РЕФЕРАТ

Пояснювальна записка: ___ с., ___ рис., ___ табл., __ дод., ___ джерел.

Об'єкт розробки: навчальна інформаційна система з вирішення «проблеми чотирьох фарб» з курсу теорії графів.

Мета кваліфікаційної роботи: **с**творити інтерактивний навчальний додаток з вирішення «проблеми чотирьох фарб» з курсу теорії графів, у якому подано структуровані теоретичні та практичні блоки, програма, яка правильно розфарбовує довільну мапу на площині, а також особистий кабінет користувача, у якому він може побачити досягненні результати проходження тестів після кожного теоретичного блоку.

У вступі розглядається аналіз та сучасний стан проблеми, конкретизується мета кваліфікаційної роботи та галузь її застосування, наведено обґрунтування актуальності теми та уточнюється постановка завдання.

У першому розділі проведено аналіз предметної галузі, визначено актуальність завдання та призначення розробки, розроблена постановка завдання, задані вимоги до програмної реалізації, технологій та програмних засобів.

У другому розділі виконано аналіз існуючих рішень, обрано вибір платформи для розробки, виконано проектування і розробка програми, наведено опис алгоритму і структури функціонування програми, визначені вхідні і вихідні дані, наведені характеристики складу параметрів технічних засобів, описаний виклик та завантаження програми, описана робота програми.

В економічному розділі визначено трудомісткість розробленої інформаційної системи, проведений підрахунок вартості роботи по створенню програми та розраховано час на його створення.

Практичне значення роботи полягає в тому, що теорія графів є корисною в дуже різноманітних сферах людської діяльності: фізика, хімія, теорія зв’язку, проектування обчислювальних машин, електротехніка, машинобудування, архітектура, дослідження операцій, генетика, психологія, соціологія, економіка, антропологія, лінгвістика тощо, а даний проект призначений для вирішення «проблеми чотирьох фарб».

Актуальність теми кваліфікаційної роботи визначається великим попитом на подібні розробки, через те, що за допомогою даної роботи користувач може розфарбувати довільну мапу на площині правильно (під поняттям «правильно» розуміється розфарбування мапи у мінімальну кількість кольорів так, щоб будь-які дві області, що мають спільну ділянку межі, не були розфарбовані в один колір).

Список ключових слів: НАВЧАЛЬНАЯ ІНФОРМАЦІЙНА СИСТЕМА, ДИСКРЕТНА МАТЕМАТИКА, ГЕОРІЯ ГРАФІВ, ПРОБЛЕМА ЧОТИРЬОХ ФАРБ.

ABSTRACT

Explanatory note: ___ pp., ___ fig., ___ table, __ appendix, ___ sources.

Object of development: educational information system for solving the "problem of four colors" from the course of graph theory.

The purpose of the qualification work: to create an interactive educational application for solving the "problem of four colors" from the course of graph theory, which presents structured theoretical and practical blocks, a program that correctly paints an arbitrary map on the plane, and a user's personal account. achieving test results after each theoretical block.

The introduction considers the analysis and the current state of the problem, specifies the purpose of the qualification work and the field of its application, provides a justification for the relevance of the topic and clarifies the problem.

In the first section the analysis of the subject area is carried out, the urgency of the task and purpose of development is defined, the statement of the task is developed, requirements to software realization, technologies and software are set.

The second section analyzes the existing solutions, selects the platform for development, performs design and development of the program, describes the algorithm and structure of the program, determines the input and output data, provides characteristics of the parameters of hardware, describes the call and download program .

The economic section determines the complexity of the developed information system, calculates the cost of work to create a program and calculates the time for its creation.

The practical significance of the work is that graph theory is useful in many different areas of human activity: physics, chemistry, communication theory, computer design, electrical engineering, mechanical engineering, architecture, operations research, genetics, psychology, sociology, economics, anthropology , linguistics, etc., and this project is designed to solve the "problem of four colors".

The relevance of the topic of qualifying work is determined by the great demand for such developments, due to the fact that with this work the user can paint any map on the plane correctly (the term "correct" means painting the map in a minimum number of colors so that any two areas have a common area of the border, have not been painted in one color).

List of key words: EDUCATIONAL INFORMATION SYSTEM, DISCRETE MATHEMATICS, GEORGE OF GRAPHS, THE PROBLEM OF FOUR COLORS.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ІС – інформаційна система;

ОС – операційна система;

ПЗ – програмне забезпечення;

ПК – персональний комп'ютер.

ЗМІСТ

ВСТУП

Ще на початку XX століття все більш вчених були зацікавлені новітньою Теорією графів. З кожним роком кількість теорем та тверджень збільшувалась, але разом з тим збільшувалась кількість проблем, які не могли розв’язати роками. Яскравим прикладом такої проблеми є «Проблема чотирьох фарб».

Проблема була сформульована у 1852 році Ф. Гутрі, але знадобилося більше 120 років, щоб її довести. Тільки у 1976 році, коли ера комп’ютерних технологій ще тільки народжувалася, два математики К. Аппель та В. Хакен довели теорему, застосувавши вперше в історії математики комп’ютер. Вони довели за допомогою обчислювальної машини, що для 1936 різноманітних мап теорема справджується, а потім довели, що не існує контр-прикладу, для якого б не було достатньо чотирьох фарб. Але доказ був сприйнятий далеко не всією спільнотою математиків з-за неможливості перевірити алгоритм. Перевірка деяких тверджень сягала 1200 годин машинного часу! До того ж, їх праця містила 85 сторінок, 2500 діаграм, 400 мікрофільмів, 24 леми та ще тисячі окремих тверджень. Багато математиків брались перевірити алгоритм, але тільки у 2004 році знов за допомогою комп’ютера математики довели, що Розв’язок проблеми було правильним.

Останнім часом графи і пов'язані з ними методи досліджень органічно пронизують на різних рівнях чи не всю сучасну математику. Теорія графів входить як окремий розділ у дискретну математику, розглядається як одна з гілок сучасної топології та має безпосереднє відношення до алгебри і теорії чисел. Без теоретико-графових алгоритмів важко уявити сучасне програмування. У цьому контексті слід відмітити задачі, пов’язані з представленням програм у вигляді теоретико-графових моделей, найважливішою з яких є графова. Особливо важливе місце вони займають при моделюванні процесів та явищ [11].

Метою кваліфікаційної роботи є створити інтерактивний навчальний додаток з вирішення «проблеми чотирьох фарб» з курсу теорії графів, у якому

подано структуровані теоретичні та практичні блоки, програма, яка правильно розфарбовує довільну мапу на площині, а також особистий кабінет користувача, у якому він може побачити досягненні результати проходження тестів після кожного теоретичного блоку.

Були поставлені наступні задачі для виконання даної роботи:

1. Розробити алгоритм правильного розфарбування довільної мапи на площині.

2. Розробити структуровані теоретичні блоки курсу з теорії графів, створити словник з теорії графів.

3. Розробити практичні блоки, аби користувач мав можливість перевіряти рівень своїх знань після успішного опанування теоретичних блоків.

4. Розробити можливість реєстрації або авторизації користувача, можливість входу в його особистий кабінет.

Робота складається з трьох частин. Перша частина – теоретичний блок, у якому подано впорядкований теоретичний матеріал з предмету, а також завдання для самостійного розв’язання та їх розв’язки. Друга частина – практичний блок із особистим кабінетом користувача, де він може авторизуватись, проходити тести після вдалого опанування кожного теоретичного блоку. Третя частина - алгоритм, що розфарбовує довільну мапу на площині правильно у якомога меншу кількість кольорів.

Програма є актуальною, оскільки за допомогою неї користувач може розфарбувати довільну мапу на площині правильно (під поняттям «правильно» розуміється розфарбування мапи у мінімальну кількість кольорів так, щоб будь-які дві області, що мають спільну ділянку межі, не були розфарбовані в один колір).

Програма може бути застосована для вивчення курсу з теорії графів. Дана навчальна інформаційна система надає можливість наочно продемонструвати сутність рішення «проблеми чотирьох фарб» з курсу теорії графів та показати приклади її рішення.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1. Загальні відомості з предметної галузі

1.1.1. Коротка історія виникнення проблеми

У XVIII столітті кожен мешканець Кенігсбергу, мабуть, намагався вирішити інтуїтивно просте питання: «Чи можна пройти по всім мостам міста, не проходячи жодний двічі?» Багато мешканців намагалися це зробити різними шляхами, але нікому не вдавалося. Тоді ця проблема зацікавила видатного, на той час російського, математика Леонарда Ейлера. У 1736 році Леонард довів, що відповідь на питання негативна. Й хоча термін «граф» був вперше введений Джеймсом Сильвестром у 1878 році, але саме Ейлер поклав початок Теорії графів, вирішивши задачу про Кенігсберзькі мости та ще довівши низку тверджень, які зараз застосовуються у Теорії графів.

Ще на початку XX століття все більш вчених були зацікавлені новітньою Теорією графів. З кожним роком кількість теорем та тверджень збільшувалась, але разом з тим збільшувалась кількість проблем, які не могли розв’язати роками. Яскравим прикладом такої проблеми є «Проблема чотирьох фарб».

Проблема була сформульована у 1852 році Ф. Гутрі, але знадобилося більше 120 років, щоб її довести. Всілякі доведення сипалися, мов град з неба, але всі вони спростовувалися. Наприклад, А. Кемп у 1879 році написав свій доказ теореми, Пітер Тет у 1880 році, але обидва докази були спростовані вже через десять років. Деякі вчені навіть намагалися довести теорему тільки для якогось числа: наприклад, Ф. Франклін к 1913 році довів теорему для 25 країн, але це, звичайно, не доводило теорему у загальному випадку. Тільки у 1976 році, коли ера комп’ютерних технологій ще тільки народжувалася, два математики К. Аппель та В. Хакен довели теорему, застосувавши вперше в історії математики комп’ютер. Вони довели за допомогою обчислювальної машини, що для 1936 різноманітних мап теорема справджується, а потім

довели, що не існує контр-прикладу, для якого б не було достатньо чотирьох фарб. Але доказ був сприйнятий далеко не всією спільнотою математиків з-за неможливості перевірити алгоритм. Перевірка деяких тверджень сягала 1200 годин машинного часу! До того ж, їх праця містила 85 сторінок, 2500 діаграм, 400 мікрофільмів, 24 леми та ще тисячі окремих тверджень. Багато математиків брались перевірити алгоритм, але тільки у 2004 році знов за допомогою комп'ютера математики довели, що Розв'язок проблеми було правильним.

1.1.2. Розфарбування графів

Розфарбовуючи географічну карту оптимально було б користуватися по можливості меншою кількістю кольорів, проте так, щоб дві країни, що мають загальну частину кордону, були пофарбовані по-різному. У 1852 році Френсіс Гутрі, складаючи карту графств Англії, звернув увагу, що для такої мети цілком вистачає чотирьох фарб. Його брат, Фредерік, повідомив про це спостереження відомому математику О. Де Морган,а той – математичної громадськості. Точне формулювання гіпотези опубліковане А. Келі.

Перший доказ з'явився через рік і належав В. Кемпе. Одинадцять років потому П. Хівуд виявив в ньому помилку. Однак з доказу Хівуд зрозумів, що п'яти фарб дійсно достатньо. Проте для будь-якої конкретної карти вистачало чотирьох фарб! За першим помилковим доказом послідкувало безліч інших. В цьому відношенні проблема чотирьох фарб поступалася лише знаменитій проблемі Ферма. До середини XX століття, хоча проблемою чотирьох фарб займалися багато видатних математиків, положення з доказом змінилося несуттєво: ідеї Дж. Д. Биркгофа дозволили П. Франкліну в 1913 році довести гіпотезу для карти з не більше ніж 25 країнами. Пізніше це число було збільшено до 38.

У 1977 році доказ гіпотези чотирьох фарб було нарешті отримано К. Аппелем і У. Хакеном і опубліковано в двох статтях. Значну частину рутинних перевірок виконав комп'ютер, і це революційне нововведення в практику

дедуктивних міркувань у чистій математики служить підставою для деякого природного скептицизму по відношенню до даного доказу і донині.

Зазвичай, коли мова заходить про розфарбування графів, майже завжди мають на увазі фарбування їх вершин. Оскільки графи, в яких є петлі, неможливо правильно розфарбувати, то вони не є предметом обговорення. Термінологія, в якій мітки називаються кольорами (фарбами), походить від розфарбовування політичних карт. Розфарбування з використанням k кольорів називається k-розфарбуванням. Найменше число кольорів, необхідне для розфарбування вершин графа G називається його хроматичним числом і часто позначається X(G). Основною проблемою при відшуканні хроматичного числа графа є те, що компоненти розбиттів або декомпозицій в існуючих напрямках теорії графів мають бути зв'язними, в той час як компоненти оптимальних розбиттів, як правило, виявляються незв'язними. Це значно ускладнює застосування традиційних методів теорії графів для розв'язання задач на розбиття [18].

Вершинне розфарбування називається власним (правильним), якщо жодні дві суміжні вершини не пофарбовані в один колір. Підмножина вершин,виділених одним кольором, називається кольоровим класом, кожен такий клас формує незалежний набір. Таким чином, k-розфарбування – це те ж саме, що і розбиття вершин на k підмножин, що не перетинаються [5] (рис. 1.1).

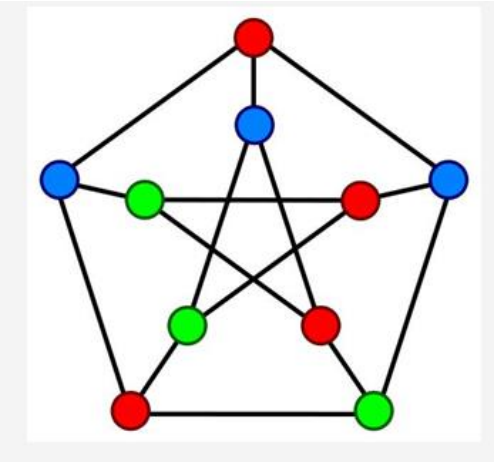


Рис. 1.1. Розфарбування графа Петерсена.

Аналогічно, власним (правильним) реберним розфарбуванням графа називається присвоювання ребрам графа певних «кольорів» таким чином, що ніякі два суміжних ребра не фарбуються в один і той же колір. Наприклад, на рисунку 2 подано реберне розфарбування графа в червоний, синій і зелений кольори. Реберно-хроматичним числом (або хроматичним індексом) графа називають найменше число кольорів, необхідне для правильного розфарбування ребер графа. При знаходженні правильних вершинних чи реберних розфарбувань графів виникає питання: чи можна правильно розфарбувати вершини (ребра) заданого графа максимум в k різних кольорів для заданого значення k або ж навпаки – знайти мінімальне можливе число кольорів [5].

Наприклад, ребра графа на рис. 12 можна правильно розфарбувати в три кольори, але не можна розфарбувати в два, отже, граф має хроматичний індекс три.

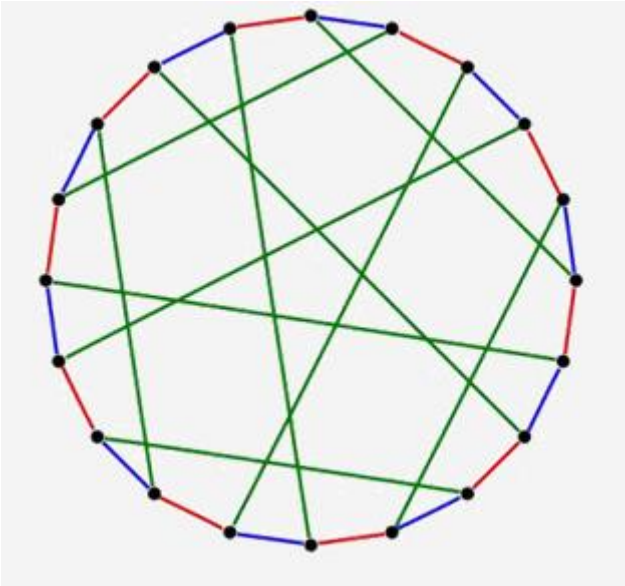


Рис. 1.2. Правильне реберне розфарбування кубічного графа

Окрім задачі відшукання найменшого числа кольорів, у які можна правильно розфарбувати вершини (ребра) графа, виникає задача пошуку усіх можливих правильних розфарбувань у задане число кольорів. Число можливих

варіантів правильного розфарбування вершин графа з використанням не більше ніж заданого числа кольорів розв’язується із використанням хроматичного многочлена [5]. Наприклад, граф на рис. 1.3 може бути розфарбований 12 способами з використанням 3 кольорів.

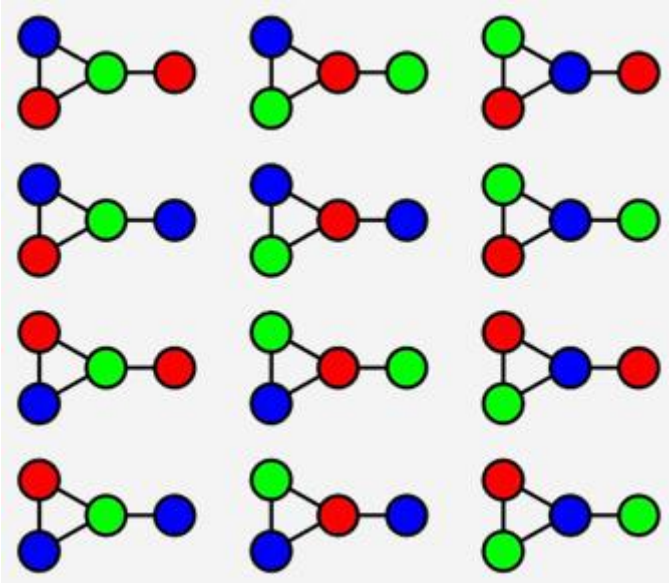


Рис. 1.3. Правильні розфарбування графа у 3 кольори

Історично поняття хроматичного числа виникло з проблемою чотирьох фарб. Проблема виникла в математиці в середині 19 століття. Спочатку питання формулювалося наступним чином: знайти найменше число кольорів для розфарбування географічної карти, якщо сусідні країни розфарбовані в різні кольори? Під географічною картою розуміють розбиття площини на скінченне число зв'язних областей (країн), межі яких складаються із замкнутих неперервних ліній без самоперетинів, а сусідніми вважаються країни, що мають спільну границю.

Досить очевидно, що трьох фарб недостатньо, тому питання формулювалося зазвичай у більш конкретному виді: чи достатньо чотирьох фарб для розфарбування будь-якої географічної карти? Це і є проблема чотирьох фарб [7] (рис. 1.4).

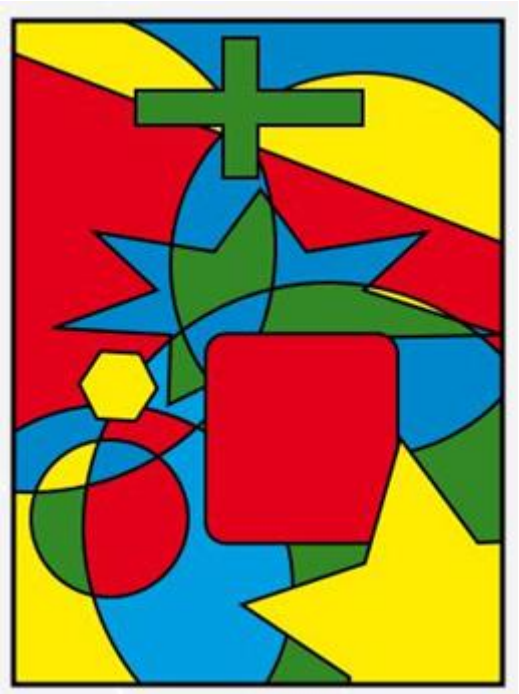


Рис. 1.4. Розфарбування плоского графа у чотири кольори

1.1.3. Задача розфарбовування карти

Задача розфарбування географічних карт зводиться до пошуку правильного розфарбування вершин плоских графів, що є двоїстими до заданої плоскої карти. Проілюструємо це зведення картою, зображеної на рис. 1.5.

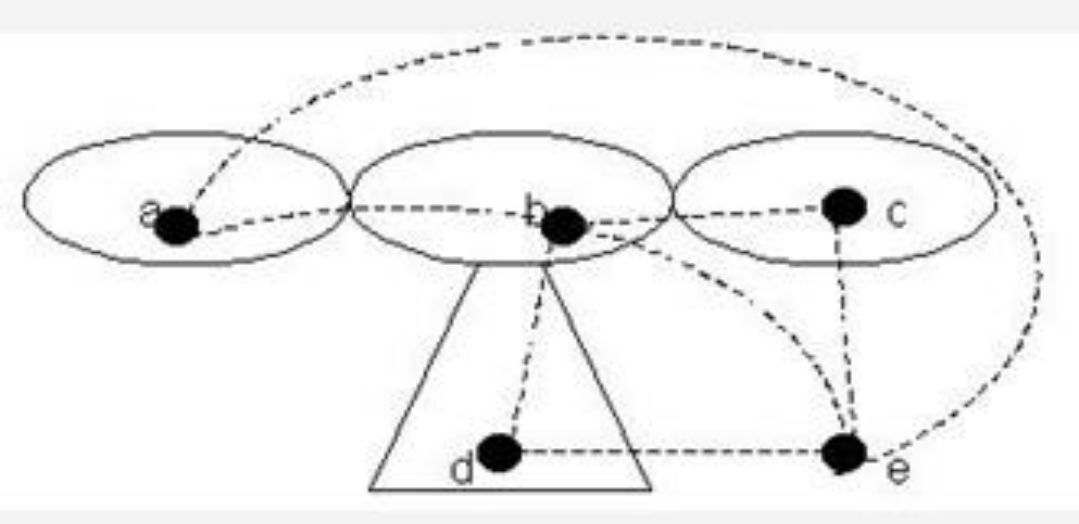


Рис. 5. Побудова двоїстого графа

На рис. 1. 5 зображено карту, що має п'ять країн (зовнішня область – теж країни). Усередині кожної області зафіксуємо точку (вершину двоїстого графа), та з’єднуємо точки дугами, якщо відповідні країни мають спільний кордон (на рис. 5 ребра проведено пунктирними лініями). При цьому ребра можна зобразити так, щоб вони не перетиналися, тобто щоб отриманий граф також був плоским. Зрозуміло, що розфарбування карти визначає правильне розфарбування вершин графа і навпаки. Тому проблему чотирьох фарб можна сформулювати наступним чином: чи достатньо чотирьох фарб для правильного розфарбування вершин плоского графа [7].

Ця проблема виникла вона при створені географічних карт. У 1852 році Френсіс Гетрі, складаючи карту графств Англії, звернув увагу, що для того, щоб розфарбувати карту так, щоб два графства, що мають спільну границю, були забарвлені по-різному, цілком вистачає чотирьох фарб. Його брат, Фредерік, повідомив про це спостереження відомого математика Августуа де Моргана, а той – математичну громадськість. Широку популярність проблема чотирьох фарб набула після того, як в 1878 році видатний математик Артур Келі повідомив, що він розмірковував над цією задачею, але так і не зумів її розв’язати [17].

Задача розфарбовування карти на площині еквівалентна аналогічній задачі на сфері. Для простих карт як правило, достатньо і трьох кольорів, четвертий колір потрібен, наприклад, тоді, коли є одна область, оточена непарним числом інших, які дотикаються одна одної, утворюючи цикл. Теорема про п'ять фарб, що стверджує, що для правильного розфарбування плоскої карти досить п'яти кольорів, мала коротке і нескладне доведення і була доведена наприкінці XIX століття. Проте обгрунтування аналогічного результату у випадку чотирьох кольорів наштовхнулося на значні труднощі.

Теорема про чотири фарби була доведена в 1976 році Кеннетом Аппелем Вольфгангом Хакеном з університету Ілінойса. Це була перша велика математична теорема, доведена за допомогою комп'ютера. Першим кроком у доведенні було обґрунтування факту, що існує певний набір з 1936 карт, жодна

з яких не може містити карту меншого розміру, яка спростовувала б теорему. К.Аппель і В. Хакен використали спеціальну комп'ютерну програму, щоб довести цю властивість для кожної з 1936 карт. Доведення цього факту зайняло сотні сторінок. Після цього К.Аппель і В.Хакен дійшли висновку, що не існує контрприкладу до теореми, тому що інакше він мав би містити якусь із цих 1936 карт. Це протиріччя говорить те, що взагалі не існує контрприкладу до відповідної теореми.

