МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних наук,
управління та адміністрування
Кафедра інформаційних технологій

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: Розробка віртуального навчального середовища для вивчення теорії графів на мобільній платформі iOS

Виконав студент 2 курсу групи МІС-20з
спеціальності 122 Комп'ютерні науки
Різниченко Олексій Миколайович

Керівник к.т.н., доцент
Терещенко Тетяна Михайлівна

Рецензент д.т.н., проф.
Мещеряков Володимир Іванович

Одеса 2021

АНОТАЦІЯ

Кваліфікаційна магістерська робота: 71 с., 28 рис., 9 табл.,11 джерел, 1 додаток.

Ключові слова: мобільна програма, навчальна програма, графи, контроль знань.

Мета дослідження – розробка мобільної програми для вивчення та контролю результатів навчання теорії графів.

Об'єкт дослідження – навчання теорії графів.

Предмет дослідження – система навчання теорії графів.

Задачі дослідження: провести аналіз і дослідження існуючих систем навчання теорії графів; розробити концептуальну та логічну моделі проектованої; розробити модулі прототипу даної системи у відповідності до обраних методів і технологій.

Результати, їх новизна, теоретичне та практичне значення: в ході виконання роботи було розроблено мобільний додаток для вивчення та теорії графів, розглянуті існуючі на час розробки аналоги проектованої системи, визначені їхні переваги та недоліки, проведено аналіз потреб користувачів та визначені функції, які необхідно реалізувати, описані функції та інші вимоги до проектованої системи, розроблено концептуальне та логічне представлення проектованої системи, та розроблено необхідні компоненти відповідно до специфікацій описаних у роботі, також було проведено тестування розробленої системи.

Подальші перспективи розвитку програмної системи пов’язані з оптимізацією рендерінгу графів, розробкою нових функцій як то: розширення списку алгоритмів які підтримує система, а також теоретичних матеріалів з теорії графів.

SUMMARY

Qualifying master's thesis: 71 pages, 28 figures, 9 tables, 11 sources, 1 appendix.

Key words: mobile program, curriculum, graphs, knowledge control.

The purpose of the study is to develop a mobile program for learning and monitoring the results of graph theory.

The object of study - graph theory learning.

The subject of research is the system of learning graph theory.

Research objectives: to analyze and study existing systems of graph theory; develop a conceptual and logical model of the projected; develop prototype modules of this system in accordance with the selected methods and technologies.

Results, their novelty, theoretical and practical significance: developed a mobile program for learning and testing the results of graph theory learning, considered existing at the time of development analogues of the designed system, identified their advantages and disadvantages, analyzed user needs and identified functions to be implemented, described functions and other requirements of the designed system, conceptual and logical representation of the designed system is developed, and necessary components are developed according to the specifications described in work, testing of the developed system was also carried out.

Further prospects for software development are related to optimizing graph rendering, developing new features such as expanding the list of algorithms supported by the system, as well as theoretical materials on graph theory.

ЗМІСТ

ВСТУП

Даний дипломний проект описує розробку програмного продукту, створеного для підвищення результативності вивчення теорії графів.

Актуальність цієї роботи полягає у тому, що така фундаментальна наука, як теорія графів, може здаватися складною на початку її вивчення.
Беручи до уваги важливість даної науки, варто приділити достатньо уваги для полегшення сприйняття даної теми.

Для цього можуть бути використані наступні підходи: тестування теоретичних знань, візуалізація алгоритмів на графах, а також перевірка правильності виконання алгоритму користувачем.

Так як аналіз існуючих на момент написання цієї роботи додатків для роботи з графами показав, що вони не створювались для вирішення вищевказаних задач, дані додатки не задовольняють потребам студентів.

Метою даної роботи є підвищення результативності навчання теорії графів, порівняно із традиційними методами навчання, тому для досягнення даної задачі буде проведено аналіз предметної області, визначено вимоги до проектованої системи, та проведено їх специфікацію для того, щоб розроблювана система відповідала наведеним вимогам та вирішувала поставлені перед нею задачі.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИМОГ ДО ПРОЕКТОВАНОЇ СИСТЕМИ

1.1 Вступ до теорії графів

Поняття графу

Граф - це абстрактне представлення об'єктів та зв’язків між ними. У загальному сенсі граф - це множина точок (вершин, вузлів), які з'єднуються множиною ліній (ребер, дуг)[6].

Теорія графів - розділ дискретної математики, що вивчає графи. В наш час теорія графів знаходить застосування у найрізноманітніших галузях науки, техніки та практичної діяльності. Вона використовується при проектуванні електричних мереж, плануванні транспортних перевезень, побудові молекулярних схем. Застосовується теорія графів й у економіці, психології, соціології, біології.

Об'єкти представляються як вершини, або вузли графа, а зв'язки як дуги, або ребра. Для різних областей застосування види графів можуть відрізнятися спрямованістю, обмеженнями на кількість зв’язків та додатковими даними про вершини або ребра.

Властивості графів

Графом $G = (V, E)$ є об’єкт, що заданий парою множин (V, E), де V – множина вершин, $E \subseteq V \times V$ – множина ребер. Множину вершин графу G позначають $V(G)$, а множину ребер – $E(G)$. Кількість вершин графу $n(G) = |V(G)|$, а кількість ребер $m(G) = |E(G)|$. Граф називається скінченним, якщо множини його вершин і ребер є скінченними. Кількість вершин $n(G)$ графу називають його порядком.

Якщо для деякого ребра $e = (v, w) \in E(G)$, то кажуть, що вершини v та w суміжні, вершини v та w інцидентні ребру e, ребро e інцидентне вершинам v і w.

Неорієнтований граф – це граф, для кожного ребра якого несуттєвий порядок двох його кінцевих вершин. Орієнтований граф – це граф, для кожного ребра якого істотний порядок двох його кінцевих вершин. Ребра орієнтованого графа називають дугами. Змішаний граф – це граф, що містить як орієнтовані, так і неорієнтовані ребра.

Кожен з перерахованих видів графа може містити одне або кілька ребер, у яких обидва кінці сходяться в одній вершині, такі ребра називаються петлями.

Різновиди графів

Графи поділяються на орієнтовані та неорієнтовані. У орієнтованих графах зв'язки є направленими, тобто пари E впорядковані. Для таких графів пари (a, b) та (b, a) - це два різні зв’язки). У свою чергу в неорієнтованому графі, зв'язки не є спрямованими, і тому якщо існує зв'язок (a, b), то є зв’язок (b, a).

Також існують зважені графи, у яких кожному ребру присвоєно число (вага).

Такі ваги можуть представляти якусь величину, залежно від контексту. Зв’язний граф, що не має циклів називається деревом.

Огляд деяких алгоритмів на графах

Пошук у глибину

Пошук у глибину (Depth-first search, DFS) -це один з способів обходу графу. Суть алгоритму полягає у тому, щоб йти «вглиб» графа, наскільки це можливо.

Алгоритм пошуку описується рекурсивно: перебираються всі ребра вихідні з поточної вершини.

У випадку коли ребро веде у вершину, яка вже була розглянута, то алгоритм виконується від даної нерозглянутої вершини.

Потім алгоритм повертається назад і продовжує перебирати ребра. Завершення роботи алгоритму відбувається тоді, коли у поточній вершині не залишилося ребер, що ведуть до нерозглянутої вершини.

Пошук у ширину

Пошук у ширину (Breadth-first search, BFS) – ще один із методів обходу графа. Нехай заданий граф G=(V,E), та виділена вихідна вершина s.

Алгоритм пошуку в ширину обходить всі ребра G для «відкриття» всіх вершин, що досяжні s, обчислюючи при цьому відстань (мінімальна кількість ребер) від s до кожної досяжної s вершини. Алгоритм працює як для орієнтованих, так і для неорієнтованих графів.

Даний алгоритм має таку назву тому, що в процесі обходу рух відбувається вшир, тобто перед тим як почати пошук вершин на відстані k+1, виконується обхід вершин на відстані k[8].

Пошук найкоротшого шляху

У даній роботі буде розглянуто хвильовий алгоритм пошуку найкоротшого шляху, який полягає у наступному: у графі позначено вершину старту і вершину фінішу.

Мета алгоритму – прокласти найкоротший шлях від стартової вершини до фінішу, якщо це, звичайно, можливо.

Від старту на всі напрямки поширюється «хвиля», причому кожна пройдена «хвилею» вершина позначається як пройдена.

«Хвиля», своєю чергою, неспроможна проходити через вершини, помічені як пройдені.

«Хвиля» рухається, доки не досягне точки фінішу, чи доки не залишиться непройдених клітин. Якщо «хвиля» пройшла всі доступні клітини, але так і не досягла клітини фінішу, то шлях від старту до фінішу прокласти неможливо.

Після досягнення «хвилею» фінішу, прокладається зворотній шлях до старту, який і є результатом роботи алгоритму.

Пошук компонент зв’язності

Для пошуку компонентів зв'язності використовується алгоритм пошуку (у глибину, або у ширину).

При запуску обходу з однієї вершини, він гарантовано відвідає всі вершини, яких можна дістатися, тобто, всю компоненту зв'язності, до якої належить початкова вершина.

Для знаходження всіх компонентів необхідно запустити обхід з кожної вершини по черзі, якщо вона ще не була пройдена раніше.

1.2 Визначення бізнес–вимог

Призначенням системи є створення навчального середовища для роботи з графами: а саме з деревами та незваженими неорієнтованими графами для мобільної платформи iOS.

Під цим навчальним додатком для роботи з графами розуміється такий додаток, що дозволяє створити на екрані довільне дерево, або, граф, призначити деяку назву, колір будь-якій вершині, змінити надану назву, або колір, згорнути, розгорнути або видалити вузол дерева разом з його дочірніми вузлами, рисувати та видаляти дуги графу, читати теоретичну або довідкову інформацію.

Крім цього, необхідно реалізувати перемальовування дерева відповідно до обраного відображення, наприклад класичне відображення дерева та відображення визначене користувачем при створенні дерева.

Також треба додати можливість «згортати», тобто ховати дочірні вузли дерева для зручності використання.

Також в даній програмі треба забезпечити візуалізацію обходу довільного графу, чи дерева в глибину та в ширину, і крім цього реалізувати алгоритм пошуку найкоротшого шляху.

Під візуалізацією у контексті даної системи мається на увазі одночасна чи послідовна зміна напису на кожній з вершин дерева чи графу на порядковий номер даної вершини при обході.

Крім того, до вищевказаного необхідно розробити пошук компонент зв'язності графа, реалізувати перевірку обходу дерева або графа користувачем, пошук найкоротшого шляху та проходження тестових завдань із теорії графів.

Наостанок необхідно реалізувати зберігання створених графів у пам'яті пристрою, а також додати можливість масштабувати дерево чи граф на екрані пристрою.

Визначення об'єкту автоматизації

Відповідно до наведених вище вимог, об'єктом автоматизації у контексті даної роботи є процес вивчення теорії графів та деяких алгоритмів на графах.

Суб'єктом автоматизації для проектованої системи є студенти, які вивчають графи.

Для візуалізації вищеописаних тверджень було створено ментальну карту об'єкта автоматизації, яку зображено на рисунку 1.1.

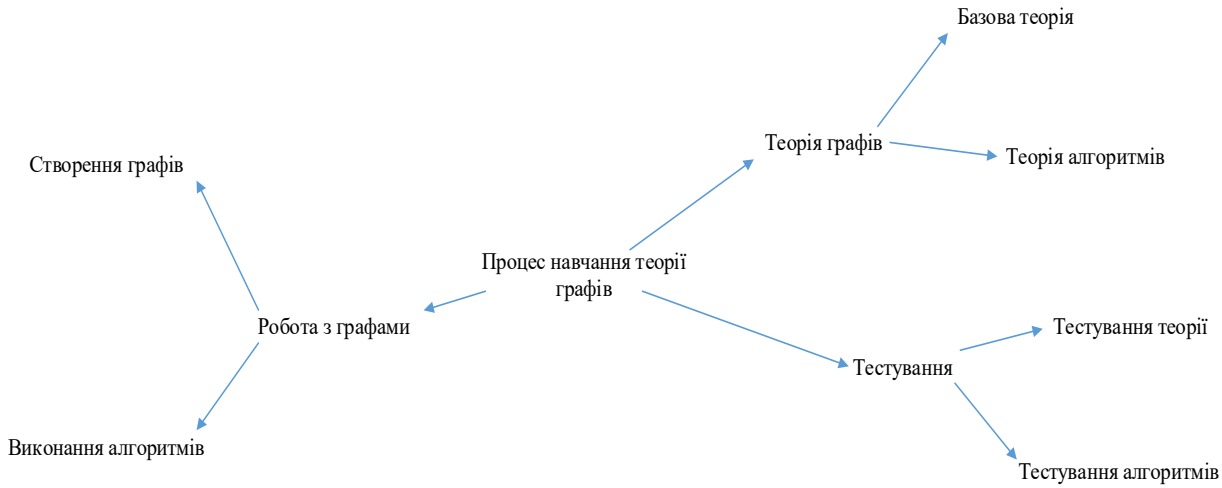

Рисунок 1.1 –Ментальна карта об'єкта автоматизації

Визначення проблем користувачів

1. Відсутність таких програмних систем, які були б створені для читання теоретичних матеріалів з графової теорії
2. Брак можливостей виконувати тестові завдання з теорії графів
3. Відсутність програмного забезпечення, яке б дозволяло тестувати правильність знаходження найкоротшого шляху
4. Відсутність програмного забезпечення, яке б дозволяло перевіряти правильність обходу дерева

Аналіз існуючих програмних рішень

В результаті пошуку схожих програмних продуктів в офіційному магазині застосувань Apple були знайдені наступні аналоги системи:

1. Graph Theory Pad

   Дане застосування дозволяє рисувати графи, міняти використовувані для рисування кольори, переглядати кількість вузлів та ребер, та зберігати та експортувати створений граф у PDF.

2. Graphing

   Дане застосування дозволяє створювати графи за допомогою редагування матриці суміжності, також задаючи кількість вузлів.

3. mGraph

   Дане застосування дозволяє рисувати графи, змінювати їх розташування, та зберігати створені графи у пам’яті пристрою.

4. GraphMaker

   Дане застосування дозволяє рисувати орієнтовані графи, будувати по ним матрицю суміжності, а також вирішувати задачі із пошуку найкоротшого шляху, мінімального остовного дерева, або максимального потоку у мережі.

