Форма № 17

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Кафедра програмних засобів

(найменування кафедри)

КУРСОВИЙ ПРОЄКТ (РОБОТА)

з дисципліни «Об'єктно-орієнтоване програмування»
(назва дисципліни)

на тему: Розробка програмного забезпечення моделювання структур графів з використанням ООП

Студента 1 курсу КНТ-113сп групи
спеціальності 121 Інженерія програмного забезпечення
освітня програма (спеціалізація) інженерія програмного забезпечення
Щедровського І. А.
(прізвище та ініціали)
Керівник асистент, Короткий О. В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Національна шкала _____
Кількість балів: _____Оцінка: ECTS _____

Члени комісії _____ Короткий О. В.
(підпис) (прізвище та ініціали)
_____ Каплієнко Т.І.
(підпис) (прізвище та ініціали)

2022 рік

Форма № 25

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Інститут, факультет Запорізький національний університет НУ «Запорізька політехніка». Факультет комп`ютерних наук і технологій ;
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації)

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
_____С.О. Субботін
"____" _____2022 року

З А В Д А Н Н Я

НА КУРСОВИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА

Щедровського Івана Андрійовича
(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка програмного забезпечення моделювання структур графів з використанням ООП
керівник проєкту (роботи) Короткий Олександр Володимирович, асистент,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом закладу вищої освіти від _____
2. Строк подання студентом проєкту (роботи) 21 грудня 2022 року
3. Вихідні дані до проєкту (роботи) розробити додаток згідно теми курсової роботи
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Огляд та аналіз сучасних методів та засобів проектування програмного забезпечення; 2. Проєктування програмного забезпечення системи; 3. Розробка програмного забезпечення системи; 4.Аналіз ефективності програмного забезпечення; 5. Розробка документів на супроводження програмного забезпечення.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5 Основна частина	Короткий О.В., асистент		

7. Дата видачі завдання 06 листопада 2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів курсового проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1.	Аналіз індивідуального завдання.	1 тиждень	
2.	Аналіз програмних засобів, що будуть використовуватись в роботі.	2 тиждень	
3.	Аналіз структур даних, що необхідно використати в курсової роботі.	3 тиждень	
4.	Затвердження завдання	4 тиждень	
5.	Вивчення можливостей програмної реалізації структур даних та інтерфейсу користувача.	5-9 тиждень	
6.	Аналіз вимог до апаратних засобів	9 тиждень	
7.	Розробка програмного забезпечення	9-13 тиждень	
8.	Проміжний контроль	10 тиждень	Розділи 1-5 ПЗ
9.	Оформлення, відповідних пунктів пояснювальної записки.	10-14 тиждень	Розділи 1-2 ПЗ
10.	Захист курсової роботи.	15 тиждень	

Студент _____ Щедровський І.А.
(підпис) (прізвище та ініціали)

Керівник проєкту (роботи) _____ Короткий О.В.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до курсової роботи містить 107 сторінок, 31 рисунок, 1 таблиця, 2 формули, 2 додатки, 8 джерел.

Пояснювальна записка складається з шести розділів.

Розділ «Огляд та аналіз сучасних методів та засобів проектування програмного забезпечення» містить в собі аналіз сучасних методів до проектування програмного забезпечення та аналіз різних сфер життя.

Розділ «Проектування програмного забезпечення системи» містить аналіз функцій системи, розробку UML діаграм використання, проєктування графічного інтерфейсу та постановку мети.

Розділ «Розробка програмного забезпечення системи» містить розробку структури програми, розробку UML діаграми класів та опис класів програмного забезпечення.

Розділ «Аналіз ефективності програмного забезпечення » містить аналіз застосунку на швидкодія та масштабованість, аналіз ефективності кожного компоненту системи, а також тестування всього програмного забезпечення.

Розділ «Розробка документів на супроводження програмного забезпечення» містить інструкцію для програміста, де є розгортка застосунку на локальній машині та на сервері, а також інструкцію користувача.

ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ, ГРАФИ, TYPESCRIPT, VITE, GIT, АЛГОРИТМИ, ОПТИМІЗАЦІЯ ВІЗУАЛІЗАЦІЇ ГРАФІВ

ЗМІСТ

ВСТУП

У сучасному світі, де технології безперервно розвиваються, моделювання структур графів стає ключовою складовою для вирішення різноманітних завдань у галузі комп'ютерних наук. Особливо актуальною стає розробка програмного забезпечення, яке не лише ефективно відтворює графічні структури, але і використовує об'єктно-орієнтований підхід для досягнення оптимальної функціональності та надійності.

Ця курсова робота спрямована на дослідження та розробку програмного забезпечення для моделювання структур графів з використанням об'єктно-орієнтованого програмування (ООП). ООП вже довго визнано як ефективний метод побудови програмних систем, але його застосування в контексті моделювання графів ще потребує ретельного вивчення та оптимізації.

Основною метою мого дослідження є створення програмного продукту, який буде не лише демонструвати високий рівень точності відображення графічних структур, але і забезпечувати зручний інтерфейс для користувача та широкі можливості взаємодії з графами. У цьому контексті я визначив ряд конкретних завдань, що включають в себе реалізацію основних операцій з графами, ефективну обробку великої кількості даних та взаємодію з іншими об'єктами програми.

Ця робота має значення для розширення можливостей в галузі моделювання структур графів, враховуючи не лише теоретичні аспекти, але і практичні можливості реалізації програмного забезпечення, що використовує ООП. Я вірю, що мій внесок допоможе вирішити виклики, пов'язані зі складністю та розміром графічних структур у сучасних інформаційних системах.

1 ОГЛЯД ТА АНАЛІЗ СУЧАСНИХ МЕТОДІВ ТА ЗАСОБІВ ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У сучасному інформаційному суспільстві велика увага приділяється інтенсивній розробці програмного забезпечення для моделювання структур графів. Це виявляється у насиченому ритмі досліджень та творчих зусиль, спрямованих на створення інструментів, сприяючих оптимізації аналізу та візуалізації графічних структур. Задача моделювання структур графів обумовлює необхідність постійного вдосконалення технічних рішень для забезпечення ефективної обробки та представлення величезних обсягів даних.

1.1 Огляд сучасного стану питання

Актуальність розробки програмного забезпечення для моделювання структур графів визначається неухильним збільшенням обсягу даних та виринаючою необхідністю в їх інтенсивному аналізі. Наголошується на тому, що в останніх наукових дослідженнях та практичних застосуваннях наростає потреба в постійному вдосконаленні методів та засобів обробки та візуалізації графічних структур. Використання об'єктно-орієнтованого програмування (ООП) у цьому контексті розкриває передові можливості для творення гнучких та легко розширюваних програм.

З огляду на географічний аспект, належить визначити, що проблеми моделювання графів залишаються актуальними на всіх континентах. Проекти та дослідження в цій сфері активно ведуться як у розвинених країнах, так і в країнах з розвиваючоюся економікою.

Останні досягнення в галузі розробки програмного забезпечення для моделювання графів характеризуються впровадженням передових методів оптимізації та візуалізації. Ключовим етапом у цьому напрямку є використання алгоритмів машинного навчання для автоматизації аналізу та класифікації графічних структур.

Дослідження також зосереджують увагу на створенні інтерактивних інтерфейсів та інструментів для полегшення взаємодії користувача з об'ємними даними. У зарубіжних наукових працях особлива увага приділяється розробці програм, які здатні ефективно працювати з розподіленими обчислювальними ресурсами.

Загальний аналіз поточного стану досліджень в галузі моделювання структур графів свідчить про активний розвиток методів та інструментів, що базуються на принципах ООП. Використання передових алгоритмів та технологій, таких як машинне навчання, сприяє оптимізації процесів аналізу та візуалізації графічних структур, зроблюючи їх більш доступними для різноманітної аудиторії.

1.2 Основні шляхи вирішення поставленої задачі

У сфері розробки програмного забезпечення для моделювання структур графів існує низка ключових напрямків, що визначають ефективний підхід до розв'язання поставлених завдань. При розгляді цих аспектів слід зосередитися на наступних напрямках:

– вдосконалення алгоритмів аналізу графічних структур;

– розробка ефективних методів візуалізації графів та їх взаємодії з користувачем;

– використання технологій машинного навчання для автоматизації обробки та класифікації графічних структур.

Вдосконалення алгоритмів аналізу графічних структур. Один із найважливіших компонентів успішного програмного продукту — це оптимізовані алгоритми аналізу графічних структур. Здійснення подальших досліджень та вдосконалення існуючих алгоритмів дозволить забезпечити точніше та швидше моделювання графів, що є важливим для задоволення високих вимог користувачів.

Розробка ефективних методів візуалізації графів та їх взаємодії з користувачем. Надійна візуалізація графів та їх зручна взаємодія з користувачем є ключовими елементами, що визначають успішність програми. Дослідження та розробка методів, спрямованих на поліпшення візуального відображення графічних структур та їх взаємодії, є необхідним етапом у процесі створення високоефективного програмного забезпечення.

Використання технологій машинного навчання для автоматизації обробки та класифікації графічних структур. Інтеграція технологій машинного навчання у розробку дозволяє автоматизувати процеси обробки та класифікації графічних структур. Використання цих інноваційних методів сприяє покращенню точності та ефективності аналізу, роблячи програмне забезпечення більш адаптованим та гнучким до різноманітних завдань.

Узагальнюючи зазначені напрямки, варто наголосити, що оптимальним шляхом розвитку є інтеграція цих підходів. Забезпечення взаємодії між алгоритмуми аналізу, методами візуалізації та застосуванням машинного навчання визначатиме високу ефективність та конкурентоспроможність розробленого програмного забезпечення для моделювання структур графів.

1.3 Формулювання мети досліджень

Основною метою цього дослідження є створення високоефективного програмного забезпечення для моделювання структур графів, використовуючи концепції об'єктно-орієнтованого програмування (ООП). Задача полягає в розробці комплексного підходу, що поєднує в собі передові алгоритми, інноваційні методи візуалізації та застосування технологій машинного навчання.

Дослідження націлене на створення високоефективних алгоритмів, спрямованих на аналіз та візуалізацію графів, враховуючи особливості об'єктно-орієнтованого програмування. Передбачається, що результатом

досліджень стануть нові методи візуалізації та аналізу, які взаємодіють із користувачем з використанням ООП.

Важливим аспектом є створення інтерактивного інтерфейсу, спрямованого на забезпечення ефективної взаємодії користувачів з графічними структурами. Цей інтерфейс буде розроблено з урахуванням передових підходів та принципів ООП, надаючи користувачам інтуїтивно зрозумілі та потужні засоби взаємодії.

Очікується, що результатами досліджень будуть не лише нові алгоритми для ефективного аналізу та візуалізації графів, але й реалізований інтерактивний інтерфейс. Використання технологій машинного навчання буде сприяти автоматизації процесів аналізу та класифікації, підвищуючи точність та ефективність обробки об'ємних даних у контексті моделювання графічних структур.

2 ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

В даному розділі розглядається процес проектування програмного забезпечення системи моделювання структур графів. Система призначена для зручного візуального моделювання графів, що може бути використана як користувачами, так і розробниками для відображення роботи різних алгоритмів.

2.1 Постановка мети

Основною метою цієї курсової роботи є розробка програмного забезпечення для моделювання структур графів з використанням об'єктно-орієнтованого програмування (ООП). Програмний продукт повинен демонструвати високий рівень точності відображення графічних структур та забезпечувати зручний інтерфейс для користувача.

2.2 Аналіз функцій системи

Аналіз функцій системи є важливим етапом у розробці програмного забезпечення для моделювання структур графів. Він дозволяє визначити основні функціональні вимоги до системи, які є необхідними для її подальшого проектування та реалізації. Детальний розгляд функцій дозволяє забезпечити повноту та ефективність розроблюваного програмного продукту.

У рамках курсової роботи передбачається розробка програмного забезпечення для моделювання структур графів з використанням об'єктно-орієнтованого програмування. Основними функціями системи є:

– відображення графа. Граф повинен відображатися на сторінці на всю сторінку. При цьому повинні бути відображені всі його елементи, а саме: вершини, ребра та їхні властивості;

– запуск алгоритмів. Алгоритми повинні можна запускати через графічний інтерфейс. При цьому повинні бути доступні всі необхідні параметри для запуску алгоритму;

– можливість створення та видалення нод. Користувач повинен мати можливість створювати нові вершини та видаляти існуючі;

– можливість створення, зміни та видалення граней. Користувач повинен мати можливість створювати нові ребра, змінювати їхні властивості та видаляти існуючі;

– міжвіконна взаємодія. Система повинна забезпечувати взаємодію між різними вікнами. Наприклад, користувач повинен мати можливість перетягувати вершини з одного вікна в інше;

– наявність декількох режимів роботи застосунку. Система повинна підтримувати кілька режимів роботи, наприклад, напрямлений та не напрямлений граф, граф з вагами та без;

– візуальне підсвічування виконання алгоритму. При виконанні алгоритму він повинен підсвічуватись на графіку. Це дозволить користувачеві простежити за його ходом;

– можливість переміщуватись по внутрішнім координатам. Користувач повинен мати можливість переміщуватись по графу, використовуючи внутрішні координати.

Відображення графа є однією з основних функцій системи. Граф повинен відображатись на сторінці на всю сторінку. При цьому повинні бути відображені всі його елементи, а саме: вершини, ребра та їхні властивості.

Відображення вершин може здійснюватися за допомогою різних способів, наприклад, у вигляді точок, прямокутників, кіл тощо. Ребра можуть відображатися у вигляді ліній, стріл тощо. Властивості вершин і ребер можуть відображатися у вигляді текстових позначок або графічних елементів.

Алгоритми повинні можна запускати через графічний інтерфейс. При цьому повинні бути доступні всі необхідні параметри для запуску алгоритму.

Параметри алгоритму можуть бути представлені у вигляді текстових полів, списків, діалогових вікон тощо. При запуску алгоритму повинні бути перевірені всі введені параметри. Якщо параметри введені неправильно, алгоритм не повинен запускатися.

Користувач повинен мати можливість створювати нові вершини та видаляти існуючі.

Створення вершини може здійснюватися за допомогою кнопки або меню. При створенні вершини користувач повинен ввести її ім'я та інші необхідні параметри.

Видалення вершини може здійснюватися за допомогою кнопки або меню. При видаленні вершини всі ребра, які ведуть до цієї вершини, також повинні бути видалені.

Користувач повинен мати можливість створювати нові ребра, змінювати їхні властивості та видаляти існуючі.

Створення ребра може здійснюватися за допомогою кнопки або меню. При створенні ребра користувач повинен вибрати початкову та кінцеву вершини ребра, а також ввести його властивості.

Зміна властивостей ребра може здійснюватися за допомогою кнопки або меню. При зміні властивостей ребра користувач повинен внести необхідні зміни.

Видалення ребра може здійснюватися за допомогою кнопки або меню. При видаленні ребра воно буде видалене з графа.

Система повинна забезпечувати взаємодію між різними вікнами. Наприклад, користувач повинен мати можливість перетягувати вершини з одного вікна в інше.

Міжвіконна взаємодія може здійснюватися за допомогою об'єктів, які можуть переміщуватись між вікнами.

2.3 Проєктування графічного інтерфейсу

Розробка графічного інтерфейсу (GUI) є ключовим етапом у створенні програмного забезпечення для моделювання структур графів. Інтерфейс повинен бути зручним, інтуїтивно зрозумілим та забезпечувати зручність взаємодії користувача з системою.

Основні елементи інтерфейсу:

- поле для відображення графа;
- меню;
- контекстне меню;
- панель на якій буде форма для запуску алгоритмів.

Поле для відображення графа - основний візуальний елемент, що представляє собою графічне відображення нод (вузлів) та ребер графу.

Меню містить кнопки, що дозволяють користувачу виконувати різні дії, такі як відкриття панелі для запуску алгоритмів або ж запускати їх напряму.

Контекстне меню – дозволяє взаємодіяти з окремими елементами графу . Воно виводиться за правим кліком мишею. Містить опції для швидкого виклику функцій, таких як видалення старих нод або ж створення нових.

Панель – блок який відображається справа на єкрані та може в собі рендерити різні форми, які потрібні для вказання даних для запуску алгоритмів.

Меню розподілене на логічні розділи, відповідальні за різні аспекти взаємодії з графом. Кнопки та опції мають текстовий та графічний вигляд для забезпечення належного рівня інтуїтивності та доступності.

Першою функцією є відкриття іншого пресету. Всього в проєкті є 4 пресети між якими можна перемикатись та редагувати кожен окремо.

Другою функцією є відображення напрямків ребер. Включення або виключення відображення напрямку ребер, що корисно при роботі з орієнтованими графами. Це не тільки візуально, алгоритми також будуть враховувати це

Третьою функцією є відображення ваги ребер. Можливість ввімкнення або вимкнення відображення ваги ребер на графі для наглядності та аналізу. Якщо вага вимкнена – алгоритми будуть вважати, що вона дорівнює 1 для всіх ребер

Контекстне меню може взаємодіяти з окремими елементами графу. Воно динамічно змінюється відповідно до контексту та містить деякі опція для зміни елемента, на якому воно було викликано.

Якщо контекстне меню було викликано в пустому місці показується тільки один елемент - Створити нову ноду. Також в контекстних меню передбачені шорткати, за допомогою яких можна швидко обирати яку саме операцію необхідно зробити.

Якщо контекстне меню було викликано на ноді показується один пункт меню - Видалити ноду.

Відображення графів відбувається в основному вікні програми. Кожна нода представляється графічним елементом, а ребра відображаються лініями між відповідними нодами. Графічний інтерфейс повинен бути плавним, зручним у використанні та забезпечувати достатню простоту для розуміння структури графу.

Графічний інтерфейс дозволяє користувачеві динамічно змінювати вигляд графу. Зокрема, можливість масштабування(через масштаб браузера) та переміщення для забезпечення оптимального перегляду великих та складних структур.

Для реалізації графічного інтерфейсу буде використано HTML та CSS.

Графічний інтерфейс розробляється з урахуванням принципів зручності використання. Це включає в себе інтуїтивний дизайн елементів управління, легкий доступ до основних функцій та зручне розташування елементів інтерфейсу.