Спочатку нетрадиційне «комп'ютерне» доведення не було прийняте математичною спільнотою, оскільки його неможливо було перевірити вручну. Проте згодом воно отримало визнання, хоча у декого з прихильників традиційних доведень ще довгий час залишалися сумніви. Щоб розвіяти їх, у 1997 році Робертсон, Сандерс, Сеймур і Томас опублікували простіше доведення, в якому використовуються аналогічні ідеї, але як і раніше виконане за допомогою комп'ютера. Крім того, в 2005 році доведення було проведене Дж. Гонтиром з використанням спеціалізованого програмного забезпечення, проте комп'ютерна частина все ще залишалася швидше предметом віри. Адже навіть перевірка роздруківок усіх програм і усіх вхідних даних не може виключити випадкові збої або приховані вади електроніки [17].

1.1.4. Доведення проблеми чотирьох фарб

Доведення Аппеля і Хакена, в цілому хоча й прийнятий математичним співтовариством, викликає й понині певний скептицизм.

Доведення Аппеля і Хакена поділялося на два громіздкі етапи:

1. Показувалося, що достатньо перевірити 1936 варіантів мап, щоб визначити, що таке розфарбовування можливе для будь-якої мапи;
2. Наводиться алгоритм, що перевіряє усі мапи.

Говорячи прямо, комп'ютерну частину доказу неможливо перевірити вручну, а традиційна частина доведення довга і складна настільки, що її ніхто цілком і не перевіряв. Тим часом доведення, яке не піддається перевірці, є

нонсенс. Погодитися з подібним доказом означає приблизно те ж саме, що просто повірити авторам. Але і тут все складніше.

Повернемося спочатку до доказів формули Ейлера і теореми про 5-розфарбуванні. Її-то начебто неважко перевірити, взявши аркуш паперу і олівець. Але міркування в ній засновані на очевидних міркуваннях типу: “Плоский граф розрізає площину на сукупність багатокутних областей, що не перекривають одна іншу". Тим часом це твердження не належить до числа аксіом або шкільних теорем плоскої геометрії, і його потрібно доводити.

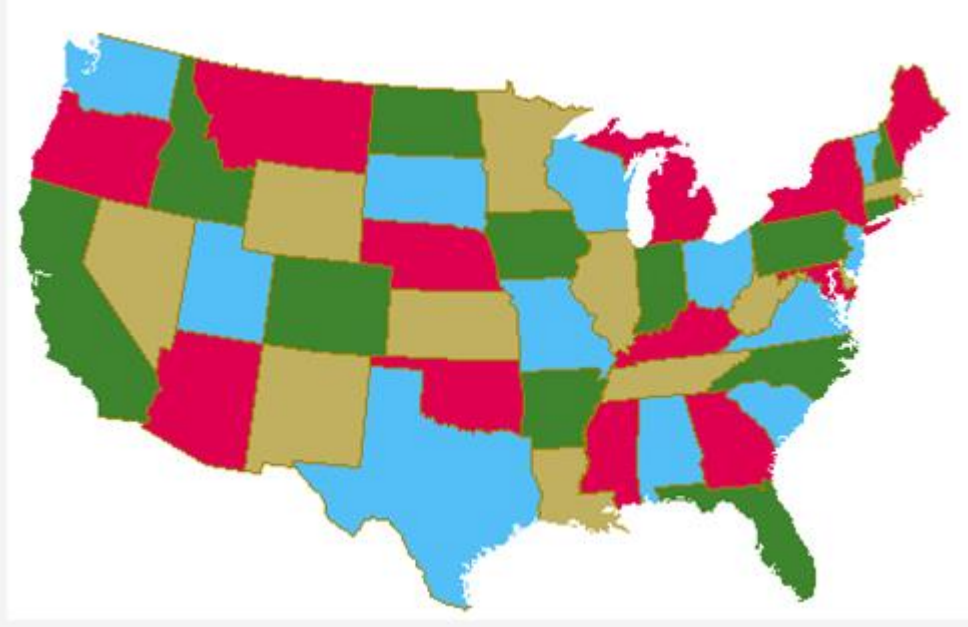


Рис. 1.4. Адміністративна карта штатів США, що розфарбовані у чотири кольори.

Відповідна теорема, сформульована К. Жорданом, доводиться дуже непросто, проте основні труднощі пов'язані з тим, як слід розуміти слова типу "розрізає", інтуїтивно цілком ясно, але насилу піддається формалізації. У світлі цього зауваження стає вже не зовсім зрозумілим, чи доведена теорема про п'ять фарб або ми повірили правдоподібним міркуванням, заснованим на інтуїтивних уявленнях про властивості геометричних фігур.

Довгий час ідеалом математичної строгості були формулювання і

доведення Евкліда, в яких здійснювалася програма виведення теорем з аксіом за певними правилами (метод дедукції, дозволяє отримувати неочевидні твердження з очевидних за допомогою ряду елементарних, очевидно законних, умовиводів). Цей зразок строгості і в наш час недосяжний в курсі шкільної математики, але для сучасної чистої математики стандарти Евкліда недостатні. Евклід, мабуть, не замислювався над тим, чому пряма ділить площину на дві частини (і що це означає), він не визначав поняття "між", вважаючи це поняття очевидним і т. д. Більша частина відповідних тверджень доведена або включена в аксіоматику геометрії тільки в останню сотню років. Формальні висновки з нової системи аксіом стали набагато довше, ніж вантичні часи.

Важко навіть уявити довжину повного виведення теореми про п'ять фарб відповідно до сучасних стандартах математичної логіки і системи аксіом геометрії. Але абсолютно точно, що таке міркування ніхто ніколи не робив через марність цього заняття: формальні висновки практично не піддаються перевірці в силу властивостей людської психології. Тому зазвичай задовольняються впевненістю в тому, що формальний висновок можливий в принципі.

У формулі Ейлера, наприклад, математики не мають сумніву. Взагалі прийняття доказу є деякий соціальний акт в широкому сенсі. "Не зрозумілі" докази можуть зіграти дуже корисну роль, стимулюючи пошуки більш доступних міркувань. Саме така історія відбувається і з доказом проблеми чотирьох фарб. Не так давно (2004 рік, Жорж Гонтьє) з'явився новий доказ причому та частина, яка виконана не на комп'ютері, вже піддається перевірці. Однак комп'ютерна частина все ще залишається скоріше предметом віри. Адже навіть перевірка роздруківок всіх програм і всіх вхідних даних не може гарантувати від випадкових збоїв або навіть від прихованих вад електроніки Мабуть, єдиний спосіб перевірки комп'ютерних результатів – написати іншу програму і для іншого типу комп'ютера. Це, звичайно, зовсім несхоже на стандартний ідеал дедуктивних міркувань, але саме так здійснюється перевірка тверджень у всіх експериментальних науках.

Таким чином, незважаючи на свою громіздкість, проблема чотирьох фарб є однією з найретельніше перевірених і доведених теорем, а також одним з найвідоміших прецедентів некласичного доведення в сучасній математиці.

1.2. Призначення розробки та галузь застосування

Останнім часом графи і пов'язані з ними методи досліджень органічно пронизують на різних рівнях чи не всю сучасну математику. Теорія графів входить як окремий розділ у дискретну математику, розглядається як одна з гілок сучасної топології та має безпосереднє відношення до алгебри і теорії чисел.

Без теоретико-графових алгоритмів важко уявити сучасне програмування. У цьому контексті слід відмітити задачі, пов’язані з представленням програм у вигляді теоретико-графових моделей, найважливішою з яких є графова. Особливо важливе місце вони займають при моделюванні процесів та явищ [11].

Окрім того, сферами застосування граф-моделей є використання обчислювальних ресурсів системи, організація великих масивів інформації, збільшення степеня паралелізму програми, підвищення ефективності роботи багатопроцесорних і багатомашинних систем. Розв’язання цих та інших задач привело до появи великої кількості граф-моделей, пов’язаних як з параметрами та структурами даних, так і з обчислювальними системами.

Окрім того, графи знайшли своє застосування в теорії планування та управління, теорії розкладів, соціології, математичній лінгвістиці, економіці, біології, медицині, географії. Широке застосування знаходять графи в таких областях, теорія скінченних автоматів, електроніка, у розв’язанні ймовірнісних і комбінаторних задач, знаходженні максимального потоку в мережах, найкоротшої відстані між вершинами графа, максимального паросполучення, перевірки планарності графа тощо.

Як особливий клас можна виділити задачі оптимізації на графах. Математичні розваги і головоломки також є частиною теорії графів. Зокрема, однією із важливих задач на графах є задача розбиття або розфарбування множини його вершин чи ребер [15], а також знаменита проблема чотирьох фарб [2].

Окрім того, сферами застосування граф-моделей є використання обчислювальних ресурсів системи, організація великих масивів інформації, збільшення степеня паралелізму програми, підвищення ефективності роботи багатопроцесорних і багатомашинних систем. Розв’язання цих та інших задач привело до появи великої кількості граф-моделей, пов’язаних як з параметрами та структурами даних, так і з обчислювальними системами.

Окрім того, графи знайшли своє застосування в теорії планування та управління, теорії розкладів, соціології, математичній лінгвістиці, економіці, біології, медицині, географії. Широке застосування знаходять графи в таких областях, теорія скінченних автоматів, електроніка, у розв’язанні ймовірнісних і комбінаторних задач, знаходженні максимального потоку в мережах, найкоротшої відстані між вершинами графа, максимального паросполучення, перевірки планарності графа тощо.

Дана програма є актуальною, оскільки за допомогою неї користувач може розфарбувати довільну мапу на площині правильно (під поняттям «правильно» розуміється розфарбування мапи у мінімальну кількість кольорів так, щоб будь-які дві області, що мають спільну ділянку межі, не були розфарбовані в один колір).

Програма може бути застосована для вивчення курсу з теорії графів. Дана навчальна інформаційна система надає можливість наочно продемонструвати сутність рішення «проблеми чотирьох фарб» з курсу теорії графів та показати приклади її рішення.

1.3. Підстава для розробки

Підставою для розробки кваліфікаційної роботи бакалавра на тему «Розробка інтерактивного навчального додатку для вирішення «Проблеми чотирьох фарб» з курсу теорії графів засобами мови C#» є наказ по Національному технічному університету «Дніпровська політехніка» від __.__. 2021р. № ____-_.

1.4. Постановка завдання

Метою кваліфікаційної роботи є створення інтерактивного навчального додатку для вирішення «проблеми чотирьох фарб» з курсу теорії графів, у якому подано структуровані теоретичні та практичні блоки, програма, яка правильно розфарбовує довільну мапу на площині. а також особистий кабінет користувача, у якому він може побачити досягненні результати проходження тестів після кожного теоретичного блоку.

Для досягнення даної мети поставлені наступні задачі:

1. Розробити алгоритм правильного розфарбування довільної мапи на площині.

2. Розробити структуровані теоретичні блоки курсу з теорії графів, створити словник з теорії графів.

3. Розробити практичні блоки, аби користувач мав можливість перевіряти рівень своїх знань після успішного опанування теоретичних блоків.

4. Розробити можливість реєстрації або авторизації користувача, можливість входу в його особистий кабінет.

Головним завданням роботи є створення програми, яка правильно розфарбовує довільну мапу на площині.

Для виконання даного завдання необхідно виконати наступні етапи:

1. Ознайомитися з теоретичним матеріалом «Проблеми чотирьох фарб» з курсу теорії графів.

2. Структурувати матеріал з даної теми.

3. Розробити алгоритм правильного розфарбування довільної мапи на площині.

4. Розробити структуру інформаційної системи.

5. Розробити алгоритм роботи програми.

6. Визначити мову програмування і середовище розробки.

7. Створити проект для рішення задачі правильного розфарбування довільної мапи на площині..

8. Розробити програму для використання на ПК із зручним інтерфейсом.

Розроблена навчальна інформаційна система має бути протестована на конкретних прикладах та з різноманітними даними, що засвідчуватиме про її працеспроможність.

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

Головним завданням роботи є створення програми, яка правильно розфарбовує довільну мапу на площині, створення інтерактивного курсу з теорії графів, у якому подано структуровані теоретичні та практичні блоки, глосарій, а також особистий кабінет користувача, у якому він може побачити досягнені результати проходження тестів після кожного теоретичного блоку.

Робота повинна складатися з трьох частин:

1. Теоретичний блок, у якому повинно бути подано впорядкований теоретичний матеріал з предмету та завдання для самостійного розв’язання та їх розв’язки.

2. Практичний блок із особистим кабінетом користувача, де він зможе авторизуватись, проходити тести після вдалого опанування кожного теоретичного блоку.

3. Програма, що реалізує алгоритм, який розфарбовує довільну мапу на площині правильно у якомога меншу кількість кольорів. Під поняттям

«правильно» розуміється розфарбування мапи у мінімальну кількість кольорів так, щоб будь-які дві області, що мають спільну ділянку межі, не були розфарбовані в один колір.

1.5.2. Вимоги до інформаційної безпеки

Для надійної роботи програмного забезпечення зі сторони операційної системи необхідно дотримуватися таких факторів:

- використання ліцензійного ПЗ;
- захист від зловмисних програм;
- архівація даних на сервері;
- встановлення блоків безперебійного живлення.

1.5.3. Вимоги до складу та параметрів технічних засобів

Для розробки даної інформаційної системи необхідно ПК з наступними характеристиками:

- тип процесора: процесор з частотою 2.2 ГГц, або більш потужний;
- ОЗУ об'ємом 4 Гб;
- 30 Мб доступного простору на жорсткому диску;
- жорсткий диск з частотою обертання 5400 об / хв.;
- дозвіл екрану не менше 1024x768;
- клавіатура;
- маніпулятор «миша».

Програмні вимоги:

- .Net Framework 4.6.1.

Підтримуючі операційні системи:

- Windows 10.
- Windows 8, Windows 8.1.
- Windows 7 with SP1.

1.5.4. Вимоги до інформаційної та програмної сумісності

Дана інформаційна система повинна бути розроблена з використанням наступних програмно-апаратних засобів:

- операційна система Microsoft Windows 77;
- за допомогою середовища програмування .NET C# (WPF + Windows Forms).

Програма повинна являти собою самостійний виконуваний модуль, бути структурована і закоментована

РОЗДІЛ 2
ПРОЄКТУВАННЯ ТА РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1. Функціональне призначення системи

Створений інтерактивний навчальний додаток призначений для вирішення «проблеми чотирьох фарб» з курсу теорії графів. В ньому у зручній формі подано структуровані теоретичні матеріали з даної теми, а практичні блоки надають можливість на прикладах застосувати отримані відомості. Розроблена програма, яка правильно розфарбовує довільну мапу на площині.

Наявність особистого кабінету користувача надає можливість відстежувати досягненні результати проходження тестів після кожного теоретичного блоку.

2.2. Опис застосованих математичних методів

2.2.1. Теорема Хівуда

Проблема чотирьох фарб, яка на перший погляд здавалася ізольованим завданням, навіть грою, мало пов'язаною з іншими розділами математики та практичними застосуваннями, знаходить у наш час найнесподіваніші застосування. Відомо понад двадцять формулювань цієї проблеми, які пов'язують її з конкретними задачами алгебри, статистичної механіки і задачами планування. І це ще раз підкреслює всюдисущий характер математики: розв’язок задачі, що вивчається з чистої цікавості, виявляється корисним в описі реальних і абсолютно різних за своєю природою об'єктів і процесів [17].

Наведемо точні формулювання й доведемо теорему Хівуда (теорема о п’яти фарбах).

Формулювання питання:

Означення 1. Граф зветься планарним, якщо можна так накреслити його на площині, щоб не було перетину ребер на рисунку не по вершинам. Приклад: повний граф K_4 є планарним. Доказом цього є його зображення, у якому немає перетину ребер не по вершинам (рис. 2.1).

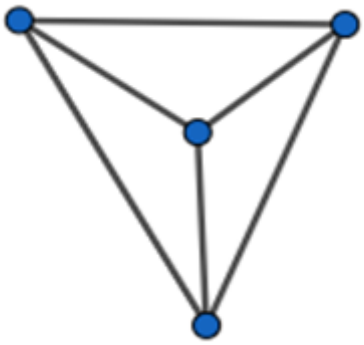
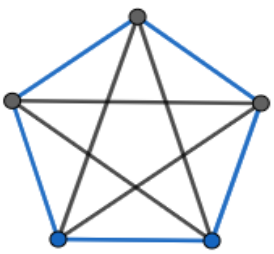
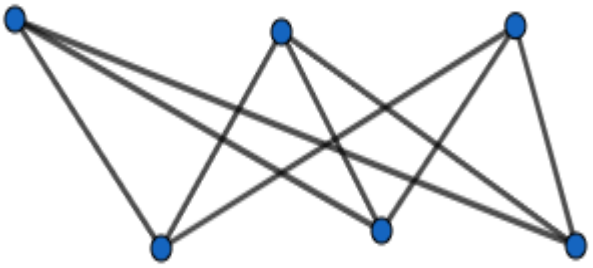


Рис. 2.1. Повний граф на 4 вершинах є планарним

В табл. 2.1 наведені приклади непланарних графів

Таблиця 2.1

Приклади непланарних графів

Повний граф на п’яти вершинах(K_5)	Повний двочастковий граф на 6 вершинах($K_{3,3}$)
	

Для того, щоб вивести критерій планарності графа, треба вивести спочатку поняття гомеоморфізму.

Означення 2.Два графи G_1,G_2гомеоморфні, якщо G_2 можна отримати шляхом додавання або видалення якихось вершин на ребрах графу G_1.

Критерій Куратовського. Граф є планарним тоді й тільки тоді, коли в ньому нема подграфів, гомеоморфних K_5 або $K_{3,3}$.

Означення 3. Назвемо планарним граф плоским, якщо він зображен на площині без перетину ребер не по вершинам

Нехай - число вершин у плоскому графі, e - число ребер у ньому. Введемо ще одну характеристику для плоских графів. Нехай f - число граней у плоскому графі. Грань - це область, що обмежена ребрами у плоскому графі, і яка не містить у собі вершин та ребер графа. Зовнішня частина площини також утворює грань.

Теорема Хівуда: можемо сформулювати проблему чотирьох фарб у вигляді наступного твердження:

Теорема 1. Будь-який плоский граф допускає правильне 4-розфарбування. Ось це питання і не могли вирішити дев’яносто вісім років. Однак, на перший погляд більш слабке твердження можна довести достатньо просто, але для цього спочатку доведемо формулу Ейлера та наслідок з неї.

Будь-який плоский граф допускає правильне 5-розфарбування.

Нехай $G := (V, E)$ - даний граф, у якого v вершин та e ребер. Спочатку спростимо граф. Якщо є кілька ребер, що з'єднують деяку пару вершин (така ситуація може виникнути, якщо дві країни мають кілька непов'язаних між собою шматків кордону) то залишимо тільки одне ребро: правильність розмальовки такого зменшеного графа все одно гарантує правильну розмальовку вихідного.

2.2.2. Теорема Ейлера

Нехай дано плоский зв'язний граф. Тоді $v - e + f = 2$. Доведемо теорему методом математичної індукції.

Оскільки за умовою нам дано зв'язний граф, то маємо, що $e \geq v - 1$ (Зв'язний граф - граф, у якого між будь-якою парою вершин існує як мінімум один ланцюг. Між будь-якими двома вершинами дерева є рівно один простий ланцюг, отже $e \geq v - 1$).

Розглянемо випадок рівності даного виразу.

Випадок $e = v - 1$ можливий, тоді й тільки тоді, коли граф є деревом. Це було доведено у 4-х еквівалентних визначеннях дерева. Тоді у випадку графа-дерева маємо, що $f = 1$, оскільки дерево має тільки зовнішню частину площини.

Отже, $v - e + f = v - (v - 1) + 1 = 2$. Для випадку $e = v - 1$ Теорема Ейлера доведена.

База індукції: $e = v - 1$, доведена вище.

Припустимо, що для всіх $e \leq k$ (при чому $k \geq v - 1$) твердження доведено.

Розглянемо довільні числа $v, f, e = k + 1$ й будь-який граф з даними параметрами (рис.2.2).

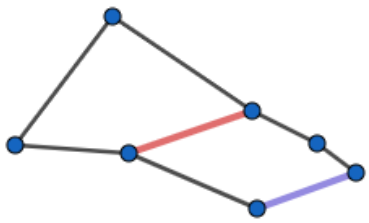


Рис. 2.2. Червоним позначено ребро, яке буде зараховано двічі, спочатку як границя верхньої циклічної грані, а потім як границя нижньої. Синім позначено ребро, яке буде зараховано як границя нижньої циклічної грані а також як границя зовнішньої нескінченої грані.

Оскільки $k + 1 \geq v \Rightarrow e \geq v$, то тоді у нашому графі є цикли.

Іншими словами, у цьому графі є обмежені грані.

Візьмемо будь-який цикл(циклічну грань), й видалимо з нього одне ребро. Тобто, спочатку ми мали граф G з параметрами v, e, f.

Після видалення ребра маємо граф G_1. Кількість вершин не змінилася, ребер стало на одиницю менше з-за видалення, а кількість граней також зменшилась на 1.

Отже отримали параметри $v, e-1, f-1$.

Оскільки у графа G_1 на одиницю менше ребер ніж у даного, то ми можемо для нього застосувати індуктивне припущення. Тобто, якщо відняти від числа вершин графа G_1 число його ребер та додати кількість граней, то отримаємо двійку.

Маємо: $v-(e-1)+f-1=2 \Rightarrow v-e+f=2$. Відповідно індукція завершена, шаг математичної індукції зроблений, а отже теорему доведено.

Наслідок з теореми Ейлера: Якщо в плоскому зв'язному графі $v \geq 3$, тоді $e \leq 3v-6$.

Нехай розмір грані - це сума кількості ребер у її границі та подвійної кількості внутрішніх ребер. Розглянемо довільну грань. Якщо її розмір не перевищує двох, тоді циклічної границі у даної грані немає, оскільки при наявності циклічної границі у розмір грані вже входить доданок «3» (не буває циклів, у яких довжина менше 3).

Оскільки грань не має циклічної границі, то вона є нескінченою, що не обмежена циклом. З того факту, що грань - нескінчена, маємо, що кількість її внутрішніх ребер не перевищує одиниці (оскільки другий доданок - подвійна кількість внутрішніх ребер). Тобто маємо одне ребро й нескінчену грань.

Оскільки граф є зв'язним, то він складається тільки з цього ребра та цієї грані. Але це суперечить умові $v \geq 3$. Отримали суперечність, а отже розмір кожної грані у даному графі не менше ніж 3.

Знайдемо суму розмірів усіх граней. З одного боку отримаємо подвійну кількість ребер. З іншого боку, виходить не менше ніж $3f$. Отже, $2e \geq 3f$.

З Теореми Ейлера маємо, що $v - e + f = 2$.

З нерівності маємо, що $f \leq \frac{2}{3}e$.

Тобто, $2 \leq v - e + \frac{2}{3}e \Rightarrow 2 \leq v - \frac{1}{3}e \Rightarrow e \leq 3v - 6$. Отже, теорему доведено.

Наслідок з Теореми 3. $K_5, K_{3,3}$ - не планарні графи $K_5: v \geq 3,$ оскільки за умовою дан повний граф на п’яти вершинах.

Якщо K_5 планарний, то тоді повинна виконуватись наступна нерівність: $e \leq 3v - 6$. Підрахуємо загальну кількість ребер у повному графі з п’тьяма вершинами: $C_5^2 = \frac{5!}{3!*2!} = \frac{20}{2} = 10$. Тобто, повинна виконуватись наступна нерівність:$10 \leq 3 * 5 - 6 \Rightarrow 10 \leq 9$. Неправильна числова нерівність, отже K_5 - не планарний граф.

$K_{3,3:} v \geq 3,$ оскільки за умовою дан повний двочастковий граф на шести вершинах.