Таблиця 2.1 – Порівняльний аналіз програмних продуктів

Продукти / Функції	Graph Theory Pad	Graphing	mGraph	GraphMaker	Проектована система
Рисування довільного графа	+	+	+	+	+
Зберігання графу у пам’яті пристрою	+	-	+	+	+
Пошук найкоротшого шляху	-	-	-	+	+
Пошук компонент зв’язності	-	-	-	-	+
Тестування набутих знань	-	-	-	-	+
Введення теоретичних даних	-	-	-	-	+

Визначення мети програмного продукту

Метою розроблюваної програмної системи є підвищення результативності навчання теорії графів за рахунок автоматизованого процесу контролю вирішення завдань з теми «Графи». Для досягнення цієї мети потрібно розв’язати такі задачі: забезпечення учня теоретичним матеріалом, візуалізація роботи алгоритмів, перевірка засвоєних знань.

1.3 Визначення функціональних вимог

Огляд варіантів використання системи

Необхідна функціональність системи розглянута на діаграмі варіантів використання, яка представлена на рисунку 1.2.

Окрім базових функцій, розглянутих у діаграмі, необхідно реалізувати деякі додаткові функції, котрі допомогли б зробити дане застосування відповідним до рекомендацій компанії Apple з дизайну та юзабіліті.

До таких функцій відноситься масштабування зображеного на екрані дерева/графу, робота з системними меню iOS, а саме використання методів для рисування меню створених компанією Apple для підтримання єдиного дизайну застосувань для iOS[4], а також робота з пам’яттю, виділеною застосуванню системою, тобто збереження та зчитування з пам’яті пристрою даних про дерево або граф.

Крім того, було необхідно реалізувати опрацювання жестів користувача, до яких відносяться різноманітні дотики на екран (довгі дотики, подвійні дотики), зведення та розведення пальців на екрані для зручності користування застосуванням.

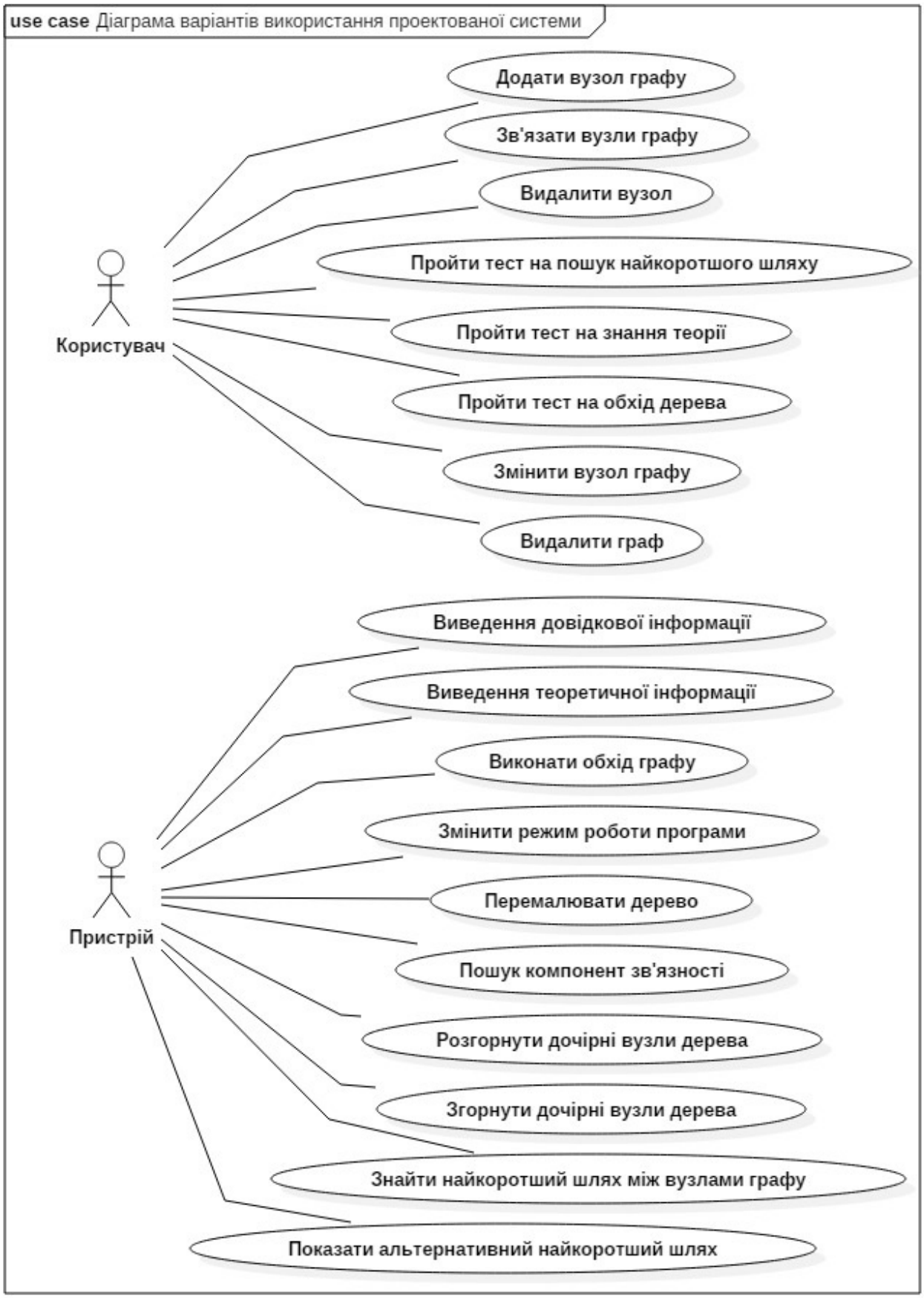

Рисунок 1.2 - Діаграма варіантів використання розробленої системи

Опис варіантів використання

1. Сценарій прецеденту «Додати вузол графу»
    - Передумова: немає.
    - Основний сценарій:
    1. Користувач вказує точку на екрані, де він бажає розмістити новий вузол.
    2. Система перевіряє можливість розміщення вузла на вибраному місці.
    3. Якщо така можливість є, система виводить діалогове вікно для введення назву вузла.
    4. Користувач вводить назву вузла.
    5. Система розміщує вузол на вибраному місці.
    6. Інакше система повідомляє про неможливість виконання даної дії.
    - Мінімальні гарантії – система повідомляє про неможливість розміщення вузла у вказаній точці.
    - Гарантії успіху – система розміщує новий вузол у вказаній точці.
2. Сценарій прецеденту «Зв’язати вузли графу»
    - Передумова: у середовищі присутні 2 вузли графу, або більше.
    - Основний сценарій:
    1. Користувач вказує точку на екрані де знаходиться вузол, який він бажає пов’язати з іншим вузлом.
    2. Система перевіряє чи вказана точка належить якому-небудь з елементів.
    3. Якщо точка належить якому-небудь з елементів, система пропонує обрати другий вузол.
    4. Інакше система повідомляє про неможливість виконання даної дії.
    5. Користувач вказує точку на екрані, де знаходиться вузол, який він бажає пов’язати з обраним раніше вузлом.

6. Система перевіряє чи вказана точка належить якому-небудь з невибраних елементів.
7. Якщо виконується попередня умова, система пов’язує обрані вузли.
8. Інакше система повідомляє про неможливість виконання даної дії.

- Мінімальні гарантії – система повідомляє про неможливість виконання даної дії.
- Гарантії успіху – система пов’язує обрані вузли.

3. Сценарій прецеденту «Видалити вузол»

- Передумова: в системі існує хоча б один вузол.
- Основний сценарій:

1. Система видаляє вузол та усі його зв’язки з іншими елементами. У випадку видалення вузла дерева, також видаляються усі його дочірні вузли.

- Мінімальні гарантії – немає.
- Гарантії успіху – система видаляє вузол та його зв’язки.

4. Сценарій прецеденту «Пройти тест на пошук найкоротшого шляху»

- Передумова: в системі існує більше двох пов’язаних між собою вузлів графу.
- Основний сценарій:

1. Користувач обирає вузли графу, найкоротший шлях між якими буде перевірятися.
2. Користувач вказує точку на екрані, де як він вважає знаходиться поточний вузол найкоротшого шляху.
3. Якщо користувач вказав неправильний вузол, система повідомляє про помилку та припиняє тест.
4. Інакше, якщо пошук найкоротшого шляху було виконано правильно, система повідомляє про успішний результат.

- Мінімальні гарантії – система повідомляє про помилку.
- Гарантії успіху – система повідомляє про успіх.

5. Сценарій прецеденту «Пройти тест на знання теорії»
    - Передумова: немає.
    - Основний сценарій:
    1. Система виводить вікно із питаннями та варіантами відповідей.
    2. Користувач обирає варіант відповіді.
    3. Система виводить результат користувача.
    - Мінімальні гарантії – система виводить вікно із питаннями та варіантами відповідей.
    - Гарантії успіху – система виводить результат користувача.
6. Сценарій прецеденту «Пройти тест на обхід дерева»
    - Передумова: користувачем було створено дерево.
    - Основний сценарій:
    1. Користувач вказує алгоритм, за яким буде відбуватися обхід
    2. Користувач вказує точку на екрані, де як він вважає знаходиться поточний вузол обходу графа.
    3. Якщо користувач вказав неправильний вузол у порядку обходу, система повідомляє про помилку та припиняє тест.
    4. Інакше, якщо обхід дерева було виконано правильно, система повідомляє про успішний обхід.
    - Мінімальні гарантії – система повідомляє про помилку.
    - Гарантії успіху – система повідомляє про успіх.
7. Сценарій прецеденту «Змінити вузол»
    - Передумова: в системі існує хоча б один вузол.
    - Основний сценарій:
    1. Система виводить діалогове вікно для введення назви вузла.
    2. Користувач вводить назву вузла.

3. Система зберігає зміни.

- Мінімальні гарантії – система виводить діалогове вікно для введення назви вузла.
- Гарантії успіху – система зберігає зміни.

8. Сценарій прецеденту «Видалити граф»

- Передумова: в системі існує хоча б один вузол.
- Основний сценарій:

1. Система видаляє усі вузли графа та усі зв’язки між ними.

- Мінімальні гарантії – немає.
- Гарантії успіху – система видаляє усі вузли графа та усі зв’язки між ними.

9. Сценарій прецеденту «Виведення довідкової інформації»

- Передумова: немає.
- Основний сценарій:

1. Система виводить вікно із довідковою інформацією по роботі з програмою.

- Мінімальні гарантії – немає.
- Гарантії успіху – система виводить вікно із довідковою інформацією по роботі з програмою.

10. Сценарій прецеденту «Виведення теоретичної інформації»

- Передумова: немає.
- Основний сценарій:

1. Система виводить вікно із теоретичною інформацією..

- Мінімальні гарантії – немає.
- Гарантії успіху – система виводить вікно із теоретичною інформацією.

11. Сценарій прецеденту «Виконати обхід графу»

- Передумова: в системі існує хоча б один вузол.

- Основний сценарій:

1. Користувач вказує алгоритм, за яким відбуватиметься обхід.
2. Система позначає вузли графу цифрами відповідно до їх порядку при обході.

- Мінімальні гарантії – немає.
- Гарантії успіху – система позначає вузли графу цифрами.

12. Сценарій прецеденту «Змінити режим роботи програми»

- Передумова: немає.
- Основний сценарій:

1. Якщо в даний момент програма працює у режимі роботи з графами, то режим роботи змінюється на режим роботи з деревами.
2. Інакше режим змінюється на режим роботи із графами.

- Мінімальні гарантії – немає.
- Гарантії успіху – система змінює режим роботи програми.

13. Сценарій прецеденту «Перерисувати дерево»

- Передумова: в системі існує хоча б один вузол.
- Основний сценарій:

1. Користувач вказує відображення за яким необхідно перерисувати дерево.
2. Система перерисовує дерево згідно із вказаним користувачем відображенням.

- Мінімальні гарантії – немає.
- Гарантії успіху – система перерисовує дерево.

14. Сценарій прецеденту «Пошук компонент зв’язності»

- Передумова: в системі існує хоча б один вузол.
- Основний сценарій:

1. Система змінює колір усіх зв’язаних між собою вузлів на випадковий колір та повторює процедуру для інших вузлів.

- Мінімальні гарантії – немає.
- Гарантії успіху – система демонструє компоненти зв’язності.

15. Сценарій прецеденту «Згорнути дочірні вузли дерева»

- Передумова: в системі існує хоча б один вузол з дочірніми вузлами.
- Основний сценарій:

1. Користувач вказує точку на екрані, де він знаходиться вузол, дочірні вузли якого він бажає згорнути.
2. Система перевіряє наявність вузла в даній точці.
3. Якщо вузол знайдено, система вилучає зображення дочірніх вузлів та їх зв’язків з екрана.

- Мінімальні гарантії – немає.
- Гарантії успіху – система згортає дочірні вузли дерева.

16. Сценарій прецеденту «Розгорнути дочірні вузли дерева»

- Передумова: в системі існує хоча б один вузол, чиї дочірні вузли було згорнуто.
- Основний сценарій:

1. Користувач вказує точку на екрані, де він знаходиться вузол, дочірні вузли якого він бажає розгорнути.
2. Система перевіряє наявність вузла в даній точці.
3. Якщо вузол знайдено, система додає зображення дочірніх вузлів та їх зв’язків на екран.

- Мінімальні гарантії – немає.
- Гарантії успіху – система розгортає дочірні вузли дерева.

17. Сценарій прецеденту «Знайти найкоротший шлях між вузлами графу»

- Передумова: в системі існує хоча б два вузли графу.

- Основний сценарій:

1. Користувач вказує вузол від якого буде відбуватися пошук найкоротшого шляху.
2. Користувач вказує вузол, до якого буде відбуватися пошук найкоротшого шляху.
3. Якщо вузли пов’язані між собою, система візуалізує найкоротший шлях між вузлами.
4. Інакше система повідомляє, що вузли не пов’язані між собою.

- Мінімальні гарантії – система повідомляє, що вузли не пов’язані між собою.
- Гарантії успіху – система візуалізує найкоротший шлях.

18. Сценарій прецеденту «Показати альтернативний найкоротший шлях між вузлами графу»

- Передумова: було знайдено найкоротший шлях між вузлами графу.
- Основний сценарій:

1. Якщо існує не показаний раніше альтернативний шлях, система візуалізує його.
2. Інакше система повідомляє, що альтернативний шлях не знайдено.