Застосунок повинен бути сумісний з різними операційними системами (Windows, Linux, MacOS) та різними браузерами (Chrome, Firefox, Opera). Це

забезпечить універсальність програмного забезпечення та розширить коло його користувачів.

2.4 Розробка структури системи

Структурна модель системи, показана на рисунку 2.1, визначає основні компоненти та їх взаємозв'язки.

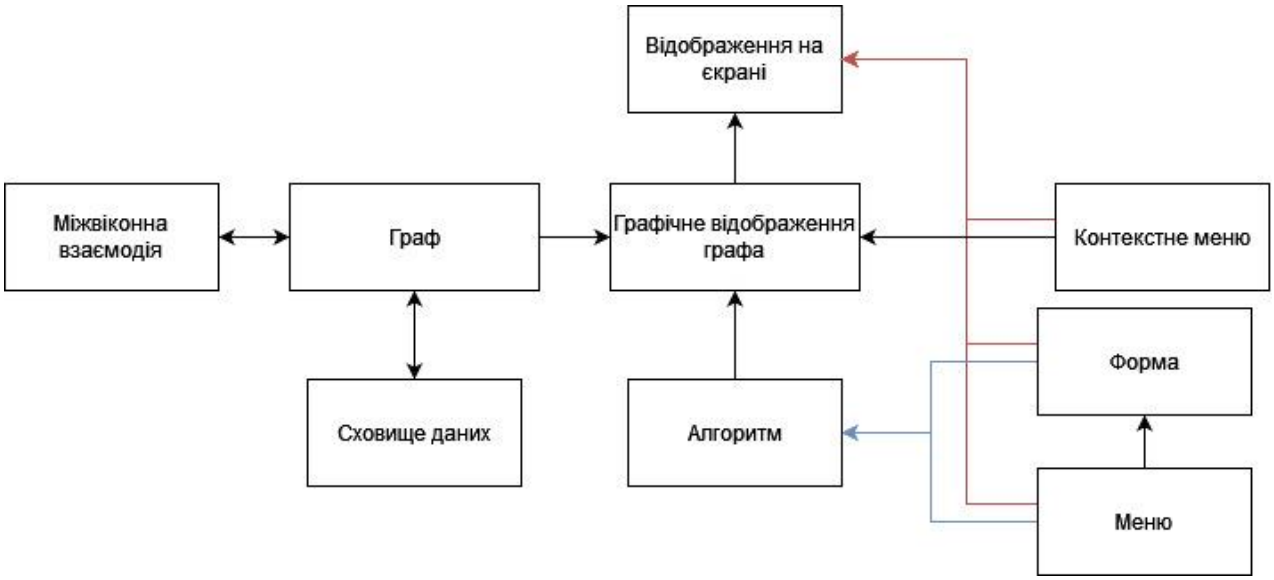


Рисунок 2.1 – Структурна модель застосунку

Система має основні компоненти, які представленні на таблиці 2.1:

Таблиця 2.1 – Опис компонентів системи

Назва компонента	Опис компонента
Відображення графа	Це основний компонент системи
Контектне меню	Потрібне для створення та видалення нод
Меню	Потрібне для відкриття форми та запуску алгоритмів. Основна точка взаємодії з користувачем
Форма для запуску алгоритму	Потрібна для запуску алгоритму з деякими вхідними даними

Продовження таблиці 2.1

Алгоритм	Алгоритм який виконується вважається окремим компонентом
Сховище даних	Потрібне для збереження стану застосунку, такого як внутрішній зсув та пресети
Міжвіконна взаємодія	Окремий складний компонент

Центральним елементом є компонент відображення графа, що виступає як основний модуль системи. Він відповідає за графічне представлення структури графа на екрані, забезпечуючи зручну взаємодію з користувачем та аналіз графічної інформації.

Контекстне меню відповідає за можливість створення та видалення вузлів графа. Цей компонент дозволяє користувачеві динамічно редагувати структуру графа, адаптуючи її під конкретні вимоги та завдання.

Меню, яке є основною точкою взаємодії з користувачем, відповідає за відкриття форми та запуск алгоритмів. Це важливий компонент, який забезпечує зручний доступ до основних функцій системи.

Форма для запуску алгоритмів використовується для введення вхідних даних та запуску алгоритму. Цей компонент дозволяє користувачеві взаємодіяти з алгоритмуми, вказуючи необхідні параметри та спостерігаючи за їхнім виконанням.

Алгоритм, що виконується, розглядається як окремий компонент системи. Цей елемент відповідає за реалізацію конкретного алгоритмічного підходу та взаємодію з іншими компонентами.

Сховище даних використовується для зберігання стану застосунку, таких як внутрішній зсув та пресети. Цей компонент забезпечує постійний доступ до поточного стану системи та можливість збереження його для подальшого використання.

Міжвіконна взаємодія розглядається як окремий складний компонент, що забезпечує гармонійну взаємодію різних вікон системи та підтримує цілісність користувацького інтерфейсу.

Система використовує об'єктно-орієнтований підхід для моделювання компонентів. Наприклад, інтерфейси та класи використовуються для представлення нод, ребер, GUI, контролера та інших об'єктів системи. Це дозволяє створювати гнучкі та розширювані об'єктні структури, сприяє повторному використанню коду та підтримці чистоти дизайну.

Компоненти системи взаємодіють динамічно, реагуючи на події та зміни, що виникають у процесі взаємодії з користувачем або виконання алгоритмів. Це забезпечує живий та відзивчивий характер системи.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

3.1 Розробка UML діаграми класів

На рисунку 3.1 представлена UML діаграма класів.

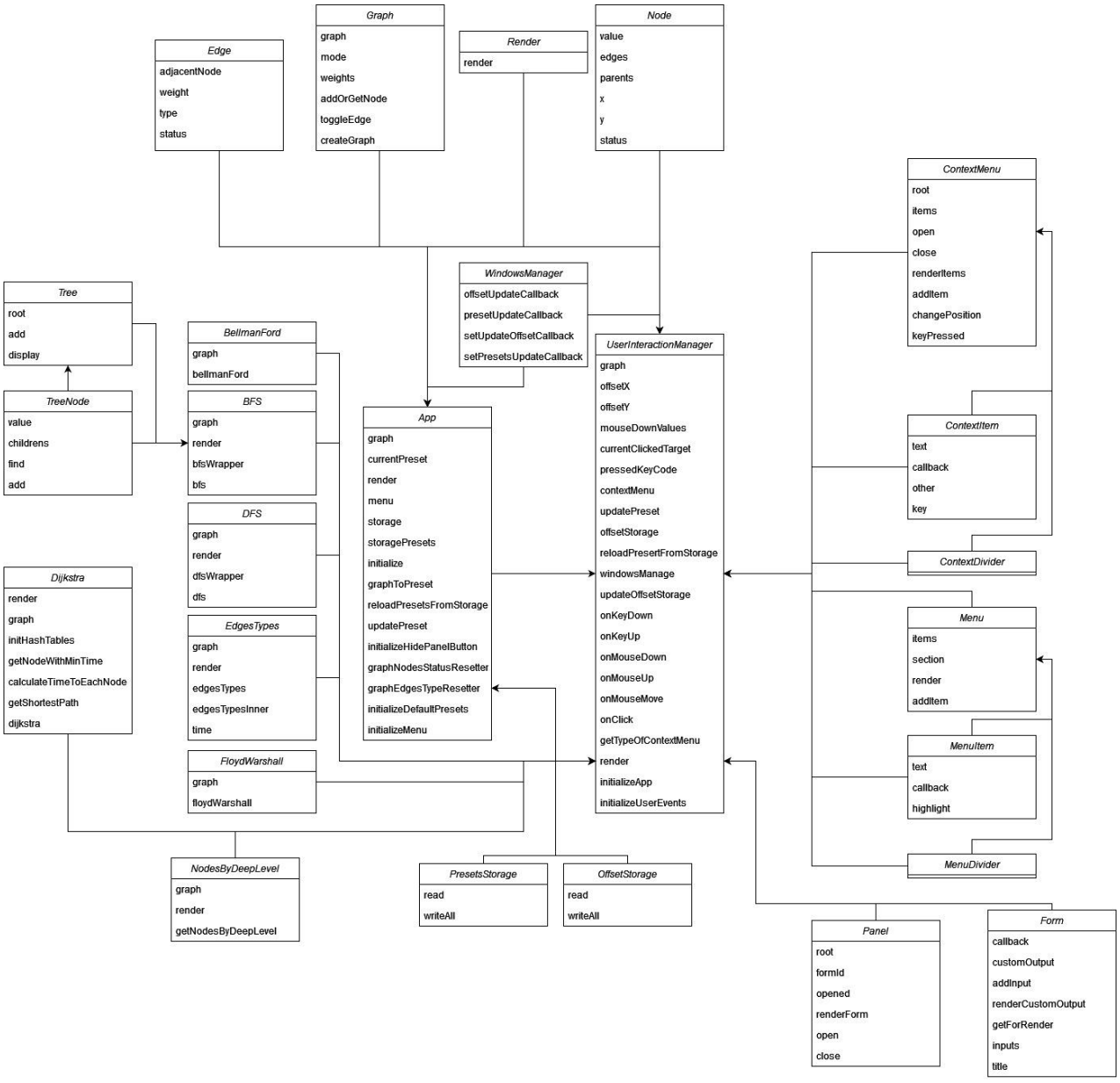


Рисунок 3.1 – UML діаграма класів

3.2 Розробка UML діаграми використання

UML діаграма використання визначає, які функції можуть бути використані користувачами. До основних входять створення та редагування графів, взаємодія з контекстним меню та основними функціями графічного інтерфейсу.

На рисунку 3.2 показана UML діаграма використання.

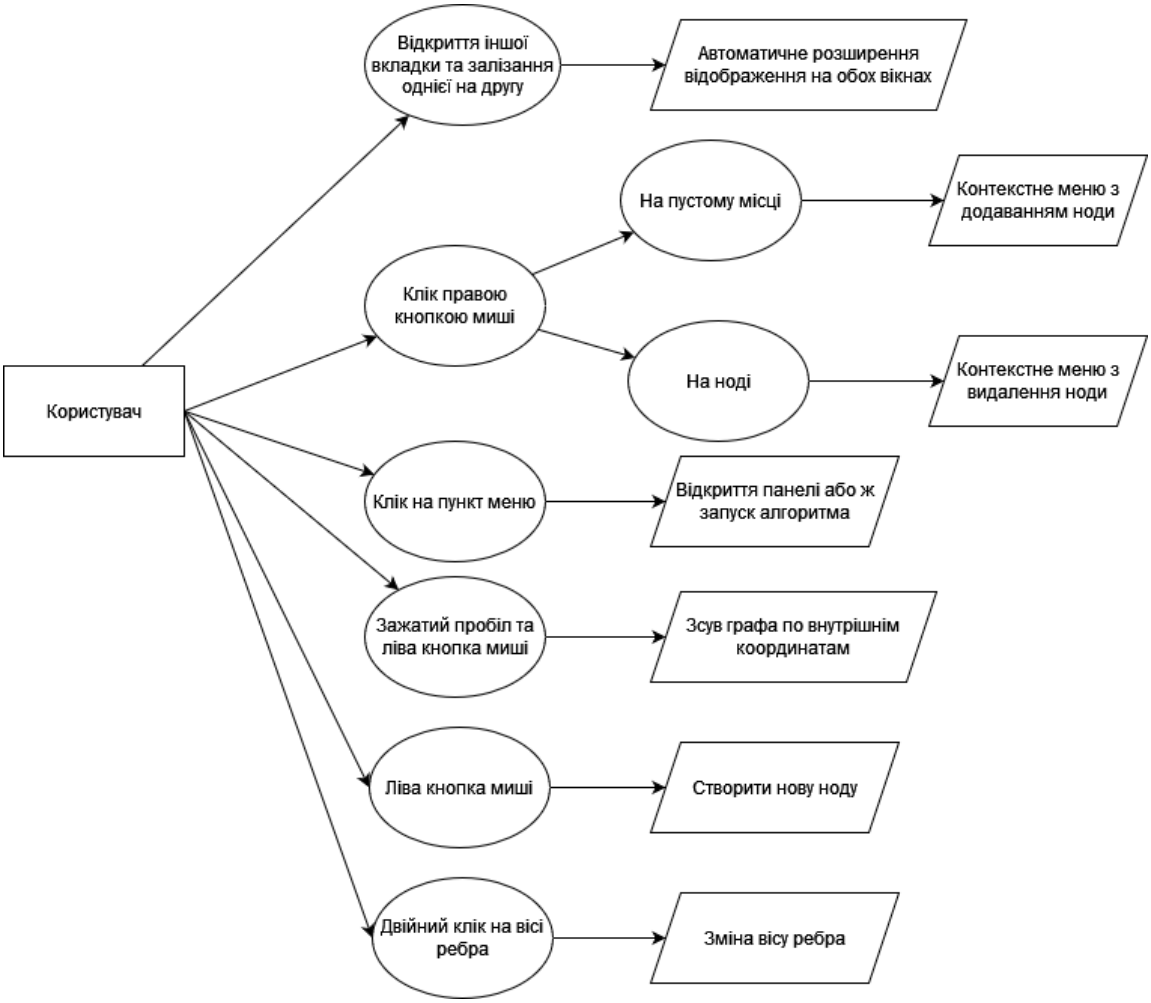


Рисунок 3.2 – UML діаграма використання

3.3 Опис класів програмного комплексу

Першим основним класом програми є точка графу – нода. Клас Node має наступні властивості:

– value – Значення ноди, яке відображається при візуалізації графа;

– edges – Множина граней які виходять з даної точки;

– parents – Map всіх батьківських нод. Батьківська нода – нода, з якої виходить грань через яку можна попасти в цю ноду;

– x – Положення на візуалізації по координаті X;

– y – Положення на візуалізації по координаті Y;

– status – Статус ноди. Потрібен для візуального відображення. Всього має 4 значення: default - біла, progress - жовта, done - зелена, passed - сіра.

Другим основним класом є грань – Edge. Грань має наступні властивості:

– adjacentNode – Нода, на яку посилається ця грань;

– weight – Вага грані. Може бути від’ємною;

– status – Статус грані. Це поле потрібно для реалізації можливості включати та виключати направлений режим. Це поле має два значення: standart – звичайна грань та no-direction – грань яка працює тільки коли виключений напрямлений режим показу;

– type – Тип грані. Потрібен для візуального виділення грані. Може мати наступні значення: default – звичайна грань, forward – рожева, cross – зелена та back – блакитна.

Третій клас на основі якого працює весь застосунок – клас Graph.

Цей клас має наступні властивості та методи:

– graph – Map, де в якості ключа число або строка, а в якості значення – нода;

– mode – Напрямлений або не напрямлений граф. Це поле потрібно для правильної роботи алгоритмів;

– weights – Чи має граф ваги, або ж ні. Це поле потрібно для правильної роботи алгоритмів;

– addOrGetNode() – Метод, який приймає данні графа, тобто Map, та значення ноди. Цей метод або ж повертає ноду, якщо вона вже існує, або ж створює її та повертає;

– toggleEdge() – Метод, який приймає три параметра: ноду, з якої буде йти грань, ноду в яку буде йти грань та вагу ребра. Цей метод створює грань, якщо між цими нодами немає зв’язку. Якщо між нодами є зв’язок, але тип цієї грані – no-direction – змінює його на standart. Якщо між нодами є зв’язок, але тип цієї грані standart – змінює на no-direction. Але якщо тип грані direction і грань яка йде з 2 ноди в першу є no-direction – обидві видаляються;

– createGraph() – Метод, який створює граф з готового пресету. Готовий пресет представляє собою масив нод та масив граней.

Розглянемо класи однієї з найважливіших функцій – класи зберігання даних

Для зберігання зсуву по внутрішнім координатам існує клас OffsetStorage. Цей клас має наступні властивості та методи:

– key – Ключ для зберігання даних в сховищі даних;

– writeAll – Метод, який приймає зсув та записує його до сховища;

– read – Метод, який читає данні зі сховища, валідує та повертає.

Для зберігання пресетів існує клас PresetStorage. Цей клас має наступні властивості та методи:

– key – Ключ для зберігання даних в сховищі даних;

– writeAll – Метод, який приймає великий об’єкт пресета та записує його до сховища даних;

– read – Метод, який читає данні зі сховища, валідує та повертає.

Однією з дуже важливих функцій програми є меню. Меню представляє собою головний клас для самої меню, а також два додаткових, один для елемента меню, другий для роздільника елементів.

Головний клас меню – Menu. Цей клас має наступні властивості та методи:

– items – Елементи меню які потрібно буде відображати;

– section – Елемент, в який будуть рендеритись;

– render() – Відрендерити меню в section;

– addItem() – Додати елемент меню до масиву елементів. Елементом меню є MenuItem або ж MenuDivider.

Для відображення напрямленого графа використовуються математичні формули. Вони використовуються саме для визначення позиції стрілки відносно ноди та її градус повороту для правильного відображення

Для цього використовуються дві математичні формули. Перша – формула знаходження кута між двома векторами, формула представлена під номером 1.

$$a = \operatorname{acos}\frac{\vec{a} * \vec{b}}{|\vec{a}| * |\vec{b}|} \quad (1)$$

Таким чином ми можемо знайти градус кута між проєкцією вектора a на вісь X. А тепер потрібно визначити позицію точки в прямокутних координатах, тобто X та Y. Для цього можна використати формулу полярних координат, яка представлена під номером 2

$$\begin{cases} x = r\cos\varphi \\ y = r\sin\varphi \end{cases} \quad (2)$$

Клас для елемента меню – MenuItem, має наступні властивості:

– text – Текст елемента меню;

– callback() – Функція зворотнього виклику, яка буде викликана після кліку по цьому елементу;

– highlight – Властивість, яка потрібна для підсвічування активного елемента меню.

Клас для роздільника меню – MenuDivider. Цей клас не має ніяких властивостей та методів.

Оскільки деякі алгоритми потребують вхідних даних для запуску – потрібна деяка форма, куди можна записувати значення та місце, де ця форма, як і інші форми або ж кастомні елементи, можуть відображатись.