Якщо K_5 планарний, то тоді повинна виконуватись наступна нерівність: $e \leq 3v - 6$.

Підрахуємо загальну кількість ребер у повному двочастковому графі на шести вершинах: $C_5^2 - 4 = \frac{6!}{2!*4!} - 4 = 15 - 4 = 9$.

Від кількості ребер у звичайному повному графі на 6 вершинах віднімаємо 4, бо маємо двочастковий граф, у якому, як відомо, не може бути ребер між вершинами однієї і тієї ж частки. Отже, повинна виконуватись наступна нерівність:$9 \leq 3 * 6 - 6 \Rightarrow 9 \leq 12$. неправильна числова нерівність, отже $K_{3,3}$не планарний граф.

2.2.3. Теорема Лема

Теорема Лема : існує вершина, ступінь якої не більше п’яти.

Як відомо, з наслідку з Теореми Ейлера маємо: якщо $v \geq 3$, тоді $e \leq 3v - 6$. Доведемо, що звідси випливає, що при $v \geq 3$ в графі є вершина, ступінь якої менше або дорівнює п’яти.

Доведемо це твердження методом від супротивного.

Нехай будемо вважати, що ступінь кожної вершини більше або дорівнює шести.

Тоді сума степенів всіх вершин не менше ніж $6 * v$, тобто $\sum_{\omega \in v} \deg\, \omega \geq 6 \cdot v$. З іншого боку, як відомо, $\sum_{\omega \in v} \deg\, \omega = 2 * |E| = 2 * e$.

Відомо, що $e \leq 3v - 6$, а отже $\sum_{\omega \in v} \deg\, \omega \leq 6 * v - 12$.

Маємо протиріччя, а отже в графі при $v \geq 3$ є вершина з ступенем не більше п’яти.

Доведемо теорему методом математичної індукції.

База індукції: $v = 3$, очевидно, що достатньо 5 кольорів.

Припустимо, що теорема доведена для $v \leq k$ вершин. Доведемо її для $v = k + 1$ вершин.

Нехай α - та сама вершина, існування якої було доведено у лемі, ступінь цієї вершини не перевищує п’яти. Якщо її ступінь не більше чотирьох, то розглянемо граф G/α, який одержується з G видаленням вершини α й всіх ребер, що є інцидентними до неї.

Граф G/α допускає правильне 5-фарбування за припущенням індукції. А так як ребра з’єднують вершину α не більш з ніж чотирма вершинами цього меншого графу, то для правильного розфарбування α залишається принаймні один колір з п’яти.

Нехай тепер $d(\alpha) = 5$. Розглянемо тепер граф $H \subset G$, що складається з вершин, які є суміжними з α, та ребер, що з’єднують ці вершини між собою. У

графі H обов'язково знайдуться дві вершини, що ребром не з'єднані. Доведемо це методом від супротивного: нехай всі п'ять вершин попарно з'єднані ребрами, тоді всього ребер: $e(H) = C_5^2 = \frac{5!}{3!*2!} = 5 * 2 = 10$. З іншого боку, оскільки граф H є підграфом основного графу, то він все одно залишається плоским, а отже за наслідком з теореми Ейлера: $e(H) \leq 3 * 5 - 6 \Rightarrow e(H) \leq 9$.

Отримали суперечність, отже є обов'язково дві вершини у H, що ребром не з'єднані.

Назвемо ці вершини β, γ.

Розглянемо граф G_1,який одержується з G/α шляхом деформації, яка ототожнює вершини β, γ.

Граф G_1- це плоский граф, оскільки при ототожненні вершин не може виникнути петлі. Але для G_1 є справедливим припущення індукції, і існує його деяке правильне розфарбування.

Роз'єднаємо вже у розфарбованому графі вершини β, γ. Тоді правильне 5-розфарбування отримує й G/α, при чому таку, що вершини β, γ розфарбовані однаково. Тобто, β, γ розфарбовані однаково, й розфарбування п'яти суміжних вершин з α графу H використовує не більше, ніж чотири кольори.

Використовуємо останній, 5-й колір для вершини α. Отже, теорему доведено.

2.3. Опис використаних технологій та мов програмування

Навчальна інформаційна система, що розроблена в даній роботі, реалізована на мові C# із застосуванням технології WPF. Це надало можливість створити зручний інтерфейс користувача, організувати інтерактивну роботу програми, забезпечити взаємодію програми з файловою системою комп'ютера.

Для розробки та дизайну користувацького інтерфейсу використана технологія WPF. Програма використовує багатопотоковість, методи, події, класи, наслідування та інші зручності для роботи великих проектів.

Для виводу теоретичної частини використовуються TreeView та Browser. Для тестування застосовується файл бази даних XML.

C# - простий, сучасний об'єктно-орієнтована і типобезпечна мова програмування. C # відноситься до широко відомої родини мов C, і здасться добре знайомим кожному, хто працював з C, C ++, Java або JavaScript.

C# є об'єктно-орієнтованою мовою, але підтримує також і компонентно-орієнтоване програмування. Розробка сучасних додатків все більше тяжіє до створення програмних компонентів у формі автономних пакетів, що реалізують окремі функціональні можливості. Важлива особливість таких компонентів - це модель програмування на основі властивостей, методів і подій. Кожен компонент має атрибути, які надають декларативні відомості про компоненті, а також вбудовані елементи документації. C# надає мовні конструкції, безпосередньо підтримує таку концепцію роботи. Завдяки цьому C# відмінно підходить для створення і застосування програмних компонентів.

Ось лише кілька функцій мови C #, що забезпечують надійність і стійкість додатків: прибирання сміття автоматично звільняє пам'ять, зайняту знищеними і невикористовуваними об'єктами; обробка виключень дає структурований і розширюваний спосіб виявляти і обробляти помилки; сувора типізація мови не дозволяє звертатися до неініціалізованих змінним, виходити за межі масиву або виконувати неконтрольоване приведення типів.

В C# існує єдина система типів. Всі типи C#, включаючи типи-примітиви, такі як int і double, успадковують від одного кореневого типу object. Таким чином, всі типи використовують загальний набір операцій, і значення будь-якого типу можна зберігати, передавати і обробляти схожим чином. Крім того, C# підтримує призначені для користувача посилальні типи і типи значень, дозволяючи як динамічно виділяти пам'ять для об'єктів, так і зберігати спрощені структури в стеці.

Щоб забезпечити сумісність програм і бібліотек C# при подальшому розвитку, при розробці C# багато уваги було приділено управлінню версіями. Багато мови програмування обходять увагою це питання, і в результаті

програми на цих мовах ламаються частіше, ніж хотілося б, при виході нових версій залежних бібліотек. Питання управління версіями істотно вплинули на такі аспекти розробки C#, як роздільні модифікатори virtual і override, правила вирішення перевантаження методів і підтримка явного оголошення членів інтерфейсу.

C# дозволяє стартувати розробку швидше, а це дозволяє швидше отримати прототип рішення. Швидкість розробки на C # на початкових етапах проекту значно вище в порівнянні з C ++.

Таким чином, в коротких малобюджетних проектах C# матиме перевагу в швидкості розробки, але в довгих і відносно дорогих дана перевага буде незначним.

C# спроектований бути кросплатформовим, однак його розвиток не пішов в цьому напрямку. Тому під Windows утворилася досить повна .net інфраструктура; на інших же платформах рівноцінної інфраструктури не з'явилося. І хоча C# можливо використовувати для побудови додатків під не-Windows платформи, проблеми, викликані використанням .net в не-Windows оточенні, зводять нанівець багато переваг вибору C#. Тому рекомендувати його для кросплатформового використання можна хіба що якщо код на C# вже написаний. При цьому треба чітко розуміти, що в перспективі це буде приносити додаткові витрати на підтримку.

Якщо говорити про сукупність суб'єктивних «простоти розробки», «краси коду» і об'єктивної продуктивності, то використовуючи C# простіше написати код, що задовольняє цим критеріям одночасно.

Відмінність асортименту C ++ і C# бібліотек в тому, що C ++ бібліотек більше, вони мають велику історію, за яку стали непогано налагоджені і оптимізовані, часто кросплатформових, багато з відкритим кодом. Однак при всіх позитивних сторонах C ++ бібліотеки як мають дуже різну, часто навіть архаїчну архітектуру, часто вже не об'єктний, а структурно-процедурний інтерфейс. Пов'язано це з тим, що багато C ++ бібліотеки це З бібліотеки.

Можна було б просто сказати, що під Window, C # помітно зручніше

налагоджувати і на цьому зупинитися.

Однак якщо з якоїсь причини у вас на ряду з mananged кодом з C # збірки використовується unmanaged, то його налагодження буде стане більш складна у порівнянні зі звичайною налагодженням unmamanged коду з C ++.

Синтаксис C#, мабуть, можна назвати спрощеною версією C ++, таким чином C#, як і будь-яке спрощення, одночасно несе і позитивний і негативний ефекти.

Варто сказати, що більш складний код часто легше пишеться і аналізується, якщо написаний більш простою мовою. З цієї позиції, використовуючи C#, менше шансів допустити помилку в принципово складному коді і більше шансів написати чистий код, володіючи тими ж ресурсами. Це може бути корисно при вирішенні досить складних, але не вимогливих до продуктивності завдань. Однак при цьому більшу кількість «синтетики» в C# робить менше оцінку продуктивності коду по його «зовнішнім виглядом».

У підтримці додатків варто розуміти, що деякі баги в додатках, написаних на C#, засобами .net виправити неможливо і при необхідності їх виправити вартість підтримки може істотно зрости.

Мабуть, основний ризик використання C# - це сильна зав'язка на Microsoft. Звичайно, навряд чи Microsoft кудись зникне в найближчому майбутньому, але варто розуміти, що Microsoft - це комерційна організація, метою якої є отримання прибутку, а для прибутку потрібні продажу своєї продукції. Тому в інтересах Microsoft розгортати розробку C# і .net так, щоб це приводило до продажу нової продукції Microsoft. Так що якщо інтереси вашої розробки будуть не відповідати інтересам Microsoft, рано чи пізно це може призвести до проблем.

2.4. Опис структури системи та алгоритмів її функціонування

2.4.1. Загальний опис структури системи

Під час виконання завдання кваліфікаційної роботи створена програма, яка правильно розфарбовує довільну мапу на площині та забезпечує роботу інтерактивного курсу з теорії графів, у якому подано структуровані теоретичні та практичні блоки, глосарій, а також особистий кабінет користувача, у якому він може побачити досягненні результати проходження тестів після кожного теоретичного блоку.

Робота складається з наступних частин (рис. 2.3):

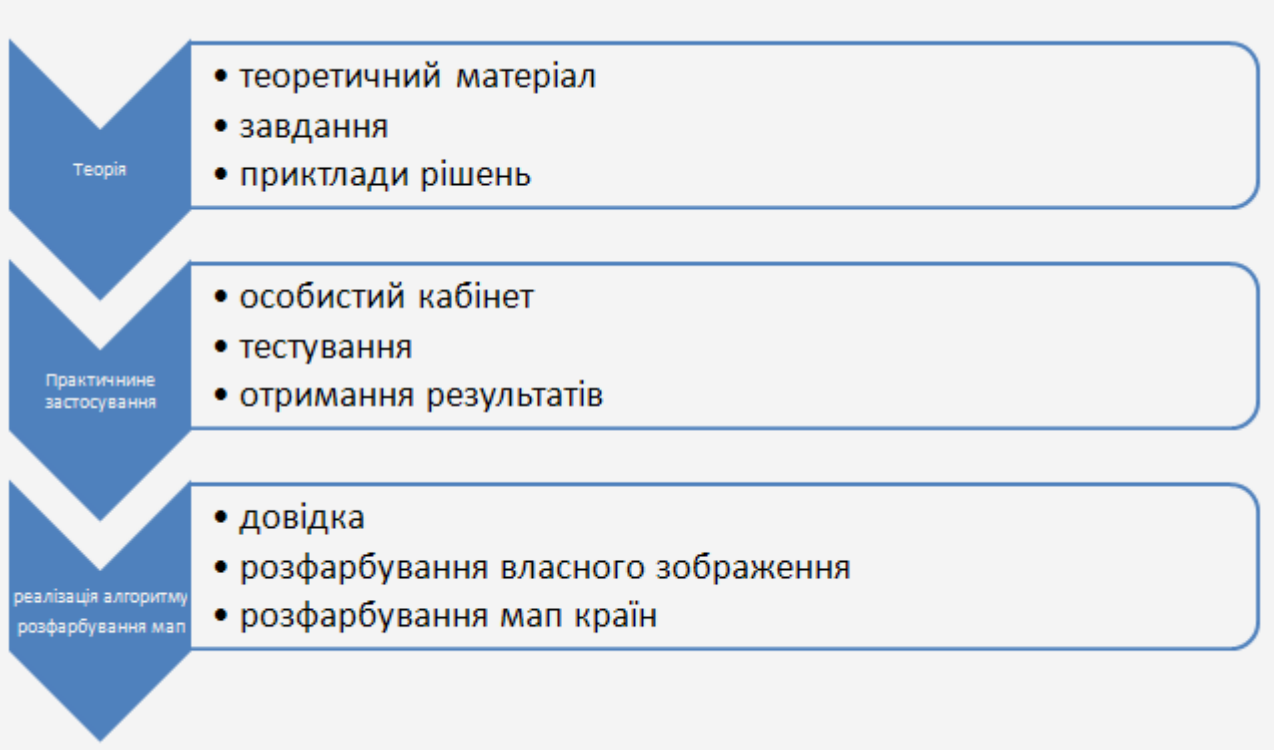


Рис. 2.3. Структура навчальної системи

1. Теоретичний блок, у якому подано впорядкований теоретичний матеріал з предмету та завдання для самостійного розв’язання та приклади їх розв’язків.

2. Практичний блок із особистим кабінетом користувача, де він може авторизуватись, проходити тести після вдалого опанування кожного теоретичного блоку.

3. Програма, що реалізує алгоритм, який розфарбовує довільну мапу на площині правильно у якомога меншу кількість кольорів (під поняттям «правильно» розуміється розфарбування мапи у мінімальну кількість кольорів так, щоб будь-які дві області, що мають спільну ділянку межі, не були розфарбовані в один колір).

2.4.2. Проєктування та реалізація алгоритму розфарбування довільної мапи на площині

На вхід програмі поступає зображення у форматі $.png$ або $.jpeg$. Умовно алгоритм роботи програми можна поділити на такі етапи:

1. Перетворення зображення у бінарну матрицю.
2. Відокремлення кожної області у матриці.
3. Перетворення матриці на граф, який представлений у матриці суміжності.
4. Перетворення представлення графа з матриці суміжності у список ребер.
5. Алгоритм розфарбування графу.
6. Перенесення результату розфарбування графу на зображення, яке мали на вході.

Розберемо концепт роботи кожного етапу.

2.4.2.1. Перетворення зображення у бінарну матрицю

Як відомо, на вхід програми поступає двокольорове зображення у форматі $.png$ або $.jpeg$. Перетворимо дане зображення на цілочисельний двовимірний масив, розмірами якого будуть розміри елементу $PictureBox$, у який було завантажене зображення. Далі, за допомогою метода $GetPixel$ класу $Bitmap$ перевіряємо колір кожного пікселя зображення.

Оскільки зображення двокольорове, то при перевірці кольору може виникнути 2 ситуації:

1. Колір пікселя – чорний (тобто цей піксель – частина границі). Тоді, у комірку з відповідними координатами новоствореного двовимірного масиву записуємо цифру «-1».
2. Колір пікселя – білий (тобто цей піксель – складова «внутрішньої частини» будь-якої області). Тоді, у комірку з відповідними координатами новоствореного двовимірного масиву записуємо цифру «0».

Відображення роботи першого етапу (розміри зображення та розміри отриманого масиву в даному прикладі не є однаковими, це зроблено для наочності та розуміння роботи етапу) відбувається за допомогою двовимірного масиву, що розміщується у таблиці 2.2.

Таблиця 2.2

Схематичне зображення двовимірного масиву Z після завершення першого етапу

-1	**-1**	**-1**	**-1**	**-1**	**-1**	**-1**	**-1**	**-1**	**-1**	**-1**
-1	0	0	0	0	0	0	**-1**	0	0	**-1**
-1	0	0	0	0	0	0	**-1**	0	0	**-1**
-1	0	0	0	0	0	0	**-1**	0	0	**-1**
-1	**-1**	**-1**	**-1**	**-1**	**-1**	**-1**	**-1**	0	0	**-1**
-1	0	0	**-1**	0	0	0	**-1**	0	0	**-1**
-1	0	0	**-1**	0	0	0	**-1**	0	0	**-1**
-1	**-1**	**-1**	**-1**	0	0	0	**-1**	0	0	**-1**
0	0	0	0	0	0	0	**-1**	0	0	**-1**
0	0	0	0	0	0	0	**-1**	**-1**	**-1**	**-1**

2.4.2.2. Відокремлення кожної області у матриці

Після виконання першого етапу отримали двовимірний масив, у якому можна «побачити» границі та області зображення за допомогою двох цифр – «0» та «-1». Але для майбутнього розфарбування мапи нам потрібно, щоб кожна область була унікальною, щоб вона відрізнялась від інших.

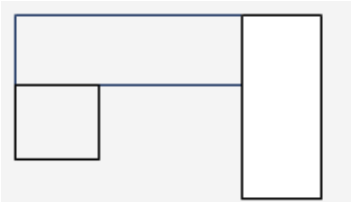


Рис. 2.4. Зображення карти, що маємо на вході.

На початковому зображенні кожна область, звичайно, є унікальною – вона має свої індивідуальні координати, тому ми й можемо їх відрізнити одну від одної. Але як зробити так, щоб у двовимірному масиві (надалі називатимемо його масив Z) кожна область також була індивідуальною? Наразі кожна область складається з нулів, й відокремлені вони одна від одної цифрами «-1».

Зробимо так, щоб тепер кожна область складалась не з нулів, а зі свого індивідуального номера. Наприклад, перша область позначена цифрами «1», друга область – цифрами «2»,… n-а область – цифрами «n».

Суть алгоритму другого етапу:

1. Знайти елемент у масиві Z, що позначений як нуль, та позначити його цифрою «1».

2. Позначити всі елементи цієї самої області також одиницями.

3. Повторити цю дію, поки ми не наддамо кожній області свій індивідуальний номер.

Реалізація алгоритму:

1. У тілі другого циклу робимо перевірку, чи є елемент масиву Z нулем.

2. У тілі умови створюємо динамічну структуру – чергу, надамо їй назву q. У цю чергу одразу записуємо точку(структура – $Point$) з координатами, що рівні індексам елементу масиву Z.

3. У самому масиві Z ми перезначаємо цей елемент, тепер він дорівнює змінній с (перед циклами створюємо цю змінну, даємо їй значення 0), а потім записуємо, що:

```
Z[i,j]=++c;
```

4. Наступним кроком є запуск циклу $while$. Цикл буде виконуватись, доки черга q не стане пустою. У циклі зчитуємо перший елемент черги за допомогою методу $Peek$:

```
Point p = q.Peek();
```

5. Надалі видаляємо цей елемент з черги за допомогою метода Pop. Призначимо змінні x та y наступним чином:

```
int x = p.X, y = p.Y;
```

6. Розуміємо, що потрібно перетворити на індивідуальний номер не тільки один елемент конкретної області, але усю «внутрішню частину». Тобто, потрібно пройти цю внутрішню частину «в ширину» - перетворити усі нулі на

одиниці (у даному випадку на одиниці, оскільки наразі працюємо з першою областю).

Щоб перетворити усі нулі на одиниці, ми можемо йти:

– вліво (тобто, зменшуючи змінну x на одиницю);

– вправо (тобто, збільшуючи змінну x на одиницю);

– вверх (тобто, зменшуючи змінну y на одиницю);

– вниз (тобто, зменшуючи змінну y на одиницю).

Розглянемо першу умову детально. Нам потрібно перевірити, чи є елемент у масиві Z, що стоїть зліва нулем. Якщо так, то його перевизначаємо одиницею, й одразу записуємо у чергу структуру $Point$ за допомогою метода $Enqueue$. Якщо ж ні (тобто цей елемент є границею), то нічого не відбувається, й ми переходимо до наступних умов.

Також не слід забувати й про область допустимих значень, які може приймати змінна x. У цьому випадку вона повинна бути додатною, щоб не вийти за межі масиву Z:

```
if (x > 0 && Z[x - 1, y] == 0)
        {
           Z[x - 1, y] = c;
           q.Enqueue(new Point(x - 1, y));
        }
```

Аналогічними є й інші три умови.

Отже, після перевірки чотирьох умов, якщо маємо ще точки у черзі q, то повторюємо вище описаний процес для наступної точки.

7. Якщо для першої області її перевизначення виконано, повертаємось до умови, де шукаємо елементи масиву Z, (п. 6) що рівні нулеві, й знов повторюємо усі вище зазначені дії для наступної області.

Алгоритм працюватиме, доки не буде перевизначено кожну область.

До виконання другого етапу мали масив Z, схему якого надано у таблиці 2.2. Після виконання другого етапу матимемо масив Z, схему якого надано у таблиці 2.3:

Таблиця 2.3

Схематичне зображення двовимірного масиву Z після завершення другого етапу

-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	2	2	2	2	2	2	-1	3	3	-1
-1	2	2	2	2	2	2	-1	3	3	-1
-1	2	2	2	2	2	2	-1	3	3	-1
-1	-1	-1	-1	-1	-1	-1	-1	3	3	-1
-1	4	4	-1	1	1	1	-1	3	3	-1
-1	4	4	-1	1	1	1	-1	3	3	-1
-1	-1	-1	-1	1	1	1	-1	3	3	-1
1	1	1	1	1	1	1	-1	3	3	-1
1	1	1	1	1	1	1	-1	-1	-1	-1

2.4.2.3. Перетворення матриці на граф, який представлений у матриці суміжності

Ініціалізуємо новий двовимірний масив для зберігання у ньому матриці суміжності, назвемо його $Graph$. Далі перетворимо масив Z у матрицю суміжності наступним чином: ініціалізуючи подвійний цикл, перевіряємо кожен елемент масиву Z, й шукаємо цифри «-1», тобто шукаємо границі між областями. Коли буде знайдено границю, зробимо наступні дії:

1. Аналізуємо елемент масиву Z, що стоїть зліва від границі, назвемо його u (тобто той, що має індекс$(i-1,j)$) й елемент, що стоїть справа від границі, назвемо його v (тобто той, що має індекс$(i+1,j)$). Якщо ці 2 елементи більше одиниці, та не є рівними між собою, то тоді у матриці суміжності $Graph$ позначаємо між цими двома вершинами ребро(тобто, у

масиві $Graph$ елементу (u, v) даємо значення «1»). Елементу (v, u) також даємо значення «1», бо граф є неорієнтований.

2. Робимо аналогічні дії, що й у першому пункті, але для випадку, коли елементи масиву Z стоять відповідно зверху(тобто той, що має індекс $(i, j + 1)$) й знизу (тобто той, що має індекс $(i, j - 1)$).

Після виконання третього етапу маємо наступний двомірний масив $Graph$ (табл. 2.4):

Таблиця 2.4

Схематичне зображення двовимірного масиву Graph після завершення третього етапу

0	1	1
1	0	0
1	0	0

2.4.2.4. Перетворення представлення графа з матриці суміжності у список ребер

1. Спочатку Ініціалізуємо наступну структуру:

List<Point>[] adj = new List<Point>[1000];

Даний рядок коду означає, що створюється 1000 посилань, й поки що вони всі є нульовими(тобто, не несуть якоїсь інформації).

2. Поки вважаємо, що кількість вершин не перевищує 1000, але, якщо потрібна більша кількість, просто змінимо кількість листів. Потім, за допомогою циклу, для кожної вершини створимо свій лист.

Тобто, $hrt[i]$ (де i – це індекс вершини) – це посилання на лист окремої вершини, у якому зберігається інформація про «сусідів» цієї самої вершини.

3. Застосуємо стандартний алгоритм для перетворення матриці суміжності список ребер, й отримаємо для матриці суміжності $Graph$ наступну структуру (табл. 2.5):

Таблиця 2.5

Структура після перетворення матриці суміжності у список ребер

Лист	Інформація, що зберігається у листу
adj[0]	1,2
adj[1]	0
adj[2]	0

2.4.2.5. Алгоритм розфарбування графу

Алгоритм розфарбування графу є типовим жадібним алгоритмом.

