Міністерство освіти і науки України
Національний університет «Києво-Могилянська академія»
Факультет інформатики
Кафедра математики

Кваліфікаційна робота

освітній ступінь – бакалавр

на тему: «**РОЗБИТТЯ ГРАФА НА ІЗОМОРФНІ ПІДГРАФИ НА СІТКАХ ПОЛІМІНО**»

Виконав студент 4-го року навчання,
Освітньої програми
«Прикладна математика», 113
Брагінець Дмитро Русланович
Керівник Дуденко М.А.
кандидат фіз.-мат. наук, ст. викладач
Рецензент _____
(прізвище та ініціали)

Кваліфікаційна робота захищена
з оцінкою _____
Секретар ЕК _____
«____» _____ 20___ р.

Київ – 2023

Графік підготовки кваліфікаційної роботи до захисту

№з/п	Перелік робіт	Термін	Підпис	Дата	Примітка
1	Отримання теми кваліфікаційної роботи	29.10.2022			
2	Ознайомлення з темою кваліфікаційної роботи	01.11.2022			
3	Розробка плану та структури роботи	08.11.2022			
4	Робота з науковою літературою, опис основних означень	14.12.2022			
5	Дослідження основних властивостей ізоморфних підграфів на сітках поліміно	09.02.2023			
6	Робота над текстовим оформленням результатів	17.03.2023			
7	Попередній аналіз роботи та виправлення помилок	13.04.2023			

ЗМІСТ

Сторінка

ВСТУП

Розбиття графа на ізоморфні підграфи на сітках поліміно – цікава та складна проблема, водночас наочна та прикладна. Особливо коли мова йде про розпізнавання розв'язку людиною як процес корисного дозвілля у форматі логічної гри для будь-якої вікової категорії.

Однак, варто зазначити, що хоча загальні означення і є оманливо простими для розуміння, пошук розв'язку, чи навіть його існування, не є тривіальною задачею. Чим більше різноманітних за формою та розміром сітки, типом побудови приєднання підграфів, тим більшим є задоволення від тієї природної закономірності в процесі пошуку розбиття для науковця чи гравця.

Більш того, сфера застосування ізоморфних підграфів відома в хімії (maximum common subgraph isomorphism [1]), в фізиці (line-graph-lattice crystal structures [2]), та дещо неочікуваних сферах комп'ютерних наук (criminal community detection based on isomorphic subgraph analytics [3]).

Розбиття графа на ізоморфні підграфи є фундаментальною проблемою теорії графів, яка має численні застосування в інформатиці, дослідженні операцій та інших галузях. Задача може бути сформульована наступним чином: «За заданим графом G визначити, чи можна його розбити на два або більше ізоморфних підграфів». У главі 2 ми обговоримо стандартні методи розбиття графа на ізоморфні підграфи, включаючи їх алгоритми, складність та обмеження.

Метою дослідження кваліфікаційної роботи є розробка алгоритму, який може ефективно розбивати заданий граф на ізоморфні підграфи на сітках поліміно. Наукова задача дослідження полягає у вивченні властивостей поліміно та їх відношення до ізоморфних підграфів, у

розробці математичного апарату для розбиття графів з використанням цих фігур.

Наукова новизна отриманих результатів полягає в розробці алгоритму розбиття графів на ізоморфні підграфи, який базується на фігурах поліміно та ідеях щодо його покращення. За допомогою отриманих результатів можна буде сформувати інтерактивний додаток – логічну гру, де упорядкованими є пазли відповідно до їх складності.

У даній роботі було проведено дослідження з розбиття графа на ізоморфні підграфи на сітках поліміно. Основною метою дослідження було розробка ефективного алгоритму для розбиття графа на ізоморфні підграфи з використанням поліміно.

У першому розділі було наведено необхідні означення та твердження, необхідні для розбиття графа на ізоморфні підграфи. Розглянуто базову задачу та її представлення у вигляді графу. Також було розглянуто поняття поліміно, які є основними фігурами для розбиття графів, а також їх узагальнення.

У другому розділі наведено ідеї алгоритмів розв’язання задачі розбиття графа на ізоморфні підграфи. Були розглянуті наївні емпіричні алгоритми, які можуть бути застосовані до простих випадків задачі. Також висвітлено ідеї щодо адаптації типових алгоритмів для розв’язання даної задачі, наведено приклад застосування комбінованого алгоритму, який поєднує різні підходи для досягнення кращих результатів.

У третьому розділі представлені практичні дослідження, що включають веб-застосунок "Polyomino Painter" та мобільну гру "Equalide". Ці розробки демонструють можливості застосування розбиття графа на ізоморфні підграфи в практичних ситуаціях. Також розглянуто зв'язок між теселяцією та розбиттям поліміно, що дозволило зрозуміти більш глибокі взаємозв'язки між цими поняттями.

РОЗДІЛ 1 НЕОБХІДНІ ОЗНАЧЕННЯ ТА ТВЕРДЖЕННЯ

1.1 Базова задача

Насамперед розглянемо задачу, з якою стикнувся ще у школі [4, с. 4]:

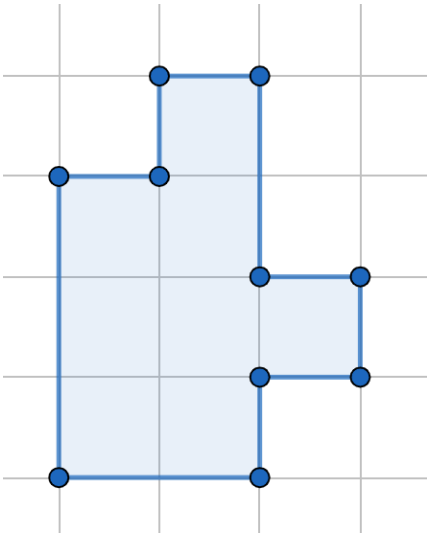


Рисунок 1.1 Базова задача

Задача: Провести ламану по лініям клітинок зошита так, щоб отримані дві фігури були рівними між собою.

Нам пояснили, що фігурою вважається набір цілісних квадратиків, власне таких самих, як і початково надана фігура. Розв’язавши на канікулах декілька простих задач, я стикнувся з неможливістю зрозуміти як розв’язувати більш складні пазли (надалі будемо вважати терміни «задача», «пазл», «головоломка» еквівалентними), а тоді просто почав проводити ламані за діагоналями, або навіть поза межами сітки.

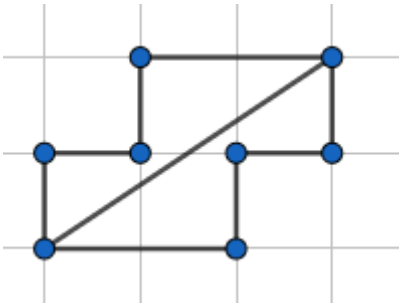


Рисунок 1.2 Приклад некоректного рішення - ламана як діагональ

Для початку варто розібратись що таке «однакові» фігури та до чого тут графи. Наведемо стандартні означення:

Граф G(V, E) складається з множини вершин V та множини ребер E, що з'єднують пари вершин.

Підграфом H(V', E') графа G(V, E) називається граф, де V' ⊆ V і E' ⊆ E.

Два графи G(V, E) та H(V', E') *ізоморфні*, якщо між їх множинами вершин існує бієкція (тобто функція "один-до-одного" та "на") така, що зберігає суміжність.

Якщо кожну клітинку (надалі поняття «клітинка» і «квадрат» вважаємо еквівалентними) базової задачі представити у вигляді вершини графа, а саме розташувати її в центрі квадрату, та з’єднати ребром тоді і тільки тоді, коли відповідні квадрати мають спільні ребра, то отримаємо репрезентацію у вигляді графа:

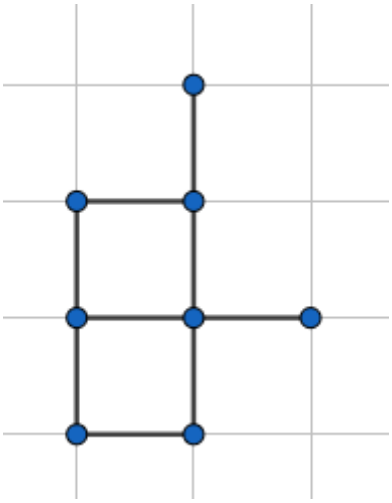


Рисунок 1.3 Базова задача у вигляді графа

Таким чином для розв’язку задачі потрібно знайти ребра, що необхідно вилучити так, щоб отриманий граф складався з двох компонент зв’язності – ізоморфних підграфів. Такий процес і будемо називати *розбиттям графу на ізоморфні підграфи*.