Для створення форми існує клас Form, який має наступні методи та властивості:

– inputs – Масив інпутів, які будуть відображатись;

– title – Назва форми;

– callback – Функція зворотнього виклику, яка буде викликана при submit форми;

– customOutput – Спеціальне поле нижче форми куди алгоритми зможуть виводити інформацію про результат виконання;

– addInput() – Додати новий елемент вводу;

– renderCustomOutput() – Відобразити для користувача спеціальне поле для виводу інформації алгоритму;

– getForRender() – Взяти готову форму, щоб потім відрендерити, наприклад, в Panel.

Користувачу зручно, якщо він клацає на пункт меню і в нього з’являється деяка панель справа де є форма для запуску алгоритму.

Для реалізації цієї панелі існує клас Panel, який має наступні властивості та методи:

– root – Елемент DOM дерева, куди буде відображатись форма;

– formId – Унікальний ідентифікатор форми, потрібний для правильної реалізації закриття панелі при повторному кліці на елемент меню з однаковим алгоритмом;

– renderForm() – Відобразити форму в панелі;

– open() – Показати панель користувачу на єкран;

– close() – Скрити панель з поля зору користувача;

– opened – Поле, яке необхідно для реалізації закриття/відкриття панелі.

Також в програмі реалізоване контекстне меню, яке потрібно для більш зручного створення та видалення нод – ContextMenu. Цей клас має наступні властивості та методи:

– root – Елемент DOM дерева, в який буде відображатись контекстне меню;

– items – Елементи контекстного меню, які будуть відображатись;

– open – Відкрити контекстне меню;

– close – Закрити контекстне меню;

– renderItems() – Відобразити елементи в контекстному меню;

– addItem() – Додати новий елемент;

– changePosition() – Змінити позицію контекстного меню;

– keyPressed() – Спрацьовує при натисканні клавіши на клавіатурі лише коли меню відкрите. Потрібно для роботи шорткатів.

Контестне меню має свої елементи, які реалізовані через клас ContextItem. Цей клас має наступні методи та властивості:

– text – Текст елемента меню;

– callback – Функція зворотнього виклику, яка спрацьовує при кліці на цей елемент, або після шорткату;

– other – Додаткові параметри;

– key – Клавіша для шорткату.

Також в контекстному меню можна розділяти елементи, які їх дуже багато або ж для зручності. Для цього існує клас ContextDivider який не має властивостей або ж методів.

Одним з головних класів програм є клас для рендеру графа на сторінку – Render. Цей клас має наступні методи:

– render() – Метод, який рендерить граф на сторінку. Головний метод класу;

– getNodeStatusForRender() – Приватний метод який визначає статус ноди для рендера;

– getRenderedCircles() – Приватний метод, який з нод робить кружки для рендера;

– getLinesForRender() – Приватний метод, який з граней робить відрізки та математично розраховує положення стрілок.

Також досить важливим класом є WindowsManager. Цей клас дозволяє реалізувати міжвіконну взаємодію та має наступні властивості та методи:

– offsetUpdateCallback – Приватна функція зворотнього виклику, яка викликається, якщо в сховищі даних offset щось змінюється;

– presetUpdateCallback – Приватна функція зворотнього виклику, яка викликається, якщо в сховищі даних preset щось змінюється;

– setUpdateOffsetCallback() – Встановлює функцію offsetUpdateCallback;

– setPresetsUpdateCallback() – Встановлює функцію presetUpdateCallback.

Головним класом застосунку є App. Цей клас має наступні властивості та методи:

– graph – Граф, на якому буде працювати весь застосунок;

– currentPreset – Пресет, який відображається зараз;

– render – фукнція для виклику рендера графа. Встановлюється з UserInteractionManager;

– menu – Меню, яке зараз відображається;

– storage – Сховище для пресетів;

– storagePresets – Локальні данні з сховища для пресетів;

– initialize() – Інітіалізація, в якій створюються UserInteractionManager, render, меню та кнопка для сховування меню;

– graphToPreset() – Перевести граф в пресет для зберігання в сховищі;

– reloadPresetsFromStorage() – Оновити локальні пресети зі сховища та перерендерити граф;

– updatePreset() – Оновити пресет та записати до сховища;

– initializeHidePanelButton() – Інітіалізація кнопки для сховування меню;

– graphNodesStatusResetter() – Поставити всім нодам статус в default;

– graphEdgesTypeResetter() – Поставити всім граням тип в standart;

– algorithmWrapper() – Спеціальний метод для полегшення використання елементів меню, який автоматично створює унікальний ідентифікатор кожного алгоритму, викликає graphNodesStatusResetter до виконання алгоритму та після завершення через деяку затримку;

– initializeDefaultPresets() – Коли в сховищі ще немає пресетів – їх потрібно інітіалізувати;

– initializeMenu() – Інітіалізація меню. Створення елементів на основі класів алгоритмів.

Також для взаємодії з користувачем існує окремий клас UserInteractionManager, який дуже сильно пов’язаний з App, але виконує окремі функції. Цей клас має наступні властивості та методи:

– graph – Граф, береться з App;

– offsetX – Зміщення по локальній координаті X;

– offsetY - Зміщення по локальній координаті Y;

– mouseDownValues – Властивість, необхідна для правильної роботи кліків миши. Також використовується для створення граней від однієї ноди до іншої;

– currentClickedTarget – – Властивість, необхідна для правильної роботи кліків миши;

– pressedKeyCode – Нажата клавіша;

– contextMenu – Контекстне меню;

– updatePreset() – Оновлення пресету після зміни графа – ця функція береться з класу App;

– offsetStorage – Сховище для локальних зміщень;

– reloadPresetsFromStorage() – Метод перезавантаження пресетів зі сховища. Використовується в мульти-віконній взаємодії;

– windowsManager – Властивість яка преставляє клас для мульти-віконних взаємодій;

– updateOffsetStorage() – Метод для оновлення сховища зсувів;

– onKeyDown() – Метод, який викликається при натисканні клавіши клавіатури;

– onKeyUp() – Метод, який викликається при відпусканні клавіши клавіатури;

– onMouseDown() – Метод, який викликається при натисканні на ліву кнопку миші;

– onMouseUp() – Метод, який викликається при підніманні лівої кнопки миші;

– onMouseMove() – Метод, який викликається при переміщенні миші;

– onClick() – Метод, який викликається при кліці на ліву кнопку миші, де клік – це опускання клавіши, а потім підіймання;

– getTypeOfContextMenu() – Метод, який повертає тип контекстного меню в залежності від елемента, на якому воно було викликано;

– onContextMenu() – Метод, який викликається при виклику контекстного меню;

– render() – Метод рендера графа;

– initializeApp() – Метод ініціалізації який викликається з класу App;

– initializeUserEvents() – Метод ініціалізації користувацьких ефектів, таких як клік, переміщення миші та інше.

Також в програмі реалізовані базові алгоритми, щоб можна було перевірити, що все працює.

Першими базовими алгоритмуми є алгоритми DFS та BFS. DFS та BFS є стандартними алгоритмуми обходу графу, які використовуються для визначення досяжності вершин та знаходження шляхів між ними. DFS використовує глибинний підхід, а BFS - широкий, що дозволяє візуально представляти структуру графу та знаходити шляхи між вершинами.

Клас алгоритму DFS має наступні властивості та методи:

– graph – Приватне поле графа. Потрібне для використання в алгоритмі;

– render() – Приватна функція рендера. Потрібна для візуального показу роботи алгоритму;

– dfsWrapper() – Обгортка над dfs, яка потрібна для точного проходження всіх піддерев;

– dfs() – Алгоритм DFS.

Клас алгоритму BFS дуже схожий на DFS і має наступні властивості та методи:

– graph – Приватне поле графа. Потрібне для використання в алгоритмі;

– render() – Приватна функція рендера. Потрібна для візуального показу роботи алгоритму;

– bfsWrapper() – Обгортка над bfs, яка потрібна для точного проходження всіх піддерев;

– bfs() – Алгоритм BFS.

Результат виконання BFS формує дерево – клас Tree.

Клас дерева зберігає в собі елементи дерева – класи TreeNode.

Клас TreeNode має в собі наступні властивості та методи:

– value – Значення ноди дерева;

– childrens – Діти ноди дерева;

– find() – Функція для знаходження заданої ноди дерева;

– add() – Функція додавання ноди дерева.

Клас дерева, Tree, має наступні властивості та методи:

– root – Головна нода дерева від якої будуть всі інші;

– add() – Метод додавання нових нод дерева;

– display() – Метод відображеня готового дерева.

Алгоритм Беллмана-Форда використовується для знаходження найкоротших шляхів в графі, навіть якщо ребра можуть мати від'ємні ваги. Цей алгоритм реалізованний в класі BellmanFord та має наступні властивості та методи:

– graph – Приватне поле графа. Потрібне для використання в алгоритмі;

– bellmanFord() – Метод запуску алгоритму Белламана-Форда.

Алгоритм Дейкстри використовується для знаходження найкоротших шляхів в графі з невід'ємними вагами ребер. Цей алгоритм дозволяє визначати оптимальні шляхи між вершинами, що є важливим для візуалізації та аналізу графів. Цей алгоритм реалізованний в класі Dijkstra та має наступні властивості та методи:

– graph – Приватне поле графа. Потрібне для використання в алгоритмі;

– render() – Приватна функція рендера. Потрібна для візуального показу роботи алгоритму;

– initHashTables();

– getNodeWithMinTime();

– calculateTimeToEachNode();

– getShortestPath();

– dijkstra() – Метод запуску алгорима Дейкстри.

Алгоритм визначення типів ребер графу створений на основі алгоритму DFS і робить аналіз типів ребер для кращого розуміння структури графу та взаємодії між вершинами. Кожен тип ребра вносить важливий внесок у відображення графу та допомагає аналізувати його особливості.

Дерев'яні ребра (Tree Edges). Ребра дерева виникають, коли DFS переходить у нову вершину, яка ще не була відвідана. Вони визначають структуру дерева обходу графу, що є ключовим для розуміння взаємозв'язків між вершинами. Ці ребра допомагають конструювати дерево та покращують візуальне представлення графу.

Ребра "назад" (Back Edges). Ребра "назад" спрямовані вище в дереві та свідчать про існування циклу в графі. Вони важливі для визначення циклічної структури та дозволяють виявити області зацикленості. Врахування цих ребер забезпечує високу точність аналізу графічних структур.

Ребра між вершинами (Edge). Ребра між вершинами є стандартними ребрами, які не входять в конструкцію дерева обходу та не свідчать про наявність циклів. Їх аналіз допомагає визначити зв'язки між вершинами та можливі напрямки обходу графу.

Ребра, що перетинаються (Cross Edges). Ребра, що перетинаються, вказують на зв'язки між вершинами, які не пов'язані в структурі дерева обходу. Ці ребра можуть виявлятися важливими для аналізу паралельних шляхів та взаємозв'язків між різними частинами графу.

Алгоритм визначення типів граней реалізованний в класі EdgesTypes та має наступні властивості та методи:

– graph – Приватне поле графа. Потрібне для використання в алгоритмі;

– render() – Приватна функція рендера. Потрібна для візуального показу роботи алгоритму;

– edgesTypes() – Метод запуску алгоритму визначення типів ребер;

– edgesTypesInner() – Приватний внутрішній метод для рекурсії.

Алгоритм Флойда-Воршелла використовується для знаходження всіх можливих шляхів між усіма парами вершин у графі. Цей алгоритм реалізованний в класі FloydWarshall та має наступні властивості та методи:

– graph – Приватне поле графа. Потрібне для використання в алгоритмі;

– floydWarshall() – Метод для запуску алгоритму Флойда-Воршелла.

Алгоритм взяття нод за рівнем вкладеності дозволяє вам визначати всі вершини, які знаходяться в певному радіусі від обраної вершини. Потрібний для аналізу навколишніх зв'язків та структури графу. Цей алгоритм реалізованний в класі NodesByDeepLevel та має наступні властивості та методи:

– graph – Приватне поле графа. Потрібне для використання в алгоритмі;

– render() – Приватна функція рендера. Потрібна для візуального показу роботи алгоритму;

– getNodesByDeepLevel() – Метод для запуску отримання всіх нод в радіусі.

4 АНАЛІЗ ЕФЕКТИВНОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Базовий аналіз ефективності програмного забезпечення

Основні аспекти базового аналізу ефективності програмного забезпечення:

- швидкодія;
- масштабованість;
- оптимізація алгоритмів та відображення;
- підтримка паралельних обчислень.

Швидкодія є одним із важливих критеріїв оцінки ефективності програмного забезпечення. Перш за все, важливо визначити та забезпечити оптимальне відображення графічної інформації на екрані. Висока реактивність системи на користувацькі дії, мінімізація затримок та ефективність роботи вбудованих алгоритмів є пріоритетами в реалізації програми. При змінах у структурі графу система повинна динамічно адаптуватися, забезпечуючи незабароме та безперервне оновлення відображення.

Масштабованість системи є важливим аспектом, оскільки програма повинна бути готовою ефективно працювати з графами різної складності та розміру. Забезпечення плавного та стабільного масштабування графічних елементів є критичним для уникнення впливу на продуктивність при великій кількості вузлів та ребер. Система повинна володіти можливістю ефективно обробляти та відображати графи, які можуть варіюватися від простих до великих та складних.

Забезпечення гнучкості відображення графу дає можливість користувачеві індивідуалізувати вигляд графу відповідно до його потреб та вимог. Переміщення графу між вікнами повинно бути інтуїтивно зрозумілим та миттєвим, надаючи користувачеві можливість зручно адаптувати відображення графу до конкретного контексту аналізу.

Ефективність роботи алгоритмів та механізмів відображення є ключовими факторами для забезпечення швидкої та продуктивної роботи програми. Оптимізація алгоритмів включає в себе використання ефективних структур даних, уникання зайвих обчислень та забезпечення максимальної продуктивності при роботі з великими обсягами даних.

З урахуванням сучасних комп'ютерних архітектур, важливим аспектом є можливість використання паралельних обчислень для підвищення продуктивності системи. Розробка програми повинна передбачати можливість розпаралелювання обчислень для використання ресурсів багатоядерних процесорів.

Враховуючи ці аспекти, забезпечується не тільки висока ефективність програмного забезпечення для моделювання структур графів, але й висока задоволеність користувачів зручністю та продуктивністю використання програми. Об'єктно-орієнтований підхід дозволяє структурувати та оптимізувати код, що сприяє підтримці високої ефективності програмного забезпечення на різних етапах розробки та експлуатації.

4.2 Аналіз ефективності компонентів програмного забезпечення

Система складається з основних компонентів, що взаємодіють між собою:

– відображення графа. Це основний компонент системи;

– контекстне меню. Потрібне для створення та видалення нод;

– меню. Потрібне для відкриття форми та запуску алгоритмів. Основна точка взаємодії з користувачем;

– форма для запуску алгоритму. Потрібна для запуску алгоритму з деякими вхідними даними;

– алгоритм. Алгоритм який виконується вважається окремим компонентом;

– сховище даних. Потрібне для збереження стану застосунку, такого як внутрішній зсув та пресети;

– міжвіконна взаємодія. Окремий складний компонент.

Ключовим аспектом аналізу ефективності відображення графа є його швидкодія. Важливо, щоб відображення графа було миттєвим та ефективним навіть при значній кількості вузлів та ребер. Це забезпечить користувачам плавний та беззатримковий перегляд графічної інформації.

Щодо контекстного меню, оптимізація полягає в швидкому та точному відгуку на дії користувача. Важливо, щоб воно активувалося швидко та надавало можливість користувачеві оперативно взаємодіяти з елементами графу, забезпечуючи зручність та ефективність.

Меню повинне бути легким та швидким у використанні. Його оптимізація полягає в швидкому доступі до функціоналу та мінімізації затримок при взаємодії з опціями. Важливо, щоб воно було ергономічним та не викликало затримок в роботі.

Швидкодія форми для запуску алгоритму полягає в зручному та інтуїтивно зрозумілому інтерфейсі для введення даних. Оптимізація полягає в миттєвому реагуванні на введення користувача та максимальній зручності використання.

В аналізі ефективності алгоритму важливо враховувати його швидкість та точність виконання завдань. Оптимізація включає в себе покращення роботи алгоритму та його взаємодії з іншими компонентами системи для забезпечення оптимальних результатів.

Швидкодія сховища даних визначається швидким доступом до необхідної інформації та ефективним управлінням даними. Важливо, щоб воно забезпечувало консистентність та ефективність при взаємодії з іншими компонентами системи.

Оптимізація міжвіконної взаємодії включає в себе мінімізацію затримок при перемиканні між вікнами та оптимальне управління ресурсами. Важливо, щоб цей компонент був ефективним у використанні та не заважав користувачеві взаємодіяти з різними частинами системи.

4.3 Тестування програмного забезпечення

Тестування програмного забезпечення є критичним етапом у розробці для переконання в його стабільності, правильності та відповідності вимогам. Даний розділ детально розглядає підходи та стратегії тестування розробленого браузерного застосунку для моделювання структур графів на базі TypeScript та Vite.

Були протестовані наступні елементи:

- відображення графа;
- контекстне меню;
- панель;
- меню;
- форма для запуску алгоритму;
- сховище даних;
- міжвіконна взаємодія.

Тестування відображення графа включає в себе:

- перевірку правильності відображення графа за заданими координатами;
- перевірку правильності відображення всіх нод та граней;
- перевірку правильності відображення ваги граней;
- перевірку правильності відображення напрямку граней.

А також разом з міжвіконною взаємодією:

- перевірку правильності позиціонування графа в просторі віртуальних координат;
- перевірку правильності відступів, які основані на позиції вікна браузера. Це тестування показано на рисунку 4.1.