Концепт алгоритму:

1. Пофарбувати першу вершину у першій колір.
2. Зробити наступне для c-1 вершин, що залишились: перейти до сусідньої вершини, й розфарбувати її у найменший колір, який ще не було застосовано до будь-якої вершини, що є сусідньою з нею. Якщо ж усі кольори вже застосовані для суміжних вершин, то пофарбувати вершину у новий колір.

Програмна реалізація алгоритму:

```
int num_vertex = c;
int[] result = new int[num_vertex + 1];
bool[] available = new bool[num_vertex + 1];
int d;
for (int i = 0; i <= num_vertex; i++)
   {
      result[i] = -1; // no color assigned
```

```
            available[i] = false;
        }
    result[0] = 0;
    for (int i = 1; i < num_vertex; i++)
        {
            for (d = 0; d < adj[i].Count; d++)
                {
                    int to = adj[i][d];
                    if (result[to] != -1)
                        {
                            available[result[to]] = true;
                        }
                }
                for (d = 0; d < num_vertex; d++)
                    if (!available[d])
                        break;
                result[i] = d;
                for (d = 0; d < num_vertex; d++)
                    available[d] = false;
        }
```

Аналіз алгоритму:

1. Спочатку Ініціалізуємо змінну num_vertex, яка чисельно дорівнює кількості вершин у графі. Потім ініціалізуємо два одновимірні масиви:

– масив $result$ у якому зберігатимемо результат розфарбування (індекси масиву – номери вершини, а значення елементу масиву – номер кольору, у який розфарбована конкретна вершина);

– масив $available$ – тимчасовий масив типу $bool$ для зберігання доступних кольорів. Якщо якийсь елемент масиву $available[a]$ приймає

значення $true$, то це означає що колір а вже присвоєно до якоїсь однієї з сусідніх вершин.

2. На початку роботи алгоритму кожній вершині присвоюємо колір «-1», що означає що ще ніякі вершини не розфарбовані. Також кожному елементу масиву $available$ призначаємо значення $false$, оскільки ще жоден колір не був застосований.

3. Потім, згідно концепту алгоритму, присвоюємо першій вершині колір «0».

4. Далі робимо обхід послідовно по вершинах, що суміжні з попередньо розглянутою вершиною(тобто, у першій ітерації розглядаємо другу вершину графу, що суміжна з першою, якій вже присвоєно колір «0»). Потім йде перевірка, чи має вершина суміжні вершини, що вже розфарбовані у якісь кольори(ця перевірка необхідна, щоб надалі виключити ці кольори з списку можливих кольорів для вершини, що розглядається).

Якщо вона має таких «сусідів», то тоді у тимчасовий масив $available$ заносимо ці кольори з позначкою $true$, що означає що такі кольори застосовані до деяких сусідніх вершин і що їх застосовувати не можна.

5. Надалі запускається цикл, який, починаючи з нуля, шукає найменший можливий колір, який ще не застосований до суміжних з вершиною, що розглядаємо вершин. Коли ж такий колір знайдено(а його обов’язково буде знайдено, оскільки навіть у випадку, якщо кольори, які вже використовувалися до цього, закінчилися, алгоритм просто візьме новий колір й присвоїть його до вершини, що розглядаємо), то він присвоюється до вершини, що розглядаємо, й ми терміново виходимо з циклу, бо вже знайдено найменший можливий колір.

6. Потім у масиві $result$ оновлюємо інформацію щодо вершини, що ми тільки-но розглянули. Присвоюємо вершині відповідний колір, що було знайдено у попередньому циклі.

7. Як і зазначалось вище, масив $available$ – тимчасовий, а отже у кінці кожної ітерації алгоритму ми «скидаємо» усі значення масиву для коректної роботи у наступній ітерації (оскільки у кожній вершині будуть свої власні «сусіди» зі своїми кольорами, які не можна буде застосовувати).

2.4.2.6. Розфарбування зображення

Кінцевий етап роботи програми – це розфарбування зображення, тобто, ми вже отримали результат – вершини графа розфарбовані внаслідок роботи п’ятого етапу. Але тепер потрібно перенести результат на зображення – розфарбувати його.

Повернемося до другого етапу. До його початку Ініціалізуємо наступну структуру:

List<Point>[] hrt = new List<Point>[1000];

Даний рядок коду означає, що створюється 1000 посилань, й поки що вони всі є нульовими(тобто, не несуть якоїсь інформації).

Зробимо деякі зміни у другому етапі. Тепер, після виявлення кожної нової області, будемо створювати новий лист для кожної конкретної області:

hrt[c] = new List<Point>();

Тепер, посилання hrt з індексом c веде до листу, у якому будуть міститися усі точки, що належать даній конкретній області.

Наповнювати кожний лист точками, що належать конкретній області будемо одразу після знаходження таких точок, тобто у тілі циклу $while$.

Після завершення роботи другого етапу ми матимемо стільки посилань на листи, скільки всього маємо областей, й у кожне посилання буде переводити нас до кожного окремого листа з інформацією про усі точки конкретної області.

У кінці програми створюємо цикл, щоб обробити кожний такий лист. Початок циклу буде з індексу, що дорівнює двійці, оскільки, як зазначено вище, одиницями «покрита» біла область, тобто та, яка є фоном зображення і не є

областю на зображенні. А самі області, які потрібно розфарбувати, починаються з індексу «2».

Потім, за допомогою $foreach$ розглядаємо кожну точку у листі конкретної області, й, за допомогою метода $SetPixel$ присвоюємо кожній точці конкретної області колір, що було отримано у результаті попереднього етапу. У п’ятому етапі було отримано для кожної вершини тільки індекс кольору. Але для зображення потрібні реальні кольори.

Перед циклом створимо масив, що буде містити 5 кольорів.

```
Color[] clrNew = { Color.Yellow, Color.DarkRed, Color.Blue, Color.Tan, Color.SlateGray};
```

А потім у методі $SetPixel$, третім аргументом слугуватиме елемент з масиву $clrNew$, що матиме індекс – номер кольору, який було отримано у 5-му етапі

2.4.3. Реалізація інтерактивного курсу з теорії графів

Друга частина навчальної системи відноситься до інтерактивного курсу з теорії графів. Перейшовши до курсу з теорії графів, користувач має змогу вивчити 4 розділи:

1. Введення. Базові поняття.
2. Дерева.
3. Теорема Келі, Ейлерові цикли.
4. Гамільтонови графи.

У кожному блоці представлений структурований теоретичний матеріал, що надається у вигляді перегляду невеликих документів.

У процесі навчання, користувач може скористатися інтерактивним словником з теорії графів. Перейшовши до словника, можна знайти за допомогою стрічки пошуку певні терміні, ввівши ключові слова. Словник стане у нагоді, якщо потрібно згадати якийсь термін при вивченні нового матеріалу.

Після опанування блоку, користувач може подивитися список практичних задач, та почати їх розв’язувати. Задля кращого засвоєння вивченого матеріалу після списку завдань подано повні, авторські розв’язки кожної задачі, з належним обґрунтуванням та відповідями.

2.4.4. Реалізація етапу перевірки знань

Після опанування кожного теоретичного блоку, бажано пройти практичний тест задля контролю знань. Щоб отримати доступ до тестів, потрібно зареєструватися в особистому кабінету. Після цього можна багаторазово авторизуватися, щоб потрапити в особистий кабінет користувача. В особистому кабінеті для кожного учня відображено його власні результати усіх практичних блоків, а також діаграма що показує його рівень знань з кожного блоку. Саме з особистого кабінету здійснюється перехід до тестів.

Перейшовши до тесту, в учня є обмежений час (20 хв.) для виконання завдань. Присутні такі типи завдань:

– завдання з вибором однієї правильної відповіді;
– завдання з кількома правильними відповідями;
– завдання з полем для введення відповіді;
– завдання за малюнками.

В кожному блоці міститься від семи до дев’яти завдань. Після виконання усіх завдань, при натисненні кнопки перевірки, одразу можна побачити результат. Також відокремлюється завдання, що були розв’язані неправильно або не були розв’язані. Користувач має право перескладати тести.

2.5. Обґрунтування та організація вхідних та вихідних даних програми

Вхідними даними програми є власне зображення, яке потрібно намалювати у полі форми програми та зберегти в форматі .png чи .jpeg, або зображення, обране з репозиторію, які подаються на вхід програми та з якими відбуваються маніпуляції.

Програма розфарбовує завантажене зображення (мапу) на площині методом чотирьох фарб та виводить його зображення на екран.

Вхідними даними для системи також є ввід логіну, паролю та імені користувача в блоці перевірки отриманих знань, а також вибір відповідей на тести.

Вихідною інформацією в блоках теоретичної та практичної підготовки є тексти з теоретичних відомостей, завдань та їх розв’язання, а також аналіз пройденого тестування. Для виводу теоретичної частини використовуються TreeView та Browser. Для тестування застосовується файл бази даних XML

2.6. Опис роботи розробленої системи

2.6.1. Використані технічні засоби

Для розробки даної ІС використовувався ПК з наступними технічними характеристиками:

– процесор із тактовою частотою 1 ГГц (x64);

– оперативна пам’ять 2 ГБ (для 64-розрядної версії);

– графічний пристрій із підтримкою DirectX 9 і драйвером WDDM 1.0 або новішим.

2.6.2. Використані програмні засоби

У програмі використані технології платформи .Net 4.0 - 4.6 та мова програмування C#.

2.6.3. Виклик та завантаження програми

Запуск додатку відбувається за допомогою файлу TRY.exe після копіювання на ПК директорії проєкту «TRY» .

2.6.4. Опис інтерфейсу користувача

Завантаживши програму користувача вітає форма, де можна подивитися головне меню та перейти до нього (рис. 2.5):

Рис. 2. 5. Зображення головного меню програми

Головне меню поділяється на 3 розділи: «Проблема чотирьох фарб», «Теоретична частина» та «Практична частина».

У розділі «Проблема чотирьох фарб» (рис. 2.6) користувач за викликом допомоги (рис. 2.7) може розфарбувати власне зображення (рис. 2.8), або завантажити реальну мапу країни з репозиторію (рис. 2.9-2.12), або обрати абстракцію з репозиторію (рис. 2.13-2.15).

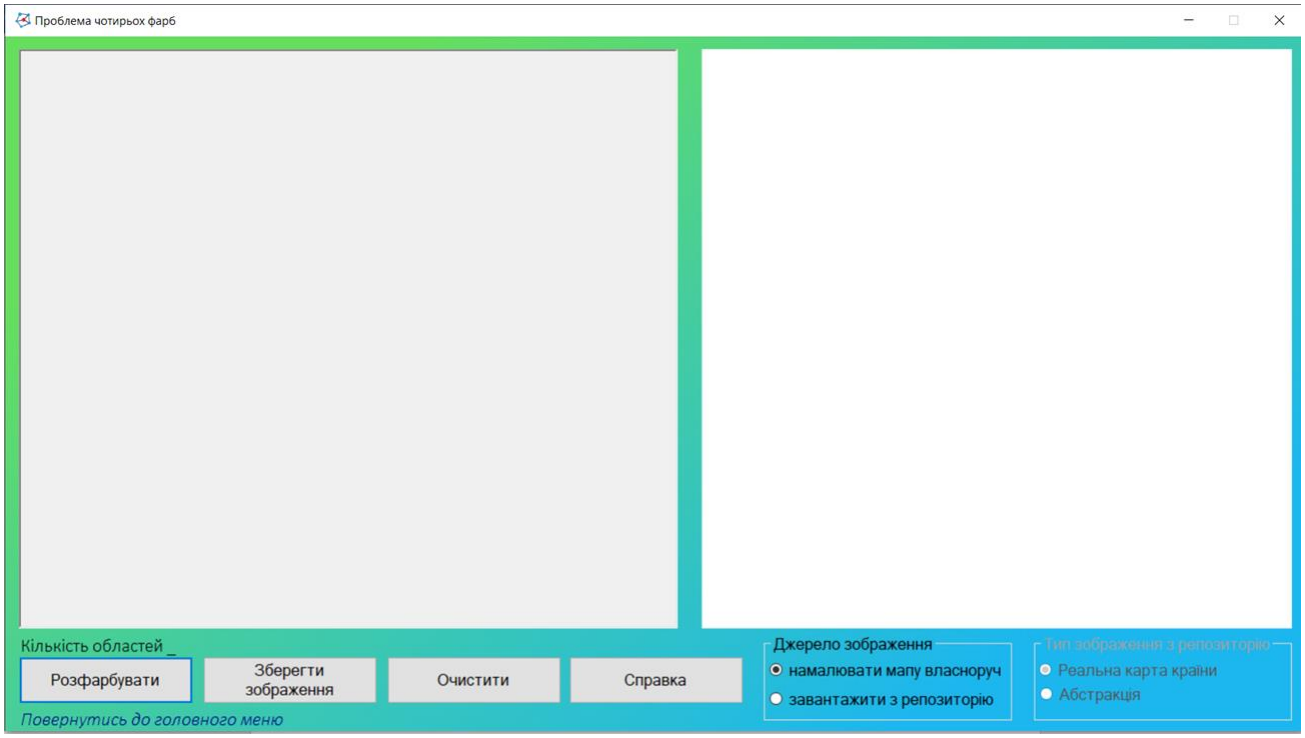


Рис. 2.6.Форма розділу «Проблема чотирьох фарб»