- Мінімальні гарантії – система повідомляє, що альтернативний шлях не знайдено.
- Гарантії успіху – система візуалізує альтернативний найкоротший шлях.

1.4 Функціональні вимоги

Перелік визначень предметної області:

Граф — множина об'єктів та зв'язків між ними.

Об'єкти — це вершини графу.

Зв'язки — це дуги\ребра графу.

Дерево — це зв'язний граф без циклів.

Корінь — це верхній вузол в дереві.

Обхід дерева – перебір елементів дерева за зв'язками між вузлами-предками і вузлами-нащадками.

Пошук у глибину – це алгоритм обходу дерева, подібної до дерева структури чи графа. Робота алгоритму починається із кореня (або іншої обраної вершини в графі) і здійснюється обхід в максимально можливу глибину до переходу на наступну вершину.

Пошук у ширину – такий алгоритм обходу дерева, або подібної до дерева структури чи графа. Робота алгоритму починається із кореня (або іншої обраної вершини в графі) і здійснюється обхід всіх досяжних із поточної вершини вершин до переходу на наступну вершину.

Зв'язність – це коли існує шлях, що з'єднує дві вершини в графі.

Компонента зв'язності графа — це деяка підмножина вершин графа, така, що для будь-яких двох вершин із цієї множини існує шлях із однієї в іншу, і не існує шляху з вершини цієї множини в вершину не з цієї множини.

Класифікація функціональних вимог

На рисунку 1.3 зображено діаграму, на якій згруповано функціональні вимоги.

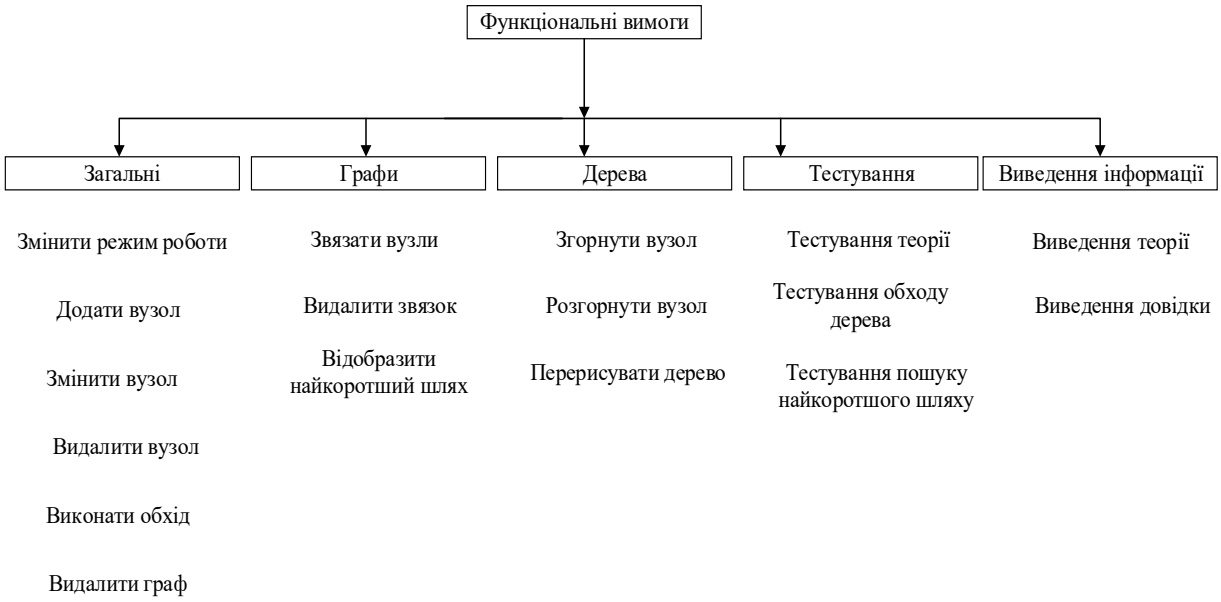

Рис. 1.3 – «Класифікація функціональних вимог»

Визначення пріоритетів за принципом «MoSCoW»

«Додати вершину» – M

«Зв’язати вершини» – S

«Видалити вершину» – C

«Пройти тест на знання теорії» – C

«Пройти тест на пошук найкоротшого шляху» – C

«Пройти тест на обхід» – C

«Змінити вершину» – W

«Видалити граф» – S

«Виведення довідки» – S

«Виведення теорії» – S

«Виконати обхід» – M

«Змінити режим роботи» – M

«Перерисувати дерево» – W

«Знайти компоненти зв’язності» – S

«Згорнути\Розгорнути дочірні вузли дерева» – C

«Знайти найкоротший шлях між вершинами» – M

«Показати найкоротший альтернативний шлях між вершинами» – M

1.5 Нефункціональні вимоги

Зовнішні інтерфейси

Шрифт, який відображається при роботі програми – системний шрифт iOS – San Francisco.

Значок додатку – стандартний значок iOS.

Написи на кнопках програми наведено англійською мовою.

Для навігації та використання функцій програми використовується системне меню операційної системи iOS.

Кольори меню програми – стандартна кольорова схема меню iOS.

Відображення повідомлень – стандартний інтерфейс відображення повідомлень у iOS.

Версії підтримуваних операційних систем – усі версії операційної системи iOS починаючи від iOS 10 та закінчуючи iOS 15.

Аналіз атрибутів якості

Атрибути якості проаналізовано у таблиці 1.3.

Таблиця 1.3 – Аналіз атрибутів якості

Набір характеристик	Властивості програмного забезпечення
Функціональність	Взаємодія: не взаємодіє з зовнішніми додатками, або веб-сервісами. Захищеність: усі дані, що необхідні системі, доступні тільки даному програмному продукту
Ефективність	Часова ефективність: рендерінг графу не більше 4 с. Ефективність використання ресурсів: система не споживатиме більше 350 Мб оперативної пам’яті пристрою при роботі.

Продовження таблиці 1.3

Надійність	Відмовостійкість: у випадку аварійного завершення роботи, зміни, внесені у поточний сеанс роботи з системою буде збережено. Здатність до відновлення: у випадку аварійного завершення роботи, наступний запуск системи відбудеться у звичайному режимі.
Переносимість	Адаптивність: система функціонуватиме на версіях операційної системи iOS починаючи з 10 версії iOS.

Опис сценаріїв якості

1. Часова ефективність:
    a) Джерело тригеру: користувач

       Тригер: рендерінг графу

       Умова: робота у нормальному режимі

       Артефакт: система

       Реакція системи: зображення результату на екрані

       Кількісна міра: < 2 секунд
    b) Джерело тригеру: користувач

       Тригер: рендерінг графу

       Умова: система у навантаженому стані

       Артефакт: система

       Реакція системи: зображення результату на екрані

       Кількісна міра: < 5 секунд
2. Ефективність використання ресурсів системи:
    a) Джерело тригеру: користувач

       Тригер: виведення довідки

       Умова:нормальна робота системи

       Артефакт: система

       Реакція системи: виведення даних

       Кількісна міра: < 200 Мб

b) Джерело тригеру: користувач

Тригер: завантаження даних

Умова: рендерінг завантаженого графу

Артефакт: система

Реакція системи: успішне виконання операції

Кількісна міра: < 350 МБ

3. Стійкість до відмов:

a) Джерело тригеру: ОС

Тригер: сигнал аварійного завершення роботи

Умова: нормальна робота системи

Артефакт: система

Реакція системи: зміни, внесені у поточному сеансі роботи з додатком буде збережено

Кількісна міра: 98%

b) Джерело тригеру: ОС

Тригер: сигнал аварійного завершення роботи

Умова: перевантаженому режим роботи

Артефакт: система

Реакція системи: зміни, що були внесені у поточному сеансі роботи з додатком буде збережено

Кількісна міра: 97%

4. Здатність до відновлення:

a) Джерело тригеру: користувач

Тригер: увімкнення системи

Умова: аварійне завершення роботи

Артефакт: система

Реакція системи: нормальний запуск системи

Кількісна міра: 98.5%

b) Джерело тригеру: ОС

Тригер: перезавантаження системи

Умова: аварійне завершення роботи

Артефакт: система

Реакція системи: нормальний перезапуск системи

Кількісна міра: 98%

5. Адаптивність:

a) Джерело тригеру: користувач

Тригер: інсталяція додатку

Умова: Операційна система iOS 10.0

Артефакт: система

Реакція системи: нормальна інсталяція додатку

Кількісна міра: 99.7%

b) Джерело тригеру: користувач

Тригер: інсталяція додатку

Умова: Операційна система iOS 15.1

Артефакт: система

Реакція системи: нормальна інсталяція додатку

Кількісна міра: 99.7%

1.6 Планування процесу розробки системи

Постановка задачі проектування програмної системи

Для проектування даної програмної системи необхідно провести узагальнення ієрархії функціональних вимог проектованої системи, а саме створити окремі компоненти для кожного із описаних у попередніх розділах типів вимог

Архітектура програмної системи

Для розробки даної програмної системи буде використано архітектурний шаблон Model View Controler.

Для цього планується створити компонент, що відповідає за зберігання даних, та зміну стану системи відповідно до змін, які вносить користувач системи у процесі роботи з розроблюваним додатком.

Також для кожного з режимів роботи програми (режим роботи із графами, деревами, режим виведення інформації, режим тестування) буде створено по окремому компоненту, відповідальному за представлення даних на екрані у вигляді візуалізованих графів, а також компоненти, що відповідають та реагування на дії користувача.

Технології розробки

Розробка даного програмного продукту проводитиметься для мобільної платформи iOS під управлінням ОС macOS.

Мобільну платформу для цієї розробки було обрано із наступних міркувань: проникність смартфонів серед студентів у віковій категорії 18-22 років перевищує 95%, а також більшість студентів у даній віковій категорії віддає перевагу використанню мобільних пристроїв у повсякденному житті, аніж використанню компютерів.

Саме тому для того, щоб популяризувати вивчення теорії графів та мати більше шансів на успіх, ніж при використанні копмютерів було прийнято рішення розробляти проектовану систему для використання на мобільній платформі.

Платформу iOS було обрано як цільову для розроблюваної системи, оскільки для операційної системи Android, яка є конкурентом iOS існує велика кількість аналогічних за функціональністю додатків.

Це може бути обумовлено нижчим порогом входу для розробки додатків через те, що платформа Android є open-source на відміну від Apple.

Дані, необхідні для роботи розроблюваної системи, зберігатимуться у файлах формату JSON.

Для розробки даної програмної системи планується використовувати інтегроване середовище розробки Xcode.

Декомпозиція робіт

В результаті аналізу проведеного у попередніх розділах даної роботи, було визначено наступні процеси, що необхідно реалізувати:

- зібрати і проаналізувати функціональні та нефункціональні вимоги до продукту,
- спланувати,
- розробити (розробити компонент що дозволяє працювати з графами, імплементувати компонент що виводить довідкову або теоретичну інформацію),
- вивчити необхідну теорію щодо використання архітектурного шаблону Model View Controller,
- здійснювати керування проектом протягом усього життєвого циклу,
- задокументувати всі етапи робіт,
- забезпечити перевірку відповідності критеріям якості
- виконати впровадження системи.

Результати вищеописаного аналізу приведено в таблиці 1.4.

Таблиця 1.4 – Декомпозиція процесів

Елемент WBS першого рівня	Стандартний розподіл	Необхідний час	Операції
Керування проектом	15%	50	Створення документації – 30
Створення робочого середовища	5%	30	Ознайомлення з MVC - 15
Керування вимогами до системи	10%	50	Збирання та аналіз вимог
Планування розробки	15%	70	

Продовження таблиці 1.4

Імплементація	25%	120	Розробка компонента роботи з графами - 50 Розробка компонента виведення текстової інформації - 30 Розробка тестування - 40
Тестування	25%	120	Тестування компонента роботи з графами - 40 Тестування компонента виведення текстової інформації - 20 Випробування режиму тестування - 50 Інтеграційне тестування - 30
Впровадження розробленої системи	5%	20	

Визначення розмірів розроблюваної системи методом UCP

Оцінювання акторів

Взаємодія з користувачем відбуватиметься за допомогою Graphical User Interface (складний).

Актор «Пристрій» – система з деяким API (простий).

UAW = 1 + 3 = 4

Оцінювання use-case за транзакціями

«Додавання вершини» – 5 транзакцій (середній).

«Зв’язування вершин» – 8 транзакцій (складний).

«Видалити вузол графу» – 2 транзакції (простий).

«Тестування теоретичних знань» - 4 транзакції (середній).

«Тестування обходу дерева» - 5 транзакцій (середній).

«Тестування пошуку найкоротшого шляху» - 5 транзакцій (середній).

«Змінити вершину» - 4 транзакції (середній).

«Видалення графу» - 2 транзакції (простий).

«Вивід довідки» - 1 транзакція (простий).

«Вивід теорії» - 1 транзакція (простий).

«Виконання обходу графа» - 3 транзакції (простий).

«Зміна режиму роботи» - 2 транзакції (простий).

«Перемалювати дерево» - 1 транзакція (простий).

«Знайти компоненти зв’язності» - 2 транзакції (простий).

«Згорнути вершини дерева» - 4 транзакції (середній).

«Розгорнути вершини дерева» - 4 транзакції (середній).

«Пошук найкоротшого шляху між вершинами» - 5 транзакцій (середній).

«Пошук альтернативного найкоротшого шляху» - 4 транзакції (середній).

UUCV = (9*5) + (9*11) +(1*14) = 40 + 80 + 15 = 155

UCP = UAW + UUCV = 2 + 155 = 157

Оцінювання технічних факторів

У таблиці 1.5 приводяться оцінки визначених технічних факторів.