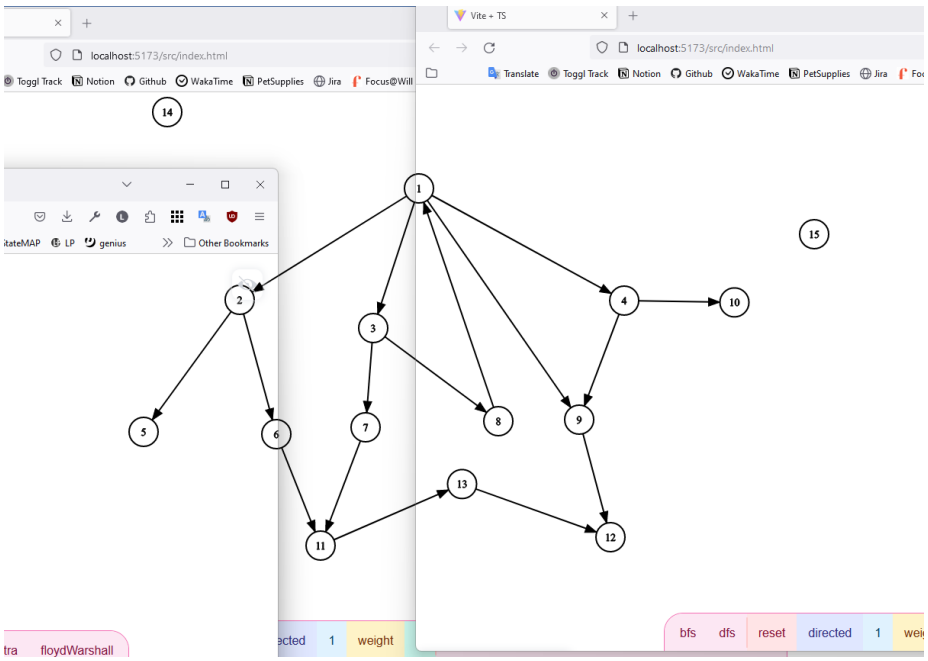


Рисунок 4.1 – Тестування міжвіконної взаємодії та правильності відступів

Перевірка контекстного меню включає в себе:

– перевірку відкриття контекстного меню. Це тестування показано на рисунку 4.2;

– перевірку правильного відображення потрібного типу меню в залежності від елементу, на якому воно було викликано;

– перевірку роботи елементів контекстного меню на виклик callback функції;

– перевірку роботи шорткатів на виклик callback функції.

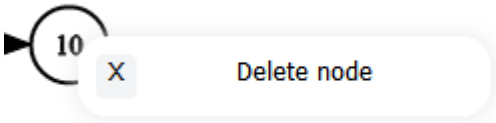


Рисунок 4.2 – Тестування відкриття контекстного меню

Перевірка меню включає в себе:

– перевірку на правильність відображення елементів, які були задані розробником. Це тестування показано на рисунку 4.3;

– перевірку на правильність відображення кольорів елементів;

– перевірку на правильність відображення роздільників елементів;

– перевірку на виклик callback функції після кліку;

– перевірка на доступність коли людина використовує сайт через клавіатуру.

Рисунок 4.3 – Тестування відображення меню

Перевірка панелі включає в себе:

– перевірка на відкриття та закриття;

– перевірка на правильність рендера контенту;

– перевірка в парі з меню, коли при кліці на один елемент двічі панель повина спочатку відкритись, а потім закритись;

– перевірка в парі з меню, коли при кліці на один елемент панель повина відкритись, а при кліці на другий елемент панель повина змінити контент.

Перевірка форми включає в себе:

– перевірка на правильність відображення всіх інпутів;

– перевірка на правильність повернення результатів при відправці форми.

Перевірка сховища даних включає в себе:

– перевірку на правильність зберігання даних;

– перевірку на запис даних та потім взяття;

– перевірку на взяття пустих даних.

Перевірка міжвіконної взаємодії включає в себе:

– перевірка виклику callback функції при зміні віртуального координатного зсуву в окремому вікні;

– перевірка виклику callback функції при зміні графу в окремому вікні.

Після проведення тестування застосунку виявлено відсутність будь-яких помилок чи несправностей в його роботі. Всі функціональні та інтерфейсні елементи працюють вірно та беззавдань згідно з передбаченими специфікаціями. Тестування вказує на високу якість програмного забезпечення, адекватну відповідь на користувацькі взаємодії та відсутність технічних недоліків.

5 РОЗРОБКА ДОКУМЕНТІВ НА СУПРОВОДЖЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Інструкція програміста

Ця інструкція відповідає на запитання як розгорнути проєкт локально та на сервері, як в нього вносити зміни та як тестувати застосунок.

Розглянемо як розгорнути проєкт локально.

Спочатку потрібно клонувати репозиторій застосунку на вашому робочому пристрої. Використовуйте команду git clone та вказуйте URL репозиторію. На рисунку 5.1 показаний приклад клонування репозиторію.

Рисунок 5.1 – Клонування репозиторію

Встановлення залежностей. Переконайтеся, що у вас встановлений пакетний менеджер pnpm. Використовуйте команду pnpm install, щоб встановити всі необхідні залежності. Це забезпечить коректну роботу застосунку. На рисунку 5.2 показаний приклад встановлення залежностей.

```
C:\home\projects\tmp\university-coursework-1-OOP>pnpm i
Lockfile is up to date, resolution step is skipped
Packages: +215
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Progress: resolved 215, reused 214, downloaded 1, added 215, done
node_modules/.pnpm/esbuild@0.19.7/node_modules/esbuild: Running postinstall script, done in 714ms

devDependencies:
+ @typescript-eslint/eslint-plugin 6.12.0
+ eslint-config-prettier 9.0.0
+ eslint-plugin-html 7.1.0
+ eslint-plugin-prettier 5.0.1
+ husky 8.0.3
+ lint-staged 15.1.0
+ prettier 3.2.4
+ typescript 5.3.2
+ vite 5.0.2

> coursework-oop@0.0.0 prepare C:\home\projects\tmp\university-coursework-1-OOP
> husky install

husky - Git hooks installed

Done in 3.3s
```

Рисунок 5.2 – Встановлення залежностей

Запуск застосунку для розробки. Використовуйте команду npm run dev, щоб запустити застосунок у режимі розробки. Це дозволить вам спостерігати за змінами в реальному часі та вносити власні виправлення. На рисунку 5.3 показаний приклад запуску застосунку.

```
C:\home\projects\tmp\university-coursework-1-OOP>npm run dev

> coursework-oop@0.0.0 dev
> vite

(!) Could not auto-determine entry point from rollupOptions or html file

  VITE v5.0.2  ready in 317 ms

  ➜  Local:   http://localhost:5173/
  ➜  Network: use --host to expose
  ➜  press h + enter to show help
```

Рисунок 5.3 – Запуск застосунку

Тепер проєкт розгорнутий локально та готовий до редагування або ж використання

Розглянемо приклад розгортання застосунку на серверах Vercel.

Спочатку нам потрібно створити свій власний github(або ж gilab, тощо) репозиторій, або можна створити форк.

Далі потрібно зайти на сайт https://vercel.com та авторизуватись. Натискаємо кнопку Add new і обираємо Project.

Після цього нас перекидує на сторінку вибору репозиторія для імпорту. Ця сторінка зображена на рисунку 5.4

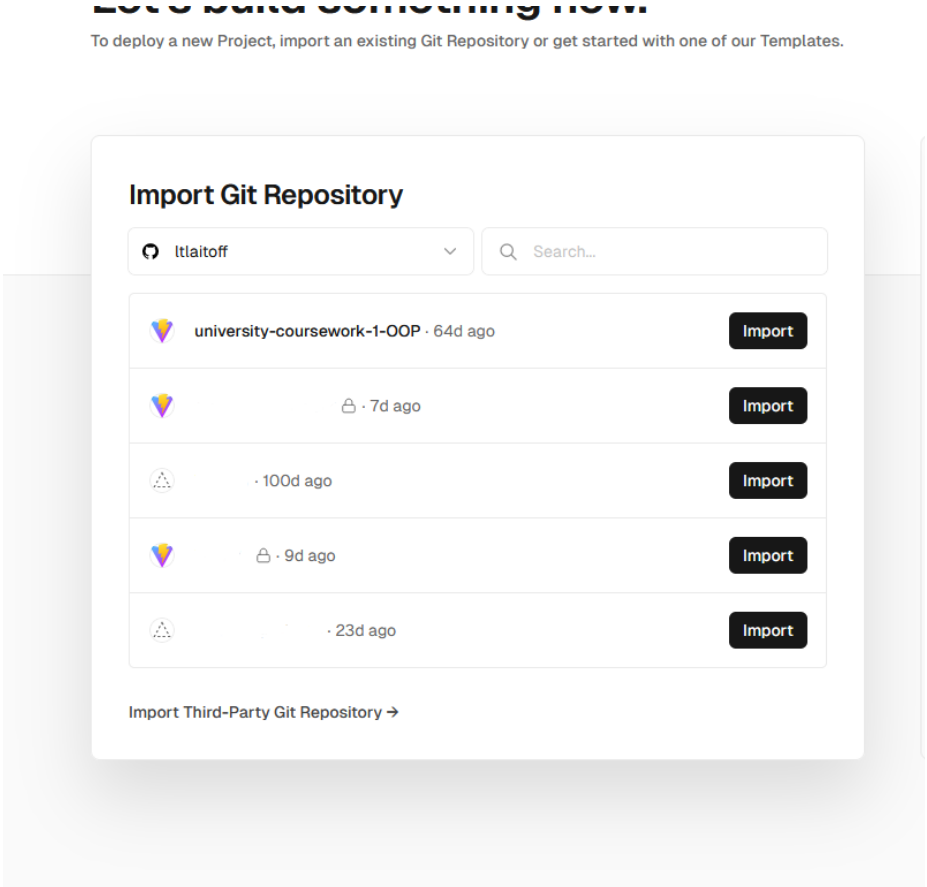


Рисунок 5.4 – Приклад вибору репозиторія для імпорту в Vercel

В списку репозиторіїв обираємо потрібний там та нажимаємо кнопку Import.

Після натискання кнопки Import на потрібному репозиторії нас перекидає на сторінку налаштування деплою застосунку. Приклад сторінки налаштування показаний на рисунку 5.5

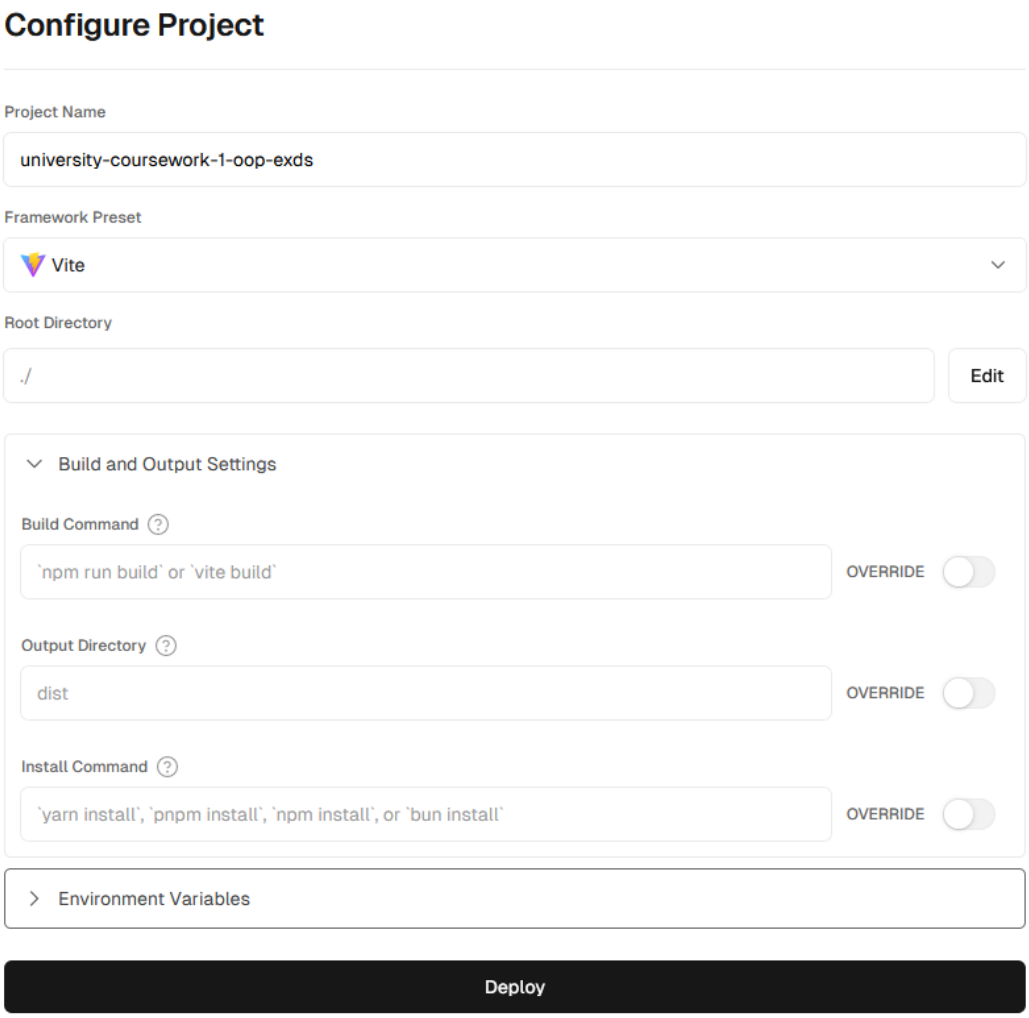


Рисунок 5.5 – Приклад сторінки налаштування проєкта перед деплоєм

Тут нічого не змінюючи натискаємо на кнопку Deploy, трохи чекаємо і застосунок тепер є в інтернеті. На рисунку 5.6 показаний приклад задеплоєного застосунку

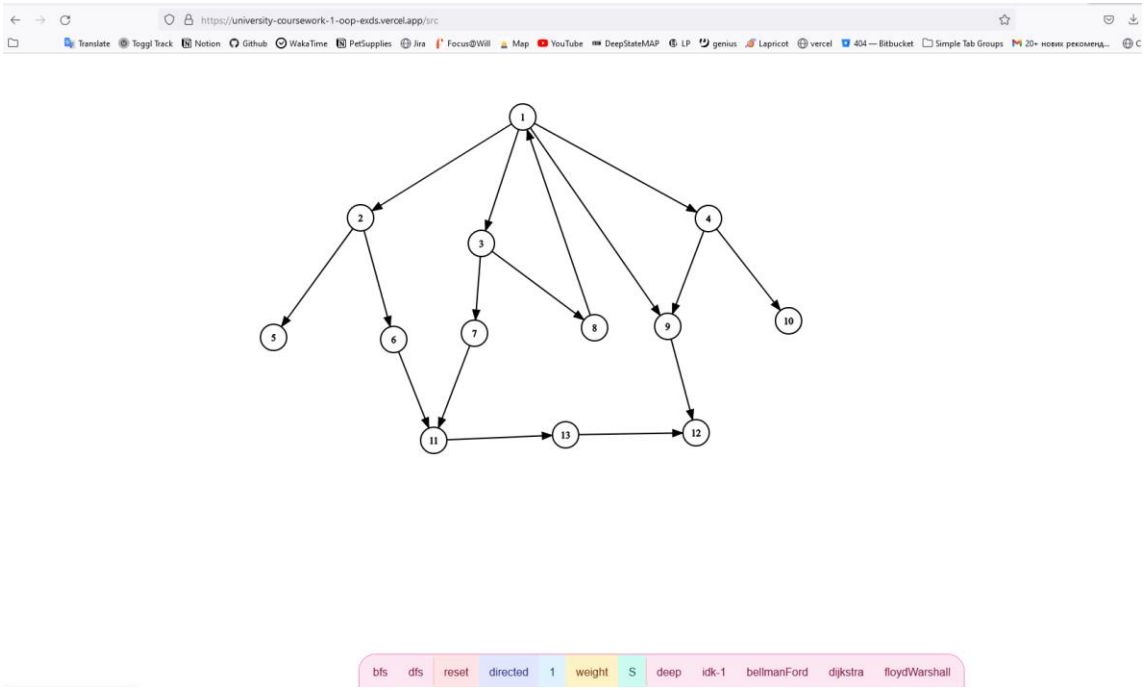


Рисунок 5.6 – Приклад задеплоєного застосунку

Короткий опис додаткових скриптів та рекомендації:

– Лінтинг коду. Періодично перевіряйте код на наявність помилок та дотримання стандартів за допомогою команди npm run lint. Це допоможе уникнути потенційних проблем та підтримувати високий стандарт коду.

– Форматування коду. Зберігайте структуру коду читабельною та однорідною, використовуючи команду npm run format. Це автоматично вирівняє код за встановленими стандартами форматування.

5.2 Інструкція користувачеві

При першому заході на сайт нас зустрічає єкран, який показаний на рисунку 5.7.