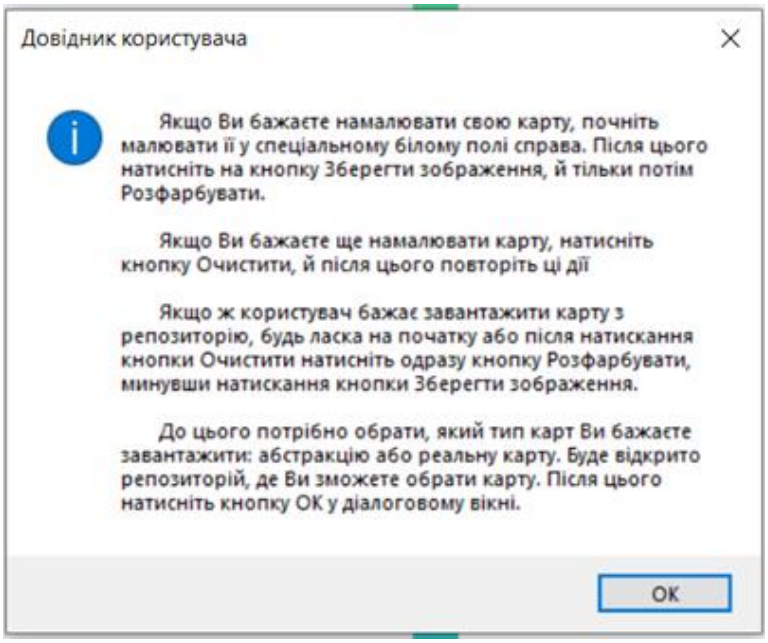


Рис. 2.7. Зміст довідки

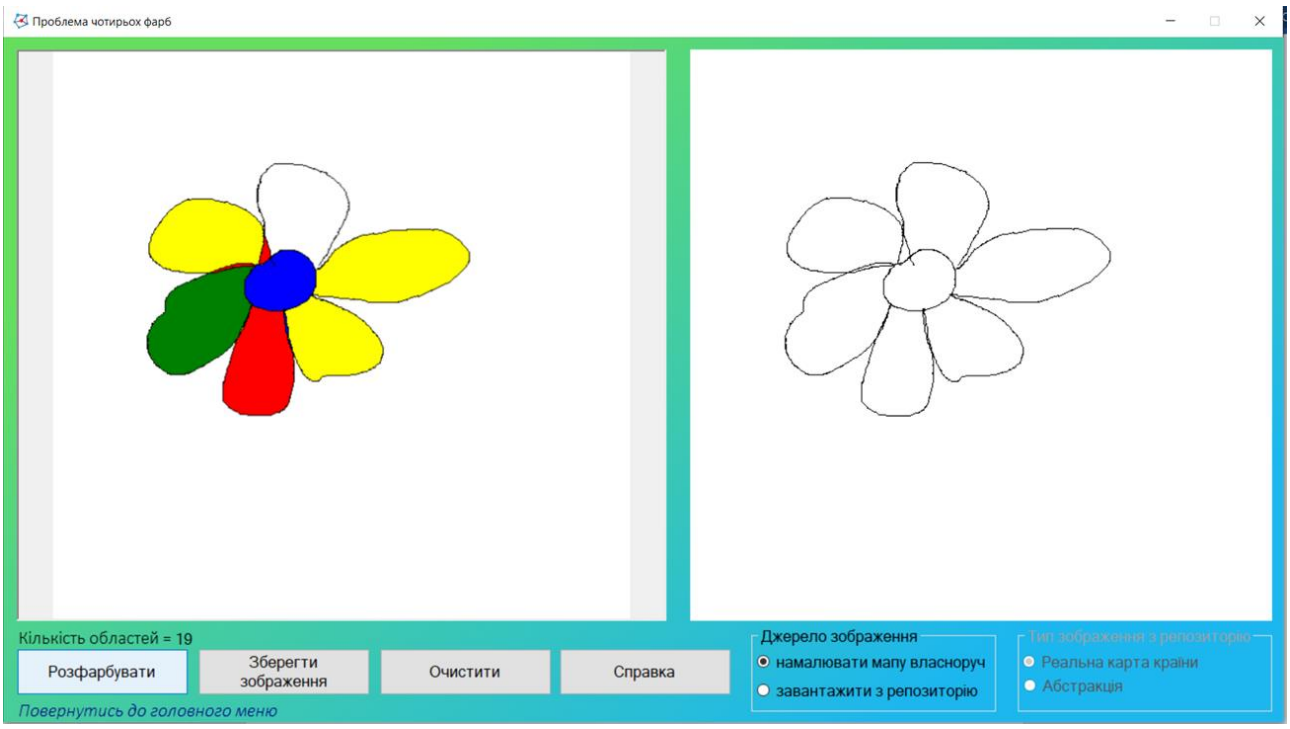


Рис. 2.8. Результат роботи програми розфарбовування власного зображення

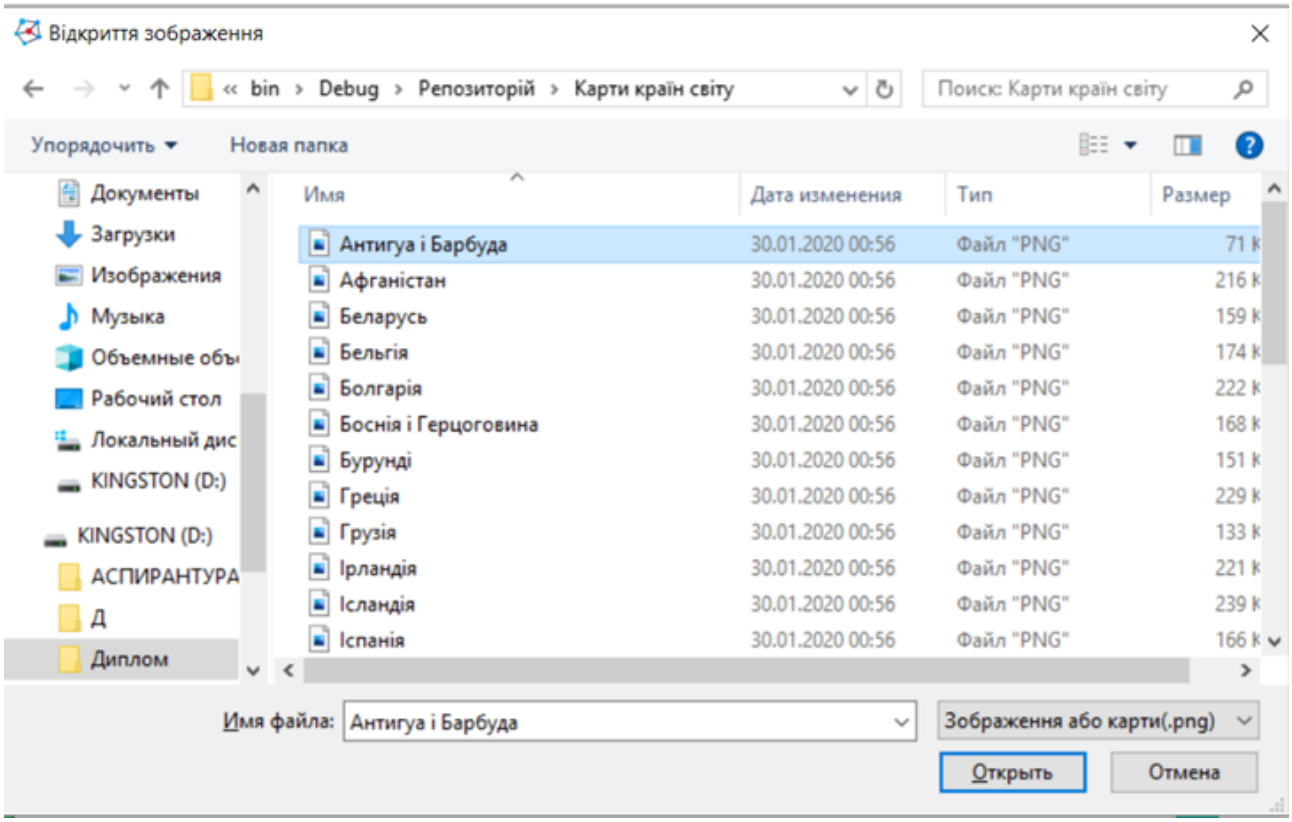


Рис. 2.9. Вибір для завантаження мап країн

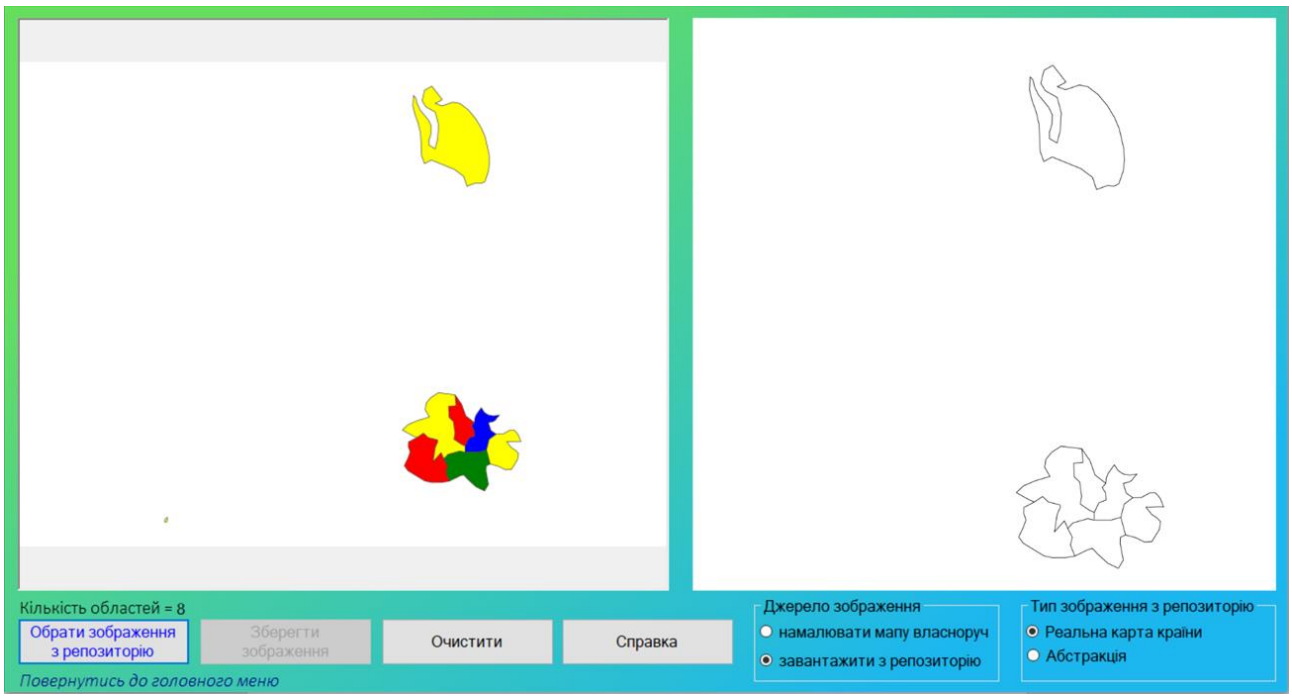


Рис. 2.10. Результат роботи програми розфарбовування мапи Антігуа і Барбуди

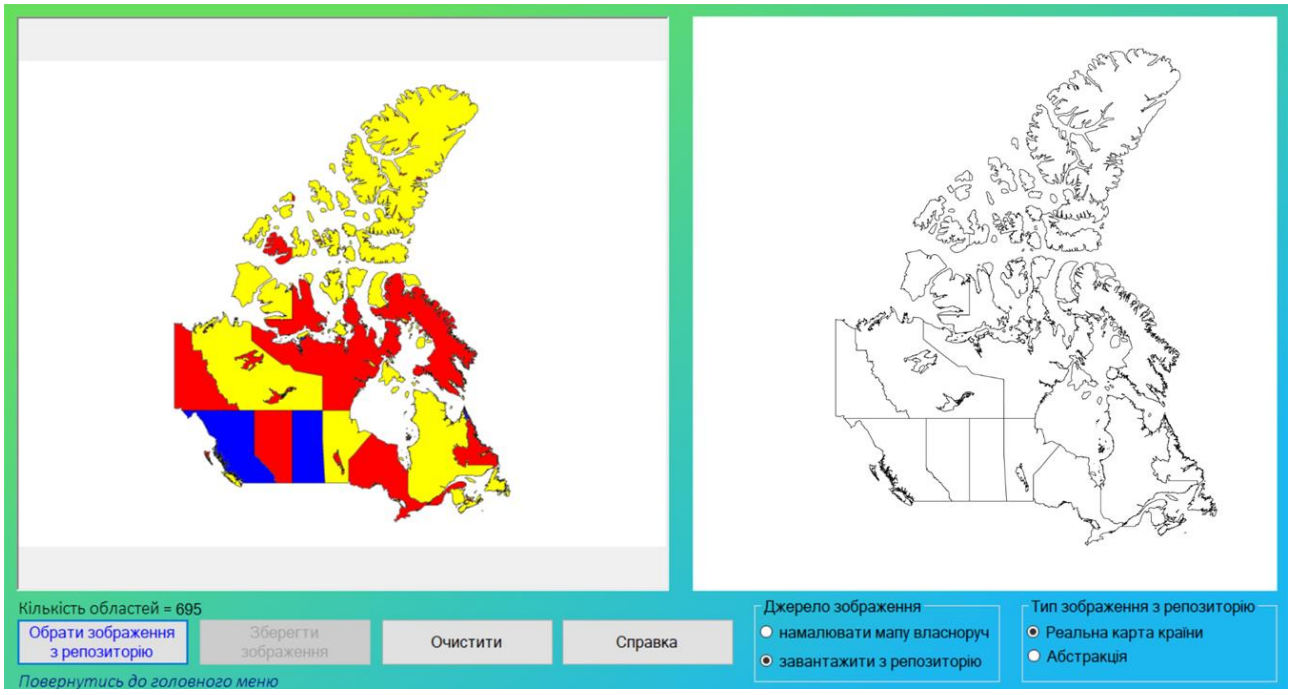


Рис. 2.11. Результат роботи програми розфарбовування мапи Канади

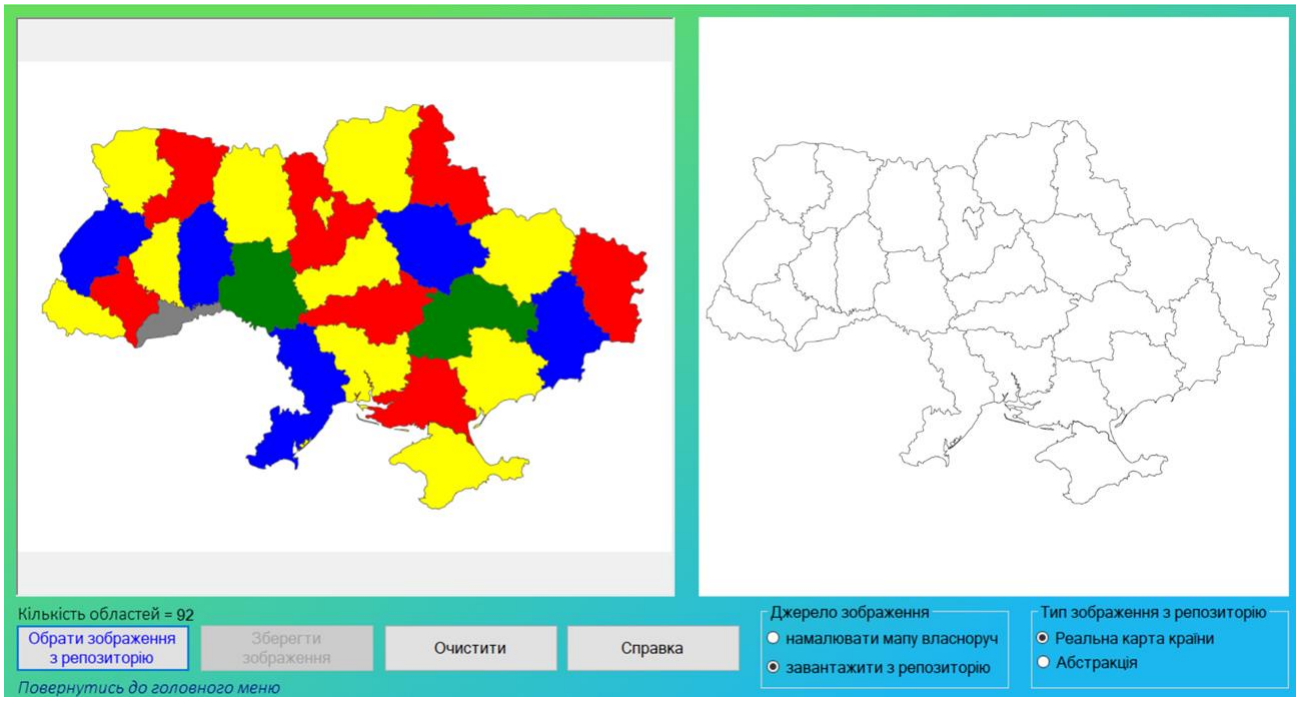


Рис. 2.12. Результат роботи програми розфарбовування мапи України

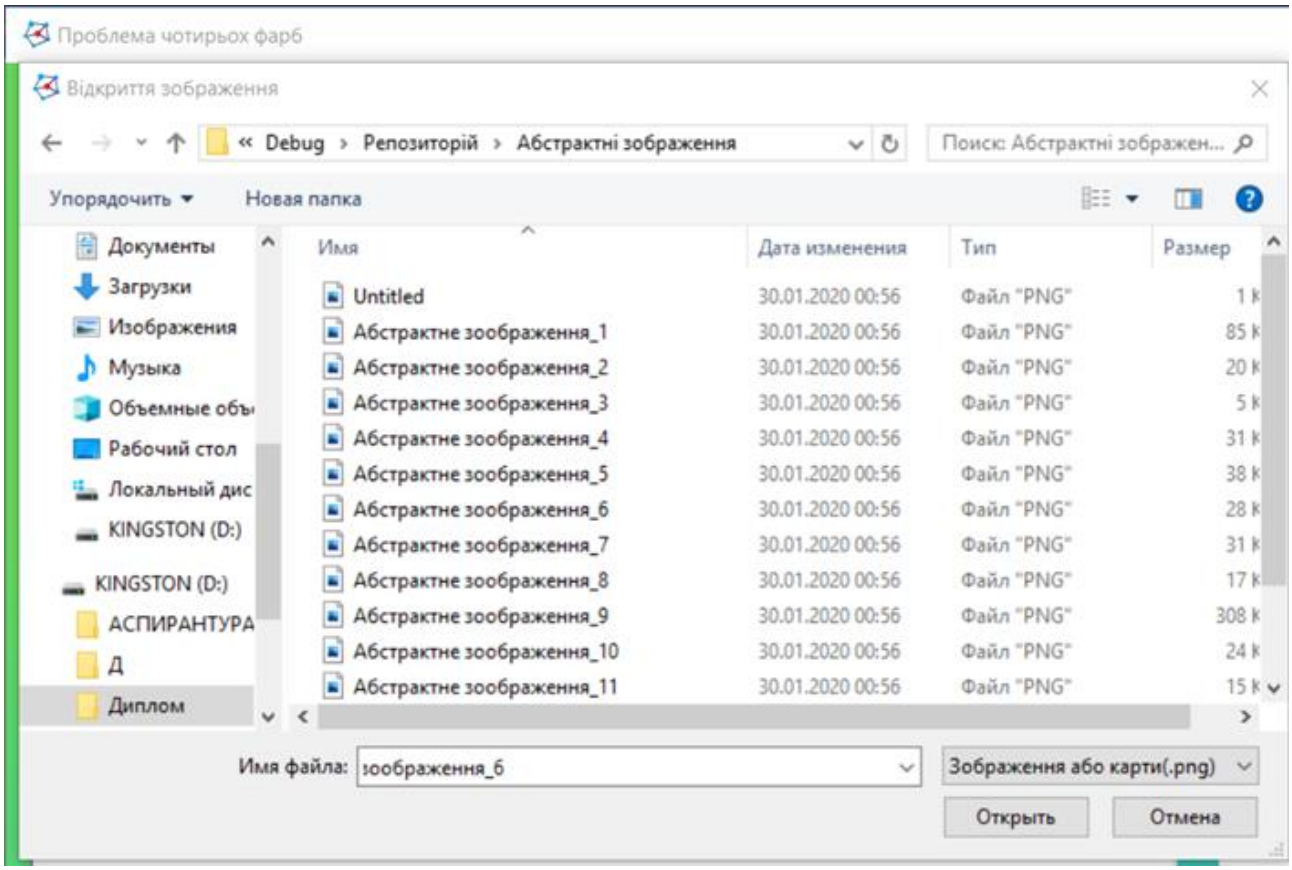


Рис. 2.13. Вибір для завантаження готових абстракційних зображень

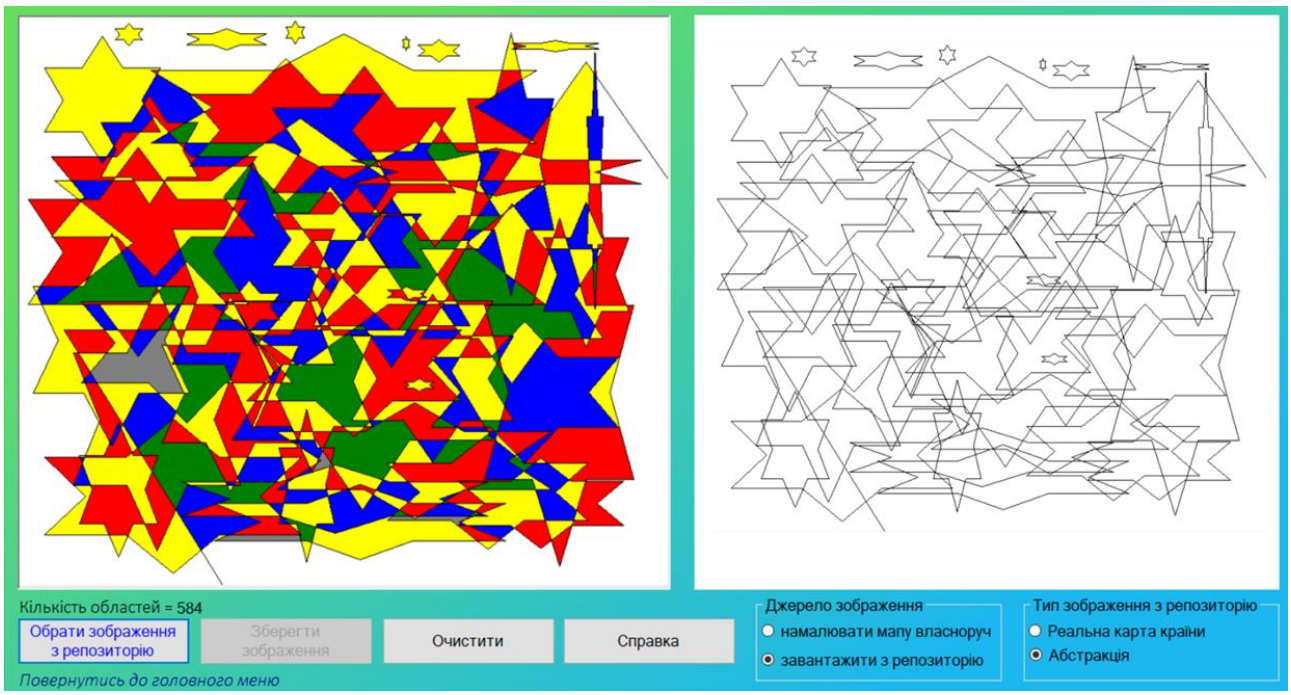


Рис. 2.14. Результат роботи програми розфарбовування абстрактного зображення 1

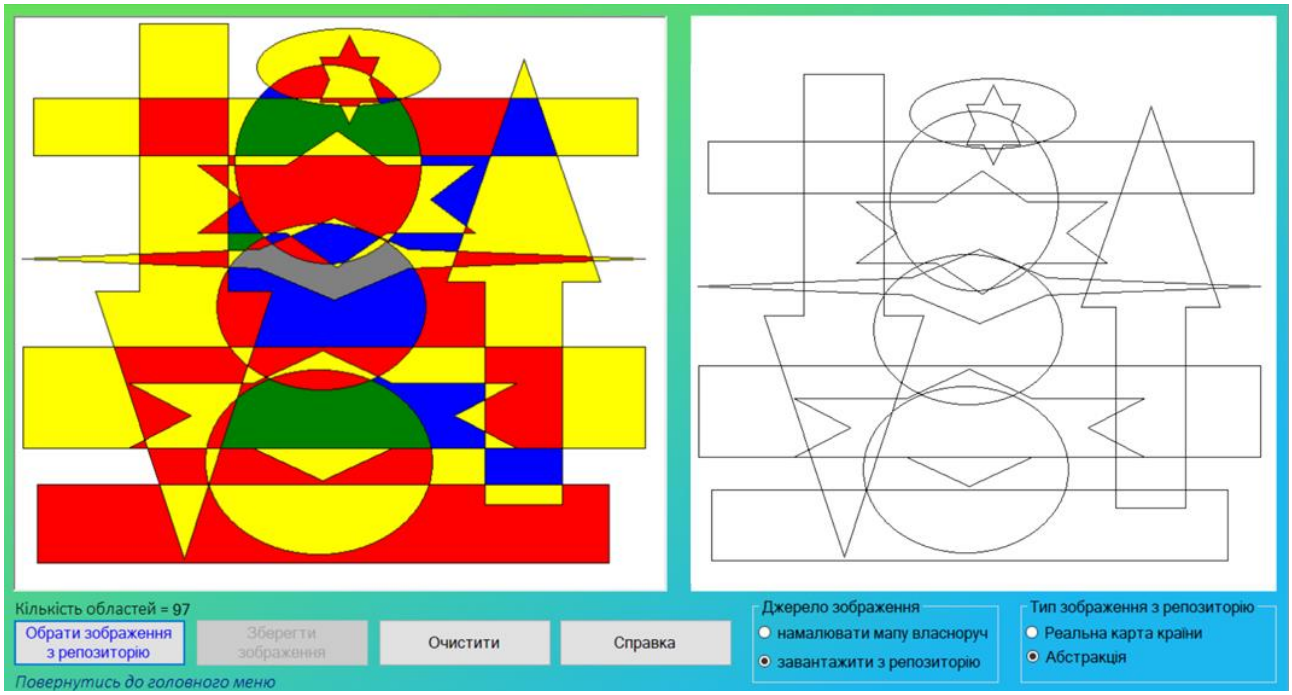


Рис. 2.15. Результат роботи програми розфарбовування абстрактного зображення 2

Друга частина навчальної системи відноситься до інтерактивного курсу з теорії графів. Перейшовши до курсу з теорії графів, користувач має змогу вивчити 4 розділи (рис. 2.16):

1. Введення. Базові поняття.
2. Дерева.
3. Теорема Келі, Ейлерові цикли.
4. Гамільтонови графи.

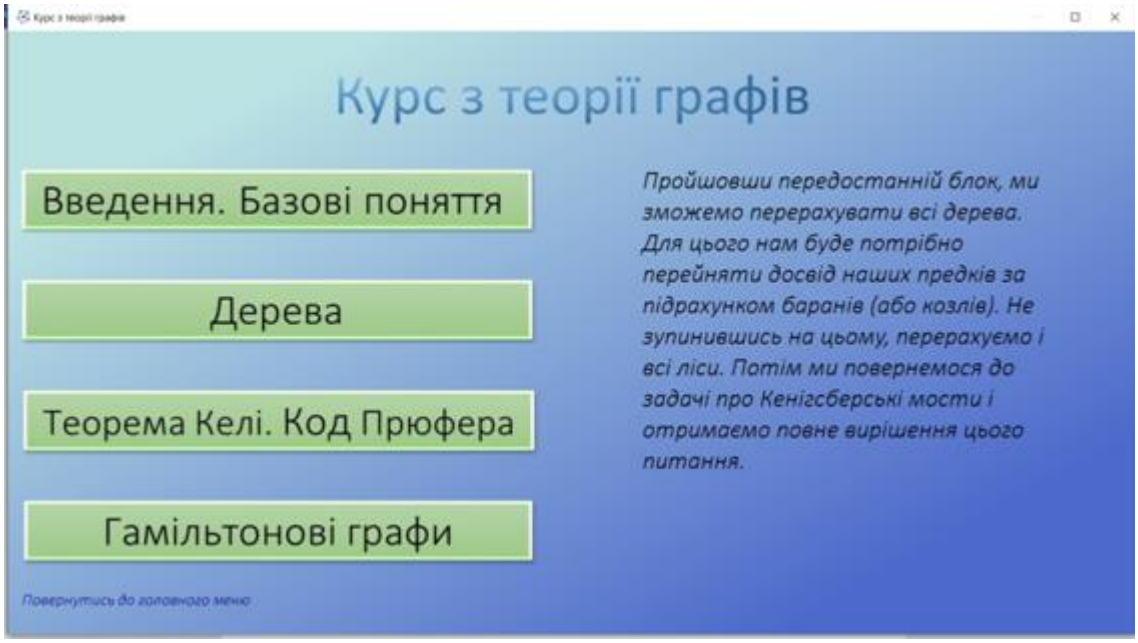


Рис. 2.16. Зображення теоретичної частини програми

Теоретична частина включає в себе як подання теоретичного матеріалу з обраної теми (рис. 2.17) та і практичні завдання (рис. 2.18) та приклади їх рішень (рис. 2.19). Вивчаючи теорію, користувачеві може знадобитися застосувати інтерактивний словник (рис. 2.20).