1.2 Поліміно та його узагальнення

У середині XX сторіччя році математик Соломон Голомб означив терміном "поліміно" геометричну фігуру, що складається з квадратів, де кожен квадрат має принаймні одну спільну грань з іншим квадратом фігури [5]. Поліміно може бути будь-якого розміру та форми, якщо воно задовольняє означення. Наприклад, пряма лінія з трьох квадратів є поліміно, так само як і L-подібна фігура з чотирьох квадратів.

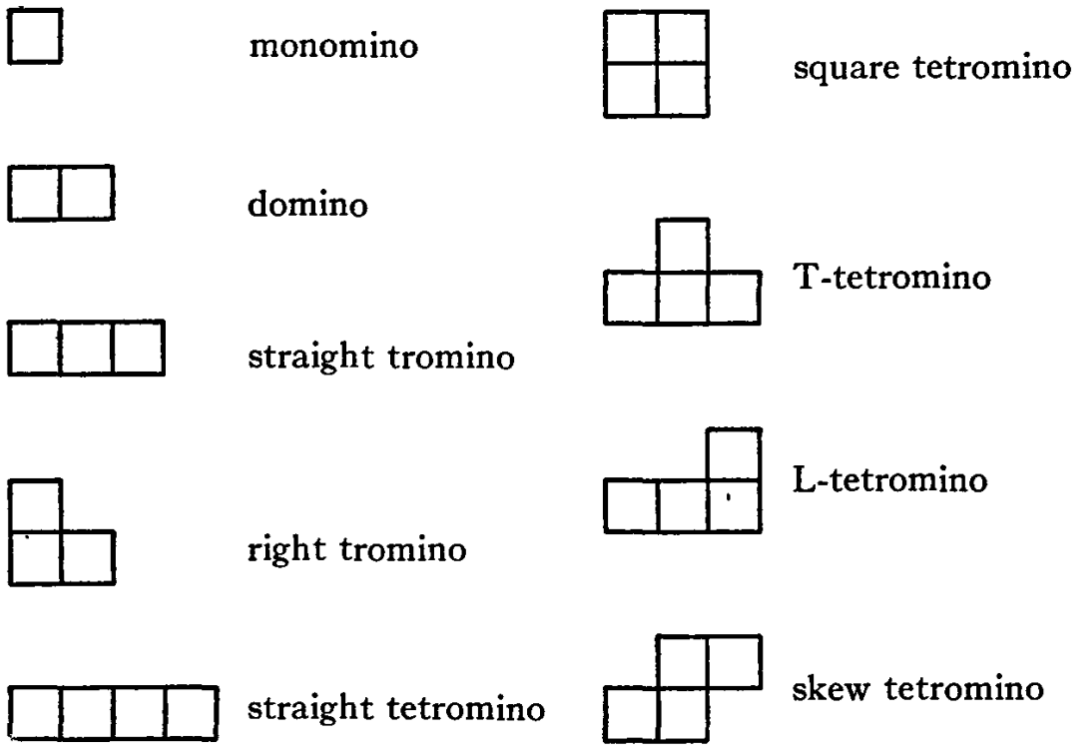


Рисунок 1.4 Приклади різноманітних поліміно

Саме такого штибу поліміно стали популярними в дозвіллі, рекреаційній математиці та олімпіадних шкільних задачах. Одним з відомих прикладів є Тетріс – відеогра, в якій гравець повинен складати поліміно, що падають, разом, щоб утворити повні ряди.

Існують також інші «поліміно – подібні», що не підпадають під строге визначення «мати одну спільну грань принаймні з іншим квадратом» – псевдо-поліміно (pseudo-polyomino) та квазі-поліміно (quasi-polyomino) [5, с. 680-681].

Псевдо-поліміно - це фігура, яка виглядає як поліміно, але не відповідає умові, щоб кожен квадрат мав хоча б одне спільне ребро з іншим квадратом. Замість цього, квадрати в псевдо-поліміно повинні мати спільний кут з іншим квадратом.

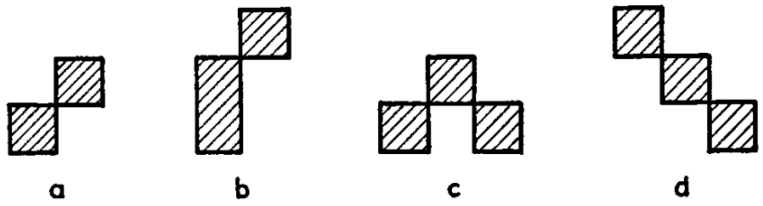


Рисунок 1.5 Псевдо-поліміно

З іншого боку, *квазі-поліміно*, задовольняє вимогу, щоб кожен квадрат мав хоча б одну спільну грань з іншим квадратом, але допускає "дірки" або "порожнечі" у фігурі. Це означає, що деякі квадрати у фігурі не торкаються інших квадратів фігури.

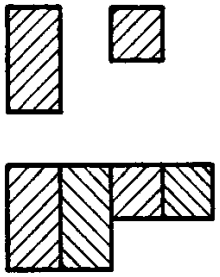


Рисунок 1.6 Квазі-поліміно

1.3 Поліформи та сітки поліміно

Оскільки площину можна заповнити не лише квадратами прямокутниками, а й правильними трикутниками або шестикутниками, з'явилась потреба в узагальнені поліміно.

Поліформа (polyform) – це геометрична фігура, що складається з однакових менших частин, так званих "поліміно", які з'єднуються разом, щоб утворити більшу фігуру.

Термін "поліформа" є узагальненням терміну "поліміно", який стосується саме фігур, що складаються з квадратів.

Поліформи можуть складатися з будь-якого типу фігур або частин, включаючи трикутники, шестикутники та інші правильні або неправильні багатокутники. Окремі частини, з яких складається поліформа, зазвичай називають "плитками", а процес їхнього з'єднання для формування більшої форми - "плиткоутворенням". Варто зазначити, що площину можна заповнити не лише рівносторонніми фігурами, як от зображено нижче:

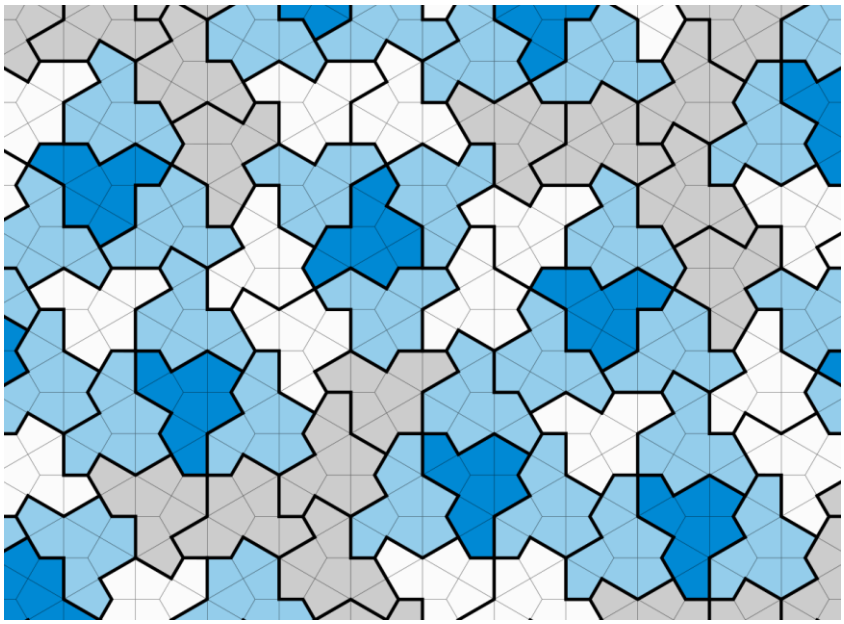


Рисунок 1.7 Плиткоутворення нерівносторонніми поліформами

Поліамант (polyiamond) – це різновид поліформи, у якій основною фігурою є рівносторонній трикутник.