Таблиця 1.5 – Оцінювання технічних факторів

Назва	Вага	Опис	Оцінка
Розподіленість системи	2	Визначає потребу програми у розподілених обчисленнях	0
Час відгуку	1	Оцінює ефективність програми, час відгуку, потік робіт	3
Ефективність кінцевого користувача	1	Оцінює ефективність роботи користувачів системи	5
Складність обробки	1	Оцінює застосовування складних алгоритмів обробки інформації	0
Повторне використання	1	Оцінює, чи буде код системи ревикористовуватись	3
Простота встановлення	0,5	Визначає метод та складність встановлення для кінцевих користувачів	5

Продовження Таблиці 1.5

Простота використання	0,5	Визначає співпадіння інтерфейсу з потребами користувачів	5
Портативність	2	Визначає, чи може програма працювати в інших середовищах	0
Простота змін	1	Оцінює простоту модифікації системи у майбутньому	5
Паралельні обчислення	1	Інформує, чи будуть мати в системі місце паралельні обчислення	0
Засоби захисту	1	Визначає необхідність спеціальних засобів захисту інформації та програми	0
Доступ до третьої сторони	1	Оцінює ступінь користування програмою зовнішніми системами або акторами	0
Потреби в спеціальному навчанні	1	Оцінює необхідність організації навчання користувачів	0

TFactor = 2 + 8 + 3 + 3,5 + 2,5 + 7 = 32

TCF = 0,6 + (0,01 * TFactor) = 0,6 + (0,01 * 32) = 0,6 + 0,32 = 0,92

Оцінювання зовнішніх факторів

У таблиці 1.6 наведено аналіз та оцінки різноманітних зовнішніх факторів що мають вплив на проектовану систему.

Таблиця 1.6 – Оцінювання зовнішніх факторів

Опис	Вага	Пояснення	Оцінка
Ознайомлення із процесами імплементації	1,5	Оцінює ступінь знайомства команди розробки з предметною областю та технічними деталями вирішення задачі	3

Продовження таблиці 1.6

Досвід схожих проектів	0,5	Загальна уява про досвід команди в розробці подібного програмного забезпечення	3
Досвід об’єктно-орієнтованої розробки	1	Досвід в проектуванні об’єктно-орієнтованих застосувань, а також в підтримці засобів для розробки інформаційних систем	3
Досвід провідного аналітика	0,5	Здатність бізнес-аналітика отримувати вимоги від клієнтів та знання щодо задач, які мають вирішуватись системою	3
Мотивація	1	Здатність команди займатись призначеною ним задачею	5
Стабільність вимог	2	Визначає, чи не будуть вимоги часто змінюватись	5
Часткова зайнятість працівників	-1	Визначає, наскільки великою є частка працівників часткової зайнятості	0
Складність мови програмування	-1	Визначає, наскільки складно вивчити мову програмування	4

EFactor = 4,5 + 1,5 + 3 + 1,5 + 5 + 10 – 4 = 16,5

EF = 1,4 + (-0,03 * EFactor) = 1,4 – 0,03 * 16,5 = 1,4 – 0,49 = 0,9

Результуючі оцінки

Кількість факторів з множини F1 – F8, оцінки яких за абсолютним значенням перевищують 3 дорівнює 3.

Одному UCP відповідає 28 робочих годин.

AUCP = UCP * TCF * EF = 149 * 0,81 * 0,9 = 109

Тривалість розробки = 109 * 28 = 3052 години.

2 МОДЕЛЮВАННЯ СИСТЕМИ

Концептуальне проектування

На рисунку 2.1 зображено діаграму концептуальних класів розроблюваної системи[7].

Модулі TreeViewController, InfoViewController та GraphViewController будуть відповідати за відображення та обробку дій користувача, Model буде здійснювати зберігання та обробку інформації, необхідної для функціонування системи, у той час як компонент GraphNode буде представляти вузол графу.

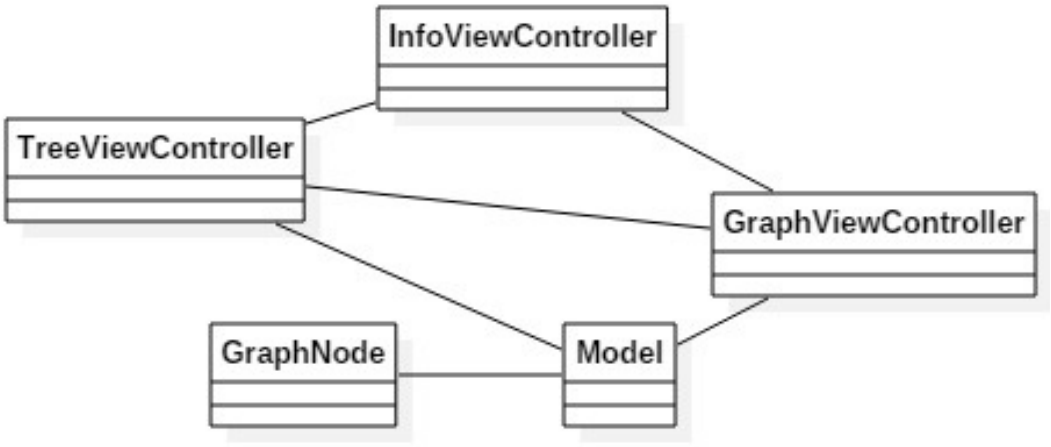

Рисунок 2.1 - Діаграма концептуальних класів розроблюваної системи

Логічне проектування

Діаграму програмних класів системи зображено на рисунку 2.2. Розглянемо зображені на рисунку 2.2 класи більш детально:

- AppDelegate – клас, який відповідає за обробку всіх повідомлень від операційної системи[1].
- InfoViewController – клас, який виводить теоретичну або довідкову інформацію.
- TreeViewController – клас, який забезпечує взаємозв’язок моделі та відображення дерева. У даному класі міститься основна логіка роботи з деревами, обробляються жести користувача та викликаються необхідні методи класу Model.

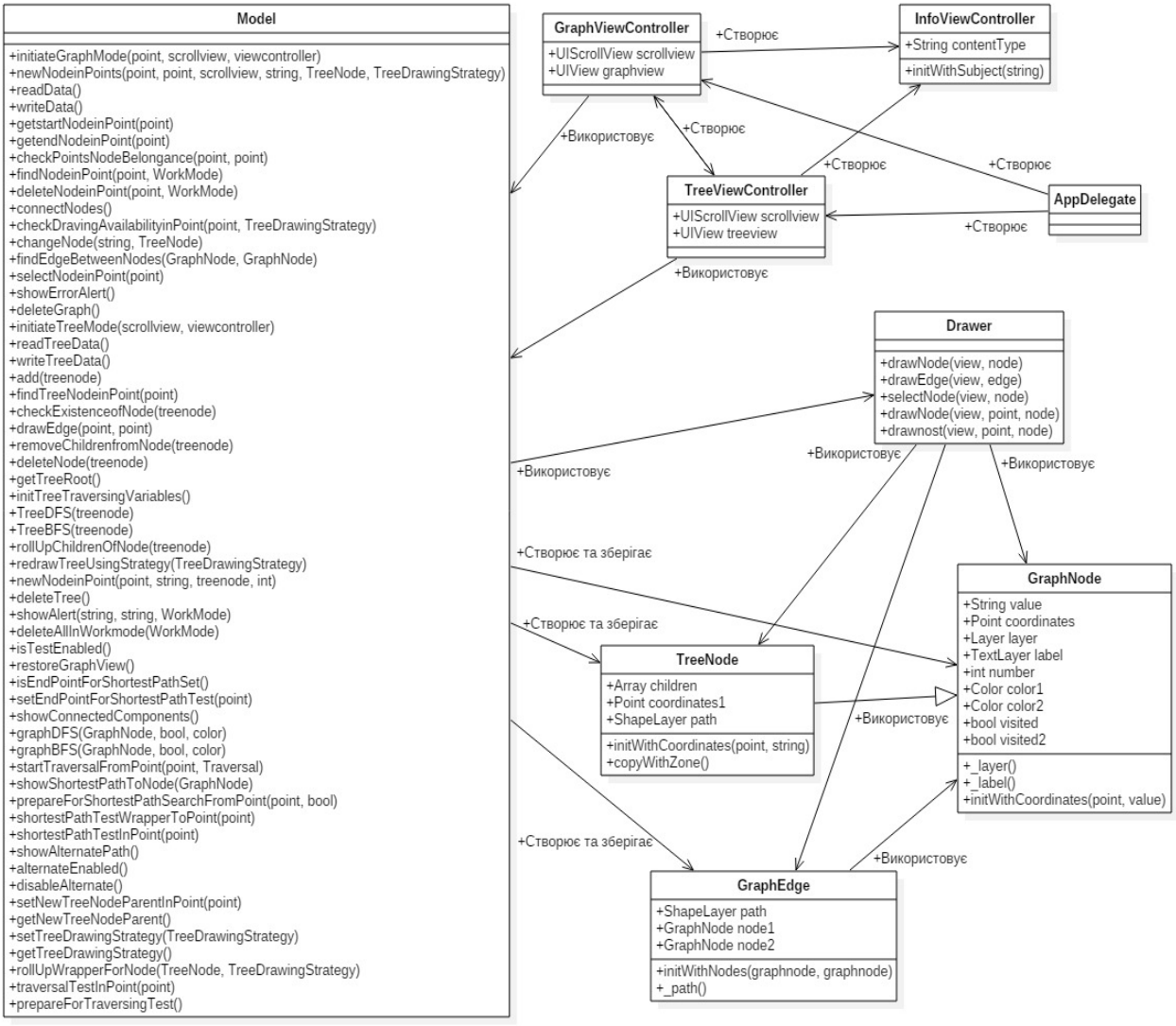

Рисунок 2.2 - Діаграма програмних класів розроблюваної системи

GraphViewController – клас, який забезпечує взаємозв’язок моделі та відображення графа. У даному класі міститься основна логіка роботи з графами, обробляються жести користувача та викликаються необхідні методи класу Model.

Model – це клас, який містить у собі усі вузли дерева та графу, виконує різноманітні функції як-то: завантаження дерева/графу з пам’яті пристрою та виклик методу для рисування дерева/графу при завантаженні програми, пошук необхідного вузла при дотику на нього, рисування ребра від вузла до його

батьківського вузла, збереження дерева/графу у пам’яті пристрою, видалення вузлів дерева/графу, та багато інших задач.

Drawer – клас, відповідальний за рисування графа або дерева на екрані пристрою, який рисує дерево у класичному порядку відображення дерева, або у порядок, визначений користувачем при рисуванні дерева, також реалізує візуальне виділення вузла графу та рисування дуг графу.

TreeNode – модельний клас вузла дерева, який містить всі необхідні відомості про нього, такі як: координати розміщення на екрані в залежності від відображення дерева, назву вузла, усі дочірні вузли, графічне представлення вузла, представлення ребра і закінчуючи кольором вузла.

GraphNode – модельний клас вузла графу, який містить всі необхідні відомості про нього, такі як: координати розміщення на екрані, назву вузла, графічне представлення вузла та представлення назви.

GraphEdge – модельний клас ребра графу, який містить всі необхідні відомості про нього, такі як: посилання на вузли, що поєднуються та графічне представлення дуги.

Застосовані шаблони проектування

Наведені вище діаграми логічного подання є результатом застосування деяких архітектурних стилів та шаблонів до проектованої системи.

Так як проектована система є інтерактивним GUI-застосуванням, то найбільш відповідним контексту системи архітектурним шаблоном виявився Model-View-Controller.

Архітектурний стиль MVC ділить застосування на 3 складові:

Модель [5] (містить дані та методи роботи з ними, реагує на запити та змінює свій стан),

представлення (відображає дані),

контролер (обробляє введені користувачем дані). Даний стиль широко використовується для підтримки людино-машинної взаємодії, тому його було обрано при проектуванні.

У даній системі роль контролера та представлення відіграють наступні класи:

- InfoViewController,
- TreeViewController
- GraphViewController.

У свою чергу роль моделі відіграють класи

- Model,
- TreeNode,
- GraphNode,
- GraphEdge.

Алгоритми

Схеми алгоритмів таких функцій як: «Додання нового вузла графу» та «Пошук компонент зв’язності» представлено рисунками 2.2 і 2.3 відповідно.

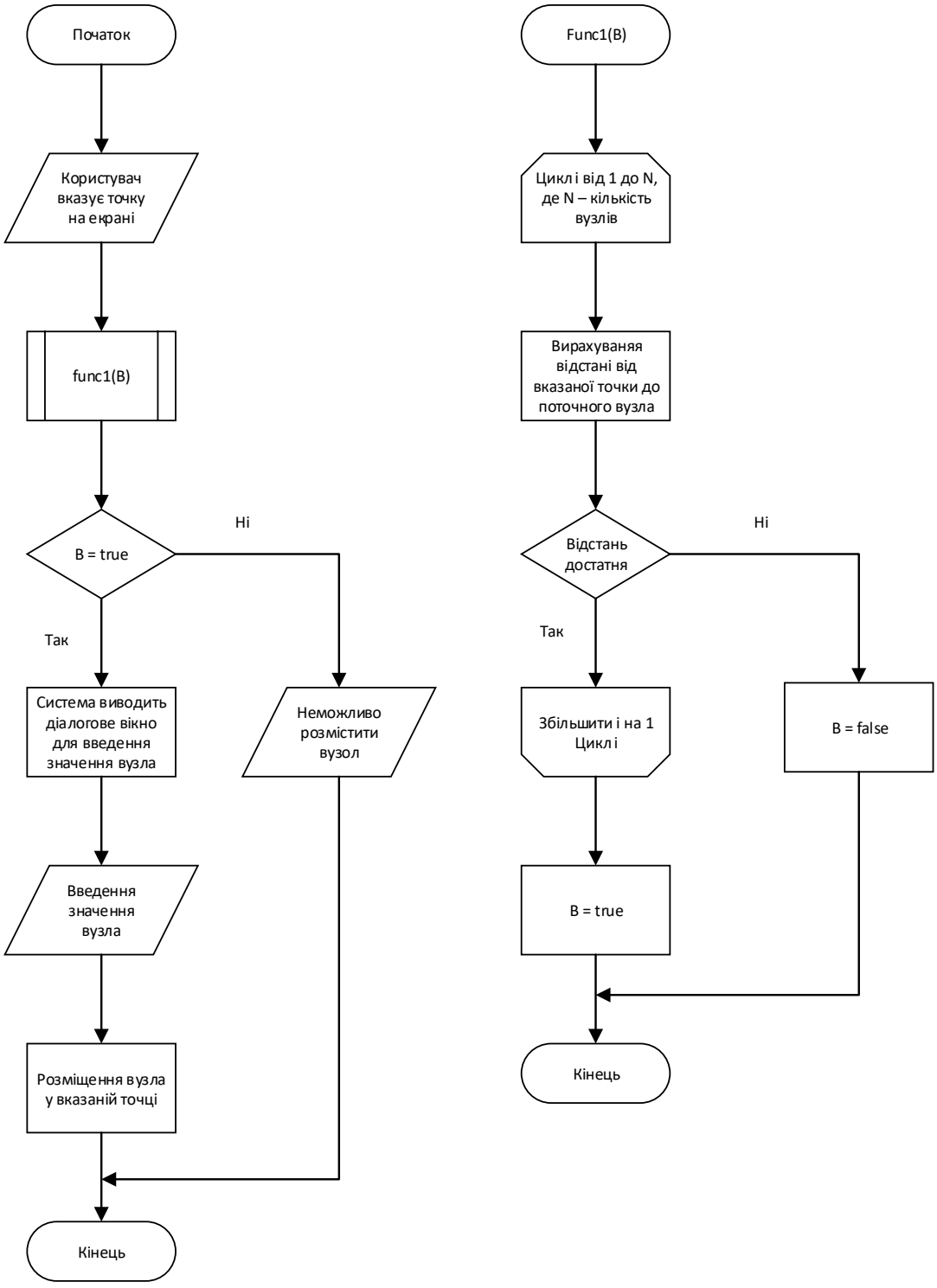

Рисунок 2.2 – Схеми алгоритмів функції «Додання нового вузла графу»