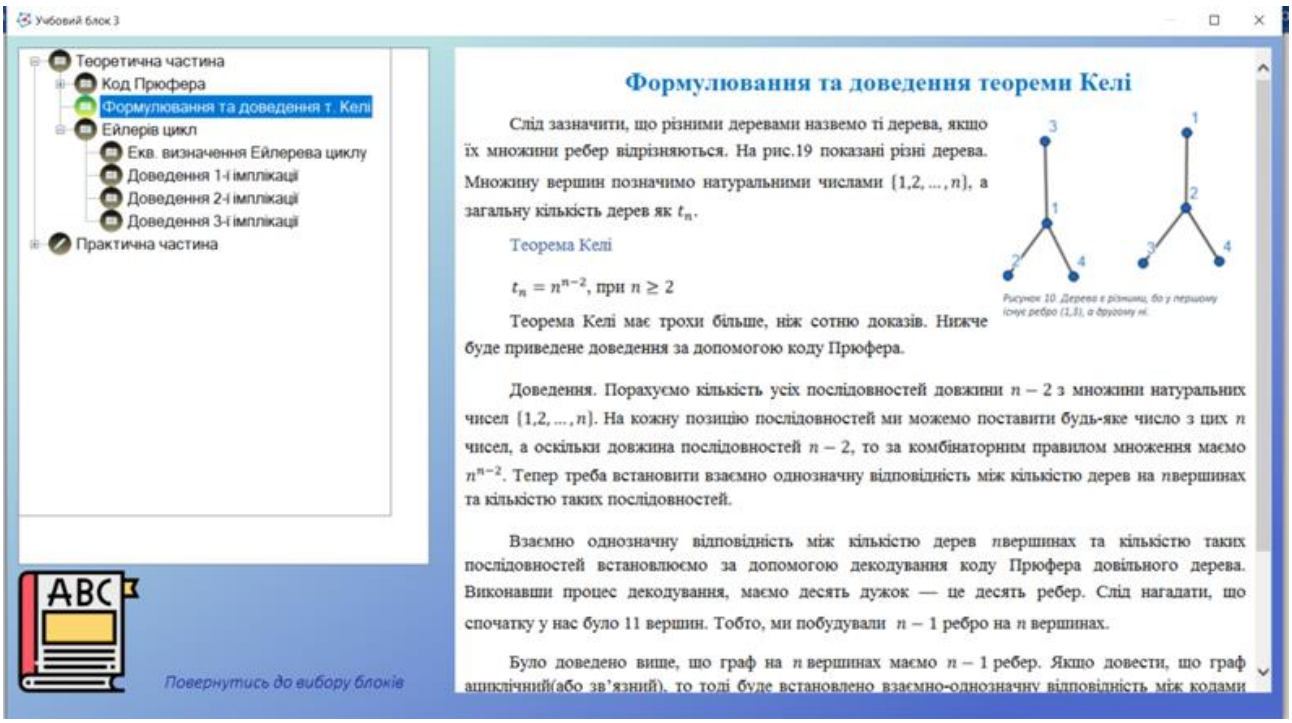


Рис. 2.17. Зображення теоретичного матеріалу з обраної теми

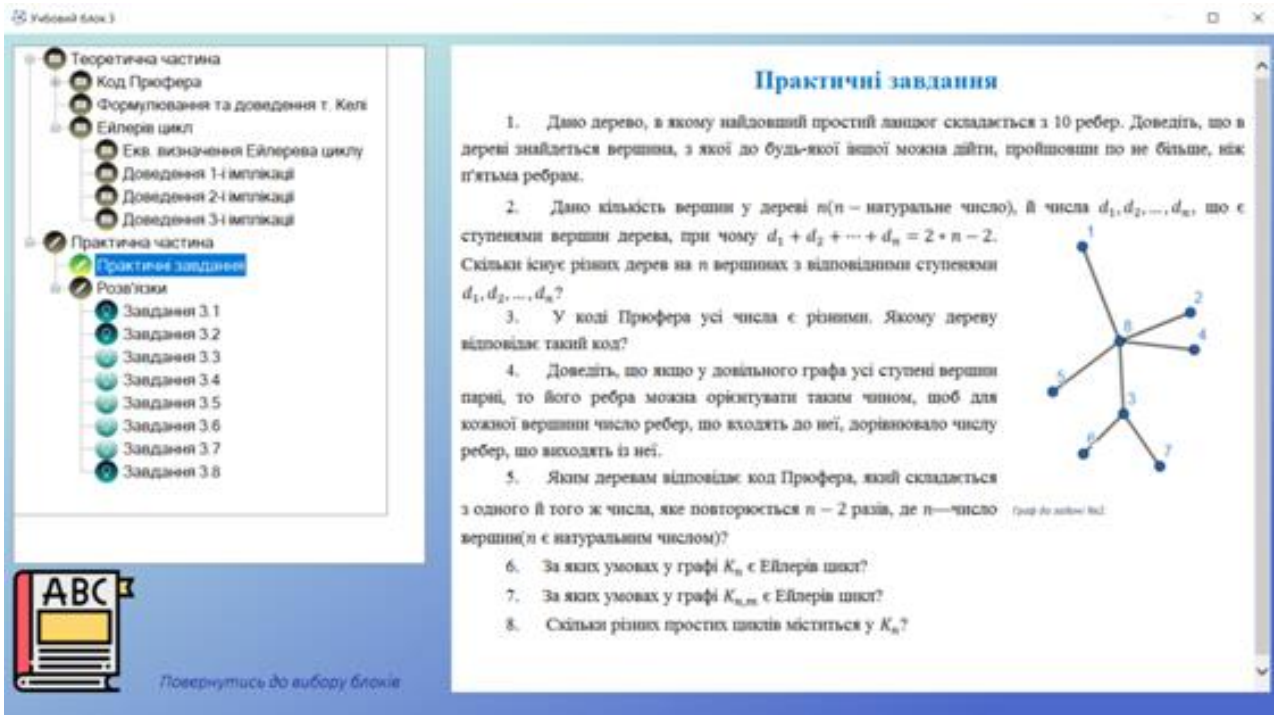


Рис. 2.18. Практичні завдання

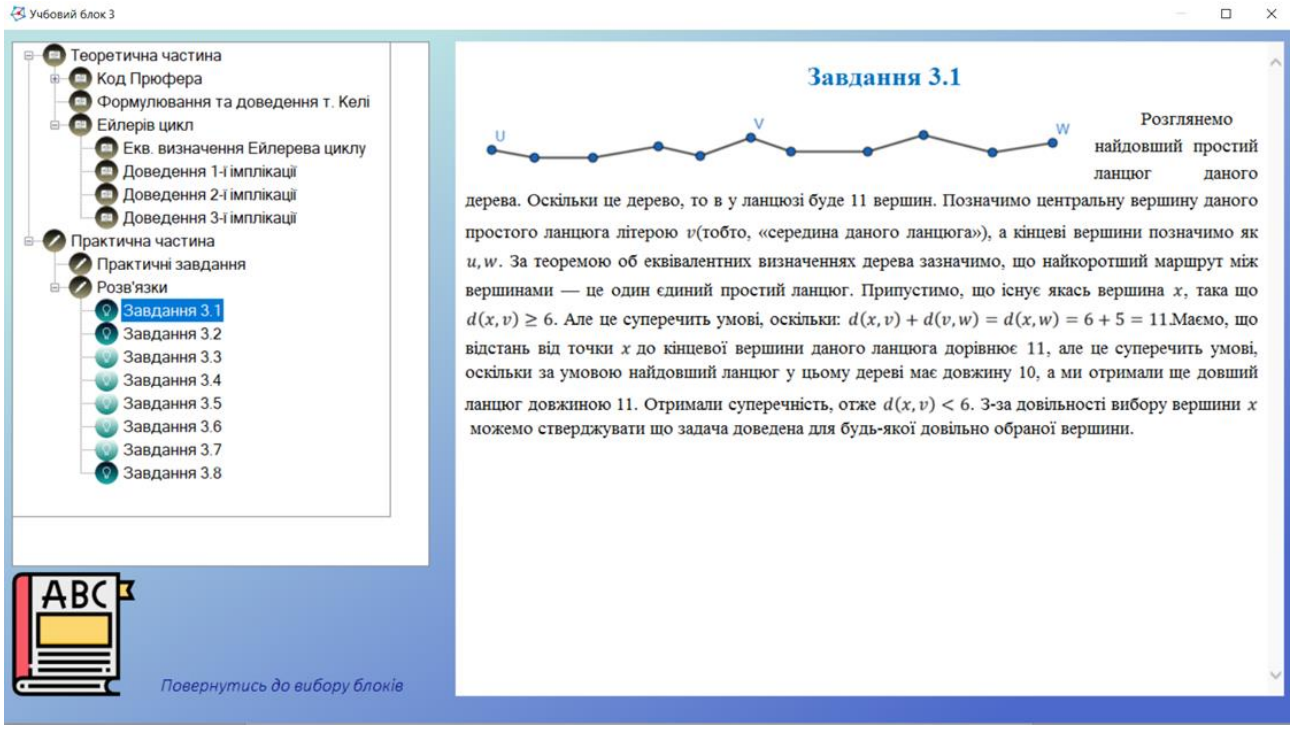


Рис. 2.19. Приклади рішень практичних завдань

Термін	Визначення
Ациклічний граф	граф, що не містить циклів.
Біграф	те ж саме, що й двочастковий граф.
Валентність вершини	дуже рідко зустрічається саме такий термін, це те ж саме що й ступінь вершини.
Вершина(вузол)	(базове поняття) фундаментальна одиниця графу, це точка, де можуть сходиться/виходити ребра.
Висота дерева	найбільший за довжиною ланцюг від коріння до листа.
Висяча вершина	вершина, ступінь якої дорівнює одиниці
Відстань між вершинами	довжини найкоротшого ланцюга, що з'єднує задані вершини. Якщо такого ланцюга(шляха) не існує, тоді відстань дорівнює нескінченності.
Гамільтонів граф	граф, у якому є Гамільтонів цикл.
Гамільтонів ланцюг	простий ланцюг у графі, що містить усі вершини графу рівно по одному разу.
Гамільтонів цикл	простий цикл у графі, що містить усі вершини графу рівно по одному разу.
Грань	область, що обмежена ребрами у плоскому графі. Зовнішня частина площини також утворює грань.

Рис. 2.20. Словник з теорії графів

Перейшовши до третього блоку програми, практичної частина, користувач спочатку повинен пройти авторизацію чи реєстрацію (рис. 2.21), після чого з'являється його особистий кабінет (рис. 2.22).

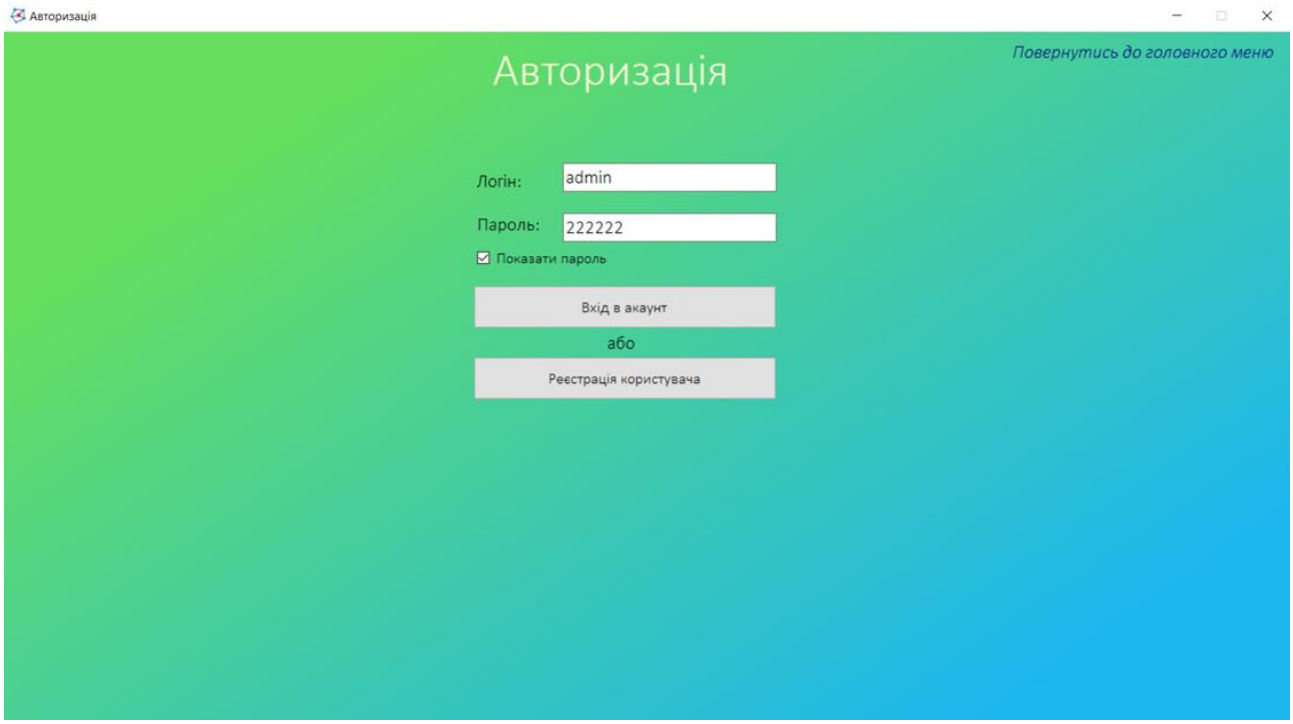


Рис. 2.21. Авторизація користувача

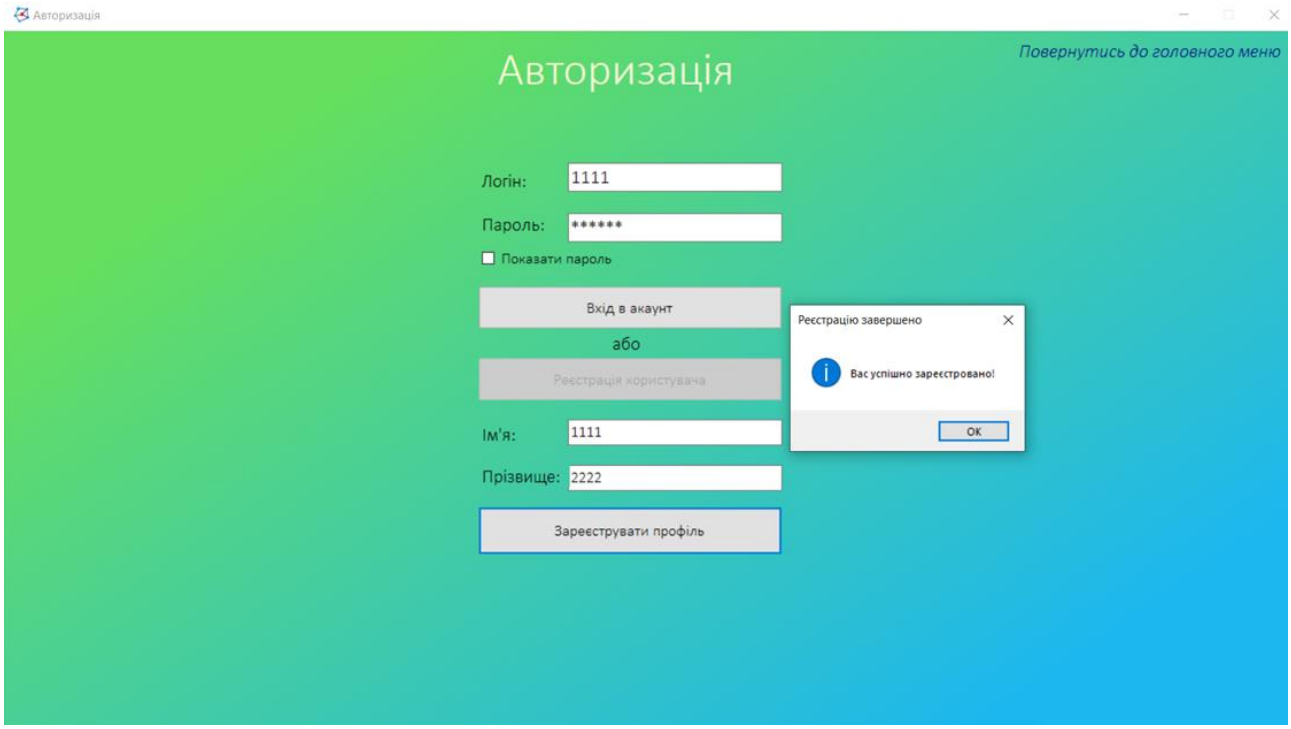


Рис. 2.22. Реєстрація нового користувача

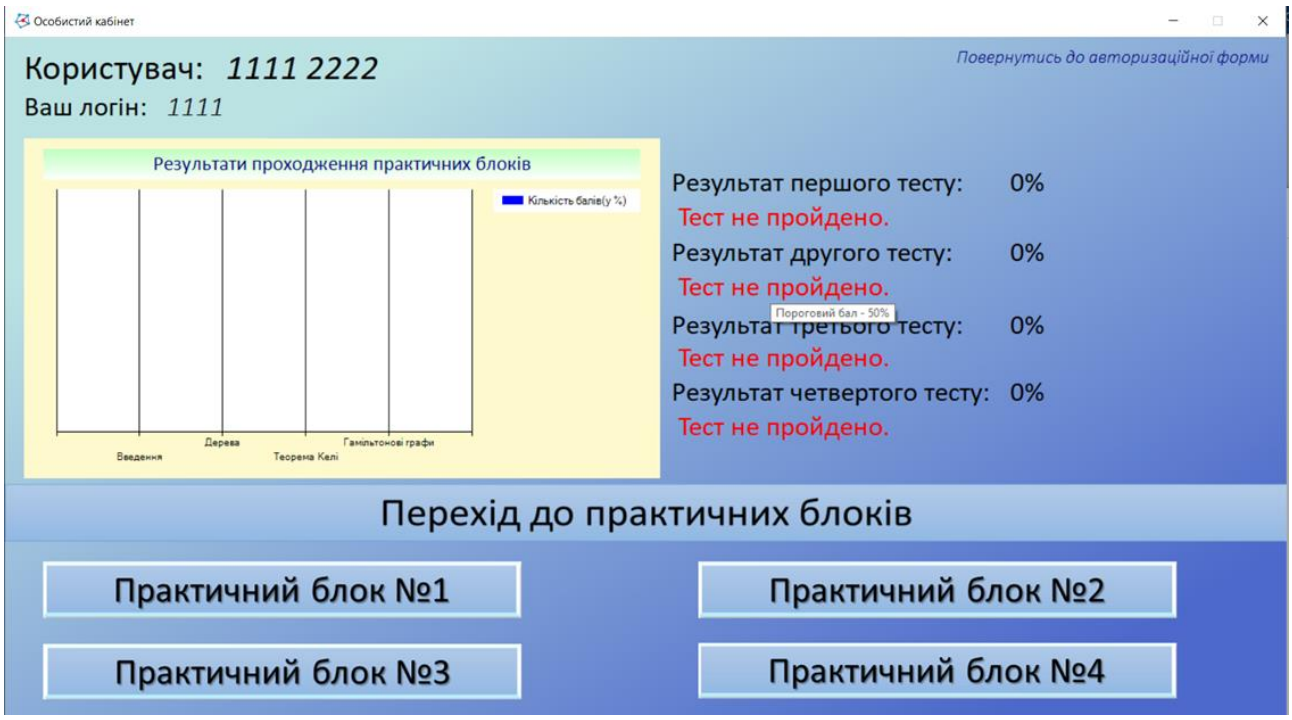


Рис. 2.23. Особистий кабінет щойно зареєстрованого користувача

В особистому кабінеті для кожного учня відображено його власні результати усіх практичних блоків, а також діаграма що показує його рівень знань з кожного блоку.

Саме з особистого кабінету здійснюється перехід до тестів (рис. 2.24).

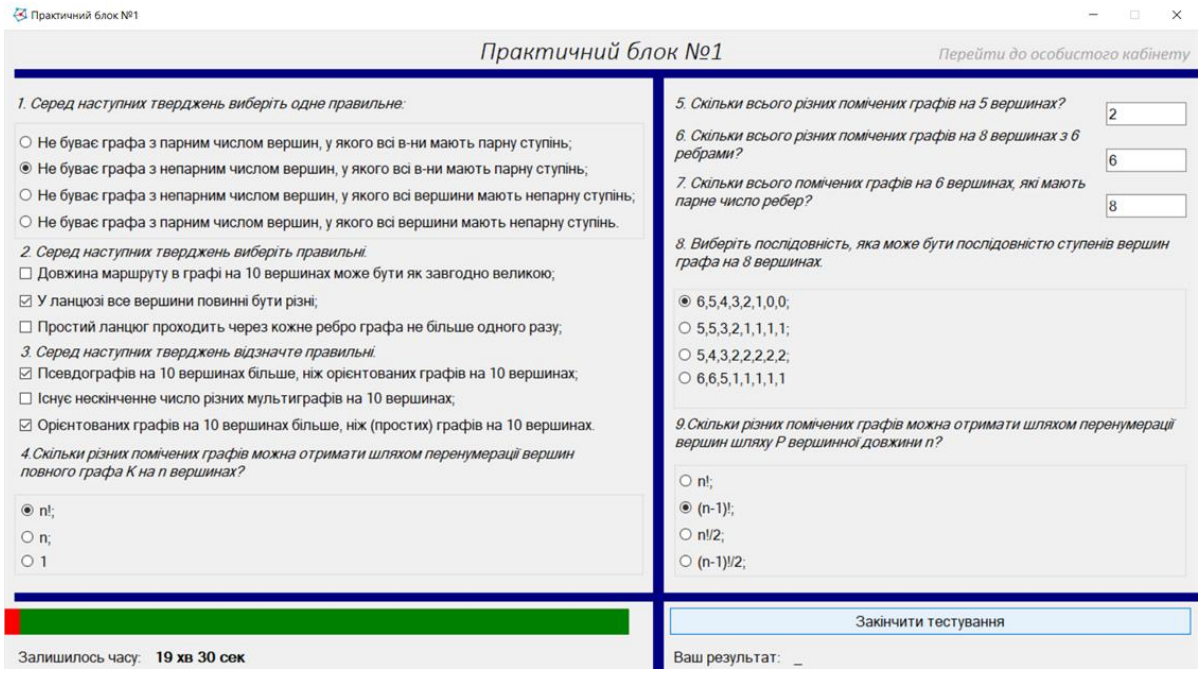


Рис. 2.24. Тестова частина програми

Перейшовши до тесту, в учня є обмежений час (20 хв.) для виконання завдань. Присутні такі типи завдань:

– завдання з вибором однієї правильної відповіді;
– завдання з кількома правильними відповідями;
– завдання з полем для введення відповіді;
– завдання за малюнками.

В кожному блоці міститься від семи до дев’яти завдань. Після виконання усіх завдань, при натисненні кнопки перевірки, одразу можна побачити результат (рис. 2.25). Також відокремлюється завдання, що були розв’язані неправильно або не були розв’язані. Користувач має право перескладати тести.

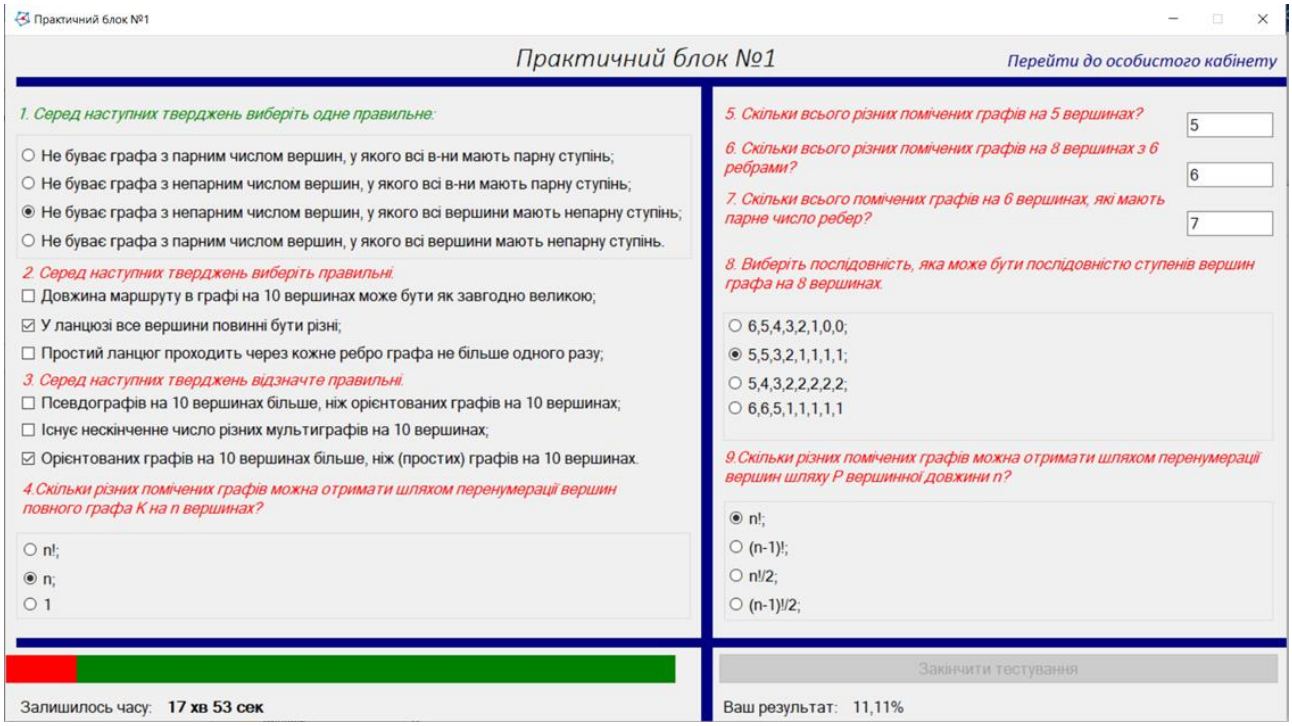


Рис. 2.25. Результат тестування

Результат тестування виситься до особистого кабінету користувача (рис. 2.26).

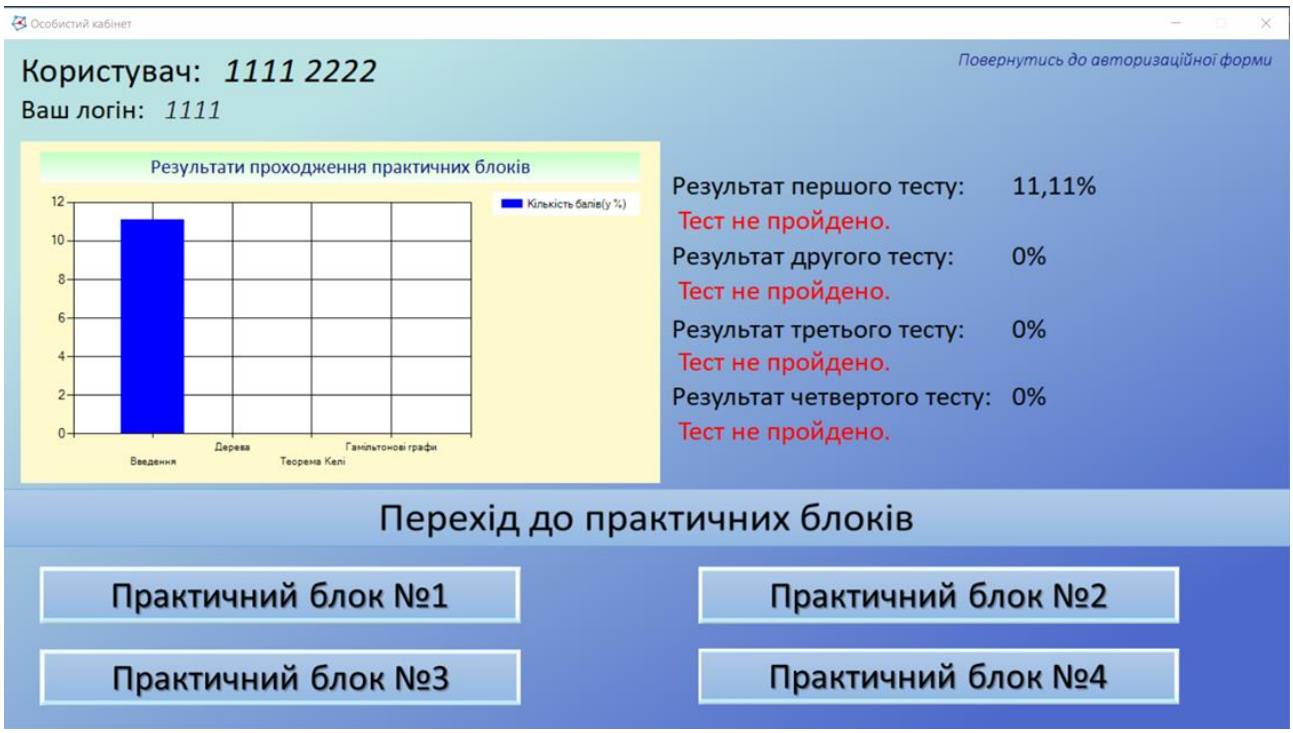


Рис. 2.26. Особистий кабінет користувача після проходження тестування

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

3.1. Розрахунок трудомісткості та вартості розробки програмного продукту

Вихідні дані розробки програмного забезпечення:

- передбачуване число операторів – 860;
- коефіцієнт складності програми – 1,25;
- коефіцієнт кореляції програми в ході її розробки - 0,1;
- середня годинна заробітна плата програміста, грн/год – 40;
- вартість машино-години ЕОМ, грн/год – 7.