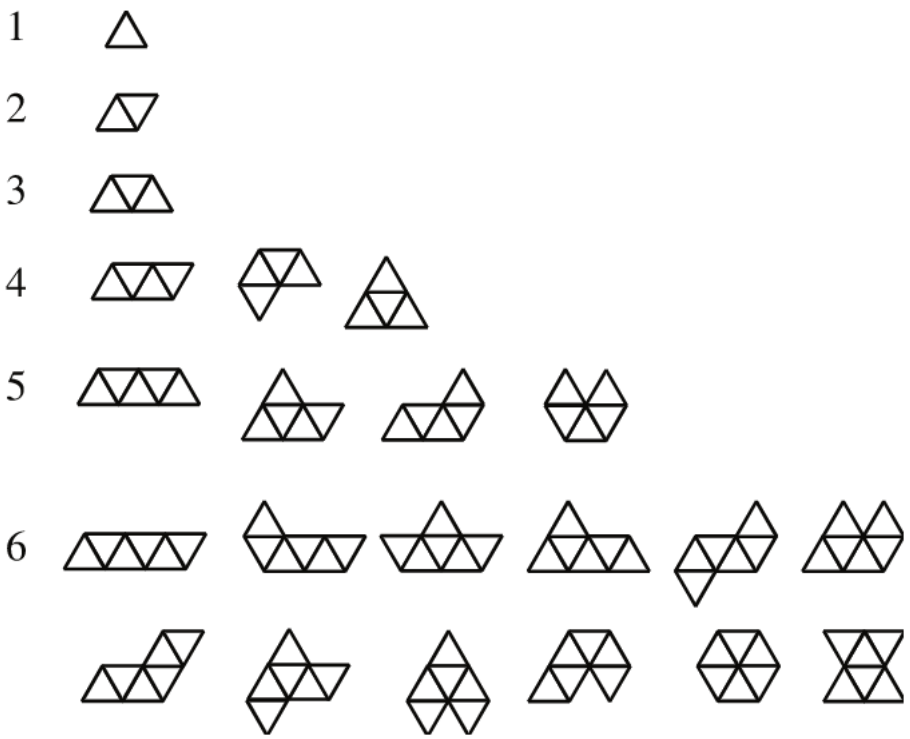


Рисунок 1.8 Приклади різноманітних поліамантів

Варто наголосити, що трикутники, які утворюють квадрати або при додаванні до яких отримуємо інші поліміно-подібні фігури - поліаболи (polyabolo), не є предметом дослідження.

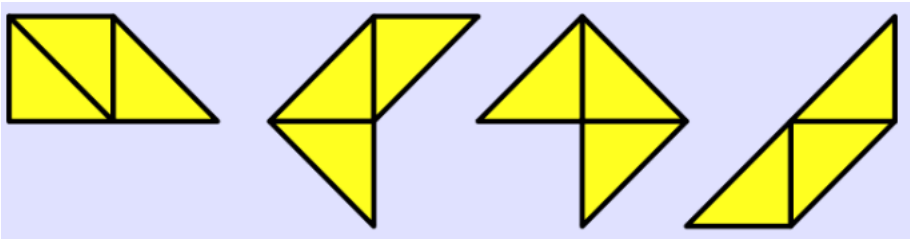


Рисунок 1.9 Приклади різноманітних поліабол

Полігекс - це різновид поліформи, утворений з правильних шестикутників.

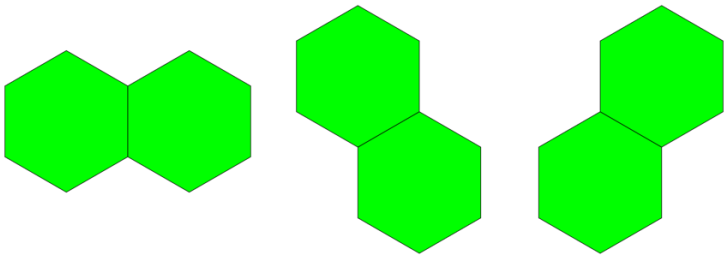


Рисунок 1.10 Приклад полігексів

За зовнішнім виглядом вони нагадують структуру бджолиних стільників (honeycomb).

Рисунок 1.11 Структура бджолиних стільників

Сіткою поліміно назвемо розбиття площини на рівносторонні фігури – поліміно, її ще називають теселяцією. Таких є лише три фігури, які ми щойно зазначили у попередніх розділах – квадрат, трикутник, шестикутник [6, с. 395]

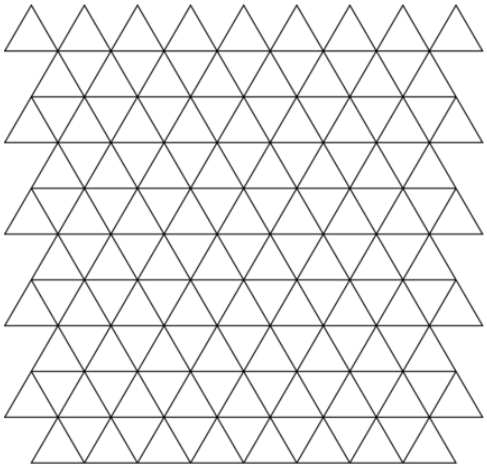


Рисунок 1.12 Трикутна сітка поліміно

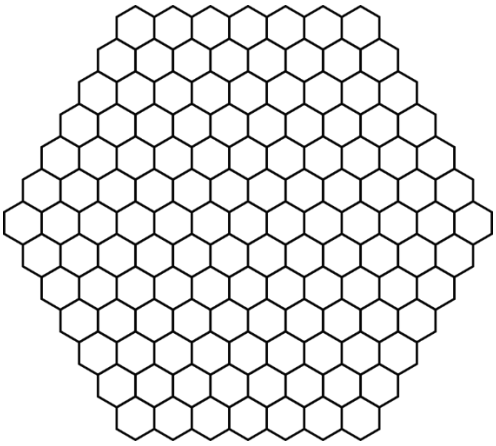


Рисунок 1.13 Шестикутна сітка поліміно

Отже, розбиттям графа на ізоморфні підграфи, що був утворений (породжений) поліміно на відповідній сітці, будемо називати *розбиттям на сітці поліміно.*

РОЗДІЛ 2 ТЕОРЕТИЧНІ АЛГОРИТМИ РОЗБИТТЯ

2.1 Наївні емпіричні алгоритми

Повертаючись до раніше згаданої задачі, необхідно порахувати кількість клітинок у фігурі – виявляється їх 8. Очевидний для задачі розв’язок досягається майже навмання, встановивши початок ламаної з будь-якої сторони фігури, можна відкреслити 4 клітинки та перевірити, чи співпадають їх форми:

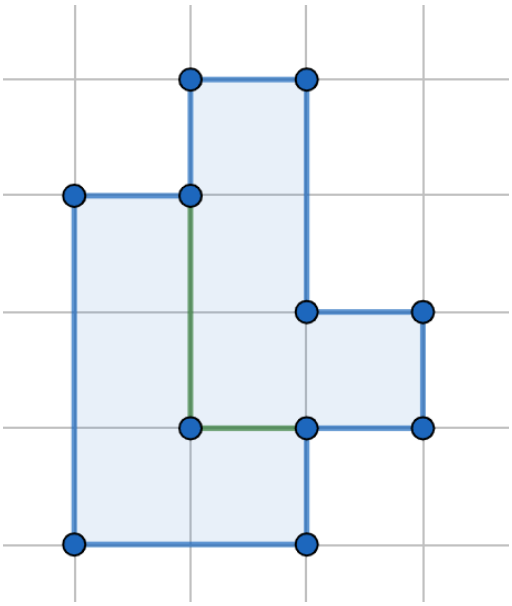


Рисунок 2.1 Розв'язок найпершої задачі

Але чи це єдиний розв’язок? На перший погляд здається, що так. Але більш уважний спостерігач побачить, що рішень задачі насправді два:

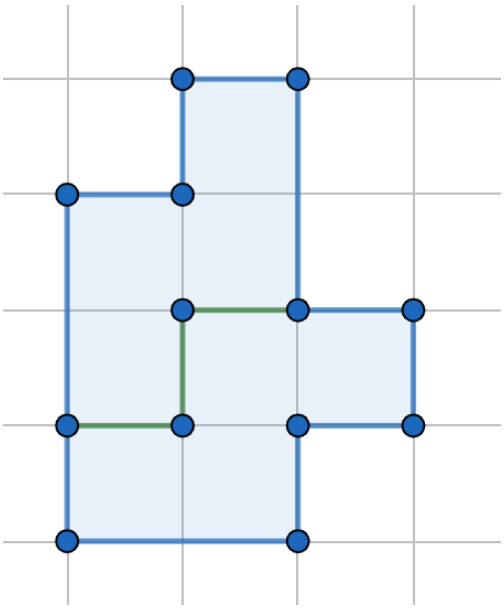


Рисунок 2.2 Ще один розв'язок найпершої задачі

Як довести існування інших розв’язків? Ще будучи школярем фізико-математичного ліцею, я сформував певні підходи, що спрощували пошук рішень такої задачі.

Кольором будемо називати клітинки з однакового розбиття фігури. Клітину будемо називати *воротами* до клітини, якщо вона однозначно має однаковий колір з відповідними клітинами.

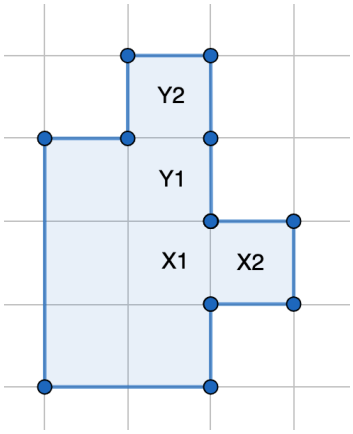


Рисунок 2.3 Аналітичне розфарбування

Якщо кольори X1 та Y1 співпадають, то розв’язок однозначний, якщо ні – клітина ліворуч від Y1 має ворота, що породжують розв’язок однозначно.

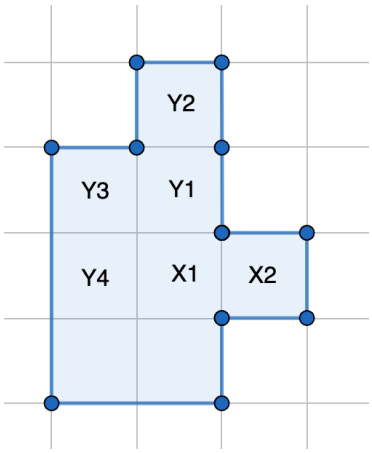


Рисунок 2.4 Розв’язок, породжений розфарбуванням

Для розв'язання цієї задачі можна використати рекурсивний алгоритм, який розбиває фігуру на менші підфігури до тих пір, поки ми не отримаємо два рівних поліміно. Алгоритм працює так:

Крок 1: Обчислити загальну кількість квадратів у фігурі.
Крок 2: Розділити фігуру на дві підфігури, які містять однакову кількість клітинок.
Крок 3: Перевірити, чи є ці дві підфігури поліміно. Якщо так, то ми знайшли розв'язок. Якщо ні, то переходимо до кроку 4.
Крок 4: Якщо дві підфігури не є поліміно, то нам потрібно повторити крок 3 для іншого набору клітин.
Крок 5: Повторюємо кроки з 2 по 4 до тих пір, поки не отримаємо два рівних поліміно.
Проте алгоритм може бути неефективним для великих фігур з великою кількістю квадратів, оскільки кількість згенерованих підфігур може бути дуже великою.

2.2 Ідеї щодо адаптації типових алгоритмів

Алгоритм грубої сили передбачає вичерпний перебір усіх можливих розбиттів заданого графа на індуковані підграфи. Для кожного розбиття алгоритм перевіряє, чи є підграфи ізоморфними. Метод грубої сили має експоненційну часову складність, що робить його непрактичним для великих графів.

Алгоритм зворотного пошуку є покращеним порівняно з методом грубої сили. Він будує дерево пошуку в першу чергу в глибину, де кожен вузол представляє частковий розділ. Щоразу, коли алгоритм досягає неізоморфного розділу або розділу з неприпустимим розміром, він повертається на попередній рівень і досліджує інші альтернативи. Алгоритм зворотного перебору ефективніший за метод грубої сили, але його найгірша часова складність все ще експоненційна [7].

Інший підхід до розбиття графа на ізоморфні підграфи полягає у використанні *канонічних позначень або інваріантів графа*.

Канонічне позначення - це унікальне представлення графа до ізоморфізму, тоді як інваріанти графа - це властивості, які зберігаються при ізоморфізмі (наприклад, степенева послідовність, спектр, циклічна структура). Обчислюючи канонічні позначення або інваріанти графа для заданого графа та його підграфів, ми можемо більш ефективно виявляти ізоморфні підграфи. Однак, знаходження канонічного позначення або обчислення інваріантів графа все ще може вимагати значних обчислювальних зусиль. [8]

Спектральні методи базуються на власних значеннях та власних векторах матриці суміжності графа або матриці Лапласа. Ізоморфні графи мають однаковий спектр (тобто однакові власні значення), який можна використовувати для розбиття графа на ізоморфні підграфи. Спектральні методи можуть бути обчислювально ефективними, але в деяких випадках їх може бути недостатньо для визначення ізоморфізму, оскільки неізоморфні графи також можуть мати такий самий спектр (коспектральні графи). [9]

Алгоритми розбиття графів, такі як алгоритми Кернігана-Ліна [10], Фідуччіа-Маттейса [11] та багаторівневі методи, можуть бути адаптовані для розбиття графа на ізоморфні підграфи шляхом накладання додаткових обмежень. Ці алгоритми спрямовані на мінімізацію цільової функції (наприклад, обрізання ребер, обсяг комунікацій), гарантуючи при цьому, що підграфи є ізоморфними. Хоча алгоритми розбиття графів можуть забезпечити хороші наближення для деяких випадків, вони не можуть гарантувати оптимальних рішень, і їх ефективність залежить від конкретної проблеми та вхідних характеристик.

Для певних класів графів, таких як дерева та графи з малою шириною, для розбиття графа на ізоморфні підграфи можна використовувати *декомпозиції дерев та методи динамічного програмування.* Деревоподібна декомпозиція - це відображення графа у деревоподібну структуру, де кожна вершина дерева асоціюється з набором вершин графа, який називається мішком [12].
Динамічне програмування може бути використане для розв'язання задачі розбиття декомпозиції дерева, використовуючи його рекурсивну структуру [13]. Ці методи можуть давати точні розв'язки для деяких класів

графів, але їх застосування обмежене графами з певними структурними властивостями.

Використання *машинного навчанн*я показало свою перспективність у вирішенні складних задач, таких як шахи чи го [14], мімікруючи поведінку людини. Існує кілька методів машинного інтелекту, які тут застосовні:

Штучні нейронні мережі - це форма машинного інтелекту, яка може розпізнавати закономірності в даних. У випадку з головоломками поліміно штучну нейронну мережу можна навчити розпізнавати патерни в поліміно, такі як симетричні фігури або групи квадратів зі схожими формами. Після навчання нейронну мережу можна використовувати для виявлення потенційних ізоморфних частин головоломки.