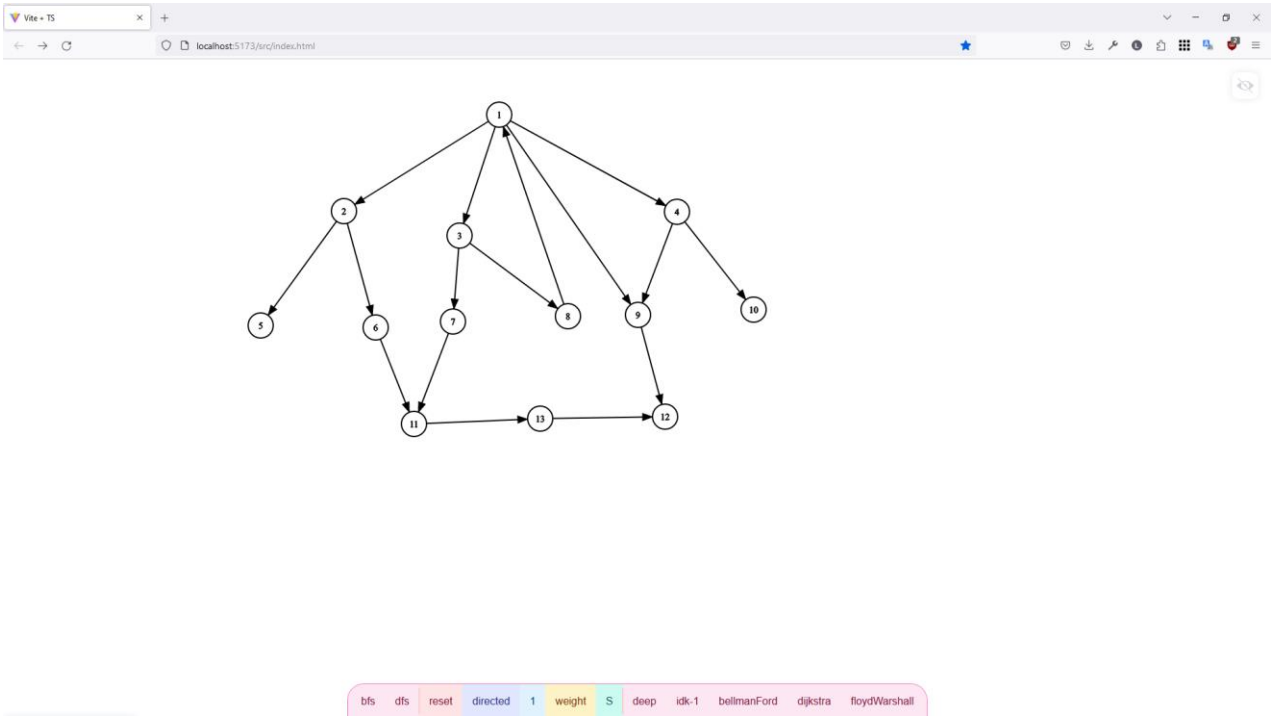


Рисунок 5.7 – Початковий єкран застосунку

При зажаті пробілу та лівої кнопки миші ми можемо переміщатись по внутрішнім координатам. На рисунку 5.8 показаний здвигнутий граф відносно меню.

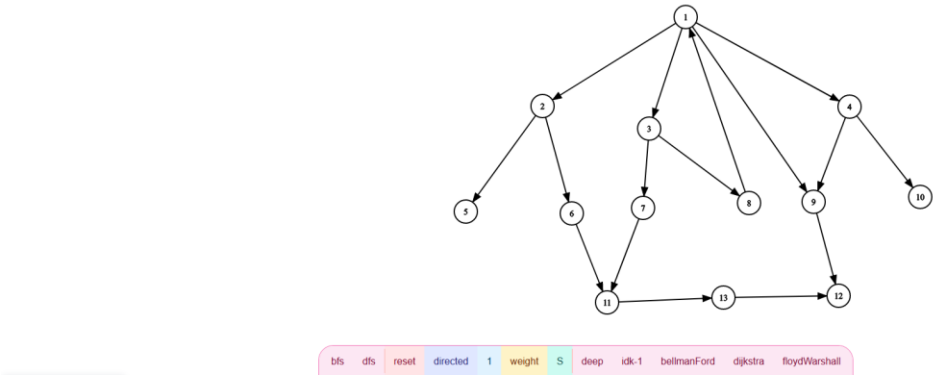


Рисунок 5.8 – Здвигнутий граф відносно меню

Щоб створити ноду можна або клацнути ліву кнопку миші в потрібному місці, або клацнути праву та викликати контестне меню, яке показано на рисунку 5.9.

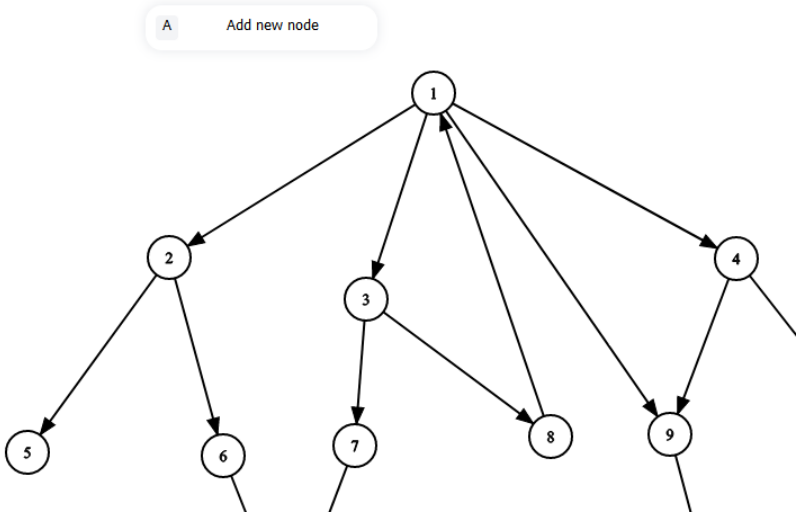


Рисунок 5.9 – Приклад виклику контекстного меню

Далі можна або натиснути на елемент цього контекстного меню, або ж нажати клавішу A на клавіатурі і нова нода буде створена. Створена нода показана на рисунку 5.10.

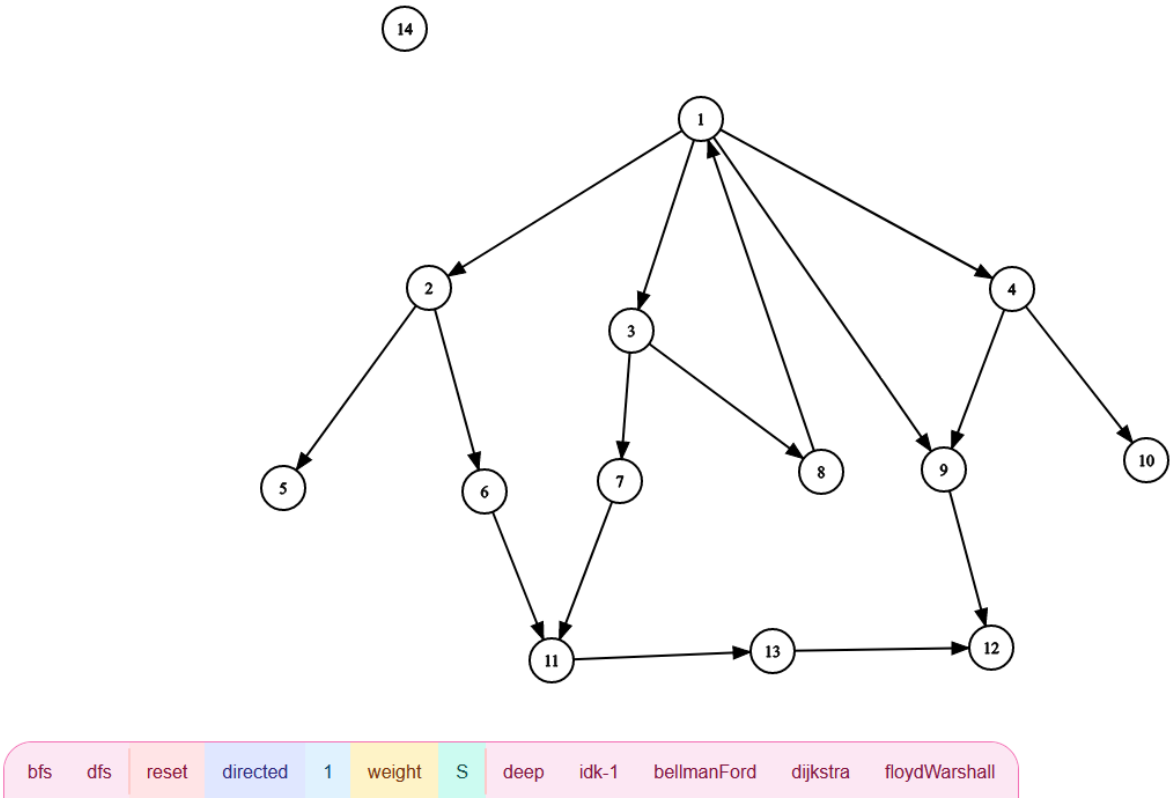


Рисунок 5.10 – Нова нода

Щоб пов’язати цю ноду з іншою потрібно натиснути на цю, вона підсвітиться червоним, це зображено на рисунку 5.11, а потім на наступну ноду і утвориться грань, це зображено на рисунку 5.12.

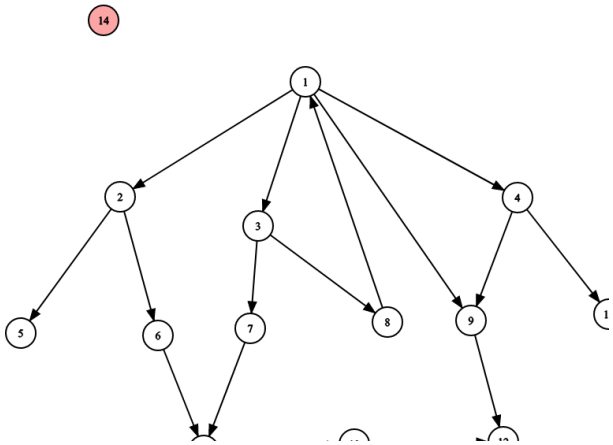


Рисунок 5.11 – Виділення ноди

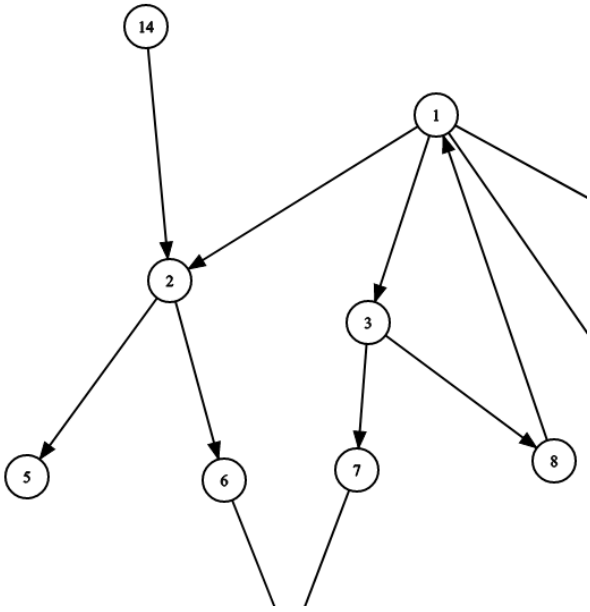


Рисунок 5.12 – Створення грані

Щоб відмінити виділення ноди потрібно натиснути клавішу Escape

Якщо в панелі меню обрати пункт вагів, weight, то на графі відобразяться ваги граней. Це показано на рисунку 5.13.

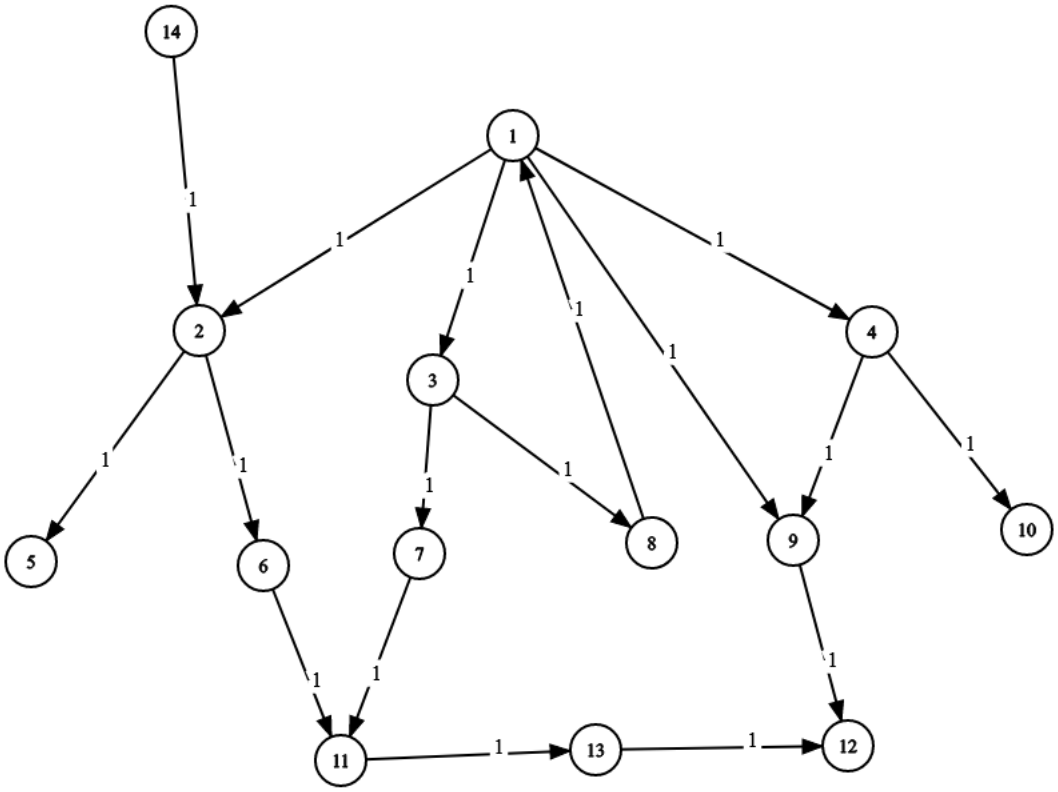


Рисунок 5.13 – Показ вагів ребер

Якщо в меню натиснути на “directed” то в графі пропадуть всі напрямлення і він стане не напрямленим

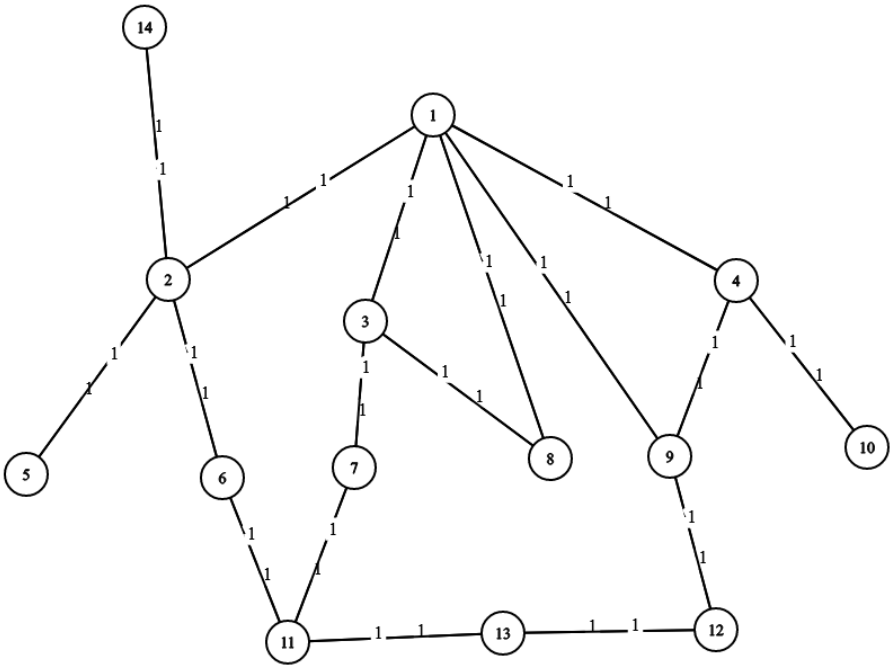


Рисунок 5.14 – Зміна типу графа на не напрямлений

Також в застосунку реалізований функціонал пресетів. При кліці на пунт меню де зображені цифри – це вибір пресета. Всього існує 4 пресети, 3 базові та 1 пустий. Приклад другого пресета показаний на рисунку 5.15.

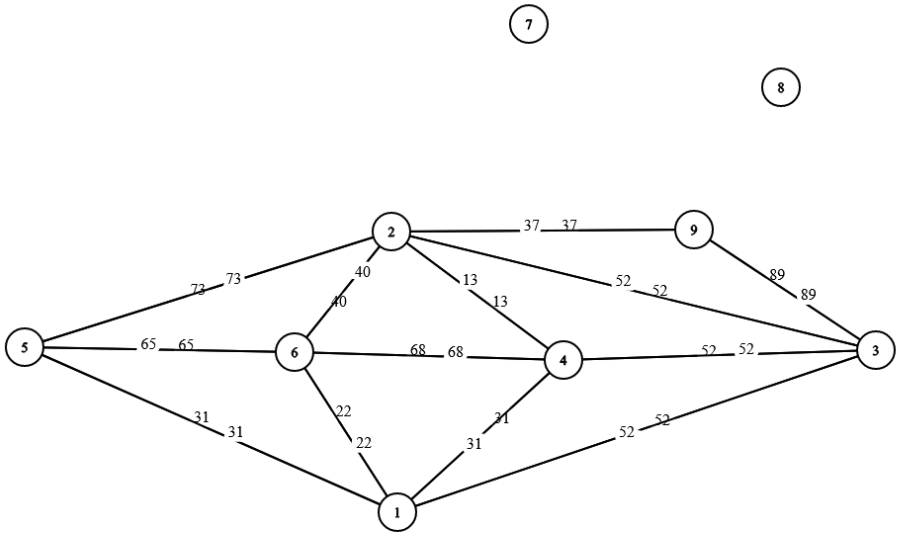


Рисунок 5.15 – Приклад другого пресета

Також в застосунку реалізована можливість запуску алгоритмів як напряму з меню, так і через відкриття окремої панелі з формою. На рисунку 5.16 показаний приклад запуску алгоритму BFS.

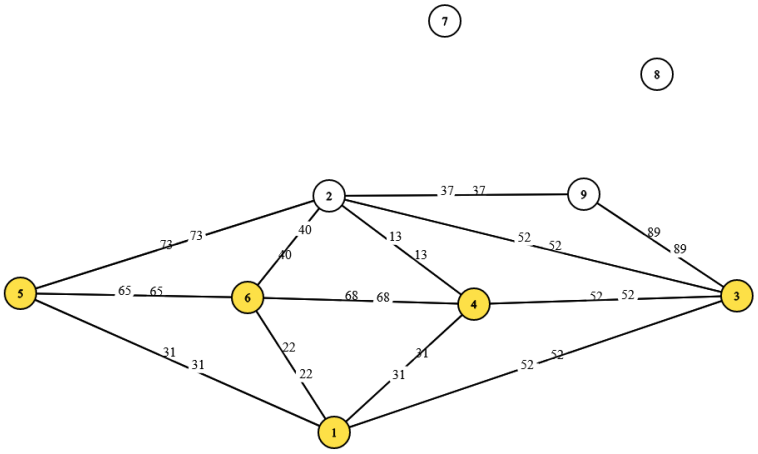


Рисунок 5.16 – Приклад запуску алгоритму BFS

На панелі меню є ще одна унікальна кнопка – S. Ця кнопка відкриває панель де можна сбросити всі пресети, а також показана матриця суміжності графа, рисунок 5.17.