Нормування праці в процесі створення ПЗ істотно ускладнено в силу творчого характеру праці програміста. Тому трудомісткість розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t = t_o + t_и + t_a + t_п + t_{отл} + t_д \text{, людино-годин,} \quad (3.1)$$

де t_o – витрати праці на підготовку й опис поставленої задачі (приймається 50),

$t_и$ – витрати праці на дослідження алгоритму рішення задачі,

t_a – витрати праці на розробку блок-схеми алгоритму,

$t_п$ – витрати праці на програмування по готовій блок-схемі,

$t_{отл}$ – витрати праці на налагодження програми на ЕОМ,

$t_д$ – витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у ПЗ, яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q \cdot C(1 + p), \tag{3.2}$$

де q – передбачуване число операторів;

С– коефіцієнт складності програми;

р– коефіцієнт кореляції програми в ході її розробки.

$$Q = 860 \cdot 1{,}25 \cdot (1 + 0{,}1) = 1182;$$

Витрати праці на вивчення опису задачі $t_и$ визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{Q \cdot B}{(75...85)K}, \text{ людино-годин}, \tag{3.3}$$

де В – коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі, В=1.2 ... 1.5;

К – коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності. до 2 – 0,8;

$$t_u = \frac{1182 \cdot 1{,}2}{85 \cdot 0{,}8} = 22, \text{ людино-годин.}$$

Витрати праці на розробку алгоритму рішення задачі:

$$t_a = \frac{Q}{(20...25)K}; \tag{3.4}$$

$$t_a = \frac{1182}{20 \cdot 0{,}8} = 73, \text{ людино-годин.}$$

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20...25)K}; \quad (3.5)$$

$$t_n = \frac{1182}{25 \cdot 0{,}8} = 59, \text{ людино-годин.}$$

Витрати праці на налагодження програми на ЕОМ:

- за умови автономного налагодження одного завдання:

$$t_{отл} = \frac{Q}{(4...5)K}; \quad (3.6)$$

$$t_{отл} = \frac{1182}{4 \cdot 0{,}8} = 369, \text{ людино-годин,}$$

- за умови комплексного налагодження завдання:

$$t_{отл}^{к} = 1{,}2 \cdot t_{отл}; \quad (3.7)$$

$$t_{отл}^{к} = 1{,}2 \cdot 369 = 443, \text{ людино-годин}$$

Витрати праці на підготовку документації:

$$t_д = t_{др} + t_{до}; \quad (3.8)$$

де $t_{др}$ – трудомісткість підготовки матеріалів і рукопису

$$t_{др} = \frac{Q}{(15...20)K}; \quad (3.9)$$

$$t_{др} = \frac{1182}{15 \cdot 0{,}8} = 98, \text{ людино-годин.}$$

$t_{до}$ – трудомісткість редагування, печатки й оформлення документації

$$t_{до} = 0{,}75 \cdot t_{др}; \quad (3.10)$$

$$t_{до} = 0{,}75 \cdot 98 = 73, \text{ людино-годин.}$$

$$t_{д} = 98 + 73 = 172, \text{ людино-годин.}$$

Отримаємо трудомісткість розробки програмного забезпечення:

t = 50 +22 + 73 +59 + 443 +172 = 821, людино-годин.

У результаті ми розрахували, що в загальній складності необхідно 821 людино-годин для розробки даного програмного забезпечення.

3.2. Розрахунок витрат на створення програми

Витрати на створення ПЗ Кпо включають витрати на заробітну плату виконавця програми Зз/п і витрат машинного часу, необхідного на налагодження програми на ЕОМ.

$$К_{ПО} = З_{ЗП} + З_{МВ}, \text{ грн,} \quad (3.11)$$

де $З_{ЗП}$ – заробітна плата виконавців, яка визначається за формулою:

$$З_{ЗП} = t \cdot С_{ПР}, \text{ грн,} \quad (3.12)$$

де t – загальна трудомісткість, людино-годин;

$С_{ПР}$ – середня годинна заробітна плата програміста, грн/година

$$З_{ЗП} = 821 \cdot 40 = 32843, \text{грн.}$$

$З_{МВ}$ – Вартість машинного часу, необхідного для налагодження програми на ЕОМ:

$$З_{МВ} = t_{отл} \cdot C_{М}, \text{ грн,} \quad (3.13)$$

де $t_{отл}$ – трудомісткість налагодження програми на ЕОМ, год.

$C_{Мч}$ – вартість машино-години ЕОМ, грн/год.

$$З_{МВ} = 443 \cdot 7 = 3104, \text{ грн.}$$

$$К_{ПО} = 32843 + 3104 = 35947 \text{ , грн.}$$

Очікуваний період створення ПЗ:

$$T = \frac{t}{B_k \cdot F_p}, \text{ мес.} \quad (3.14)$$

де B_k - число виконавців,

F_p - місячний фонд робочого часу (при 40 годинному робочому тижні Fр=176 годин).

$$T = \frac{821}{1 \cdot 176} = 4{,}6 \text{ міс.}$$

Таким чином, очікувана тривалість розробки складе 4,6 місяця, а витрати на створення програмного забезпечення 35947 грн.

ВИСНОВКИ

Головним завданням роботи було створення програми, яка правильно розфарбовує довільну мапу на площині, створення інтерактивного курсу з теорії графів, у якому подано структуровані теоретичні та практичні блоки, глосарій, а також особистий кабінет користувача, у якому він може побачити досягненні результати проходження тестів після кожного теоретичного блоку.

Вхідними даними програми є власне зображення, яке потрібно намалювати у полі форми програми та зберегти в форматі .png чи .jpeg, або зображення, обране з репозиторію, які подаються на вхід програми та з якими відбуваються маніпуляції. Програма розфарбовує завантажене зображення (мапу) на площині методом чотирьох фарб та виводить його зображення на екран.

У програмі використані технології платформи .Net 4.0 - 4.6 та мова програмування C#.

Створений інтерактивний навчальний додаток призначений для вирішення «проблеми чотирьох фарб» з курсу теорії графів. В ньому у зручній формі подано структуровані теоретичні матеріали з даної теми, а практичні блоки надають можливість на прикладах застосувати отримані відомості. Розроблена програма, яка правильно розфарбовує довільну мапу на площині. Наявність особистого кабінету користувача надає можливість відстежувати досягненні результати проходження тестів після кожного теоретичного блоку.

Аналогів для даної програми немає і вона може використовуватися у багатьох галузях і для багатьох цілей та як навчальний посібник для людей, які бажають ознайомитися з дослідженням проблеми чотирьох фарб та її осмислення.

В економічному розділі проведені обчислення трудомісткості та витрат для розробки програмного забезпечення. Для створення даної інформаційної системи знадобиться 4,6 місяця, трудомісткість розробки ПЗ – 821 людино-годин, а витрати на її створення програмного забезпечення - 35947 грн.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бібліографічний запис. Бібліографічний опис. Загальні вимоги та правила складання : (ГОСТ 7.1-2003, IDT) : ДСТУ ГОСТ 7.1:2006. – Чинний з 2007–07–01. – К. : Держспоживстандарт України, 2007. – 47 с. – (Система стандартів з інформації, бібліотечної та видавничої справи) (Національний стандарт України).

2. Болтянський В.Г., Єфремович В.А. Наочна топологія. - Москва: Наука, 1982 – 160 с.

3. Бусигін Б.С., Коротенко Г.М., Коротенко Л.М. Прикладна інформатика. Підручник для студентів комп'ютерних спеціальностей. – Дніпропетровськ: Видавництво НГУ, 2004. – 559 с. URL: http://www.programmer.dp.ua/book-ua-k01.php. дата звернення: 2.03.2021.

4. Бусыгин Б.С., Дивизинюк М.М., Коротенко Г.М., Коротенко Л.М. Введение в современную информатику. Учебник. – Севастополь: Издательство СНУЯЭиП, 2005. – 644 с. / URL: http://www.programmer.dp.ua/book-ru-k02.php. дата звернення: 15.01.2018.

5. Графы. Красиво, наглядно, интересно URL: http://www.tofmal.ru/projects/graphs/kraski.html дата звернення: 2.03.2021.

6. Донець Г.О., Шор Н.З. Про алгебраїчний підхід до проблеми чотирьох фарб. - Київ: Наукова думка, 1982 – 144 с.

7. Дискретная математика URL: http://pgap.chat.ru/zap/zap253.htm дата звернення: 2.03.2021.

8. Документація. Звіти у сфері науки і техніки. Структура і правила оформлення : ДСТУ 3008-95. – Чинний від 1996–01–01. – К. : Держстандарт України, 1996. – 39 с.

9. ДСТУ 2394-94 Інформація та документація. Комплектування фонду, бібліографічний опис, аналіз документів. Терміни та визначення. – Чинний від 01.01.1995. - Київ : Держстандарт України, 1994. – 88 с.

10. Зиков О.О. Основи теорії графів. - Москва: Вузовська книга, 2000. – с. 367-386

11. Касьянов В.Н., Евстигнеев В. А. Графе в программировании: обработка, визуализация и применение.- С.Пб.:БХВ-Петербург, 2003.-1104с.

12. Методичні вказівки з виконання економічного розділу в дипломних проектах студентів спеціальності “Комп’ютерні системи ” / Укладачі О.Г. Вагонова, Нікітіна О.Б. Н.Н. Романюк – Дніпропетровськ: Національний гірничий університет. – 2013. – 23с.

13. Методичні рекомендації до виконання кваліфікаційних робіт бакалаврів напряму підготовки 122 «Комп’ютерні науки» галузі знань 12 Інформаційні технології/, Л.М. Коротенко , О.С. Шевцова; Нац. гірн. ун-т. – Д : ДВНЗ НГУ, 2019. – 65 с.

14. Новиков Ф.А. Дискретная математика для программистов. - С.Пб. и др.: Питер, 2003.- 301с.

15. Офіційний сайт середовища розробки Visual Studio Code URL: https://code.visualstudio.com/docs дата звернення: 2.03.2021.

16. Проблема чотирьох фарб URL: http://dspace.nuft.edu.ua/jspui/bitstream/123456789/16985/1/371.pdf дата звернення: 2.03.2021.

17. Протасов І.В., Протасова К.Д. Розкладність графів: Навчальний посібник. - Видавничо - поліграфічний центр «Київський університет», 2003. - 73с.

18. Самохін А.В. Проблема чотирьох фарб: незакінчена історія докази – Соросівський освітній журнал. - 1998. - №7 – с. 91-96

19. Скорочення слів в українській мові у бібліографічному описі. Загальні правила та вимоги : ДСТУ 3582-97. – Чинний від 1998–07–01. – К. : Держстандарт України, 1998. – 24 с. – (Державний стандарт України).

20. Теорія графів URL : http://uk.wikipedia.org/wiki/Теорія_графів. дата звернення: 2.03.2021.

ДОДАТОК А

КОД ПРОГРАМИ

FormAuthorization.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO;
namespace Try1
{
   public partial class FormAuthorization : Form
   {
      public static FormMenuPracticeLe PracticeLeMenu = new FormMenuPracticeLe();
      public static string globallogin, globalpass, globalname, globalsurname; public static int globalres1, globalres2, globalres3, globalres4, the_number_of_line;
      public static FormMenu menu_ff = new FormMenu();
      private void CheckBox1_CheckedChanged(object sender, EventArgs e)
      {
         if (checkBox1.Checked == true){
            textBox2.PasswordChar = '\0';
         }
         else
         {
            textBox2.PasswordChar = '*';
         }
      }
      private void Label7_MouseHover(object sender, EventArgs e)
      {
```

```
    label7.ForeColor = Color.DarkGreen;
}
private void Label7_MouseLeave(object sender, EventArgs e)
{
    label7.ForeColor = Color.Navy;

}
// int k = 0;
private void Label7_Click(object sender, EventArgs e)
{
}

private void Label6_Click(object sender, EventArgs e)
{

}

private void FormAuthorization_FormClosed(object sender, FormClosedEventArgs e)
{
    Application.Exit();
}
public FormAuthorization()
{
    InitializeComponent();
    AddOwnedForm(PracticeLeMenu);
}

private void Form3_FormClosing(object sender, FormClosingEventArgs e)
{
    if (e.CloseReason == CloseReason.UserClosing)
    {
        e.Cancel = true;
        Hide();
    }
}
```

```
private void Form3_Load(object sender, EventArgs e)
{
   label4.Visible = false;
   label5.Visible = false;
   textBox3.Visible = false;
   textBox4.Visible = false;
}
private void ButtonCheck_Click(object sender, EventArgs e)
{
   PracticeLeMenu.ShowInTaskbar = false;
   string[] ast = File.ReadAllLines("logpass.bmp");
   int n = ast.Length;
   bool mbs = false;
   bool vhod = false;
   string log_from_box = textBox1.Text, pass_from_box = textBox2.Text;
   for(int i = 0; i < n; i++)
   {
      string[] sl = ast[i].Split('|');
      if ((String.Equals(sl[0], log_from_box)) && String.Equals(sl[1], pass_from_box))
      {
         vhod = true;
         the_number_of_line = i;
         globallogin = sl[0];
         globalpass = sl[1];
         globalres1 = Convert.ToInt16(sl[2]);
         globalres2 = Convert.ToInt16(sl[3]);
         globalres3 = Convert.ToInt16(sl[4]);
         globalres4 = Convert.ToInt16(sl[5]);
         globalname = sl[6];
         globalsurname = sl[7];
         break;
      }
   }
   if (vhod)
   {
```

```
            PracticeLeMenu.StartPosition = FormStartPosition.CenterScreen;
            PracticeLeMenu.Show();
             this.Close();
        }
        else
        {
            MessageBox.Show("Ви ввели неправильний логін або/та пароль", "Увага",
MessageBoxButtons.OK, MessageBoxIcon.Warning);
        }
    }

    private void Button2_Click(object sender, EventArgs e)
    {
        label4.Visible = true;
        label5.Visible = true;
        textBox3.Visible = true;
        textBox4.Visible = true;
        button3.Visible = true;
        button2.Enabled = false;
      //  buttonCheck.Enabled = false;
    }
    private void Button3_Click(object sender, EventArgs e)
    {
        PracticeLeMenu.ShowInTaskbar = false;
        // button3.Enabled = false;
        bool empty = false;
      if (textBox1.Text.Contains(" ") || textBox3.Text.Contains(" ") || textBox4.Text.Contains(" "))
        {
            empty = true;
        }
        string log_signup = textBox1.Text;
        string pass_signup = textBox2.Text;
        bool forb = false;
        string name_signup = textBox3.Text;
        string surname_signup = textBox4.Text;
```

```
        if (pass_signup.Contains("|") || log_signup.Contains("|") || name_signup.Contains("|") || surname_signup.Contains("|"))
        {
            forb = true;
        }
      //  string[] ast = File.ReadAllLines("logpass.txt");

        string[] ast = File.ReadAllLines("logpass.bmp");
        int n = ast.Length;
        bool theaccexists = false;

        for (int i = 0; i < ast.Length; i++)
        {
            string[] sl = ast[i].Split('|');
            if (String.Equals(sl[0], log_signup))
            {
                theaccexists = true;
                break;
            }
        }
        if (theaccexists)
        {
            MessageBox.Show("Користувач з таким логіном вже є у системі. Введіть, будь-ласка, інший логін." ,"Увага", MessageBoxButtons.OK, MessageBoxIcon.Warning);
        }
        else if (forb)
        {
            MessageBox.Show("Символ | заборонено використувати! Будь-ласка, замініть його. ", "Увага", MessageBoxButtons.OK, MessageBoxIcon.Warning);
        }
        else if (empty)
        {
            MessageBox.Show("Одне з полів містить пробіли!Змініть такі поля!", "Увага!", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
```

```
            }
            else
            {
                the_number_of_line = n;
                globallogin = log_signup;
                globalpass = pass_signup;
                globalres1 = 0;
                globalres2 = 0;
                globalres3 = 0;
                globalres4 = 0;
                globalname = name_signup;
                globalsurname = surname_signup;
                //  File.AppendAllText("logpass.txt", Environment.NewLine);
                string write_to_file = log_signup + "|" + pass_signup + "|" + "0" + "|" + "0" + "|" + "0" +
"|" + "0" + "|" + name_signup + "|" + surname_signup;
               // File.CreateText("logpass.bmp");
                //StreamWriter sw5 = new StreamWriter("logpass.bmp");
                //sw5.Close();
                File.AppendAllText("logpass.bmp", write_to_file + Environment.NewLine);
                MessageBox.Show("Вас успішно зареєстровано!", "Реєстрацію завершено",
MessageBoxButtons.OK, MessageBoxIcon.Asterisk);
                PracticeLeMenu.StartPosition = FormStartPosition.CenterScreen;
                PracticeLeMenu.Show();
               this.Close();
                button3.Enabled = false;
            }
        }
    }
}
```

FormColouringMap.cs

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Windows.Forms;
```

```
using System.IO;
namespace Try1
{
   public partial class FormColouringMap : Form
   {
      public FormMenu Menu_form;
      static int[,] Z;
      static int H, W;
      string fileName;
      Bitmap bmp2, myBM;
      Graphics g, g1;
      int xx = -1;
      int yy = -1;
      int k = 0;
      int o;
      bool moving = false, checkButtonClick = false, ch1 = false;
      Pen pen;
      string path = Application.StartupPath + "\\Власні зображення\\";
      public FormColouringMap()
      {
         InitializeComponent();
         g = pictureBox2.CreateGraphics();
         g1 = pictureBox1.CreateGraphics();
         pen = new Pen(Color.Black, 1);
      }

      private void PictureBox2_MouseUp(object sender, MouseEventArgs e)
      {
         moving = false;
         xx = -1;
         yy = -1;
      }

      private void PictureBox2_MouseMove(object sender, MouseEventArgs e)
      {
```

```
    if (moving && xx != -1 && yy != -1)
    {
        g.DrawLine(pen, new Point(xx, yy), e.Location);
        xx = e.X;
        yy = e.Y;
        pictureBox2.Image = bmp2;
    }
}

private void PictureBox2_MouseDown(object sender, MouseEventArgs e)
{
    moving = true;
    xx = e.X;
    yy = e.Y;
}

private void ButtonSaveImage_Click(object sender, EventArgs e)
{
    pictureBox2.DrawToBitmap(bmp2, pictureBox2.ClientRectangle);
 //   l
    bmp2.Save(path + "YourOwnDrawing" + k + ".png");
    checkButtonClick = true;
}

private void FormColouringMap_FormClosed(object sender, FormClosedEventArgs e)
{
    Application.Exit();
}

private void radioButton2_CheckedChanged(object sender, EventArgs e)
{
    groupBox2.Enabled = true;
    buttonSaveImage.Enabled = false;
    button1.Text = "Обрати зображення з репозиторію";
    button1.ForeColor = Color.Blue;
```

```
}
private void radioButton1_CheckedChanged(object sender, EventArgs e)
{
   groupBox2.Enabled = false;
   buttonSaveImage.Enabled = true;
   button1.Text = "Розфарбувати зображення";
   button1.ForeColor = Color.Black;
}

private void Label3_Click(object sender, EventArgs e)
{
   this.Hide();
}

private void Label3_MouseHover(object sender, EventArgs e)
{
   label3.ForeColor = Color.DarkGreen;
}

private void Label3_MouseLeave(object sender, EventArgs e)
{
   label3.ForeColor = Color.Navy;
}

private void radioButton3_CheckedChanged(object sender, EventArgs e)
{
}

private void radioButton4_CheckedChanged(object sender, EventArgs e)
{
}

private void Button3_Click(object sender, EventArgs e)
{
}
```

```
private void Button2_Click(object sender, EventArgs e)
{
    g.Clear(Color.White);
    pictureBox2.Image = bmp2;
    g1.Clear(Color.White);
    k++;
}

private void Form1_Load(object sender, EventArgs e)
{
    toolTip1.SetToolTip(this.label1, "Мається на увазі кількість замкнутих областей");
   bmp2  = new Bitmap(pictureBox2.ClientSize.Width, pictureBox2.ClientSize.Height);
   g = Graphics.FromImage(bmp2);
}

private void Form1_FormClosing(object sender, FormClosingEventArgs e)
{
    /*
    if (e.CloseReason == CloseReason.UserClosing)
    {
        e.Cancel = true;
        Hide();
    }
    */
}

private void button1_Click(object sender, EventArgs e)
{
    string path11;
    if (radioButton3.Checked == true)
    {
        o = 2;
        path11 = Application.StartupPath + "\\Репозиторій\\Карти країн світу";

    }
```

```
else
{
   o = 1;
   path11 = Application.StartupPath + "\\Репозиторій\\Абстрактні зображення";
}
if (checkButtonClick)
{
   checkButtonClick = false;
   myBM = new Bitmap(path + "YourOwnDrawing" + k + ".png");
}
else
{
   openFileDialog1.InitialDirectory =path11;
   openFileDialog1.Filter = "Зображення або карти(.png)|*.png";
   openFileDialog1.Title = "Відкриття зображення";
   openFileDialog1.FileName = "Абстрактне зоображення_6.png";
   openFileDialog1.CheckFileExists = true;
   if (openFileDialog1.ShowDialog() == DialogResult.OK)
   {
      fileName = openFileDialog1.FileName;
      Text = fileName;
   }
      myBM = new Bitmap(fileName);
}
Color[] clrNew = { Color.Yellow, Color.Red, Color.Blue, Color.Green, Color.Gray, Color.Sienna, Color.Salmon, Color.Aqua};
List<Point>[] hrt = new List<Point>[100000]; //1000 ссылок( если области не появились -- ссылки нулевые)
pictureBox1.Image = myBM;
H = myBM.Size.Height; W = myBM.Size.Width;
Color cl = myBM.GetPixel(1, 1);
Z = new int[H + 2, W + 2];
for (int i = 0; i < H; i++)
{
   for (int j = 0; j < W; j++)
```

```
    {
        if (myBM.GetPixel(j, i) !=cl) //поскольку конвертируем карту в матрицу
        {
            Z[i + 1, j + 1] = -1; //black
        }
    }
}

int c = 0;
for (int i = 0; i < H + 2; i++)
{
    for (int j = 0; j < W + 2; j++)
    {
        if (Z[i, j] == 0) // breadth first search
        {
            Queue<Point> q = new Queue<Point>();
            q.Enqueue(new Point(i, j)); // c++ push_back
            Z[i, j] = ++c;
            hrt[c] = new List<Point>();
            while (q.Count > 0)
            {
                Point p = q.Peek(); // reading the head
                hrt[c].Add(p);
                q.Dequeue(); // pop
                int x = p.X, y = p.Y;
                // тут идем по матрице
                if (x > 0 && Z[x - 1, y] == 0) //вверх
                {
                    Z[x - 1, y] = c;
                    q.Enqueue(new Point(x - 1, y));
                }
                if (x < H + 1 && Z[x + 1, y] == 0) //вниз
                {
                    Z[x + 1, y] = c;
                    q.Enqueue(new Point(x + 1, y));
```

```
                }
                if (y > 0 && Z[x, y - 1] == 0) //влево
                {
                    Z[x, y - 1] = c;
                    q.Enqueue(new Point(x, y - 1));
                }
                if (y < W + 1 && Z[x, y + 1] == 0) //вправо
                {
                    Z[x, y + 1] = c;
                    q.Enqueue(new Point(x, y + 1));
                }
            }
        }
    }
}
StreamWriter sw10 = new StreamWriter("Результати етапів роботи/Результат після 2-го етапу.txt");
for (int i = 0; i < H+2; i++)
{
    for (int j = 0; j < W+2; j++)
    {
        sw10.Write(Z[i,j]);
    }
    sw10.WriteLine();
}
sw10.Close();
bool mn = false;
int[,] Graph = new int[c+3, c+3];
for (int i = 1; i <=H; i++)
{
    for (int j = 1; j <=W; j++)
    {
        try
        {
            if (Z[i, j] == -1)
```

```
            {
                int u = Z[i - o, j];
                int v = Z[i + o, j];
                if (u > 1 && v > 1 && u != v)
                {
                    Graph[u - 2, v - 2]++;
                    Graph[v - 2, u - 2]++;
                }
                u = Z[i, j - o];
                v = Z[i, j + o];
                if (u > 1 && v > 1 && u != v)
                {
                    Graph[u - 2, v - 2]++;
                    Graph[v - 2, u - 2]++;
                }
            }
        }
        catch (IndexOutOfRangeException) {
            mn = true;
            break;

        }

    }
}
if (mn)
{
    MessageBox.Show("Не можна малювати поза границь.");
}
  c--;
label2.Text = "= " + c.ToString();
for (int i = 0; i <c; i++)
{
    for (int j = 0; j < c; j++)
    {
```

```
            if (Graph[i, j] > 0) //1
            {
                Graph[i, j] = 1;
            }
        }
    }
    StreamWriter sw = new StreamWriter("Результати етапів роботи/Результат після 3-го етапу.txt");
    sw.WriteLine("c=" + c.ToString());
    for (int i = 0; i <c; i++)
    {
        for (int j = 0; j<c; j++)
        {
            sw.Write((Graph[i, j]).ToString() + "\t");
        }
        sw.WriteLine();
    }
    sw.Close();

    StreamWriter sw1 = new StreamWriter("Результати етапів роботи/Результат після 4-го етапу.txt");

    List<int>[] adj = new List<int>[100000];
    for (int i = 0; i < c; i++)
    {
        adj[i] = new List<int>();
    }
    for (int i = 0; i < c; i++)
    {
        for (int j = 0; j < c; j++)
        {
            if (i < j && Graph[i, j] == 1)
            {
            sw1.Write(((i).ToString() + " " + (j).ToString() + "\t"));
            adj[i].Add(j);
```

```
        adj[j].Add(i);
        }
    }
    sw1.WriteLine();
}
for (int i = 0; i < c; i++)
{
    sw1.Write(i.ToString() + " ");
    foreach(int y in adj[i])
    {
        sw1.Write(y.ToString());
    }
    sw1.WriteLine();
}

sw1.Close();
//the algorithm starts here
int num_vertex = c;
int[] result = new int[num_vertex + 1];
bool[] available = new bool[num_vertex + 1];
int d;
for (int i = 0; i <= num_vertex; i++)
{
      result[i] = -1; // no color assigned
       available[i] = false;
}
result[0] = 0;
for (int i = 1; i < num_vertex; i++)
{
    for (d = 0; d < adj[i].Count; d++)
    {
        int to = adj[i][d];
        if (result[to] != -1)
        {
            available[result[to]] = true;
```

```
                }
            }
            for (d = 0; d < num_vertex; d++)
                if (!available[d])
                    break;
            result[i] = d;
            for (d = 0; d < num_vertex; d++)
                available[d] = false;
        }
        StreamWriter swres = new StreamWriter("Результати етапів роботи/Результат після 5-го етапу.txt");
        for(int i = 0; i < num_vertex; i++)
        {
            swres.WriteLine("Вершина " + i + "Колір ---> " + result[i]);
        }
        for (int i = 2; i < c+2; i++)
        {
            foreach(Point t in hrt[i])
            {
                myBM.SetPixel(t.Y - 1, t.X - 1, clrNew[result[i - 2]]); // меняем местами поскольку конвертируем матрицу в карту
            }
        }
        swres.Close();
    }
  }
}
```