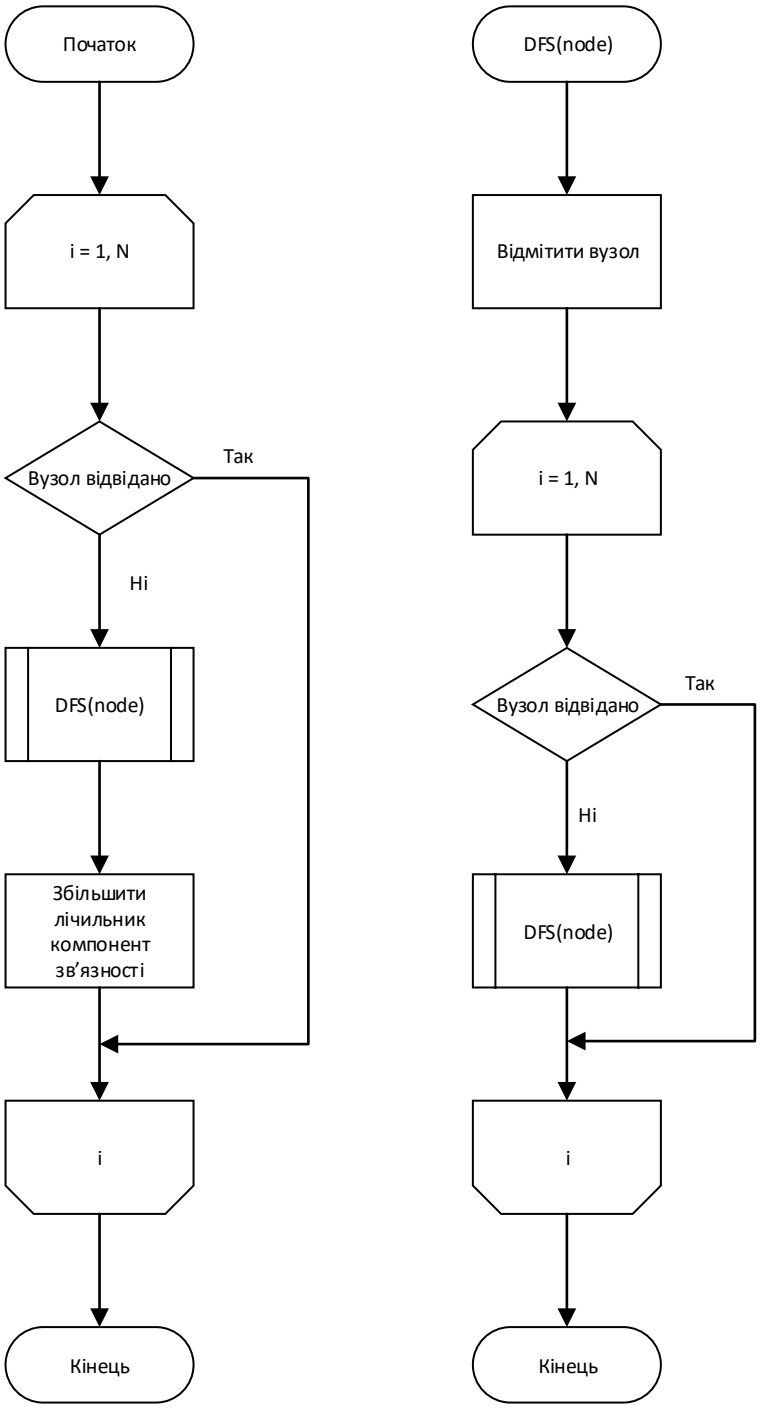

Рисунок 2.3 – Схеми алгоритмів функції «Пошук компонент зв'язності»

Алгоритм додання нового вузла графу працює наступним чином: користувач вказує точку на екрані, де він планує розмістити новий вузол.

У разі, якщо точка знаходиться на достатній відстані від інших вузлів, користувачу пропонується ввести назву вузла у спеціальному вікні, і після

цього вузол з’являється на екрані. Інакше система повідомляє про неможливість розміщення вузла у даній локації.

Принцип роботи алгоритму пошуку компонент зв’язності полягає у наступому: система виконує обхід у глибину від кожного вузла графу, при цьому розфарбовуючи їх у один колір.

Якщо в результаті обходу є невідвідані вузли, лічильник компонент зв’язності збільнується, та виконується обхід від невідвіданого вузла. Процедура повторюється, поки усі вузли не буде відвідано.

Оцінка складності алгоритмів

Оцінка складності першого алгоритму – O(n), де n – кількість вершин графу.

Оцінка складності другого алгоритму – O(n^2), де n – кількість вершин графу.

Тактики забезпечення якості

Також були враховані деякі тактики для забезпечення характеристик якості, визначених в попередніх частинах роботи.

Тактики реалізації готовності:

Для виявлення несправностей класів я обрав тактику «Винятки», яка полягає у стиканні з винятками, які виникають при несправності.

Для відновлення після несправності найбільш ефективною в умовах моєї задачі тактикою була обрана тактика «Контрольна точка», яка являє собою запис послідовності станів, який створюється у відповідь на певні події.

Тактики реалізації модифікованості:

Для запобігання хвильового ефекту є сенс використати тактику підтримки існуючих інтерфейсів шляхом відділення інтерфейсу від реалізації.

Тактики реалізації продуктивності:

Для управління ресурсами я вважаю за доцільне використати тактику кешування даних.

Тактики реалізації зручності використання:

Для підвищення зручності використання я обрав тактику періоду проектування, а саме відокремлення інтерфейсу користувача від решти програми для локалізації змін до нього.

Висновки до розділу

В результаті моделювання програмного продукту було створено діаграми концептуальних та програмних класів системи, описано використані для цього архітектурні шаблони та тактики, а також була оцінена складність використаних алгоритмів.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

Опис програмних технологій

Через специфічність платформи, для якої ведеться розробка, виникла необхідність використовувати мову програмування Objective-C. Також для запуску створеного програмного забезпечення необхідним є такий інструмент як симулятор iPhone.

Опис програмних бібліотек

Для розробки було використано наступні фреймворки Apple, такі як:

- Foundation – це стандартний фреймворк, що надає базові класи та структури даних.
- UIKit – надає необхідну інфраструктуру для створення iOS-додатків.
- AppKit – бібліотека для створення графічних інтерфейсів.
- Core Data – бібліотека для управління об’єктами.
- QuartzCore – системна бібліотека для роботи з графікою та анімацією[10].

Особливості мови програмування використаної для розробки

Особливістю мови програмування Objective-C є те, що кожний програмний модуль (клас) складається з двох файлів, наприклад ClassName.h ClassName.m.

У файлах з розширенням .h міститься опис класу, його атрибутів та методів. Класи з розширенням .m містять реалізацію оголошених методів.

Підключення необхідних класів та бібліотек відбувається за допомогою директиви #import[3].

Особливості створення структур даних

Граф зберігається у пам’яті у вигляді списку ребер, тобто списку, де кожному ребру графу відповідає рядок, в якій зберігаються дві вершини, інцидентні ребру[9].

Під час роботи програми дані про вузли дерева, вузли та ребра графу зберігаються у відповідних колекціях NSMutableArray.

Клас NSMutableArray оголошує програмний інтерфейс для об'єктів, які керують змінюваним масивом об'єктів. Цей клас додає операції вставки та видалення до базової поведінки обробки масивів, успадкованої від NSArray, який є батьківським класом NSMutableArray[2].

Між запусками програми дані зберігаються у серіалізованому вигляді у файлах налаштувань користувача, доступ до яких надається через інтерфейс NSUserDefaults.

Клас NSUserDefaults надає програмний інтерфейс для взаємодії з системними налаштуваннями. Системні налаштування дозволяють програмі налаштувати свою поведінку відповідно до вподобань користувача. Програми зберігають ці параметри, призначаючи значення набору параметрів у базі даних користувача за замовчуванням. Параметри називаються параметрами за замовчуванням, оскільки вони зазвичай використовуються для визначення стану програми за замовчуванням під час запуску[2].

Теоретичні, довідкові дані, а також дані для тестів зберігаються у файлі JSON. Вибір даного формату обумовлено тим, що для підтримки системи та можливості змін у програмі без внесення змін до вихідного коду програми необхідно зберігати дані у форматі зрозумілому людині.

JSON — це відкритий стандартний формат файлу та формат обміну даними, який використовує зрозумілий людині текст для зберігання та передачі об’єктів даних, що складаються з пар атрибут-значення та масивів (або інших значень, які можна серіалізувати). Це поширений формат даних із різноманітним використанням в електронному обміні даними, включаючи веб-додатки із серверами[11].

Структура файлу що використовувався системою виглядає наступним чином:

{

```
    "theory" : "...",
    "help" : "...",
    "questions":[ { "question" : "...", "correct answer" : "..." },
{...},{...} ]
}
```

Елемент «theory» містить у собі рядок із теоретичними даними, які доступні для вивчення користувачу при відкритті відповідного розділу у розроблюваній системі.

Елемент «help» містить рядок із довідковою інформацією, необхідною для роботи із системою.

Елемент «questions» зберігає у собі масив обєктів типу «question», що містить у собі рядок з питанням, та правильною відповіддю на поставлене питання.

Розробка проводилась у відповідності з пріоритетами визначеними у попередніх розділах роботи. Для розробки було використано гнучку методологію розробки програмного забезпечення «Scrum». Спочатку було імплементовано більш пріоритетний функціонал, а потім він тестувався паралельно із розробкою менш пріоритетних задач.

Тестування під час розробки проводилось за допомогою юніт-тестів та компонентних тестів. Відлагоджування розроблюваної системи проводилось за допомогою вбудованих відлагоджувальних засобів інтегрованого середовища розробки «Xcode», що дозволяло поставити точку зупинки під час роботи програми у емуляторі, або на реальному пристрої.

Для контролю версій було використано систему контролю версій «Git». Це дозволило мати локальну історію кожного файлу з проекту розроблюваної системи та мати можливість повернутися до останньої працездатної версії системи у випадку внесення помилок при імплементації нового функціоналу системи.

Модульне тестування

У таблиці 3.1 представлено тестування методом «білого ящика» функції «Додання нового вузла графу», схема алгоритму якої зображує рисунок 2.3.

Таблиця 3.1 – Тестування «білого ящика»

Test Case	Вхідні дані	Очікуваний результат	Результат тестування
1	100, 100	Розміщення вузла	Passed
2	700, 322	Розміщення вузла	Passed
3	700, 228	Неможливо розмістити	Passed
4	88, 322	Розміщення вузла	Passed

Нижче представлено тестування методом «чорного ящика» функції «Пошук компонент зв’язності», схему алгоритму якої зображено на рисунку 2.4.

Таблиця 3.2 – Тестування «чорного ящика»

Вхідна умова	Правильний клас еквівалентності
В системі існує принаймні один вузол	Правильний пошук компонент зв’язності

4 ТЕСТУВАННЯ ТА РОЗГОРТАННЯ ПРОГРАМНОГО ПРОДУКТУ

4.1 Функціональне тестування

Для тестування вимог користувача було описано тест-кейси та проведено тестування результат якого приведено у таблиці 4.1.

Таблиця 4.1 – Опис тест-кейсів що підтверджують вимоги користувача

Дія	Очікуваний результат	Результат тестування
Вказання точки на екрані, що знаходиться ближче ніж на відстань діаметра вузла до інших вузлів.	Система повідомляє про неможливість розміщення вузла у вказаній точці	Passed
Вказання точки на екрані, що знаходиться більше ніж на відстань діаметра вузла від інших вузлів.	Система розміщує новий вузол у вказаній точці.	Passed
Вказання точки, що не належить жодному з елементів.	Система повідомляє про неможливість виконання даної дії.	Passed
Вказання точки, що належить якому-небудь з невибраних елементів.	Система пов’язує обрані вузли	Passed
Видалення вузла.	Система видаляє вузол та його зв’язки	Passed
Пройти тест на знання теорії.	Система виводить результат користувача.	Passed
Пройти тест на обхід дерева.	Система виводить результат користувача.	Passed
Зміна вузла.	Система зберігає зміни.	Passed
Видалення графу.	Система видаляє усі вузли графа та усі зв’язки між ними	Passed
Виведення довідкової інформації.	Система виводить вікно із довідковою інформацією.	Passed

Продовження таблиці 4.1

Виведення теоретичної інформації.	Система виводить вікно із теоретичною інформацією.	Passed
Виконання обходу графу.	Система позначає вузли графу цифрами	Passed
Зміна режиму роботи програми.	Система змінює режим роботи програми.	Passed
Перерисовування дерева.	Система перерисовує дерево.	Passed
Пошук компонент зв’язності.	Система демонструє компоненти зв’язності.	Passed
Згортання дочірніх вузлів дерева.	Система згортає дочірні вузли дерева.	Passed
Розгортання дочірніх вузлів дерева.	Система розгортає дочірні вузли дерева.	Passed
Пошук найкоротшого шляху між вузлами графу.	Система візуалізує найкоротший шлях.	Passed
Пошук найкоротшого шляху між непов’язаними вузлами графу.	Система повідомляє, що вузли не пов’язані.	Passed

Тестування бізнес-вимог користувача

Так як метою розроблюваної програмної системи є підвищення ефективності навчання теорії графів за рахунок автоматизованого процесу контролю вирішення завдань з теми «Графи», цю мету можна вважати досягнутою, так як були розв’язані необхідні для її досягнення задачі, а саме: забезпечення учня теоретичним матеріалом, візуалізація роботи алгоритмів, перевірка засвоєних знань.