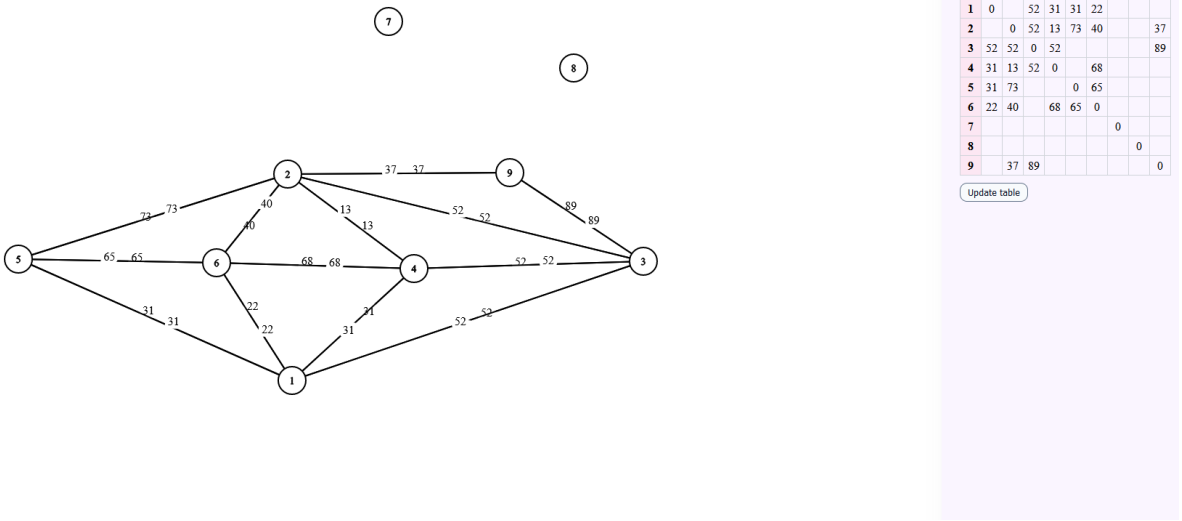


Рисунок 5.17 – Панель з матрицею суміжності

На рисунку 5.18 показаний приклад запуску алгоритму Дейкстри з точки 5 до точки 9.

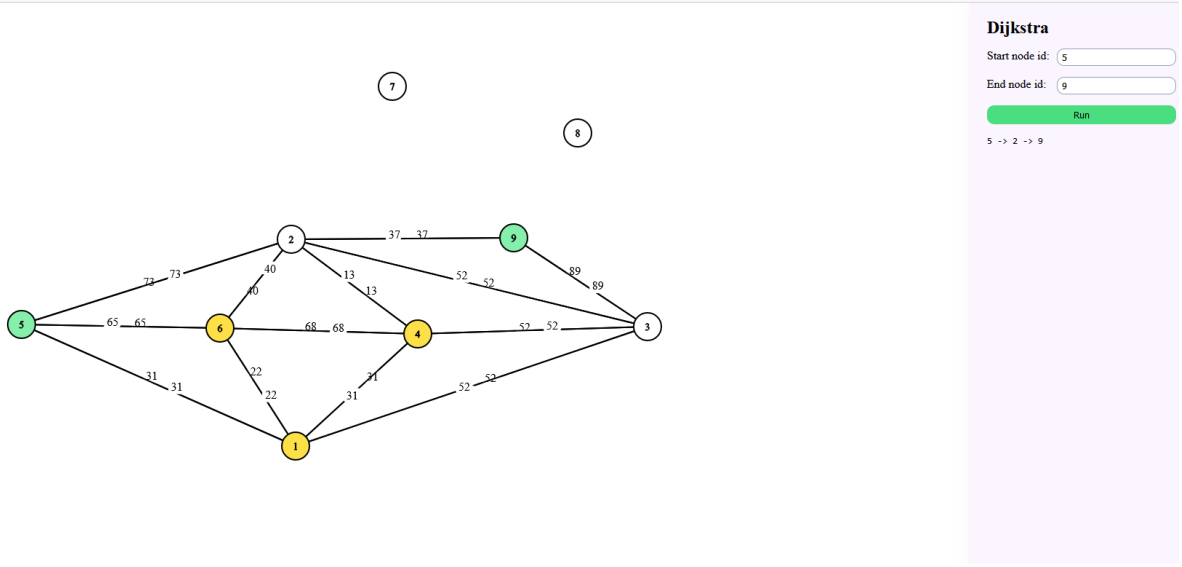
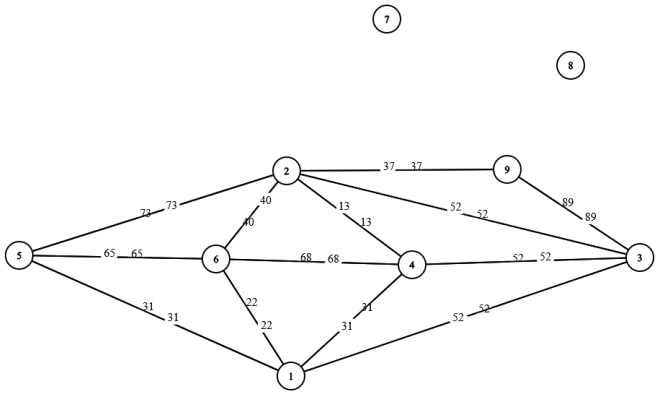


Рисунок 5.18 – Приклад запуску алгоритму Дейкстри

На рисунку 5.19 показаний приклад запуску алгоритму Флойда-Воршелла.



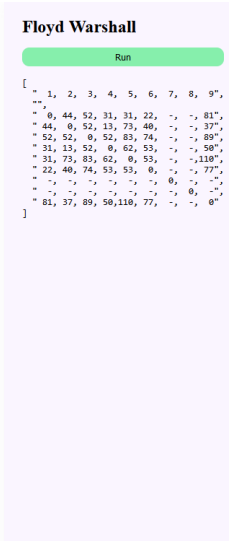


Рисунок 5.19 – Приклад запуску алгоритму Флойда-Воршелла

В застосунку в правому верхньому куті є малозамітна кнопка, показана на рисунку 5.20, яка при активації скриває меню, рисунок 5.21 та 5.22.

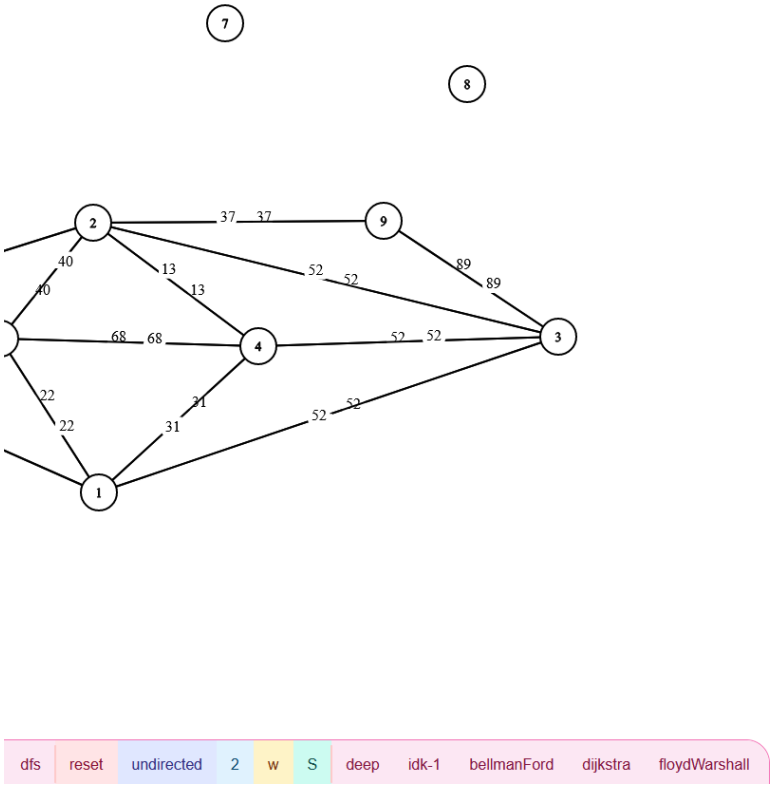


Рисунок 5.20 – Кнопка для скривання меню

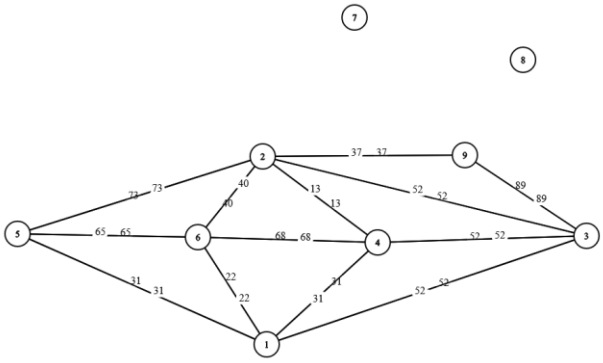


Рисунок 5.21 – Приклад застосунку зі скритою меню

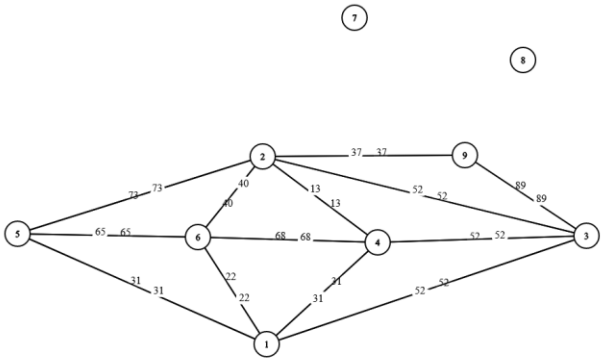


Рисунок 5.22 – Приклад застосунку зі скритою меню та наведенням на кнопку

Однією з цікавих функцій застосунку є міжвіконна взаємодія. Це означає, що ми можемо відкрити застосунок в 2 різних вікнах і граф буде показуватись у обох. Міжвіконна взаємодія показана на рисунку 5.23

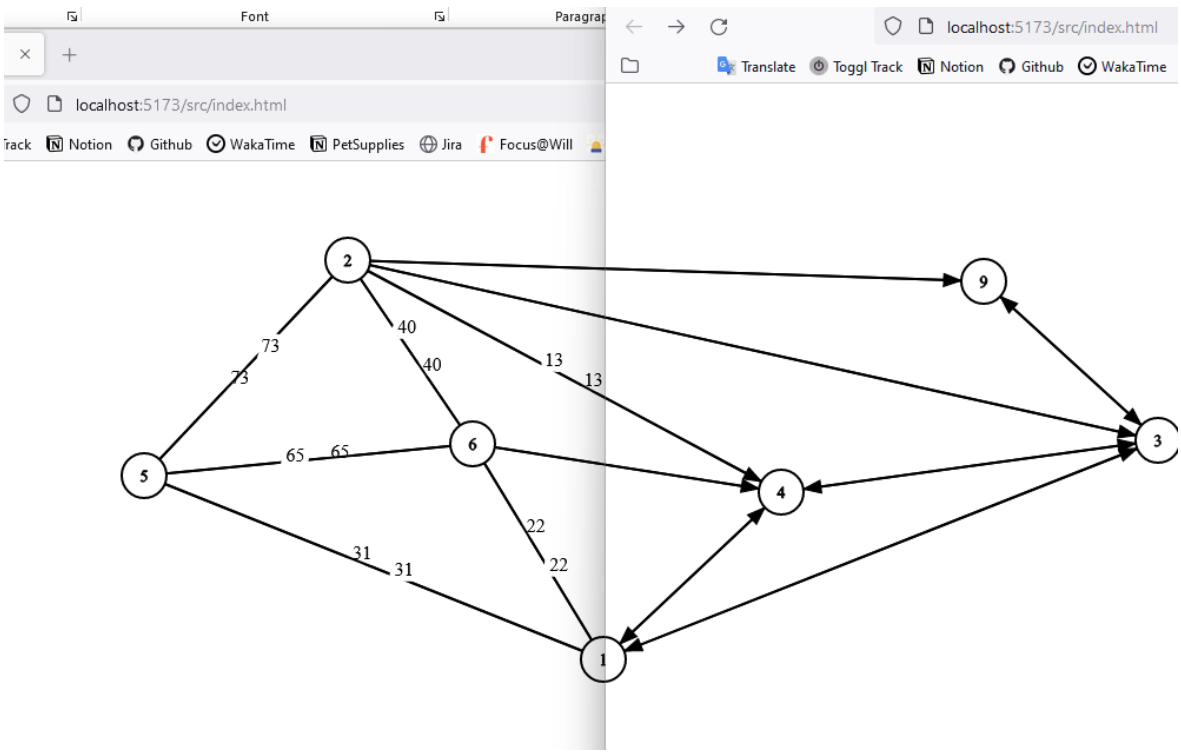


Рисунок 5.23 – Приклад міжвіконної взаємодія

Звичайно, що в застосунку можна переміщувати ноди, для цього потрібно нажати на потрібну ноду лівою кнопкою миші та перетащити. Приклад переміщеної ноди показаний на рисунку 5.24.

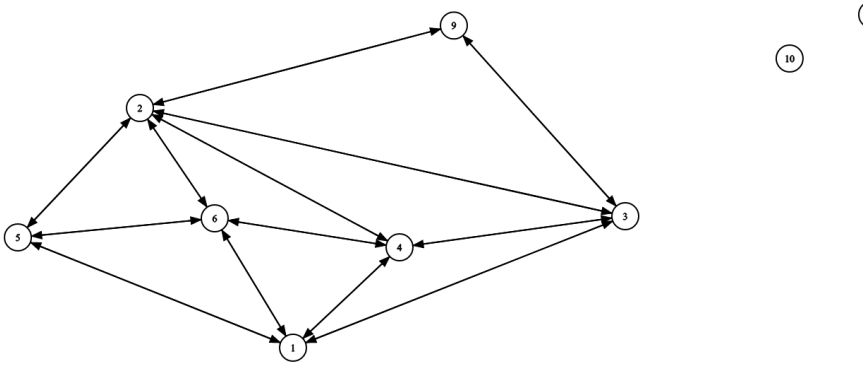


Рисунок 5.24 – Приклад переміщеної ноди

Також в застосунку реалізований функціонал зміни ваги ребра. Для цього потрібно натиснути на нього лівою кнопкою миші та в полі ввести потрібне значення. Зміна ваги ребра продемонстрована на рисунку 5.25

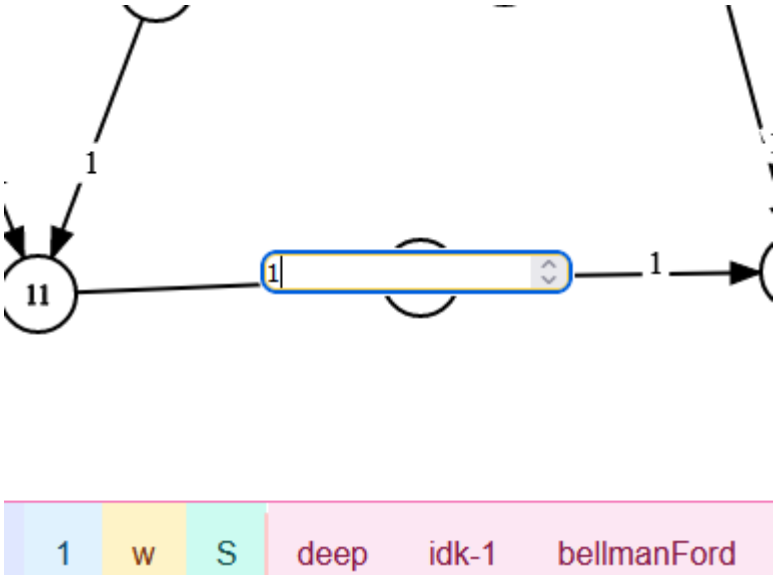


Рисунок 5.25 – Приклад зміни ваги ребра

ВИСНОВКИ

В результаті виконання цієї курсової роботи було створено сервіс, який надає можливість візуально моделювати графи та спрощує задачу візуалізації алгоритмів на графі за рахунок надання зручного та швидкого інтерфейсу.

В результаті огляду та аналізу сучасних методів та засобів проектування програмного забезпечення було встановлено, що зараз йде активний розвиток багатьох сфер життя і всі ці сфери потребують в тій чи іншій мірі роботу з графами.

Визначено мету створення програми та її основні функції.

Виконано проектування програмного забезпечення, де було проаналізовано функції системи, розроблемо діаграми використання, був спроєктований графічний інтерфейс. Визначено мету створення програми та її основні функції.

Розроблено застосунок згідно з аналізами та проєктами, розроблена структура системи, описані всі класи програмного комплексу та розроблена діаграма класів.

Проведено аналіз ефективності програмного забезпечення. Застосунок було перевірено на швидкодію та масштабованість, проаналізована ефективність кожного окремого компонента та виконано тестування.

Також розроблено методику роботи користувача з системою окремо для інших розробників, окремо для звичайних користувачів

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Hatch, S.V. Computerized Engine Controls / S.V. Hatch. – Boston: Cengage Learning, 2016. – 688 p.

2. Czichos, H. Measurement, Testing and Sensor Technology. Fundamentals and Application to Materials and Technical Systems / H. Czichos. – Berlin: Springer, 2018. – 213 p.

3. Kaźmierczak, J. Data Processing and Reasoning in Technical Diagnostics / J. Kaźmierczak, W. Cholewa. – Warszawa: Wydawnictwa Naukowo-Techniczne, 1995. – 186 p.