FormDictionary.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
```

```
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO;
using Microsoft.VisualBasic;
namespace Try1
{
   public partial class FormDictionary : Form
   {
      public FormTheoryLesson1 Lesson1;
      public FormDictionary()
      {
         InitializeComponent();
      }
      DataSet ds;
      DataView dv1, dv2;
      public static string nameXMLfile = "Knowledge.xml";
      public static int currentRow = 0;
      public static int currentRow2 = 0;
      bool isChangeSaved = true;

      void LoadXmlFile()
      {
         ds = new DataSet();
         FileStream fsReadXml = new FileStream(nameXMLfile, FileMode.Open);
         ds.ReadXml(fsReadXml, XmlReadMode.InferTypedSchema);
         fsReadXml.Close();
         dv1 = new DataView(ds.Tables[0]);
         dataGridView1.DataSource = dv1;
      }

      private void FormDictionary_Load(object sender, EventArgs e)
      {
         dataGridView1.DataSource = dv1;
         nameXMLfile = "Knowledge.xml";
```

```
            LoadXmlFile();
            dataGridView1.Columns[0].Width = 250;
            dataGridView1.Columns[1].Width = 1070;
            dataGridView1.Columns[0].HeaderText = "Термін";

            dataGridView1.RowsDefaultCellStyle.BackColor = Color.LightGreen;
            dataGridView1.AlternatingRowsDefaultCellStyle.BackColor = Color.Aquamarine;
            dataGridView1.Columns[1].HeaderText = "Визначення";
            dataGridView1.BorderStyle = BorderStyle.Fixed3D;

        }

        private void FormDictionary_FormClosing(object sender, FormClosingEventArgs e)
        {
            if (e.CloseReason == CloseReason.UserClosing)
            {
                e.Cancel = true;
                Hide();
            }
        }

    private void DataGridView1_RowsAdded(object sender, DataGridViewRowsAddedEventArgs e)
        {
            dataGridView1.Rows[e.RowIndex].Height = 50;
        }

        private void ToolStripButton1_Click(object sender, EventArgs e)
        {
            string s = Microsoft.VisualBasic.Interaction.InputBox("Введіть точний термін, для якого бажаєте найти визначення", "Пошук у словнику");
            if (s != "")
            {
                string strSort = dv1.Sort;
                dv1.Sort = "Termin";
                int index = dv1.Find(s);
```

```
        if (index != -1)
        {
            dataGridView1.ClearSelection();
            dataGridView1.Rows[index].Selected = true;
        }
        else
        {
            dv1.Sort = strSort;
            MessageBox.Show("Такого визначення немає у словнику!", "Пошук у словнику");
        }

    }
}

private void Button1_Click(object sender, EventArgs e)
{
    if (radioButton1.Checked)
    {
        if (textBox1.Text != "") dv1.RowFilter = "Termin Like '%" + textBox1.Text + "%'";
    }
    if (radioButton2.Checked)
    {
        if (textBox1.Text != "") dv1.RowFilter = "Definition Like '%" + textBox1.Text + "%'";
    }
    int tt = dataGridView1.RowCount;
    if (tt == 0)
    {
        MessageBox.Show("Нічого не знайдено!");
    }
}

private void Button2_Click(object sender, EventArgs e)
{
    LoadXmlFile();
}
```

```
        private void FormDictionary_FormClosed(object sender, FormClosedEventArgs e)
        {
            Application.Exit();
        }

        private void Label3_MouseHover(object sender, EventArgs e)
        {
            label3.ForeColor = Color.DarkGreen;
        }

        private void Label3_MouseLeave(object sender, EventArgs e)
        {
            label3.ForeColor = Color.Navy;
        }

        private void Label3_Click(object sender, EventArgs e)
        {
            this.Hide();
        }

        private void DataGridView1_CellContentClick(object sender, DataGridViewCellEventArgs e)
        {

        }
    }
}
```

FormLessonsTheory.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
```

```
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Try1
{
    public partial class FormLessonsTheory : Form
    {
        public FormTheoryLesson1 Lesson1 = new FormTheoryLesson1();
        public FormMenu Menu_form;
        public FormLessonsTheory()
        {
            InitializeComponent();
            AddOwnedForm(Lesson1);
        }

        private void Form2_FormClosing(object sender, FormClosingEventArgs e)
        {
            if (e.CloseReason == CloseReason.UserClosing)
            {
                e.Cancel = true;
                Hide();
            }
        }

        private void PBLesson1_MouseHover(object sender, EventArgs e)
        {
            label1.Visible = true;
            label1.Text = "У першому блоці курсу ми познайомимося з поняттям графа, навчимося відрізняти граф від його зображення, поговоримо про різні види графів. Ми згадаємо, з чого почалася теорія графів, навчимося представляти у вигляді графа структуру Інтернету. Ми обговоримо такі важливі поняття, як маршрути в графах, ступінь вершини, зв'язність, а також почнемо говорити про важливий клас графів - дерева.";
        }
```

```
private void PBLesson1_MouseLeave(object sender, EventArgs e)
{
   label1.Visible = false;
}

private void PBLesson2_MouseHover(object sender, EventArgs e)
{
   label1.Visible = true;
   label1.Text = "У цьому блоці ми навчимося визначати дерева чотирма різними способами, і поговоримо про те, як правильно розфарбовувати географічні карти. Ми згадаємо відому теорему про чотири фарби, а також критерій Куратовського про те, як визначити, чи можна намалювати даний граф на площині без перетину ребер.";
}
private void PBLesson2_MouseLeave(object sender, EventArgs e)
{
   label1.Visible = false;

}

private void PBLesson3_MouseHover(object sender, EventArgs e)
{
   label1.Text = "Пройшовши передостанній блок, ми зможемо перерахувати всі дерева. Для цього нам буде потрібно перейняти досвід наших предків за підрахунком баранів (або козлів). Не зупинившись на цьому, перерахуємо і всі ліси. Потім ми повернемося до задачі про Кенігсберські мости і отримаємо повне вирішення цього питання.";
   label1.Visible = true;
}

private void PBLesson3_MouseLeave(object sender, EventArgs e)
{
   label1.Visible = false;
}

private void PBLesson4_MouseHover(object sender, EventArgs e)
{
```

label1.Text = "В останньому блоці ми закінчимо обговорювати цикли, що проходять через весь граф. На цей раз ми поговоримо про цикли, що проходять через всі вершини графа. На відміну від ейлерових циклів, тут немає необхідного і достатнього критерію наявності такого циклу. Є тільки достатні умови, і ми обговоримо два таких умови, досить різних за своєю природою.";

```
    label1.Visible = true;
}

private void PBLesson4_MouseLeave(object sender, EventArgs e)
{
    label1.Visible = false;
}

private void Label2_Click(object sender, EventArgs e)
{
    this.Hide();
}

private void Label2_MouseHover(object sender, EventArgs e)
{
    label2.ForeColor = Color.DarkGreen;
}

private void Label2_MouseLeave(object sender, EventArgs e)
{
    label2.ForeColor = Color.Navy;
}

private void PBLesson1_MouseClick(object sender, MouseEventArgs e)
{
    this.Hide();
    Lesson1.ShowInTaskbar = false;
    Lesson1.StartPosition = FormStartPosition.CenterScreen;
    Lesson1.Text = "Учбовий блок 1";
    Lesson1.Show();
```

```
    Lesson1.treeView1.BringToFront();
}

private void FormLessonsTheory_Load(object sender, EventArgs e)
{
}

private void PBLesson2_Click(object sender, EventArgs e)
{
    this.Hide();
    Lesson1.ShowInTaskbar = false;
    Lesson1.StartPosition = FormStartPosition.CenterScreen;
    Lesson1.Text = "Учбовий блок 2";
    Lesson1.Show();
    Lesson1.treeView2.BringToFront();
}

private void PBLesson3_Click(object sender, EventArgs e)
{
    this.Hide();
    Lesson1.ShowInTaskbar = false;
    Lesson1.StartPosition = FormStartPosition.CenterScreen;
    Lesson1.Text = "Учбовий блок 3";
    Lesson1.Show();
    Lesson1.treeView3.BringToFront();
}

private void PBLesson4_Click(object sender, EventArgs e)
{
    this.Hide();
    Lesson1.ShowInTaskbar = false;
    Lesson1.StartPosition = FormStartPosition.CenterScreen;
    Lesson1.Text = "Учбовий блок 4";
    Lesson1.Show();
    Lesson1.treeView4.BringToFront();
```

```
        }

        private void FormLessonsTheory_FormClosed(object sender, FormClosedEventArgs e)
        {
            Application.Exit();
        }

        private void PBLesson1_Click(object sender, EventArgs e)
        {

        }
    }
}
```

FormMain.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
namespace Try1
{
    public partial class FormMain : Form
    {
        public static FormMenu Menu_form = new FormMenu();

        public FormMain()
        {
            InitializeComponent();
            AddOwnedForm(Menu_form);
```

```
        }

        private void FormMain_Load(object sender, EventArgs e)
        {
            this.Cursor = Cursors.Arrow;
        }

        private void pictureBox1_Click(object sender, EventArgs e)
        {
            Menu_form.StartPosition = FormStartPosition.CenterScreen;
            Menu_form.Show();
            this.Hide();
        }

        private void FormMain_MouseHover(object sender, EventArgs e)
        {
            this.Cursor = Cursors.Hand;
        }

        private void pictureBox1_MouseLeave(object sender, EventArgs e)
        {
            pictureBox1.Cursor = Cursors.Arrow;
        }

        private void PictureBox1_MouseHover(object sender, EventArgs e)
        {
            pictureBox1.Cursor = Cursors.Hand;
        }
    }
}
```

FormMenu.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
```

```
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Try1
{
    public partial class FormMenu : Form
    {
        public FormColouringMap One_Form = new FormColouringMap();
        public FormLessonsTheory Two_Form = new FormLessonsTheory();
        public FormAuthorization Three_Form = new FormAuthorization();
        public FormMain Main_form;
        public FormMenu()
        {
            InitializeComponent();
            AddOwnedForm(Main_form);
            AddOwnedForm(One_Form);
            AddOwnedForm(Two_Form);
            AddOwnedForm(Three_Form);
        }

        private void FormMenu_Load(object sender, EventArgs e)
        {
        }

        private void FormMenu_FormClosing(object sender, FormClosingEventArgs e)
        {
            Application.Exit();
        }

        private void PictureBox1_Click(object sender, EventArgs e)
        {
```

```
            One_Form.ShowInTaskbar = false;
            One_Form.StartPosition = FormStartPosition.CenterScreen;
            One_Form.Show();
        }

        private void PictureBox2_Click(object sender, EventArgs e)
        {
            Two_Form.ShowInTaskbar = false;
            Two_Form.StartPosition = FormStartPosition.CenterScreen;
            Two_Form.Show();
        }

        private void PictureBox3_Click(object sender, EventArgs e)
        {
            Three_Form.ShowInTaskbar = false;
            Three_Form.StartPosition = FormStartPosition.CenterScreen;
            Three_Form.Show();
        }
    }
}
```

FormMenuPracticeLe.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO;
namespace Try1
{
    public partial class FormMenuPracticeLe : Form
```

```
{
    public FormPracticeLesson1 less1_practice   = new FormPracticeLesson1();
    public FormPracticeLesson2 less2_practice2  = new FormPracticeLesson2();
    public FormPracticeLesson3 less3_practice   = new FormPracticeLesson3();
    public FormPracticeLesson4 less4_practice = new FormPracticeLesson4();
    public static FormMenu menu_f = new FormMenu(); //
    public static FormAuthorization Authform = new FormAuthorization();
    public FormMenuPracticeLe()
    {
        InitializeComponent();
    }
    private void FormMenuPracticeLe_Load(object sender, EventArgs e)
    {
        toolTip1.SetToolTip(this.pictureBox1, "Перейти до тестуваня");
        toolTip1.SetToolTip(this.pictureBox3, "Перейти до тестуваня");
        toolTip1.SetToolTip(this.pictureBox4, "Перейти до тестуваня");
        toolTip1.SetToolTip(this.pictureBox5, "Перейти до тестуваня");
        toolTip1.SetToolTip(this.label5, "Пороговий бал - 50%");
        toolTip1.SetToolTip(this.label6, "Пороговий бал - 50%");
        toolTip1.SetToolTip(this.label7, "Пороговий бал - 50%");
        toolTip1.SetToolTip(this.label8, "Пороговий бал - 50%");
        //   pictureBox1.BringToFront();
        string[] ast = File.ReadAllLines("logpass.bmp");
            int n = ast.Length;
            string loginn = null, namee = null, surnamee = null;
            int ress1, ress2, ress3, ress4;
            string[] sl = ast[FormAuthorization.the_number_of_line].Split('|');
            loginn = sl[0];
            namee = sl[6];
            surnamee = sl[7];
            ress1 = Convert.ToInt16(sl[2]);
            ress2 = Convert.ToInt16(sl[3]);
            ress3 = Convert.ToInt16(sl[4]);
            ress4 = Convert.ToInt16(sl[5]);
        labelName.Text = (namee + " " + surnamee).ToString();
```

```
label12.Text = loginn.ToString();
const int nt = 4;
string[] tests = { "Введення", "Дерева", "Теорема Келі", "Гамільтонові графи" };
double[] re = { ress1, ress2, ress3, ress4 };
chart1.Series[0].Points.Clear();
chart1.ChartAreas[0].AxisX.Interval = 1;

for (int i = 0; i < nt; i++)
{
   chart1.Series[0].Color = Color.Blue;
   chart1.Series[0].Points.AddXY(tests[i], re[i]);
}
label1.Text = ress1.ToString() + "/" + "9";
label2.Text = ress2.ToString() + "/" + "9";
label3.Text = ress3.ToString() + "/" + "9";
label4.Text = ress4.ToString() + "/" + "9";
if (ress1 < 5)
{
   label5.Text = "Тест не пройдено.";
   label5.ForeColor = Color.Red;
}
else
{
   label5.Text = "Тест успішно пройдено.";
   label5.ForeColor = Color.Green;
}
if (ress2 < 5)
{
   label6.Text = "Тест не пройдено.";
   label6.ForeColor = Color.Red;
}
else
{
   label6.Text = "Тест успішно пройдено.";
   label6.ForeColor = Color.Green;
```

```
    }
    if (ress3 < 5)
    {
        label7.Text = "Тест не пройдено.";
        label7.ForeColor = Color.Red;
    }
    else
    {
        label7.Text = "Тест успішно пройдено.";
        label7.ForeColor = Color.Green;
    }
    if (ress4 < 5)
    {
        label8.Text = "Тест не пройдено.";
        label8.ForeColor = Color.Red;
    }
    else
    {
        label8.Text = "Тест успішно пройдено.";
        label8.ForeColor = Color.Green;
    }
}

private void FormMenuPracticeLe_FormClosing(object sender, FormClosingEventArgs e)
{

    if (e.CloseReason == CloseReason.UserClosing)
    {
        e.Cancel = true;
        Hide();
    }
}

private void PictureBox1_Click(object sender, EventArgs e)
{
```

```
    //less1_practice = new FormPracticeLesson1();
    less1_practice.ShowInTaskbar = false;

    less1_practice.StartPosition = FormStartPosition.CenterScreen;
    less1_practice.Show();
    this.Hide();
}

private void FormMenuPracticeLe_Activated(object sender, EventArgs e)
{
    string[] ast = File.ReadAllLines("logpass.bmp");
    int n = ast.Length;
    string loginn = null, namee = null, surnamee = null;
    int ress1, ress2, ress3, ress4;
    string[] sl = ast[FormAuthorization.the_number_of_line].Split('|');
    loginn = sl[0];
    namee = sl[6];
    surnamee = sl[7];
    ress1 = Convert.ToInt16(sl[2]);
    ress2 = Convert.ToInt16(sl[3]);
    ress3 = Convert.ToInt16(sl[4]);
    ress4 = Convert.ToInt16(sl[5]);
    labelName.Text = (namee + " " + surnamee).ToString();
    label12.Text = loginn.ToString();
    const int nt = 4;
    string[] tests = { "Введення", "Дерева", "Теорема Келі", "Гамільтонові графи" };
    double percent1 = Math.Round(Convert.ToDouble(((ress1 / 9.0) * 100)), 2);
    double percent2 = Math.Round(Convert.ToDouble(((ress2 / 9.0) * 100)), 2);
    double percent3 = Math.Round(Convert.ToDouble(((ress3 / 7.0) * 100)), 2);
    double percent4 = Math.Round(Convert.ToDouble(((ress4 / 8.0) * 100)), 2);
    double[] re = { percent1, percent2, percent3, percent4 };
    chart1.Series[0].Points.Clear();
    chart1.ChartAreas[0].AxisX.Interval = 1;
    for (int i = 0; i < nt; i++)
    {
```

```
    chart1.Series[0].Color = Color.Blue;
    chart1.Series[0].Points.AddXY(tests[i], re[i]);
}
        label1.Text = percent1.ToString() + "%";
label2.Text = percent2.ToString() + "%";
label3.Text = percent3.ToString() + "%";
label4.Text = percent4.ToString() + "%";
if (ress1 < 5)
{
    label5.Text = "Тест не пройдено.";
    label5.ForeColor = Color.Red;
}
else
{
    label5.Text = "Тест успішно пройдено.";
    label5.ForeColor = Color.Green;
}
if (ress2 < 5)
{
    label6.Text = "Тест не пройдено.";
    label6.ForeColor = Color.Red;
}
else
{
    label6.Text = "Тест успішно пройдено.";
    label6.ForeColor = Color.Green;
}
if (ress3 < 5)
{
    label7.Text = "Тест не пройдено.";
    label7.ForeColor = Color.Red;
}
else
{
    label7.Text = "Тест успішно пройдено.";
```

```
        label7.ForeColor = Color.Green;
    }
    if (ress4 < 5)
    {
        label8.Text = "Тест не пройдено.";
        label8.ForeColor = Color.Red;
    }
    else
    {
        label8.Text = "Тест успішно пройдено.";
        label8.ForeColor = Color.Green;
    }
}

private void Chart1_Click(object sender, EventArgs e)
{
}
private void pictureBox3_Click(object sender, EventArgs e)
{
    less2_practice2.ShowInTaskbar = false;
    less2_practice2.StartPosition = FormStartPosition.CenterScreen;
    less2_practice2.Show();
    this.Hide();
}
private void pictureBox4_Click(object sender, EventArgs e)
{
    less3_practice.ShowInTaskbar = false;
    less3_practice.StartPosition = FormStartPosition.CenterScreen;
    less3_practice.Show();
    this.Hide();
}

private void pictureBox5_Click(object sender, EventArgs e)
{
    less4_practice.ShowInTaskbar = false;
```

```
        less4_practice.StartPosition = FormStartPosition.CenterScreen;
        less4_practice.Show();
        this.Hide();
    }
    private void label9_Click(object sender, EventArgs e)
    {
      //  menu_f.ShowInTaskbar = false;
      //  menu_f.Show();
        this.Hide();
        Authform.Show();
        Authform.ShowInTaskbar = false;
    }
    private void FormMenuPracticeLe_FormClosed(object sender, FormClosedEventArgs e)
    {
        Application.Exit();
    }
    private void FormMenuPracticeLe_MouseHover(object sender, EventArgs e)
    {
    }
    private void Label9_MouseHover(object sender, EventArgs e)
    {
        label9.ForeColor = Color.DarkGreen;
    }
    pivate void Label9_MouseLeave(object sender, EventArgs e)
    {
        label9.ForeColor = Color.Navy;
    }
   private void Label1_Click(object sender, EventArgs e)
    {
    }

    private void ToolTip1_Popup(object sender, PopupEventArgs e)
    {
    }
}
```

```
}
```

Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;
namespace Try1
{
   static class Program
   {
      /// <summary>
      /// The main entry point for the application.
      /// </summary>
      [STAThread]
      static void Main()
      {
         Application.EnableVisualStyles();
         Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new FormMain());
        //  Application.Run(new FormColouringMap());
         // Application.Run(new FormPracticeLesson2());
         // Application.Run(new FormAuthorization());
          // Application.Run(new FormDictionary());
          //  Application.Run(new FormPracticeLesson1());
        //  Application.Run(new FormAuthorization());

      }
   }
}
```

ДОДАТОК Б

ВІДГУК КЕРІВНИКА ЕКОНОМІЧНОГО РОЗДІЛУ

ДОДАТОК В

ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ

Ім’я файлу	**Опис**
Пояснювальні документи	
Диплом_ doc	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Диплом_.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF
Програма	
Program.rar	Архів. Містить коди програми і откомпільовану програму
Презентація	
Презентація_.ppt	Презентація кваліфікаційної роботи