Генетичні алгоритми - це ще одна форма машинного інтелекту, яку можна використовувати для вирішення головоломок поліміно. У генетичному алгоритмі створюється популяція рішень, яка потім еволюціонує протягом декількох поколінь за допомогою таких процесів, як мутація і відбір. Найкращі рішення відбираються для наступного покоління і процес повторюється доти, доки не буде знайдено задовільного рішення. У випадку з головоломками поліміно генетичний алгоритм можна використовувати для створення потенційних ізоморфних частин головоломки, а потім еволюціонувати ці частини протягом декількох поколінь, поки не буде знайдено оптимальне рішення [15].

2.3 Приклад застосування комбінованого алгоритму

Спираючись на алгоритм зворотного відстеження, людина інколи використовує зворотнє відстеження з обмеженнями на поліміно, який враховує обмеження на зв'язність та форму поліміно під час процесу розбиття графа. Цей метод будує дерево пошуку в першу чергу в глибину, де кожен вузол представляє часткове розбиття. Щоразу, коли алгоритм досягає неізоморфного розбиття або розбиття, яке порушує обмеження поліміно, він повертається на попередній рівень і досліджує інші альтернативи. Такий підхід дозволяє більш ефективно обрізати простір пошуку, зменшуючи обчислювальну складність порівняно з алгоритмом зворотного відстеження за замовчуванням.

Ось приклад пазлу, яке ми хочемо розділити на рівні частини:

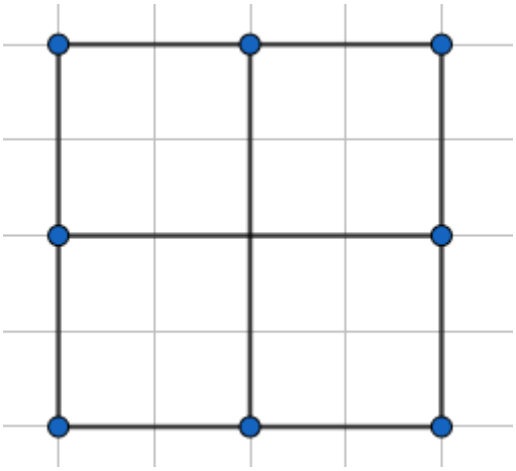


Рисунок 2.5 Базовий пазл з розбиття на 4 частини

Виберемо початкову точку для розділу і додамо перший квадрат до розділу:

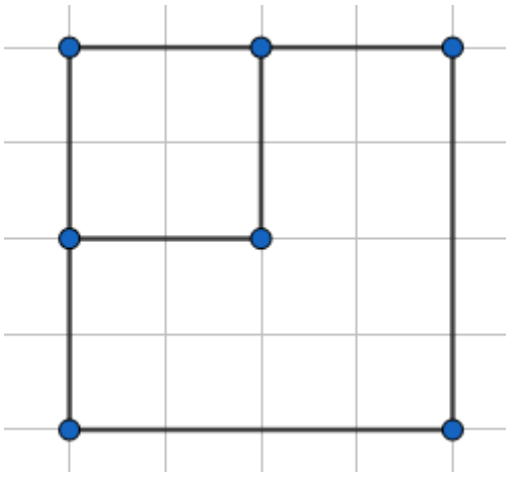


Рисунок 2.6 Обраний перший квадрат

Виберемо напрямок для додавання наступного квадрата. Спробуємо додати квадрат праворуч:

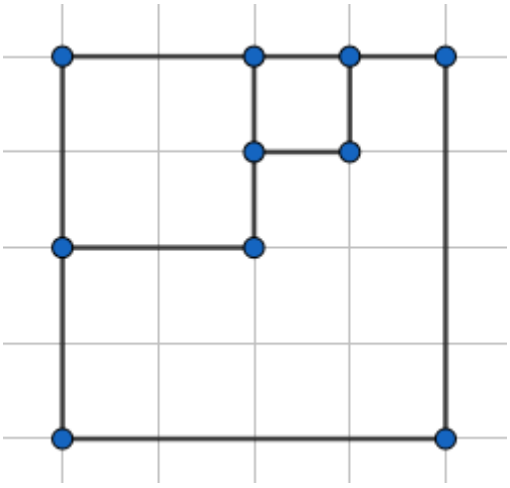


Рисунок 2.7 Заданий напрямок додавання наступного квадрата

Перевіримо, чи поточна частина ізоморфна початковому поліміно. У цьому випадку розбивка не є ізоморфною, оскільки містить лише два типи квадратів замість чотирьох. Нам потрібно відступити і спробувати інший напрямок.

Спробуємо почати додавати квадрат знизу:

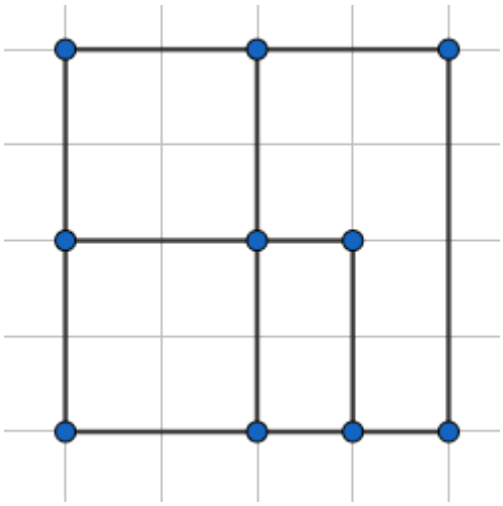


Рисунок 2.8 Спроба додавання квадрату знизу

Перевіряємо, чи поточна частина ізоморфна вихідному поліміно. У цьому випадку розбивка не є ізоморфною, оскільки містить лише три типи квадратів замість чотирьох. Нам потрібно відступити і спробувати інший напрямок. Спробуємо додати квадрат справа:

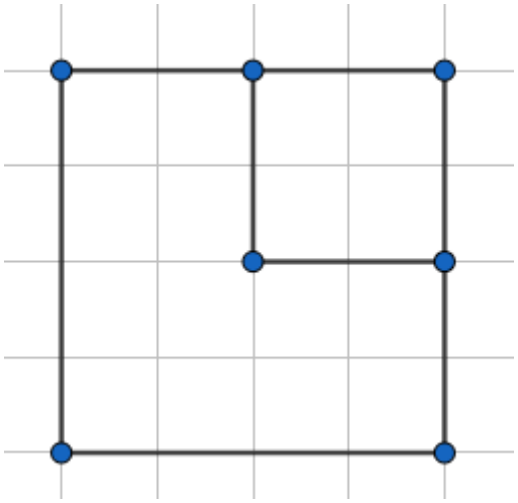


Рисунок 2.9 Зміна початкового напряму

Перевіяємо, чи поточна частина ізоморфна вихідному поліміно. У цьому випадку розбивка не є ізоморфною, оскільки містить лише два типи

квадратів замість чотирьох. Нам потрібно відступити і спробувати інший напрямок. Спробуємо замість цього додати квадрат:

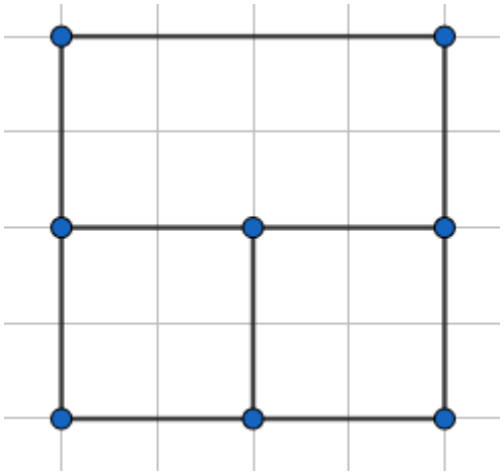


Рисунок 2.10 Додавання наступного квадрату за новим напрямом

Перевіряємо, чи поточна частина ізоморфна вихідному поліміно. У цьому випадку розбиття не є ізоморфним, оскільки містить лише три типи квадратів замість чотирьох. Нам потрібно відступити і спробувати інший напрямок. Спробуємо видалити квадрат ліворуч:

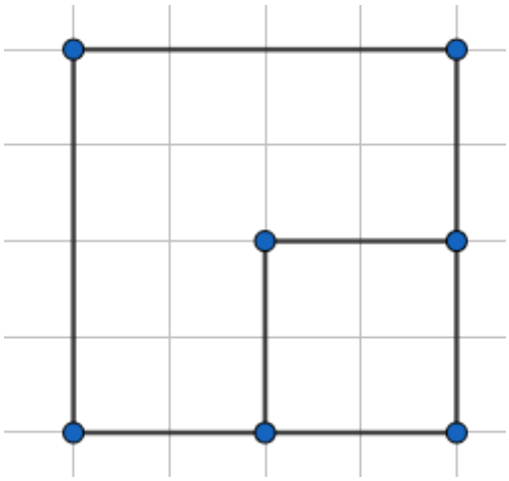


Рисунок 2.11 Залишковий квадрат при видаленні ліворуч

Перевіряємо, чи поточна частина ізоморфна оригінальному поліміно. У цьому випадку розбивка ізоморфна вихідному поліміно, оскільки містить чотири типи квадратів і розбита на рівні частини. Ми знайшли розв'язок!

