графів, усе ще не мають ефективних алгоритмів для їх вирішення, знаходження таких алгоритмів може спричинити великий стрибок як у розвитку самої теорії графів, так і значно розширити сфери її застосування.

ЛІТЕРАТУРА

1. K. Appel and W. Haken. Every Planar Map is Four Colorable. *Bull. Amer. Math. Soc.* 82, 1976. Pp. 711–712.
2. Narasingh Deo. Graph theory with applications to engineering and computer science, Prentice Hall of India, 1990.
3. Jonathan L. Gross, Jay Yellen. Graph Theory and Its Applications: 2nd edition, Chapman and Hall/CRC, 2005.
4. The C4 model for visualising software architecture. URL: https://c4model.com.
5. John P. Hayes. A graph Model for Fault Tolerant Computing Systems, IEEE, September 1976.
6. Perri Mehonen, Janne Riihijarvi, Marina Petrova. Automatic Channel allocation for small wireless area networks using graph coloring algorithm approach. IEEE, 2004.
7. Tom Leighton. Mathematics for Computer Science, MIT, 2010. URL: https://ocw.mit.edu/courses/6-042j-mathematics-for-computer-science-fall-2010/pages/readings/.
8. Nygren Erik, Sitaraman Ramesh K., Sun Jennifer. The Akamai Network: A Platform for High-Performance Internet Applications.
9. Understand the.NET Compiler Platform SDK model. URL: https://docs.microsoft.com/en-us/dotnet/csharp/roslyn-sdk/compiler-api-model.
10. Shariefuddin Pirzada and Ashay Dharwadker. Applications of graph theory. *Journal of the Korean Society for Industrial and applied Mathematics*, Volume 11, No. 4, 2007.

REFERENCES

1. K. Appel and W. Haken. Every Planar Map is Four Colorable, Bull. Amer. Math. Soc. 82, 1976. Pp. 711–712.
2. Narasingh, Deo. Graph theory with applications to engineering and computer science, Prentice Hall of India, 1990
3. Jonathan L. Gross, Jay Yellen. Graph Theory and Its Applications: 2nd edition, Chapman and Hall/CRC, 2005.
4. The C4 model for visualising software architecture. Retrieved from: https://c4model.com.
5. John P. Hayes. A graph Model for Fault Tolerant Computing Systems, IEEE, September 1976.
6. Perri Mehonen, Janne Riihijarvi, Marina Petrova. Automatic Channel allocation for small wireless area networks using graph coloring algorithm approach. IEEE, 2004.
7. Tom Leighton. Mathematics for Computer Science, MIT, 2010. Retrieved from: https://ocw.mit.edu/courses/6-042j-mathematics-for-computer-science-fall-2010/pages/readings/.
8. Nygren, Erik, Sitaraman, Ramesh K., Sun, Jennifer. The Akamai Network: A Platform for High-Performance Internet Applications.
9. Understand the.NET Compiler Platform SDK model. Retrieved from: https://docs.microsoft.com/en-us/dotnet/csharp/roslyn-sdk/compiler-api-model.
10. Shariefuddin Pirzada and Ashay Dharwadker. Applications of graph theory. Journal of the Korean Society for Industrial and applied Mathematics, Volume 11, No. 4, 2007.