Інструкція з встановлення

Для встановлення розробленої системи необхідно здійснити пошук даної системи у магазині додатків Apple “App Store” та дотримуючись вказівок на екрані інсталювати дану систему на пристрій.

Після виконання вищеописаних дій створена система повністю готова до роботи.

4.2 Інструкція з використання

На рисунку 4.1 нарисоване дерево у порядку, у якому його розмістив користувач.

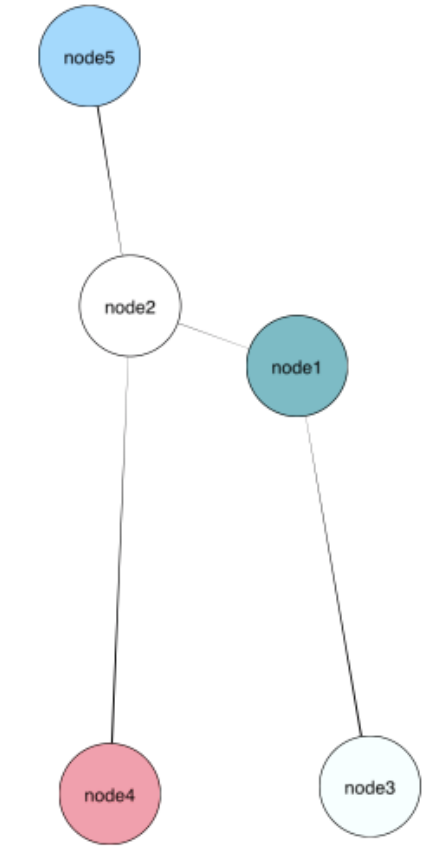

Рисунок. 4.1 – Головне вікно програми у режимі роботи з деревами

Нижче знаходиться те ж саме дерево, тільки перерисоване у класичному представленні дерева.

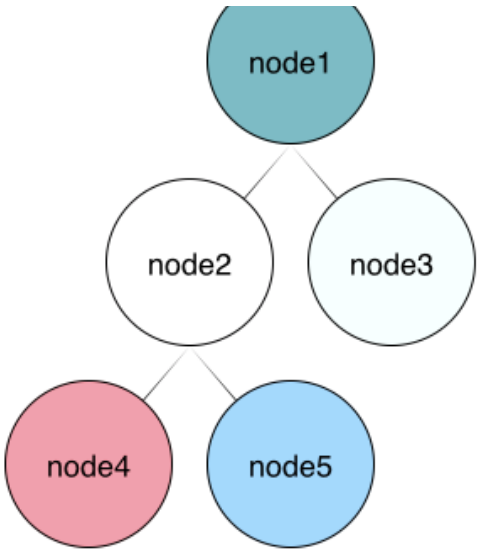

Рисунок 4.2 – Дерево у класичному представленні

При запуску програми у режимі роботи з деревами на екрані відображається дерево, збережене під час останнього використання програми, або тільки кореневий вузол у разі відсутності даних у пам’яті.

Дерево відображається у тому порядку, у якому вузли розмістив користувач.

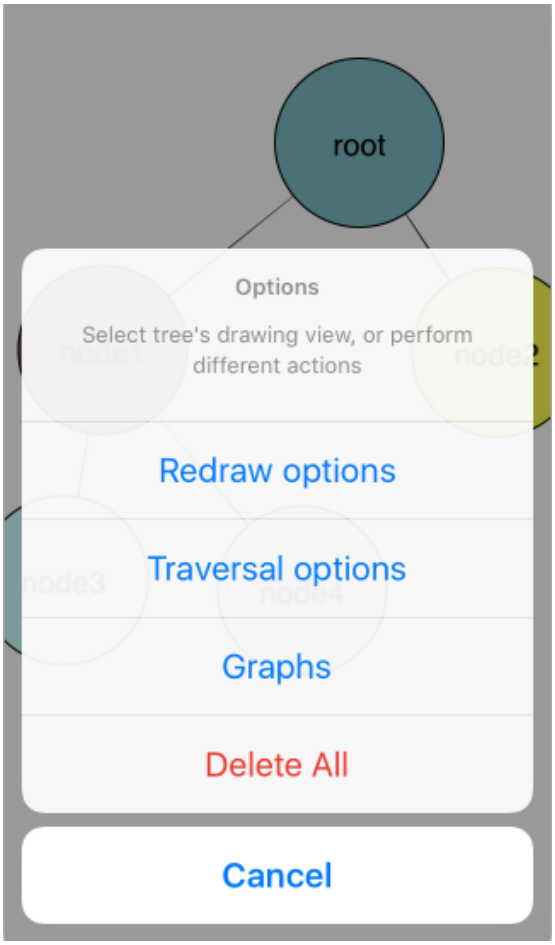

Рисунок 4.3 – Головне меню у режимі роботи з деревами

При натисканні та утриманні на вузлі дерева відкривається меню вузла, у якому користувачу пропонується змінити назву вузла, його колір або видалити вузол разом із дочірніми вузлами.

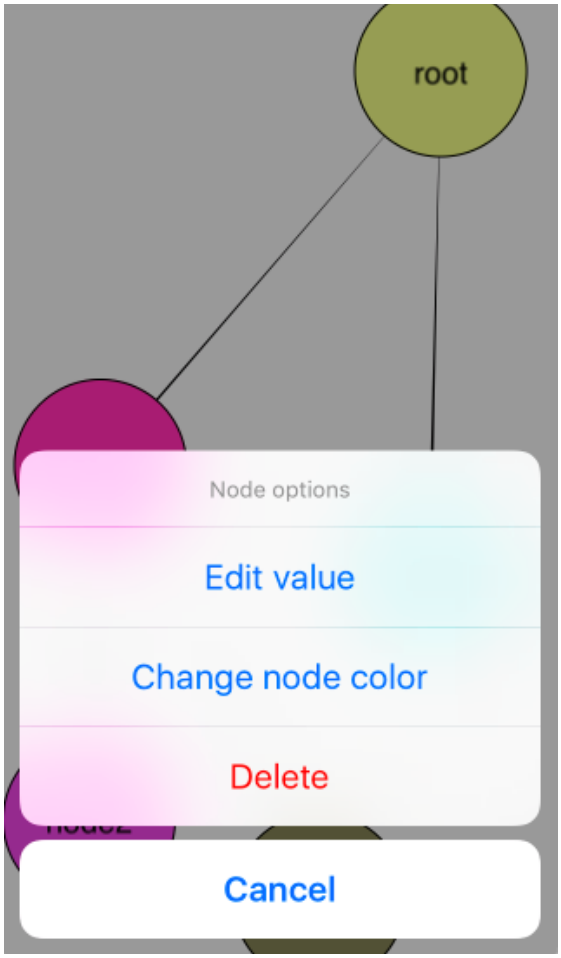

Рисунок 4.4 – Меню вузла у режимі роботи з деревами

На рисунку 4.5 зображено головне меню програми у режимі роботи з деревами та на фоні дерево, нарисоване з використанням користувальницького представлення.

Нижче знаходиться меню вузла дерева.

Воно містить пункти для зміни назви вузла дерева, зміни кольору вузла дерева, або видалити всі вузли дерева, окрім кореневого.

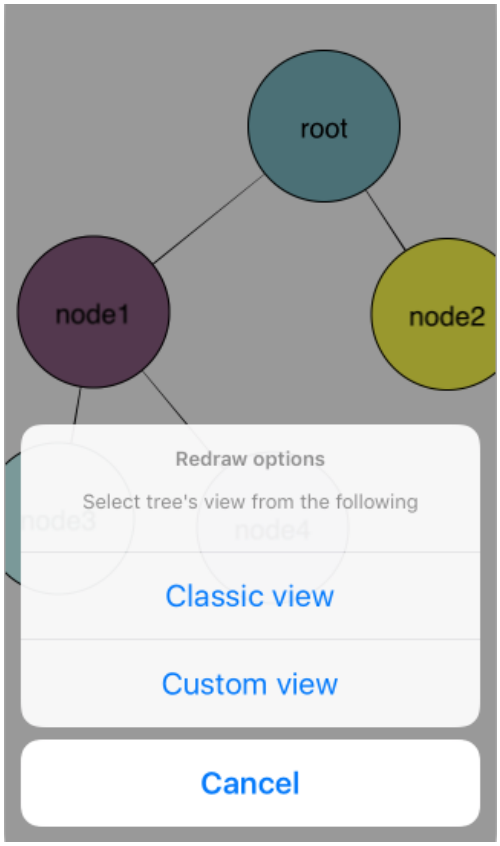

Рисунок 4.5 – Головне меню програми у режимі роботи з деревами

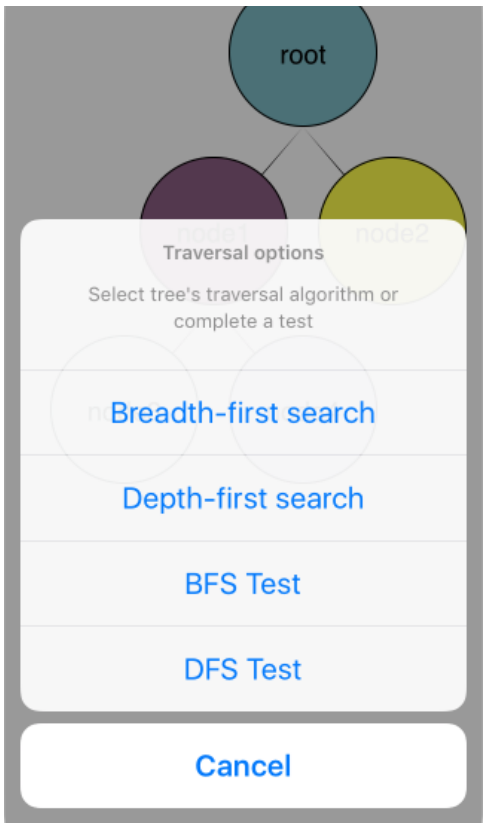

Рисунок 4.6 – Меню перерисування та обходу дерева

На рисунку 4.6 зображені меню перерисування дерева та меню обходів дерева.

Для створення нового вузла необхідно натиснути на екран пристрою і утримувати палець на якому-небудь вузлі. Після цього необхідно пересунути палець на бажане місце розміщення нового вузла.

Після виконання вищевказаних дій відкриється вікно для введення назви вузла. У вікні з введенням назви вузла необхідно ввести бажану назву вузла. Після введення назви новостворений вузол з’являється на екрані у місці яке було вказано користувачем.

У разі, якщо точка, вказана при створенні нового вузла, знаходиться занадто близько до інших вузлів дерева, що заважає розміщенню вузлів без накладання одне на одного, система повідомить, що розміщення вузла на цьому місці неможливе через брак місця.

Для переходу до головного меню режиму роботи з деревами необхідно два рази швидко натиснути по будь-якій області екрана, на якій відсутні вузли дерева.

Дане меню пропонує користувачу виконати наступні дії:

- перерисувати дерево, використовуючи класичне представлення дерева,
- перерисувати дерево у порядку, у якому вузли розмістив користувач,
- виконати обхід дерева у ширину,
- виконати обхід дерева у глибину,
- пройти тест на правильність обходу дерева у ширину або у глибину,
- змінити режим роботи на режим роботи із графами,
- видалити усі вузли дерева, окрім кореневого.

На рисунку 4.7 зображено результат обходу дерева у ширину.

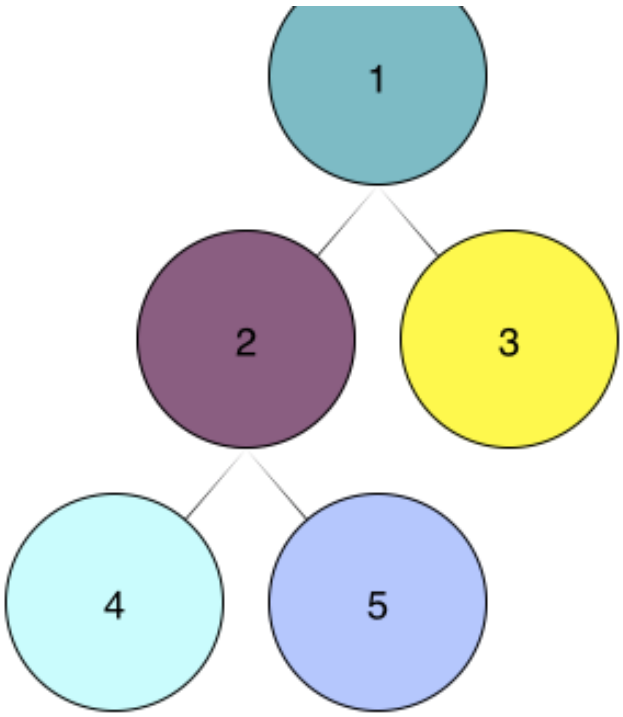

Рисунок 4.7 – Обхід дерева у ширину

При запуску програми у режимі роботи з графами на екрані відображається граф, збережений під час останнього використання програми, або білий фон екрану, у разі, якщо у пам’яті немає збережених графів.

Для переходу до головного меню режиму роботи з графами необхідно два рази натиснути по області екрана, на якій відсутні вузли графу.

Дане меню пропонує користувачу:

- відкрити вікно з теоретичними відомостями,
- відкрити вікно з довідкою по програмі,
- виконати пошук компонент зв’язності,
- відкрити вікно з тестовими питаннями,
- переключити режим роботи програми на режим роботи з деревами
- видалити всі вузли графа.