Остаточне розбиття має вигляд рисунку 2.5

Алгоритм розфарбовування - ще один метод, який можна використовувати для розділення поліміно на рівні частини. Алгоритм полягає у розфарбовуванні клітинок поліміно різними кольорами так, щоб клітинки одного кольору належали до однієї частини.

Алгоритм працює так:

1. Обираємо початковий квадрат і розфарбовуємо у колір 1.
2. Розфарбовуємо сусідню клітинку у колір 2.
3. Повторюємо крок 2 для всіх клітинок з кольором 1, поки всі клітинки не будуть зафарбовані.
4. Перевіряємо, чи отримана розбивка ізоморфна початковому поліміно. Якщо так, то переходимо до наступного кроку. Якщо ні, то повертаємося назад і спробуємо зафарбувати початковий квадрат іншим кольором.
5. Якщо отримана розбивка має таку ж кількість клітинок, як і вихідне поліміно, і є ізоморфною йому, то розбивка завершена.
6. Якщо отримана розбивка має менше клітинок, ніж початкове поліміно, повторюємо кроки 1-4 з іншим кольором початкової клітинки.
7. Якщо отримана частина має більше клітинок, ніж початкове поліміно, повертаємося назад і спробуємо вибрати інший колір для початкової клітинки, щоб видалити клітинки з частини.

8. Повторюємо кроки 1-7 до тих пір, поки не будуть досліджені всі можливі розбиття або не буде знайдено розв'язок.

Алгоритм розфарбовування - це простий та інтуїтивно зрозумілий метод розбиття поліміно на рівні частини. Він особливо корисний для поліміно правильної форми і без отворів. Проте алгоритм може бути повільним і неефективним для поліміно неправильної форми або з великою кількістю отворів. [4, с.11]

Однією з переваг алгоритму розфарбовування є те, що він гарантує розв'язок, оскільки будуть розглянуті всі можливі варіанти розфарбовування. Хоча, для знаходження оптимального рішення може знадобитися перебрати велику кількість розмальовок, що може бути дорого коштувати з точки зору обчислень.

РОЗДІЛ 3 ПРАКТИЧНІ ДОСЛІДЖЕННЯ

3.1 Веб-застосунок Polyomino Painter

На початку дослідження для внутрішніх потреб було зроблено застосунок для моделювання поліміно, на який за потреби можна додатково імплементувати додатковий функціонал:

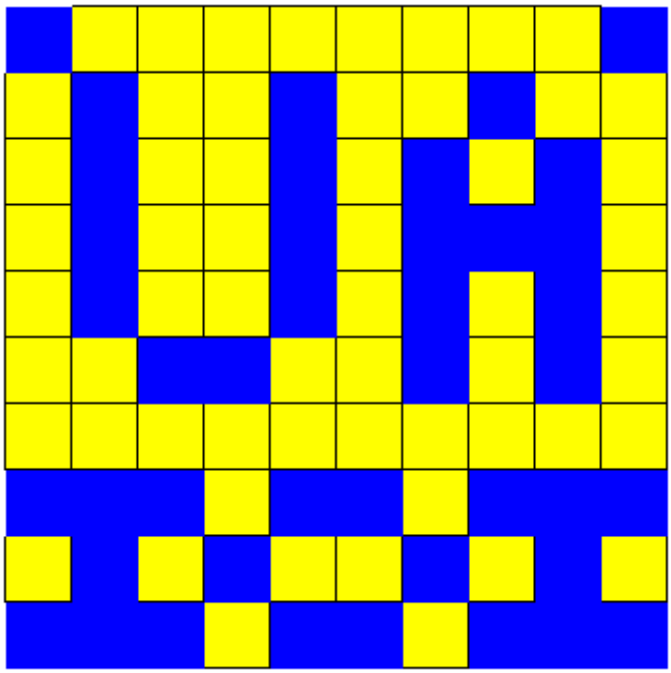


Рисунок 3.1 Приклад використання Polyomino Painer

Додаток складається з каталогу файлів:

index.html – головної сторінки програми

script.js – JavaScript файл, що генерує табличку

style.css – файл, що відповідає за тему, тобто стилі та кольори

logic.js – приклад нереалізованого функціоналу деяких операцій з поліміно

Програма сумісна з усіма сучасними браузерами, використовує технології, що не є залежними від будь-яких додаткових сторонніх бібліотек.

Нижче наведено лістинг коду:

index.html

```
<html>
  <head>
    <title>10x10 Polyomino Grid</title>
    <link rel="stylesheet" type="text/css"
href="style.css">
  </head>
  <body>
    <table class="grid"></table>
    <script src="script.js"></script>
  </body>
</html>
```

script.js

```
const rows = 10;
const cols = 10;
const grid = document.querySelector(".grid");

for (let i = 0; i < rows; i++) {
  let row = document.createElement("tr");
  for (let j = 0; j < cols; j++) {
    let cell = document.createElement("td");
    cell.className = "cell";
    cell.addEventListener("click", () => {
      // Toggle the background color of the cell
      if (cell.style.backgroundColor === "blue") {
        cell.style.backgroundColor = "yellow";
        cell.classList.remove("blue");
```

```
      } else {
        cell.style.backgroundColor = "blue";
        cell.classList.add("blue");
      }

      // Check if the blue cells form a complete shape
      const blueCells =
document.querySelectorAll(".cell.blue");
      if (blueCells.length > 0) {
        let minX = Infinity;
        let minY = Infinity;
        let maxX = -Infinity;
        let maxY = -Infinity;
        blueCells.forEach((blueCell) => {
          const x = blueCell.cellIndex;
          const y = blueCell.parentNode.rowIndex;
          minX = Math.min(minX, x);
          minY = Math.min(minY, y);
          maxX = Math.max(maxX, x);
          maxY = Math.max(maxY, y);
        });
        if (maxX - minX === blueCells.length - 1 && maxY -
minY === blueCells.length - 1) {
          console.log("Blue cells form a complete
shape.");
        }
      }
    });
```

```
    row.appendChild(cell);
  }
  grid.appendChild(row);
}
```

style.css

```
.cell {
    width: 30px;
    height: 30px;
    background-color: yellow;
    border: 1px solid black;
    cursor: pointer;
  }

  /* Add border-collapse to the grid */
  .grid {
    display: inline-block;
    border-collapse: collapse;
  }

  /* Hide the border of blue cells */
  .cell.blue {
    border-color: transparent;
  }
```