4. Diagnostics as a Reasoning Process: From Logic Structure to Software Design / [M. Cristani, F. Olivieri, C. Tomazzoli, L. Vigano, M. Zorzi] // Journal of Computing and Information Technology. – 2018. – Vol. 27 (1). – P. 43-57.

5. Wieczorek, A.N. Analysis of the Possibility of Integrating a Mining Right-Angle Planetary Gearbox with Technical Diagnostics Systems / A.N. Wieczorek // Scientific Journal of Silesian University of Technology. Series Transport. – 2016. – Vol. 93. – P. 149-163.

6. Tso, B. Classification Methods for Remotely Sensed Data / B. Tso, P.M. Mather. – Boca Raton : CRC Press, 2016. – 352 p.

7. Oppermann, A. Regularization in Deep Learning – L1, L2, and Dropout [Electronic resource]. – Access mode: https://www.deeplearning-academy.com/p/ai-wiki-regularization.

8. Classic Regularization Techniques in Neural Networks [Electronic resource]. – Access mode: https://medium.com/@ODSC/classic-regularization-techniques-in-neural-networks-68bccee03764.

ДОДАТОК А

src\main.ts

```
import './style.css'
import { Render } from './modules/Render.ts'

import { resetActiveId } from './helpers'
import { BFS } from './algorithms/BFS'
import { DFS } from './algorithms/DFS'
import { Dijkstra } from './algorithms/Dijkstra.ts'
import { BellmanFord } from './algorithms/BellmanFord.ts'
import { FloydWarshall } from './algorithms/FloydWarshall.ts'
import { NodesByDeepLevel } from './algorithms/NodesByDeepLevel.ts'
import { DEFAULT_PRESET, MINUS_WEIGHTS_PRESET, WEIGHTS_PRESET } from './presets'
import { Menu, MenuDivider, MenuItem } from './modules/Menu.ts'
import { Graph } from './models/Graph.ts'
import { SidePanel } from './modules/Panel.ts'
import { Form } from './modules/Form.ts'
import { EdgesTypes } from './algorithms/EdgesTypes.ts'
import { ContextItem, ContextMenu } from './modules/ContextMenu.ts'
import { PresetStorage } from './utils/PresetStorage.ts'
import { StoragePresets } from './types/StoragePresets.ts'
import { Preset, PresetEdge } from './types/PresetItem.ts'
import { OffsetStorage } from './utils/OffsetStorage.ts'
import WindowManager from './utils/WindowManager.ts'

class UserInteractionManager {
      graph: Graph
      offsetX = 0
      offsetY = -105

      mouseDownValues: {
            active: boolean
            target: HTMLElement | null
            innerOffsetX: number
            innerOffsetY: number
      } = {
            active: false,
            target: null,
            innerOffsetX: 0,
            innerOffsetY: 0
      }

      currentClickedTarget: HTMLElement | null = null

      pressedKeyCode: string | null = null
      contextMenu: ContextMenu

      updatePreset: (graph: Graph) => void

      offsetStorage: OffsetStorage

      reloadPresetsFromStorage: () => void
      windowsManaget

      constructor(
            graph: Graph,
            updatePreset: (graph: Graph) => void,
            reloadPresetsFromStorage: () => void
      ) {
            this.graph = graph
            this.contextMenu = new ContextMenu()
            this.updatePreset = updatePreset
```

```
        this.offsetStorage = new OffsetStorage()
        const offsetStorageValue = this.offsetStorage.read()
        this.reloadPresetsFromStorage = reloadPresetsFromStorage

        if (offsetStorageValue) {
            this.offsetX = offsetStorageValue.offsetX
            this.offsetY = offsetStorageValue.offsetY
        }

        this.windowsManaget = new WindowManager()

        this.windowsManaget.setUpdateOffsetCallback(() => {
            const offsetStorageValue = this.offsetStorage.read()

            this.offsetX = offsetStorageValue.offsetX ?? 0
            this.offsetY = offsetStorageValue.offsetY ?? 0

            this.render()
        })

        this.windowsManaget.setPresetsUpdateCallback(() => {
            console.log('presets update')
            this.reloadPresetsFromStorage()
            this.render()
        })

        let oldX = window.screenX
        let oldY = window.screenY

        setInterval(() => {
            if (oldX != window.screenX || oldY != window.screenY) {
                this.render()
            }

            oldX = window.screenX
            oldY = window.screenY
        }, 50)

        window.addEventListener('resize', () => {
            this.render()
        })

        this.render()
    }

    updateOffsetStorage(offsetX: number, offsetY: number) {
        this.offsetStorage.writeAll({
            offsetX,
            offsetY
        })
    }

    onKeyDown(event: KeyboardEvent): void {
        this.pressedKeyCode = event.code

        if (event.code === 'Escape') {
            this.currentClickedTarget = null
            this.render()
        }
    }

    onKeyUp(event: KeyboardEvent): void {
        if (this.pressedKeyCode === event.code) {
            this.pressedKeyCode = null
```

```
                this.render()
            }
        }

        onMouseDown(e: MouseEvent) {
            this.mouseDownValues = {
                active: true,
                target: e.target as HTMLElement,
                innerOffsetX:
                    e.clientX - (e.target as
HTMLElement).getBoundingClientRect().x - 20,
                innerOffsetY:
                    e.clientY - (e.target as
HTMLElement).getBoundingClientRect().y - 20
            }
        }

        onMouseUp() {
            this.mouseDownValues = {
                active: false,
                target: null,
                innerOffsetX: 0,
                innerOffsetY: 0
            }
        }

        onMouseMove(e: MouseEvent) {
            if (!this.mouseDownValues.active) return

            if (this.pressedKeyCode === 'Space') {
                console.log(e)
                this.offsetX += e.movementX
                this.offsetY += e.movementY

                this.updateOffsetStorage(this.offsetX, this.offsetY)

                this.render()
            } else {
                if (!this.mouseDownValues.target) return
                if (!this.mouseDownValues.target.dataset.elementid) return
                const node = this.graph.graph.get(
                    this.mouseDownValues.target.dataset.elementid
                )

                if (!node) return

                node.x = e.clientX - this.offsetX -
this.mouseDownValues.innerOffsetX
                node.y = e.clientY - this.offsetY -
this.mouseDownValues.innerOffsetY

                this.updatePreset(this.graph)
                this.render()
            }
        }

        onClick(e: MouseEvent) {
            console.log('click')

            const target = e.target as HTMLElement

            if (target.tagName !== 'svg') {
                if (target.dataset.edgeTextId !== undefined) {
                    const [adjacentNodeValue, weight, status] =
```

```
                        target.dataset.edgeTextId?.split('-') ?? []

                  const                  adjacentNode                  =
this.graph.graph.get(adjacentNodeValue)
                  if (!adjacentNode) return

                  const                      edge                      =
Array.from(adjacentNode.parents.values()).find(
                        edge   =>   edge.weight   ===   Number(weight)   &&
edge.status === status
                  )

                  if (!edge) return

                  const { top, left } = target.getBoundingClientRect()

                  const input = document.createElement('input')
                  input.type = 'number'
                  input.value = String(edge.weight)
                  input.className = 'input-change-weight'

                  input.setAttribute(
                        'style',
                        `position:absolute;    top:    ${top}px;    left:
${left}px`
                  )

                  input.addEventListener('blur', e => {
                        const value = (e.target as HTMLInputElement).value

                        const valueNumber = Number(value)

                        console.log(value, valueNumber)

                        if (isNaN(valueNumber)) {
                              edge.weight = 0
                        } else {
                              edge.weight = valueNumber
                        }

                        this.updatePreset(this.graph)
                        input.remove()
                        this.render()
                  })

                  document.body.append(input)
                  input.focus()

                  return
            }

            if (target.dataset.elementid) {
                  console.log('currentClikedTarget:                    ',
this.currentClickedTarget)

                  if (
                        this.currentClickedTarget !== null &&
                        this.currentClickedTarget !== e.target
                  ) {
                        const nodePrev = this.graph.graph.get(
                              this.currentClickedTarget.dataset.elementid
|| ''
                        )
```

```
                              const nodeCurrent = this.graph.graph.get(
                                    target.dataset.elementid || ''
                              )

                              if (!nodePrev || !nodeCurrent) return

                              this.graph.toggleEdge(nodePrev, nodeCurrent)
                              this.updatePreset(this.graph)

                              //          const          findedEdge          =
[...nodePrev.edges].find(edge => {
                              //    return edge.adjacentNode === nodeCurrent
                              // })

                              // if (!findedEdge) {
                              //    nodePrev.edges.add(new Edge(nodeCurrent, 1))
                              // } else {
                              //    nodePrev.edges.delete(findedEdge)
                              // }

                              this.currentClickedTarget = null

                              this.render()
                              return
                        }

                        this.currentClickedTarget = target

                        this.render()
                  }

                  return
            }

            console.log(e)
            console.log({
                  x: e.x,
                  y: e.y,
                  offsetX: this.offsetX,
                  offsetY: this.offsetY
            })

            const  lastElement  =  [...this.graph.graph.values()].reduce((acc,
node) => {
                  const asNumber = Number(node.value)

                  if (isNaN(asNumber)) {
                        return acc
                  }

                  return asNumber > acc ? asNumber : acc
            }, -1)

            this.graph.addOrGetNode(
                  this.graph.graph,
                  String(lastElement + 1),
                  e.clientX - this.offsetX,
                  e.clientY - this.offsetY
            )
            this.updatePreset(this.graph)

            this.render()
      }
```

```
	getTypeOfContextMenu(e: MouseEvent) {
		if ((e.target as HTMLElement).dataset.elementid) {
			return 'node'
		}

		return 'back'
	}

	onContextMenu(e: MouseEvent) {
		e.preventDefault()

		const typeOfContextMenu = this.getTypeOfContextMenu(e)

		if (typeOfContextMenu === 'node') {
			if (!(e.target as HTMLElement).dataset.elementid) return

			const nodeId = (e.target as HTMLElement).dataset.elementid ||
''

			this.contextMenu.addItem(
				new ContextItem(
					'Delete node',
					async (_, nodeId) => {
						const node = this.graph.graph.get(nodeId)

						if (!node) return

						this.graph.graph.forEach(graphNode => {
							graphNode.edges = new Set(

	[...graphNode.edges].filter(edge => {
									return   edge.adjacentNode
!== node
								})
							)
						})

						this.graph.graph.delete(nodeId)
						this.updatePreset(this.graph)

						this.contextMenu.close()
						this.render()
					},
					nodeId,
					{
						name: 'X',
						code: 'KeyX'
					}
				)
			)
		}

		if (typeOfContextMenu === 'back') {
			this.contextMenu.addItem(
				new ContextItem(
					'Add new node',
					async () => {
						const              lastElement              =
[...this.graph.graph.values()].reduce(
							(acc, node) => {
								const          asNumber          =
Number(node.value)

								if (isNaN(asNumber)) {
```

```
                                                return acc
                                          }

                                          return asNumber > acc ? asNumber
: acc
                                    },
                                    -1
                              )

                              this.graph.addOrGetNode(
                                    this.graph.graph,
                                    String(lastElement + 1),
                                    e.clientX - this.offsetX,
                                    e.clientY - this.offsetY
                              )
                              this.updatePreset(this.graph)

                              this.contextMenu.close()
                              this.render()
                        },
                        null,
                        {
                              name: 'A',
                              code: 'KeyA'
                        }
                  )
            )
      }

      this.contextMenu.changePosition(e.clientX, e.clientY)
      this.contextMenu.renderItems()
}

render() {
      const render = new Render()

      render.render(
            this.graph,
            this.offsetX - window.screenLeft,
            this.offsetY - window.screenTop,
            this.pressedKeyCode,
            this.currentClickedTarget
      )
}

initializeApp() {
      this.#initializeUserEvents()
}

#initializeUserEvents() {
      document.addEventListener('mousedown', (e: MouseEvent) => {
            if (this.contextMenu.open) {
                  this.contextMenu.close()
                  return
            }

            this.onMouseDown(e)
      })
      document.addEventListener('mouseup', () => this.onMouseUp())
      document.addEventListener('mousemove', (e: MouseEvent) =>
            this.onMouseMove(e)
      )

      document.addEventListener('contextmenu', (e: MouseEvent) =>
```

```
                this.onContextMenu(e)
            )
            document.addEventListener('click', (e: MouseEvent) => {
                if (this.contextMenu.open) {
                    this.contextMenu.close()
                    return
                }

                this.onClick(e)
            })

            document.addEventListener('keydown', (e: KeyboardEvent) => {
                if (this.contextMenu.open) {
                    if (e.code === 'Escape') {
                        this.contextMenu.close()
                        return
                    }

                    this.contextMenu.keyPressed(e.code)
                    return
                }

                this.onKeyDown(e)
            })
            document.addEventListener('keyup',     (e:     KeyboardEvent)     =>
this.onKeyUp(e))
        }
}

class App {
        graph: Graph
        currentPreset: string = '1'
        render: () => void = () => {}
        menu: Menu | null = null
        storage: PresetStorage
        storagePresets: StoragePresets

        constructor() {
            this.graph = new Graph()

            this.storage = new PresetStorage()

            if (this.storage.read() === null) {
                this.initializeDefaultPresets()
            }

            this.storagePresets = this.storage.read()
            this.graph.graph = this.graph.createGraph(
                this.storagePresets[this.currentPreset]
            )
        }

        initialize() {
            const userInteractionManager = new UserInteractionManager(
                this.graph,
                this.updatePreset.bind(this),
                this.reloadPresetsFromStorage.bind(this)
            )

            userInteractionManager.initializeApp()

            this.render                                                    =
userInteractionManager.render.bind(userInteractionManager)
            this.menu = this.initializeMenu()
```

```
            this.initializeHidePanelButton()
      }

      graphToPreset(graph: Graph): Preset {
            const nodes = Array.from(graph.graph.values()).map(nodeInfo => {
                  return {
                        value: nodeInfo.value,
                        x: nodeInfo.x ?? 0,
                        y: nodeInfo.y ?? 0
                  }
            })

            const edges = Array.from(graph.graph.values())
                  .map(node => {
                        const base = {
                              from: node.value
                        }

                        return Array.from(node.edges)
                              .map(edge => {
                                    if (edge.status === 'no-direction') return
null

                                    return {
                                          ...base,
                                          to: edge.adjacentNode.value,
                                          weight: edge.weight
                                    }
                              })
                              .filter(item => item !== null) as PresetEdge[]
                  })
                  .flat()

            return {
                  nodes,
                  edges
            }
      }

      reloadPresetsFromStorage() {
            this.storagePresets = this.storage.read()

            this.graph.graph = this.graph.createGraph(
                  this.storagePresets[this.currentPreset]
            )
      }

      updatePreset(graph: Graph) {
            this.storagePresets[this.currentPreset] = this.graphToPreset(graph)

            this.storage.writeAll(this.storagePresets)
            this.storagePresets = this.storage.read()
      }

      initializeHidePanelButton() {
            const hidePanelButton = document.querySelector('#hidePanelButton')
            if (!hidePanelButton) return

            hidePanelButton.addEventListener('click', e => {
                  e.stopPropagation()

                  if (!this.menu) return

                  this.menu.section.classList.toggle('menu--hidden')
```

```
                ;(e.currentTarget as HTMLElement).classList.toggle('hide-
menu--active')
            })
      }

      graphNodesStatusResetter(id: number) {
            if (window.algorithmActiveId !== id) return

            this.graph.graph.forEach(node => {
                  if (window.algorithmActiveId !== id) return

                  node.status = 'default'
            })

            this.render()
      }

      graphEdgesTypeResetter(id: number) {
            if (window.algorithmActiveId !== id) return

            this.graph.graph.forEach(node => {
                  if (window.algorithmActiveId !== id) return

                  node.edges.forEach(edge => {
                        edge.type = 'default'
                  })
            })

            this.render()
      }

      async algorithmWrapper(callback: () => Promise<void>) {
            resetActiveId()

            this.graphNodesStatusResetter(window.algorithmActiveId)

            await callback()

            this.graphNodesStatusResetter(window.algorithmActiveId)
      }

      initializeDefaultPresets() {
            this.storage.writeAll({
                  '1': DEFAULT_PRESET,
                  '2': WEIGHTS_PRESET,
                  '3': MINUS_WEIGHTS_PRESET,
                  '4': {
                        nodes: [],
                        edges: []
                  }
            })
      }

      initializeMenu() {
            const menu = new Menu()

            const bfsElement = new MenuItem(
                  'bfs',
                  async () => {
                        this.algorithmWrapper(async () => {
                              const bfsAlgorithm = new BFS(this.graph,
this.render)
                              await
bfsAlgorithm.bfsWrapper(window.algorithmActiveId)
```

```
                        })
                },
                true
        )

        const dfsElement = new MenuItem(
                'dfs',
                async () => {
                        this.algorithmWrapper(async () => {
                                const dfsAlgorithm = new DFS(this.graph,
this.render)
                                await
dfsAlgorithm.dfsWrapper(window.algorithmActiveId)
                        })
                },
                true
        )

        const resetElement = new MenuItem('reset', async () => {
                window.algorithmActiveId = -1

                this.graphNodesStatusResetter(window.algorithmActiveId)
                this.graphEdgesTypeResetter(window.algorithmActiveId)
        })

        const modeElement = new MenuItem('directed', async target => {
                if (this.graph.mode === 'directed') {
                        this.graph.mode = 'undirected'
                } else {
                        this.graph.mode = 'directed'
                }

                target.textContent = this.graph.mode
                this.render()
        })

        const changeGraphPreset = new MenuItem(this.currentPreset, async
target => {
                const presetValues = Object.keys(this.storagePresets)

                console.log(presetValues)

                const index = presetValues.indexOf(this.currentPreset)

                const next =
                        index === presetValues.length - 1
                                ? presetValues[0]
                                : presetValues[index + 1]

                console.log(target, next)

                if (!next) return

                target.textContent = next
                this.currentPreset = next
                this.graph.graph =
this.graph.createGraph(this.storagePresets[next])

                this.render()
        })

        const weightElement = new MenuItem('weight', async target => {
                if (this.graph.weights === true) {
                        this.graph.weights = false
```

```
                } else {
                        this.graph.weights = true
                }

                target.textContent = this.graph.weights ? 'w' : 'nw'

                this.render()
        })

        const sidePanel = new SidePanel('panel')

        const settingsElement = new MenuItem('S', async () => {
                const FORM_KEY = 'deep'

                if (sidePanel.formId === FORM_KEY && sidePanel.opened) {
                        sidePanel.close()
                        return
                }

                const resetAllPresets = document.createElement('button')
                resetAllPresets.className = 'button'
                resetAllPresets.textContent = 'Reset all presets'

                const getTable = () => {
                        const nodes = Array.from(this.graph.graph.values())
                        const numNodes = nodes.length

                        const distances: number[][] = Array.from({ length:
numNodes }, () =>