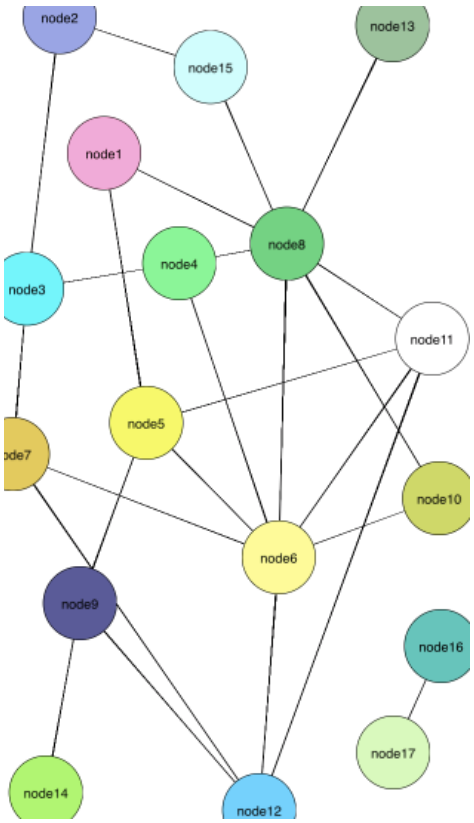

Рисунок 4.8 – Головне вікно програми у режимі роботи з графами

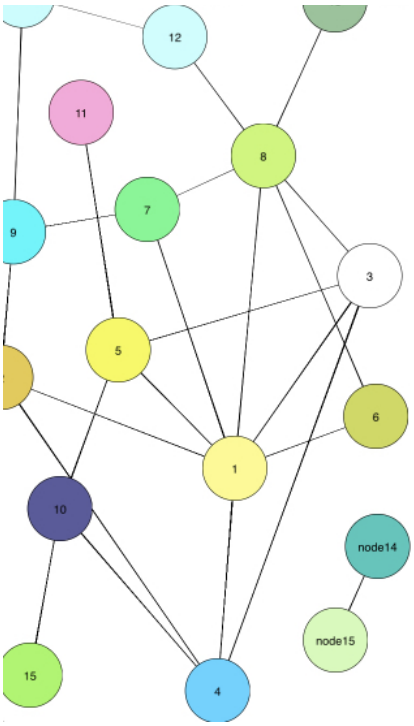

Рисунок 4.9 – Головне вікно програми після виконання алгоритму пошуку у ширину від вершини node8

На рисунку 4.8 зображено головне меню програми у режимі роботи з графами.

Нижче, на рисунку 4.9 зображено той самий граф, після виконання алгоритму пошуку у ширину від вершини node8.

В результаті виконання алгоритму пошуку у ширину від вершини node8, програма змінює назви всіх вершин на порядковий номер вершини при обході в ширину від вершини node8 до кожної іншої вершини, якої можна дістатися із вершини node8.

При цьому вершини, із яких не можна дістатися до вершини node8 не змінюють своїх назв, оскільки є недосяжними для даного алгоритму.

На цьому засновано алгоритм пошуку компонент звязності, розглянутий нижче.

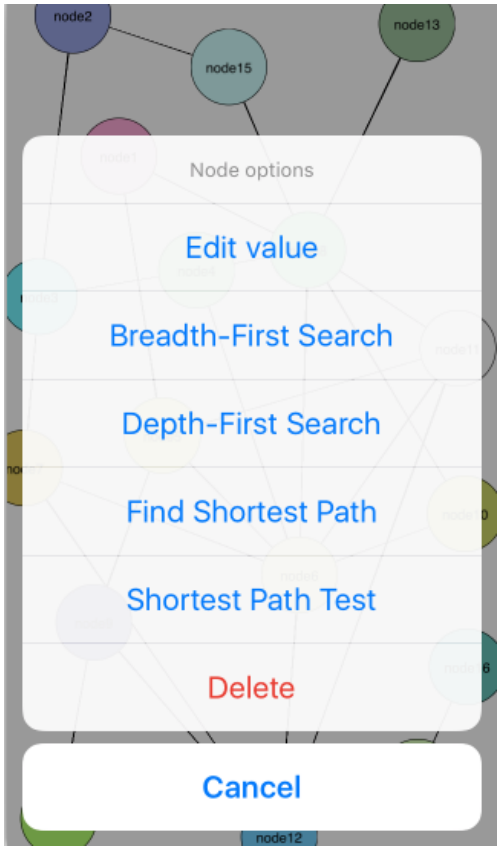

Рисунок 4.10 – Головне меню у режимі роботи з графами

На рисунку 4.10 зображено головне меню програми у режимі роботи з графами.

Дане меню містить такі пункти як:

- змінити назву вузла,
- виконати пошук у ширину, починаючи від цього вузла,
- виконати пошук у глибину від цього вузла,
- знайти найкоротший шлях до якого-небудь вузла,
- протестувати власні навички знаходження найкоротшого шляху,
- вивести теоретичні дані,
- вивести довідкові дані,
- виконати алгоритм пошуку компонент зв’язності,
- пройти тест теоретичних знань,
- перейти до режиму роботи з деревами,
- показати альтернативний шлях (стає активним тільки безпосередньо після пошуку найкоротшого шляху).

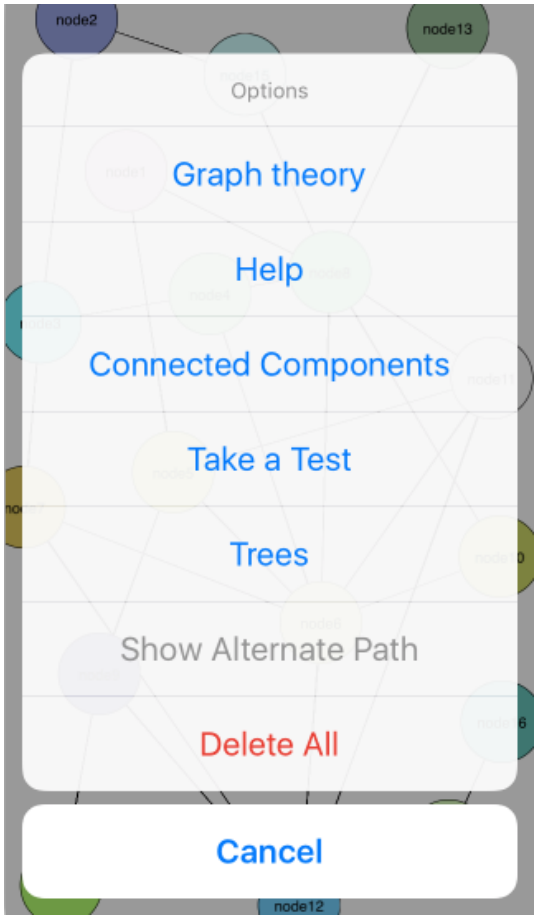

Рисунок 4.11 – Меню вузла у режимі роботи з графами

На рисунку 4.11 також наведено меню вузла у режимі роботи з графами.

При натисканні та утриманні на вузлі графа відкривається меню вузла. Дане меню містить наступні пункти:

- змінити назву вузла,
- виконати обхід у ширину,
- виконати обхід у глибину,
- почати пошук найкоротшого шляху із даного вузла
- видалити вузол разом із його зв’язками з іншими вузлами

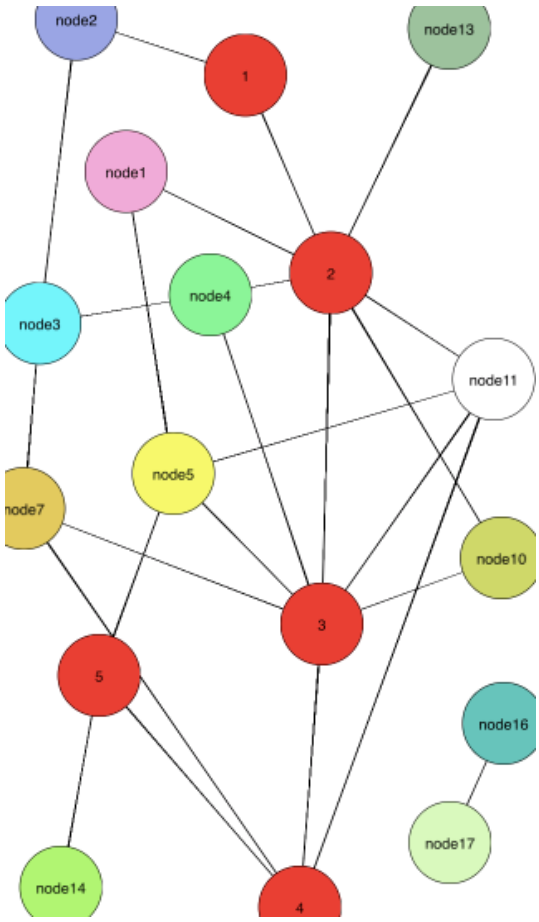

Рисунок 4.12 – Пошук найкоротшого шляху від вершини node6 до node9

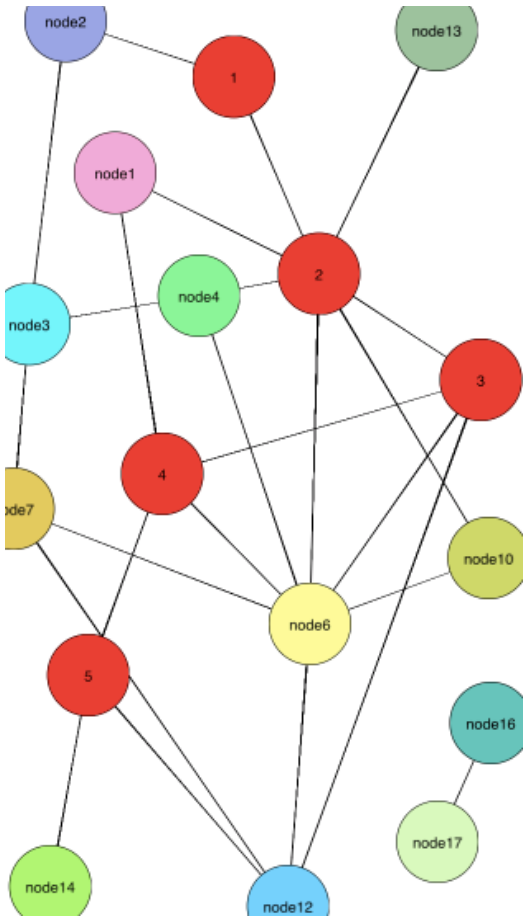

Рисунок 4.13 – Пошук альтернативного найкоротшого шляху від вершини node6 до node9

На рисунку 4.12 зображено результат виконання алгоритму пошуку найкоротшого шляху між визначеними вузлами та на рисунку 4.13 зображено альтернативний найкоротший шлях між цими вузлами.

Для створення нового вузла необхідно натиснути і утримувати палець на якому-небудь вузлі і пересунути палець на місце розміщення нового вузла.

Також можна додати вузол без зв’язку з іншими вузлами. Для цього необхідно натиснути і утримувати палець на області екрана у якій немає інших вузлів графа.

Після цього відкриється вікно для введення назви вузла і після введення новостворений вузол з’являється на екрані.

Рисунок 4.14 – Результат правильного пошуку найкоротшого шляху між вершинами node16 та node12

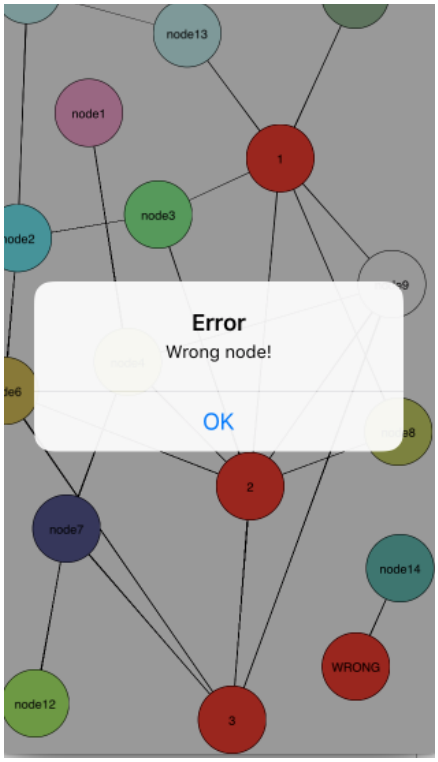

Рисунок 4.15 – Результат неправильного пошуку найкоротшого шляху між вершинами node16 та node12

На рисунках 4.14 та 4.15 зображено результати правильного та неправильного пошуку найкоротшого шляху.

При подвійному дотику на область екрана у якій знаходиться вузол графу, вузол візуально виділяється і при подвійному дотику на будь-який інший вузол графу створюється зв’язок, якщо ці вузли не були попередньо поєднані, або видаляється зв’язок, якщо вузли були поєднані.

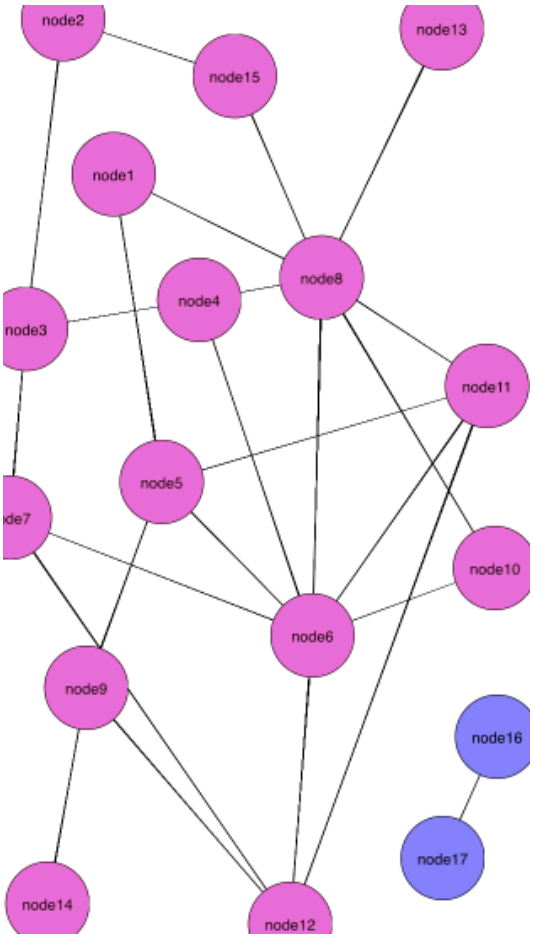

Рисунок 4.16 – Пошук компонент зв’язності

На рисунку 4.16 зображено результат виконання алгоритму пошуку компонент зв’язності.

Інтерфейси виведення даних та проходження тестів уявляють собою вікна, що подаються поверх інших вікон (модальні вікна у термінології Apple [2]).

Вікна що виводять теоретичні дані окрім власне текстових даних мають кнопку у верхньому лівому куті для повернення до попереднього вікна. За таким же принципом працюють і вікна з довідковою інформацією.