logic.js

```
export function getPolyomino() {
    const GRID = document.querySelector(".grid");
```

```
    const BLUE_CELLS =
GRID.querySelectorAll(".cell.blue");
    const WIDTH =
GRID.querySelectorAll("tr")[0].cells.length;
    const BLUE_CELLS_SET = new Set(BLUE_CELLS);
    const IS_POLYOMINO = BLUE_CELLS.length === WIDTH *
(BLUE_CELLS.length / WIDTH) && BLUE_CELLS_SET.size ===
BLUE_CELLS.length;
    return { cells: BLUE_CELLS, isPolyomino: IS_POLYOMINO
};
  }

  function partitionPolyomino() {
    const { cells, isPolyomino } = getPolyomino();
    if (isPolyomino) {
      const WIDTH = Math.sqrt(cells.length);
      const HEIGHT = WIDTH;
      const SET_A = new Set();
      const SET_B = new Set();
      for (let i = 0; i < WIDTH; i++) {
        for (let j = 0; j < HEIGHT; j++) {
          const INDEX = i * WIDTH + j;
          if (cells[INDEX].classList.contains("green") ||
cells[INDEX].classList.contains("yellow")) {
            SET_A.add(cells[INDEX]);
          } else {
            SET_B.add(cells[INDEX]);
          }
```

```
        }
      }
      cells.forEach((cell) => {
        cell.classList.remove("green", "yellow", "red");
        if (SET_A.has(cell)) {
          cell.classList.add("green");
        } else if (SET_B.has(cell)) {
          cell.classList.add("yellow");
        } else {
          cell.classList.add("red");
        }
      });
    } else {
      cells.forEach((cell) => {
        cell.classList.remove("green", "yellow", "red");
        cell.classList.add("blue");
      });
    }
  }

  document.querySelector(".partition-
button").addEventListener("click", partitionPolyomino);
```

3.2 Гра Equalide

За власною ідеєю та описаною у цій роботі матеріальною частиною розроблено мобільну пазл-гру головоломку Equalide [16].

Гру протестовано на деяких Android пристроях (смартфонах та планшетів) версій 7 – 12. Триває адаптація для сучасних девайсів що працюють на версіях 13 та 14 beta. Застосунок на момент публікації знаходиться у режимі раннього доступу за програмою Google Play Early Access. Оскільки частини коду є комерційною таємницею (відповідно до умов співпраці), публікую деякі скріншоти та опис розробки застосунку.

У розробці використано ігровий рушій Unity, оптимізована до жестів сенсорних екранів. При завантаженні користувач ознайомлюється з коротким туторіалом, далі гру можна проходити як рівень за рівнем, так і обираючи пазл серед наступних трьох нерозв’язаних. Заключним рівнем гри є бонусний, процедурно згенерований нескінченний рівень іншого типа, опис якого, з міркувань заохочення до проходження, публікувати не будемо.

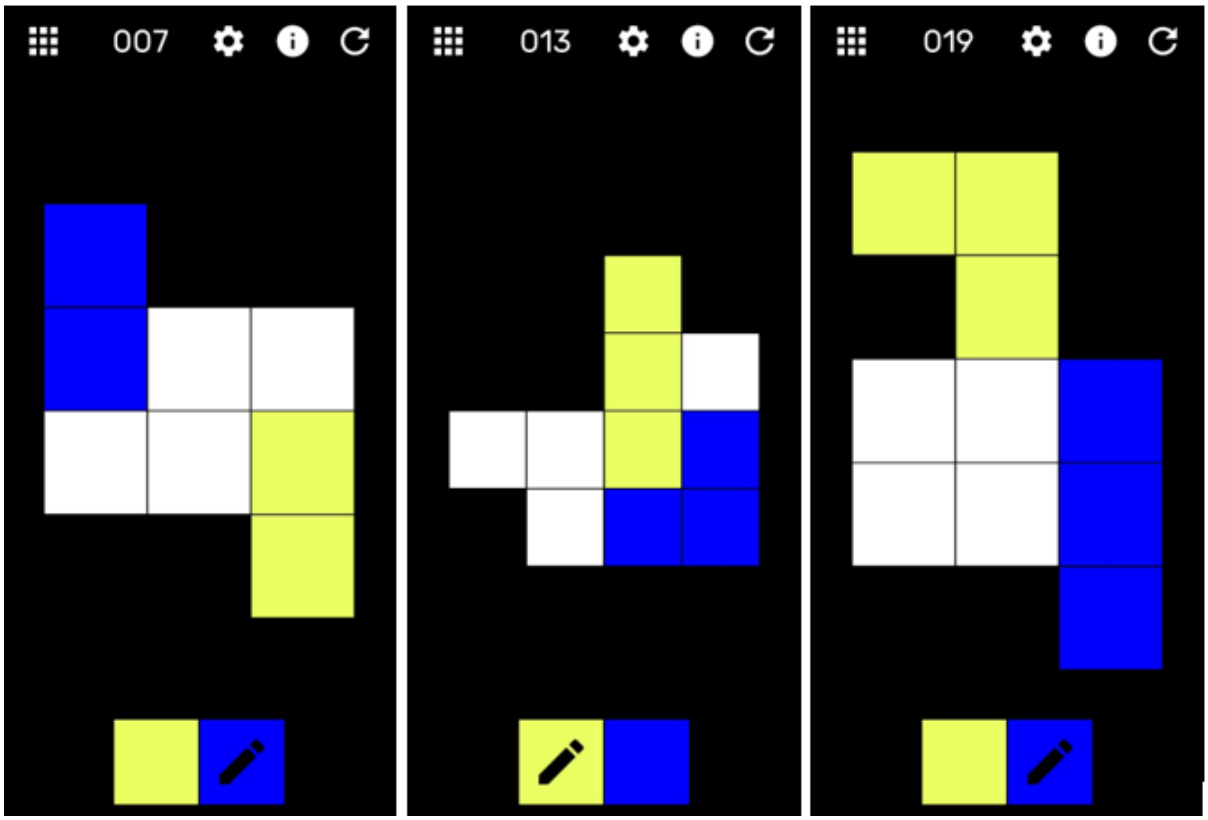


Рисунок 3.2 Скріншоти гри Equalide

На рисунку зображені деякі початкові рівні, загальна кількість переважає 200, а також додатковий бонусний нескінченний рівень після проходження всієї гри.

За допомогою цієї гри, на більш ніж 100 тестувальниками встановлений алгоритм розфарбування як оптимальний для рекреаційних користувачів.

Водночас з’ясовано шляхом опитування після проходження рівнів гри, що ключовим підходом до пошуку розв’язку виявились пошук схожої форми самого підграфа, що було приємною несподіванкою, тому що виходило за межі раніше зробленого теоретичного дослідження алгоритмів.

3.3 Зв’язок між теселяцією та розбиттям поліміно

Раніше було зазначено що не лише рівносторонні фігури можуть замащувати («плиткоутворювати») площину.
Слід зазначити, що у математиці проблема теселяції, тобто розбиття площини на однакові частини відома давно, проте саме практичне тестування Equalide виявило цікавий результат:

Якщо фігура розбивається на однакові поліміно, чи замащує вона площину? Якщо не замащує, то які фактори дозволяють бути поданою як розбиття на менші частини, проте не на всю площину до деякого обмеження? [17]

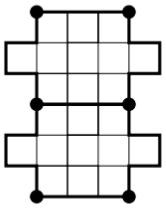
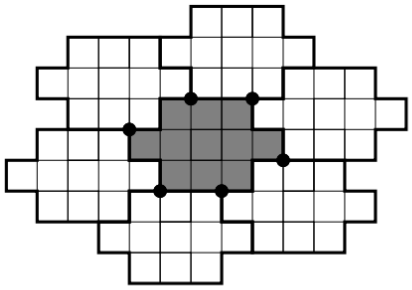
(a):  (b): 

Рисунок 3.3 Плитки (a) та розбиття площини (b) [18]

Вочевидь, при збільшенні кількості фігур розбиття постає питання, а чи схожий алгоритм пошуку на розбиття самої площини?
Отже, *теселяція площини* і *розбиття поліміно* - два тісно пов'язані поняття. При теселяції площина розбивається на конгруентні фігури (так звані плитки), які повністю покривають площину, не перекриваючись і не залишаючи проміжків. При розщепленні поліміно ділиться на ізоморфні складові частини.

Досліджуючи працю Srecko Brlek та Xavier Proven [18] я зрозумів певні закономірності:
один із способів розглянути зв'язок між плоскою тесселяцією та розбиттям на поліміно полягає в тому, що поліміно можна розглядати як окремий випадок плитки в плоскої тесселяції.

Зокрема, поліміно можна розглядати як плитку, яка складається зі скінченної кількості квадратів, розташованих у певному порядку. У цьому сенсі розщеплення поліміно можна розглядати як окремий випадок плоскої тесселяції, де плитки спеціально підібрані так, щоб бути ізоморфними частинами поліміно.

Факторизація (про Beauquier-Nivat Factorization [14]) розкладає граничне слово поліміно на три слова, кожне з яких має певну властивість. Нехай w ≡ ABCAbBbCb - факторизація граничного слова поліміно P. У цій факторизації A, B і C є допустимими множниками, тобто вони задовольняють певним умовам, включаючи насиченість і однакову довжину.

Факторизація дозволяє знаходити гомологічні фактори в поліміно, тобто пари факторів, які відповідають гомологічним сторонам. Наприклад, якщо у нас є поліміно P з граничним словом w, ми можемо обчислити найдовший спільний множник ww і wwd, щоб знайти гомологічні множники. Ці гомологічні множники відповідають гомологічним сторонам P, які можна використати для визначення способу розкладання поліміно.

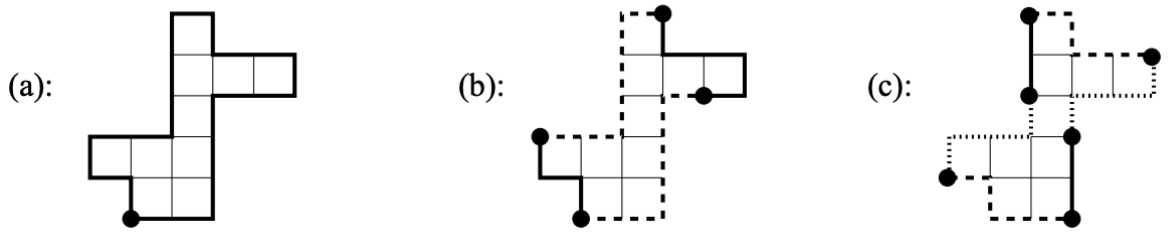


Рисунок 3.4 Поліміно (a) та його найдовший гомологічний фактор (b) та факторизація (c) [18]

Оскільки фактори A, B і C є допустимими, ми можемо бути впевнені, що вони відповідають дійсним гомологічним сторонам і можуть бути використані в процесі розбиття на плитки.

Однак застосування такого методу є досить непростою академічною задачею, що потребує більш глибок знань.

ВИСНОВКИ

1. Отримані результати свідчать про те, що розбиття графа на ізоморфні підграфи на сітках поліміно є складною задачею, але має широкі застосування в різних галузях. Розроблений алгоритм розбиття графів на ізоморфні підграфи, заснований на використанні поліміно, відкриває нові можливості для розв’язання цієї проблеми.

2. Дослідження показали зв'язок між розбиттям графа на ізоморфні підграфи та теселяцією площини. Поліміно можна розглядати як окремий випадок плитки в плоскої теселяції, де плитки є ізоморфними частинами поліміно. Це дає можливість застосування ідей теселяції площини при розбитті графів на ізоморфні підграфи.

3. Факторизація граничного слова поліміно дозволяє знаходити гомологічні фактори в поліміно, що відповідають гомологічним сторонам. Ці гомологічні фактори можуть бути використані для визначення способу розкладання поліміно на ізоморфні підграфи.

4. Подальші дослідження можуть розширити область застосування розбиття графа на ізоморфні підграфи та вдосконалити алгоритми розв’язання задачі.

5. Для удосконалення та поглиблення результатів доцільно провести більш скрупульозний аналіз властивостей поліміно та їх відношення до ізоморфних підграфів. Ці дослідження сприятимуть подальшому розвитку теорії графів та її застосуванню в різних галузях.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Duesbury E., Holliday J., Willett P. Comparison of Maximum Common Subgraph Isomorphism Algorithms for the Alignment of 2D Chemical Structures. ChemMedChem. 2017. Vol. 13, no. 6. P. 588–598. URL: https://doi.org/10.1002/cmdc.201700482
2. Line-graph-lattice crystal structures of stoichiometric materials / C. S. Chiu et al. Physical Review Research. 2022. Vol. 4, no. 2. URL: https://doi.org/10.1103/physrevresearch.4.023063
3. Sangkaran T., Abdullah A., Jhanjhi N. Criminal Community Detection Based on Isomorphic Subgraph Analytics. *Open Computer Science*. 2020. Vol. 10, no. 1. P. 164–174. URL: https://doi.org/10.1515/comp-2020-0112
4. Брагінець Д. «Проблема розбиття графа на ізоморфні підграфи», Курсова робота, Факультет Інформатики НаУКМА, Київ, 2022
5. Golomb S. W. Checker Boards and Polyominoes. *The American Mathematical Monthly*. 1954. Vol. 61, no. 10. P. 675. URL: https://doi.org/10.2307/2307321
6. Leon A. C. Mathematics: From the Birth of Numbers, Jan Gullberg, W. W. Norton and Company, New York, 1997. ISBN 0-393-04002-X. Statistics in Medicine. 1998. Vol. 17, no. 23. P. 2806–2807. URL: https://doi.org/10.1002/(sici)1097-0258(19981215)17:23%3C2806::aid-sim3%3E3.0.co;2-3
7. Kondrak G., van Beek P. A theoretical evaluation of selected backtracking algorithms. Artificial Intelligence. 1997. Vol. 89, no. 1-2. P. 365–387. URL: https://doi.org/10.1016/s0004-3702(96)00027-6

8. Babai L., Luks E. M. Canonical labeling of graphs. *the fifteenth annual ACM symposium*, Not Known. New York, New York, USA, 1983. URL: https://doi.org/10.1145/800061.808746
9. Coja-Oghlan A. Graph Partitioning via Adaptive Spectral Techniques. *Combinatorics, Probability and Computing*. 2009. Vol. 19, no. 2. P. 227–284. URL: https://doi.org/10.1017/s0963548309990514
10. Kernighan B. W., Lin S. An Efficient Heuristic Procedure for Partitioning Graphs. *Bell System Technical Journal*. 1970. Vol. 49, no. 2. P. 291–307. URL: https://doi.org/10.1002/j.1538-7305.1970.tb01770.x
11. Fiduccia C. M., Mattheyses R. M. A Linear-Time Heuristic for Improving Network Partitions. *19th Design Automation Conference*, Las Vegas, NV, USA, 14–16 June 1982. 1982. URL: https://doi.org/10.1109/dac.1982.1585498
12. Arnborg S., Proskurowski A. Linear time algorithms for NP-hard problems restricted to partial k-trees. Discrete Applied Mathematics. 1989. Vol. 23, no. 1. P. 11–24. URL: https://doi.org/10.1016/0166-218x(89)90031-0
13. Eddy S. R. What is dynamic programming?. Nature Biotechnology. 2004. Vol. 22, no. 7. P. 909–910. URL: https://doi.org/10.1038/nbt0704-909
14. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play / D. Silver et al. Science. 2018. Vol. 362, no. 6419. P. 1140–1144. URL: https://doi.org/10.1126/science.aar6404
15. Spears W. M., De Jong K. A. Using genetic algorithms for supervised concept learning. [1990] 2nd International IEEE Conference on Tools for Artificial Intelligence, Herndon, VA, USA. URL: https://doi.org/10.1109/tai.1990.130359

16. Braginets D. “Equalide” – Divide Equally!, Mobile Puzzle Game, Google Play Store. 2023 URL: https://play.google.com/store/apps/details?id=com.braingets.equalide
17. Beauquier D., Nivat M. On translating one polyomino to tile the plane. Discrete & Computational Geometry. 1991. Vol. 6, no. 4. P. 575–592. URL: https://doi.org/10.1007/bf02574705
18. Brlek S., Provençal X., Fédou J.-M. On the tiling by translation problem. Discrete Applied Mathematics. 2009. Vol. 157, no. 3. P. 464–475. URL: https://doi.org/10.1016/j.dam.2008.05.026