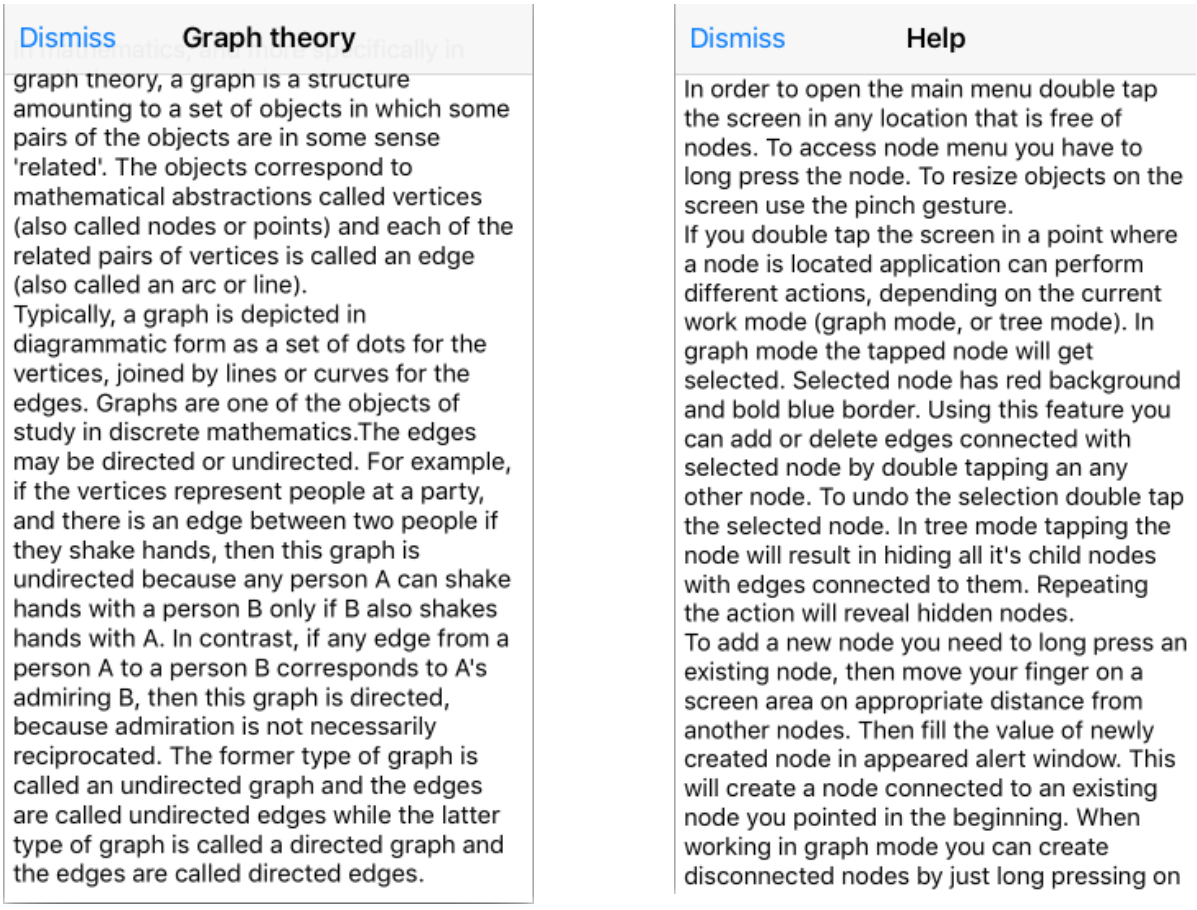

Рисунок 4.17 – Вікно виведення теоретичної та довідкової інформації

На рисунку 4.17 зображено вікна виведення теоретичної та довідкової інформації.

Вікно з інтерфейсом для проходження тестів окрім вказаних вище елементів також має кнопки вибору варіантів відповіді.

При натисканні на одну з цих кнопок система виведе повідомлення про правильність відповіді. Після цього на екрані з’явиться наступне питання.

Після проходження всіх тестових завдань виведеться вікно їх підсумковою оцінкою користувача.

Після цього система повертає користувача на перше питання та скидає його попередній результат.

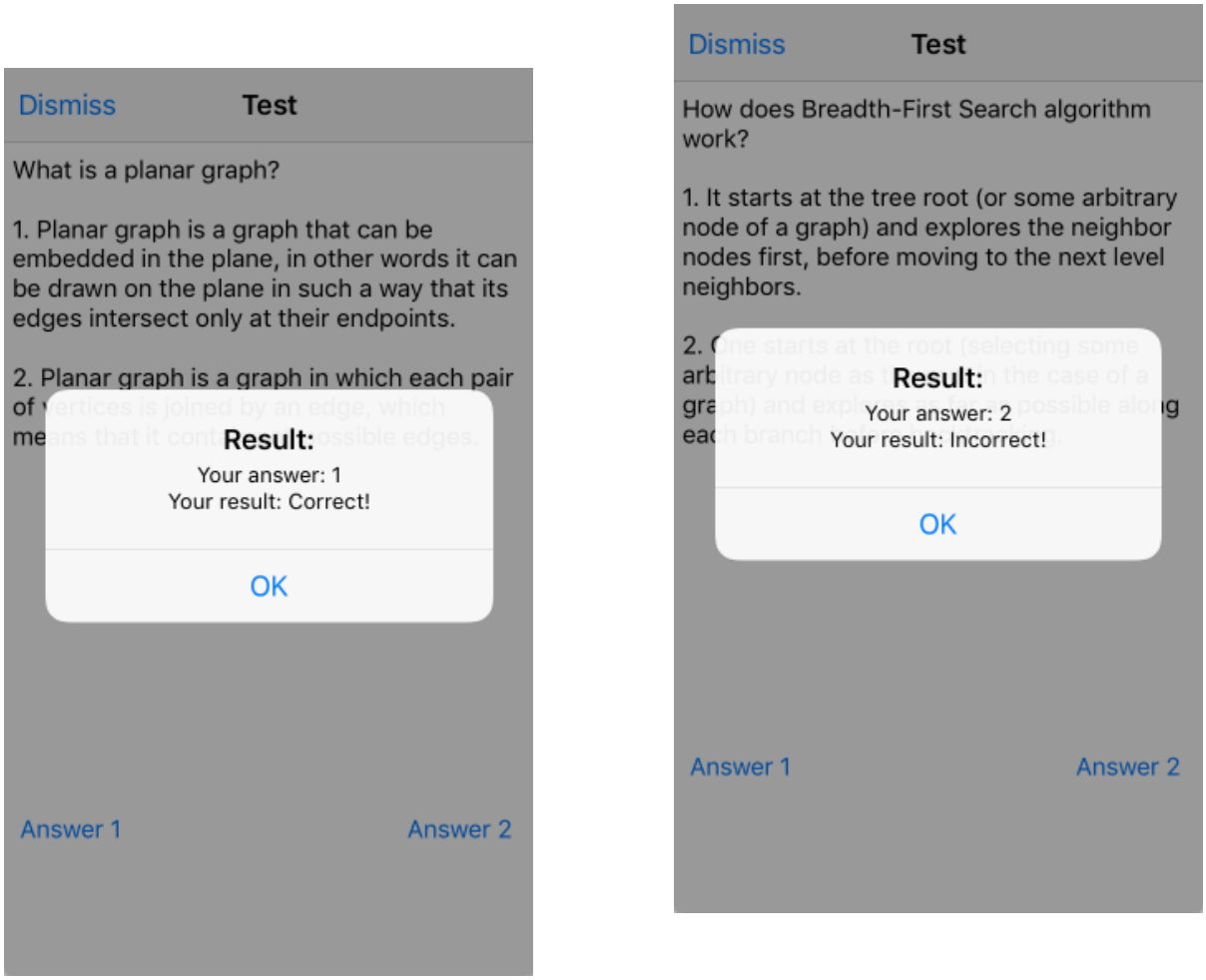

Рисунок 5.11 – Вікно тестового контролю

На рисунку 5.11 зображено вікна тестового контролю на яких оцінюються відповіді на тестові питання.

5 ВИЗНАЧЕННЯ РЕЗУЛЬТАТИВНОСТІ НАВЧАННЯ ТЕОРІЇ ГРАФІВ ЗА ДОПОМОГОЮ РОЗРОБЛЕНОЇ СИСТЕМИ

Опис проведеного дослідження

Результатом виконання вищенаведених розділів даної роботи є функціонуюча програмна система, що виконує поставлені задачі, поставлені у першому розділі роботи.

Так як дана система створювалась для полегшення вивчення основ теорії графів, було проведено дослідження впливу даної системи на результативність навчання студентів.

Для проведення даного дослідження було сформовано дві фокус-групи. Кожна із груп містила по п’ять студентів першого курсу напряму підготовки «Комп’ютерні науки». Обов’язковою умовою участі у наведеному дослідженні була відсутність у вказаних студентів знань прямо, чи опосередковано пов’язаних із теорією графів та наявність пристрою з операційною системою iOS. Наявність знань перевірялася шляхом особистої співбесіди з кожним із студентів, що брали участь у дослідженні.

Хід дослідження

Для визначення ефективності розробленої системи, першій групі студентів було вказано інсталювати розроблену систему для вивчення теорії графів. Після встановлення, їм було запропоновано прочитати наведену у системі стислу теоретичну довідку з теорії графів, ознайомитись із довідковою інформацією щодо використання програми, та почати роботу у середовищі для роботи з графами.

Після початку роботи з системою, студенти мали можливість ознайомитись із інтерфейсом користувача, створити власний граф, виконати на ньому ті алгоритми, що підтримуються системою, а також спробувати виконати ті ж алгоритми власноруч, без допомоги системи.

Другій групі студентів було запропоновано використовуючи наведені у мережі «Інтернет» самотужки вивчити такі теми як:

- Вступ до теорії графів. Поняття графу
- Різновиди графів
- Алгоритми обходу графів
- Алгоритми пошуку найкоротшого шляху

Обидвом групам було надано дві академічні години на вивчення теорії графів з використанням наданих їх групам засобів та матеріалів.

Перевірка результатів навчання

По закінченню відведеного на навчання часу, знання з теорії графів було перевірено у кожного із студентів, хто брав участь у дослідженні.

Знання перевірялись способом особистої співбесіди, яка включала в собі типові питання по вищеперерахованим темам та виставлення оцінки знань кожного із студентів за десятибальною шкалою.

Серед студентів які користувалися розробленою системою також було проведено опитування стосовно їх вражень від користування розробкою, а саме чи є такий спосіб навчання їх думку більш зручним порівняно із традиційними методами навчання як-то книги, лекції і тому подібне.

В свою чергу серед студентів, які вивчали теорію графів самостійно також було проведено опитування, чи зручно їм було використовувати матеріали з мережі «Інтернет», та яким методам навчання вони б надали перевагу.

Результати навчання та інтерпретація результатів

У підсумку проведеного дослідження було отримано наступні результати: середня оцінка за результатами перевірки знань по теорії графів для групи, що користувалася розробленою системою склала 4.4.

У той час як для групи, що вивчала теорію графів самостійно, середня оцінка склала 3.8.

За результатами даного дослідження, результативність вивчення теорії графів у групи, що користувалася розробленою системою зросла в середньому на 15%, порівняно із групою, що вивчала теорію графів самостійно.

Також, відповіді на додаткові запитання показали, що студентам, які користувалися розробленою системою було зручніше користуватися мобільним додатком, аніж традиційними методами навчання. Студенти, які вивчали графи самостійно, під час співбесіди також надали перевагу навчанню за допомогою мобільним додатком.

Це може свідчити про те, що мету створення даного програмного продукту було успішно досягнуто.

ВИСНОВКИ

У результаті виконання даної роботи було створено програмний продукт, що дозволяє користувачу створювати довільні графи, або дерева, виконувати деякі алгоритми на графах, тестувати знання алгоритмів користувачем, вивчати теорію та перевіряти знання теорії.

Під час розробки даного програмного продукту було проаналізовано предметну область, зібрано та проаналізовано вимоги до проектованої системи, було проаналізовано існуючі на момент розробки аналоги, специфіковано вимоги, проведено планування розробки, змодельовано проектовану систему із використанням UML-діаграм, описано етапи імплементації, проведено тестування створеної системи, а також проведено дослідження результативності вивчення теорії графів із використанням розробленої системи порівняно із традиційними методами навчання.

У результаті проведеного дослідження було визначено, що результативність вивчення теорії графів, а саме середня оцінка знань з теорії графів більша у групі тих студентів, котрі використовували для вивчення теорії графів представлену розробку, порівняно із групою студентів, які користувались тільки традиційними методами навчання.

Такі результати можуть свідчити про те, що мету даної роботи було успішно досягнуто.

СПИСОК ЛІТЕРАТУРИ

1. Кочан С. Программирование на Objective-C 2.0 / Стивен Кочан., 2014. – 608 с.
2. iOS Developer Library [Електронний ресурс] // Apple – Режим доступу до ресурсу: https://developer.apple.com/library/ios.
3. Программирование под iPhone, iPad [Електронний ресурс] – Режим доступу до ресурсу: http://www.imaladec.com.
4. iOS [Електронний ресурс] // Вікіпедія. – 2015. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/IOS.
5. Peres R. Model-View-Controller (MVC) in iOS: A Modern Approach [Електронний ресурс] / Rui Peres // raywenderlich.com. – 2016. – Режим доступу до ресурсу: https://www.raywenderlich.com/132662/mvc-in-ios-a-modern-approach.
6. Белов В. В. Теория графов / В. В. Белов, Е. М. Воробьев, В. Е. Шаталов. – Москва: Высшая школа, 1976. – 392 с.
7. Буч, Г. Язык UML. Руководство пользователя / Г. Буч. – М.: ДМК пресс, 2007. – 496 с.
8. Дистель Р. Теория графов / Рейнгард Дистель. – Новосибирск: Издательство института математики, 2002. – 336 с.
9. Структуры и алгоритмы компьютерной обработки данных [Електронний ресурс] // Академия Microsoft – Режим доступу до ресурсу: http://www.intuit.ru/studies/courses/648/504/lecture/11474.
10. Thompson M. CGGeometry [Електронний ресурс] / Mattt Thompson // NSHipster. – 2012. – Режим доступу до ресурсу: http://nshipster.com/cggeometry/.
11. JSON [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/JSON.