                                Array(numNodes).fill(Infinity)
                        )

                        nodes.forEach((node, i) => {
                                distances[i][i] = 0

                                node.edges.forEach(edge => {
                                        if (
                                                this.graph.mode === 'directed' &&
                                                edge.status === 'no-direction'
                                        ) {
                                                return
                                        }

                                        const j = nodes.indexOf(edge.adjacentNode)

                                        distances[i][j] = edge.weight
                                })
                        })

                        const text = [
                                `<tr>${[{ value: '' }, ...nodes]
                                        .map(
                                                node =>
                                                        '<th class="table-cell table-
cell--head">' +
                                                        node.value +
                                                        '</th>'
                                        )
                                        .join('\n')}</tr>`,
                                ...distances.map((item, index) => {
                                        const result: string[] = [
                                                `<th class="table-cell table-cell--
head">${nodes[index].value}</th>`,
                                                ...item.map(subItem => {
```

```
                                                if (subItem === Infinity) {
                                                      return `<td class="table-
cell"></td>`
                                                }

                                                return    `<td    class="table-
cell">${subItem}</td>`
                                          })
                                    ]

                                    return `<tr>${result.join('\n')}</tr>`
                              })
                        ]

                        const table = document.createElement('table')
                        table.className = 'table'
                        table.innerHTML = text.join('\n')

                        return table
                  }

                  let table = getTable()

                  const rerenderTable = document.createElement('button')
                  rerenderTable.className = 'button'
                  rerenderTable.textContent = 'Update table'
                  rerenderTable.setAttribute('style', 'margin-top: 10px')

                  rerenderTable.addEventListener('click', () => {
                        const oldTable = table
                        table = getTable()

                        oldTable.replaceWith(table)
                  })

                  const result = document.createElement('div')
                  result.append(resetAllPresets)
                  result.append(table)
                  result.append(rerenderTable)

                  sidePanel.renderForm(result, FORM_KEY)
                  sidePanel.open()
            })

            const deepElement = new MenuItem('deep', async () => {
                  const FORM_KEY = 'deep'

                  if (sidePanel.formId === FORM_KEY && sidePanel.opened) {
                        sidePanel.close()
                        return
                  }

                  const deepForm = new Form('Deep level', async data => {
                        const startNode = this.graph.graph.get(data.startNodeId)
                        if (!startNode || data.endNodeId === undefined) return

                        resetActiveId()
                        startNode.status = 'done'

                        const maxLevel = Number(data.endNodeId) || 2

                        const          nodesByDeepLevel          =          new
NodesByDeepLevel(this.graph, this.render)
```

```
                        const value = await
nodesByDeepLevel.getNodesByDeepLevel(
                              startNode,
                              [],
                              window.algorithmActiveId,
                              maxLevel
                        )

                        console.log(value)

                        this.graphNodesStatusResetter(window.algorithmActiveId)
                  })

                  deepForm.addInput('startNodeId', 'Start node id:')
                  deepForm.addInput('endNodeId', 'Deep level:')

                  sidePanel.renderForm(deepForm.getForRender(), FORM_KEY)
                  sidePanel.open()
            })

            const idk1Element = new MenuItem('idk-1', async () => {
                  const FORM_KEY = 'idk-1'

                  if (sidePanel.formId === FORM_KEY && sidePanel.opened) {
                        sidePanel.close()
                        return
                  }

                  const deepForm = new Form('IDK-1', async data => {
                        const startNode = this.graph.graph.get(data.startNodeId)
                        if (!startNode) return

                        resetActiveId()
                        startNode.status = 'done'

                        const edgesTypes = new EdgesTypes(this.render,
this.graph)

                        const value = await
edgesTypes.edgesTypes(window.algorithmActiveId)



                        console.log(value)

                        this.graphEdgesTypeResetter(window.algorithmActiveId)
                        this.graphNodesStatusResetter(window.algorithmActiveId)
                  })

                  deepForm.addInput('startNodeId', 'Start node id:')

                  sidePanel.renderForm(deepForm.getForRender(), FORM_KEY)
                  sidePanel.open()
            })

            const bellmanFordElement = new MenuItem('bellmanFord', async () => {
                  const FORM_KEY = 'bellmanFord'

                  if (sidePanel.formId === FORM_KEY && sidePanel.opened) {
                        sidePanel.close()
                        return
                  }
```

```
                const bellmanFordForm = new Form('Bellman Ford', async data =>
{
                      const startNode = this.graph.graph.get(data.startNodeId)
                      if (!startNode) return

                      resetActiveId()
                      startNode.status = 'done'

                      const bellmanFord = new BellmanFord(this.graph)

                      const           distances           =           await
bellmanFord.bellmanFord(startNode)

                      if (distances) {
                            const result = [...distances.entries()].reduce(
                                  (acc, [node, distance]) => {
                                        return {
                                              ...acc,
                                              [node.value]: distance
                                        }
                                  },
                                  {}
                            )

                            console.log(result)

      bellmanFordForm.renderCustomOutput(JSON.stringify(result, null, 2))
                      } else {
                            bellmanFordForm.renderCustomOutput('Paths     not
found')
                      }

                      this.graphNodesStatusResetter(window.algorithmActiveId)
                })

                bellmanFordForm.addInput('startNodeId', 'Start node id:')

                sidePanel.renderForm(bellmanFordForm.getForRender(),
FORM_KEY)
                sidePanel.open()
          })

          const dijkstraElement = new MenuItem('dijkstra', async () => {
                const FORM_KEY = 'dijkstra'

                if (sidePanel.formId === FORM_KEY && sidePanel.opened) {
                      sidePanel.close()
                      return
                }

                const dijkstraForm = new Form('Dijkstra', async data => {
                      const startNode = this.graph.graph.get(data.startNodeId)
                      const endNode = this.graph.graph.get(data.endNodeId)

                      if (!startNode || !endNode) return

                      resetActiveId()

                      startNode.status = 'done'
                      endNode.status = 'done'

                      const dijkstra = new Dijkstra(this.graph, this.render)
                      const   result   =   await   dijkstra.dijkstra(startNode,
endNode)
```

```
                        console.log(result)
                        if (result === null) {
                              dijkstraForm.renderCustomOutput('Not found!')
                        } else {
                              dijkstraForm.renderCustomOutput(
                                    result?.map(item => item.value).join(' -> ')
                              )
                        }

                        this.graphNodesStatusResetter(window.algorithmActiveId)
                  })

                  dijkstraForm.addInput('startNodeId', 'Start node id:')
                  dijkstraForm.addInput('endNodeId', 'End node id:')

                  sidePanel.renderForm(dijkstraForm.getForRender(), FORM_KEY)
                  sidePanel.open()
            })

            const floydWarshallElement = new MenuItem('floydWarshall', async ()
=> {
                  const FORM_KEY = 'floydWarshall'

                  if (sidePanel.formId === FORM_KEY && sidePanel.opened) {
                        sidePanel.close()
                        return
                  }

                  const floydWarshallForm = new Form('Floyd Warshall', async ()
=> {
                        resetActiveId()

                        const                        nodes                        =
Array.from(this.graph.graph.values()).sort((a, b) => {
                              return Number(a.value) - Number(b.value)
                        })
                        console.log('%c●', 'color: #731d1d', nodes)

                        const floydWarshall = new FloydWarshall(this.graph)
                        const result = await floydWarshall.floydWarshall(nodes)

                        result.unshift([])
                        result.unshift(nodes.map(node => node.value))

                        const text = JSON.stringify(
                              [
                                    ...result.map(item => {
                                          const tmp: string[] = []

                                          item.map(subItem => {
                                                let s = String(subItem)

                                                if (subItem === Infinity) {
                                                      s = '-'
                                                }

                                                if (s.length < 2) {
                                                      s = ' ' + s
                                                }

                                                if (s.length < 3 && s.length ===
2) {
                                                      s = ' ' + s
```

```
                                                }

                                                tmp.push(s)
                                          })

                                          return tmp.join(',')
                                    })
                              ],
                              null,
                              2
                        )

                        floydWarshallForm.renderCustomOutput(text)

                        this.graphNodesStatusResetter(window.algorithmActiveId)
                  })

                  sidePanel.renderForm(floydWarshallForm.getForRender(),
FORM_KEY)
                  sidePanel.open()
            })

            menu.addItem(bfsElement)
            menu.addItem(dfsElement)
            menu.addItem(new MenuDivider())
            menu.addItem(resetElement, 'rose')
            menu.addItem(modeElement, 'indigo')
            menu.addItem(changeGraphPreset, 'sky')
            menu.addItem(weightElement, 'amber')
            menu.addItem(settingsElement, 'teal')
            menu.addItem(new MenuDivider())
            menu.addItem(deepElement)
            menu.addItem(idk1Element)
            menu.addItem(bellmanFordElement)
            menu.addItem(dijkstraElement)
            menu.addItem(floydWarshallElement)

            menu.render()
            return menu
      }
}

const app = new App()

app.initialize()


src\vite-env.d.ts
/// <reference types="vite/client" />


src\algorithms\BellmanFord.ts
import { Node } from '../models/Node.ts'
import { Graph } from '../models/Graph.ts'

export class BellmanFord {
      graph: Graph

      constructor(graph: Graph) {
            this.graph = graph
      }

      async bellmanFord(startNode: Node) {
            const distances = new Map<Node, number>()
```

```
            for (const node of this.graph.graph.values()) {
                  distances.set(node, Infinity)
            }

            distances.set(startNode, 0)

            for (let i = 0; i < this.graph.graph.size; i++) {
                  for (const currentNode of [...this.graph.graph.values()]) {
                        for (const edge of currentNode.edges) {
                              if (
                                    this.graph.mode === 'directed' &&
                                    edge.status === 'no-direction'
                              ) {
                                    continue
                              }

                              const newDistance = distances.get(currentNode)! +
edge.weight

                              if                  (newDistance                  <
distances.get(edge.adjacentNode)!) {
                                    distances.set(edge.adjacentNode,
newDistance)
                              }
                        }
                  }
            }

            for (const currentNode of this.graph.graph.values()) {
                  if (currentNode === startNode) {
                        continue
                  }

                  for (const edge of currentNode.edges) {
                        if (this.graph.mode === 'directed' && edge.status ===
'no-direction') {
                              continue
                        }

                        if (
                              distances.get(currentNode)! + edge.weight <
                              distances.get(edge.adjacentNode)!
                        ) {
                              console.error('Graph contains a negative cycle.')
                              return
                        }
                  }
            }

            return distances
      }
}



## BellmanFord

{
    graph: Graph;
    bellmanFord: (startNode: Node) => Promise<Map<Node, number> | undefined>;
}

src\algorithms\BFS.ts
```

```
import { Node } from '../models/Node.ts'
import { sleep, setNodeStatus } from '../helpers'
import { DELAY } from '../config/delay.ts'
import { Graph, GraphValue } from '../models/Graph.ts'

class TreeNode {
	value: GraphValue
	childrens: TreeNode[] = []

	constructor(value: GraphValue) {
		this.value = value
	}

	find(node: TreeNode, value: GraphValue): TreeNode | undefined {
		if (node.value === value) return node

		for (const child of node.childrens) {
			const result = this.find(child, value)

			if (result) {
				return result
			}
		}

		return undefined
	}

	add(node: TreeNode) {
		this.childrens.push(node)
	}
}

class Tree {
	root: TreeNode

	constructor(value: GraphValue) {
		this.root = new TreeNode(value)
	}

	add(node: GraphValue, value: GraphValue) {
		const prevNode = this.root.find(this.root, node)

		if (!prevNode) return

		prevNode.add(new TreeNode(value))
	}

	display() {
		console.log(this.root)
	}
}

export class BFS {
	graph: Graph
	render: () => void

	constructor(graph: Graph, render: () => void) {
		this.graph = graph
		this.render = render
	}

	async bfsWrapper(id: number) {
		const visited: Node[] = []
```

```
            for (const item of this.graph.graph.values()) {
                  if (!visited.includes(item)) {
                        if (window.algorithmActiveId !== id) {
                              return
                        }

                        await this.bfs(item, visited, id)
                  }
            }

            this.render()
      }

      async bfs(node: Node, visited: Node[], id: number) {
            const tree = new Tree(node.value)

            const queue = [node]

            while (queue.length > 0) {
                  if (window.algorithmActiveId !== id) {
                        return
                  }

                  const item = queue.shift()
                  if (!item) return null

                  visited.push(item)

                  item.edges.forEach(edge => {
                        if (this.graph.mode === 'directed' && edge.status ===
'no-direction')
                              return

                        const adjacentNode = edge.adjacentNode

                        if          (!visited.includes(adjacentNode)          &&
!queue.includes(adjacentNode)) {
                              tree.add(item.value ?? -1, adjacentNode.value ??
-1)
                              queue.push(adjacentNode)
                        }
                  })

                  await setNodeStatus(
                        item,
                        {
                              status: 'progress',
                              sleep: false
                        },
                        this.render
                  )
                  await sleep(DELAY)
            }

            tree.display()

            this.render()
      }
}


src\algorithms\DFS.ts
import { Node } from '../models/Node.ts'
import { sleep, setNodeStatus } from '../helpers'
```

```
import { DELAY } from '../config/delay.ts'
import { Graph } from '../models/Graph.ts'

export class DFS {
	graph: Graph
	render: () => void

	constructor(graph: Graph, render: () => void) {
		this.graph = graph
		this.render = render
	}

	async dfsWrapper(id: number) {
		const visited: Node[] = []

		for (const item of this.graph.graph.values()) {
			if (!visited.includes(item)) {
				if (window.algorithmActiveId !== id) {
					return
				}

				await this.dfs(item, visited, id)
			}
		}

		this.render()
	}

	async dfs(node: Node, visited: Node[], id: number) {
		const jungle: Node[] = []
		const stack = [node]

		while (stack.length > 0) {
			if (window.algorithmActiveId !== id) {
				return
			}

			const item = stack.pop()
			if (!item) return null

			visited.push(item)
			jungle.push(item)
			;[...item.edges].toReversed().forEach(edge => {
				if (this.graph.mode === 'directed' && edge.status === 'no-direction') {
					return
				}

				const adjacentNode = edge.adjacentNode

				if (!visited.includes(adjacentNode) && !stack.includes(adjacentNode)) {
					stack.push(adjacentNode)
				}
			})

			await setNodeStatus(
				item,
				{
					status: 'progress',
					sleep: false
				},
				this.render
			)
```

```
                await sleep(DELAY)
            }

            this.render()

            console.log('DFS:', jungle.map(item => item.value).join(', '))
        }
}


src\algorithms\Dijkstra.ts
import { Node } from '../models/Node.ts'
import { setNodeStatus } from '../helpers'
import { Graph } from '../models/Graph.ts'

export class Dijkstra {
      render: () => void
      graph: Graph

      constructor(graph: Graph, render: () => void) {
            this.graph = graph
            this.render = render
      }

      async initHashTables(
            start: Node,
            unprocessedNodes: Set<Node>,
            timeToNodes: Map<Node, number>
      ) {
            for (const node of this.graph.graph.values()) {
                  unprocessedNodes.add(node)
                  timeToNodes.set(node, Infinity)
            }

            timeToNodes.set(start, 0)
      }

      async getNodeWithMinTime(
            unprocessedNodes: Set<Node>,
            timeToNodes: Map<Node, number>
      ) {
            let nodeWithMinTime: Node | null = null

            let minTime = Infinity

            for (const node of unprocessedNodes) {
                  const time = timeToNodes.get(node)

                  if (time !== undefined && time < minTime) {
                        minTime = time
                        nodeWithMinTime = node
                  }
            }

            return nodeWithMinTime
      }

      async calculateTimeToEachNode(
            unprocessedNodes: Set<Node>,
            timeToNodes: Map<Node, number>
      ) {
            while (unprocessedNodes.size > 0) {
```

```
                const node = await this.getNodeWithMinTime(unprocessedNodes,
timeToNodes)

                if (!node) return
                if (timeToNodes.get(node) === Infinity) return

                await setNodeStatus(
                      node,
                      {
                            status: 'progress',
                            statusForChange: 'default'
                      },
                      this.render
                )

                for (const edge of node.edges) {
                      if (this.graph.mode === 'directed' && edge.status ===
'no-direction') {
                            continue
                      }

                      const adjacentNode = edge.adjacentNode

                      if (unprocessedNodes.has(adjacentNode)) {
                            const nodeTime = timeToNodes.get(node)
                            if (nodeTime === undefined) continue

                            const              adjacentNodeTime              =
timeToNodes.get(adjacentNode)
                            if (adjacentNodeTime === undefined) continue

                            const timeToCheck = nodeTime + edge.weight

                            if (timeToCheck < adjacentNodeTime) {
                                  timeToNodes.set(adjacentNode, timeToCheck)
                            }
                      }
                }

                unprocessedNodes.delete(node)
          }
     }

     async getShortestPath(
          start: Node,
          end: Node,
          timeToNodes: Map<Node, number>
     ) {
          const path = []
          let node = end

          while (node !== start) {
                const minTimeToNode = timeToNodes.get(node)
                path.unshift(node)

                for (const [parent, parentEdge] of node.parents.entries()) {
                      if (timeToNodes.has(parent) === undefined) continue

                      const previousNodeFound =
                            Number(parentEdge.weight                          +
(timeToNodes.get(parent) ?? 0)) ===
                            minTimeToNode

                      if (previousNodeFound) {
```

```
                              timeToNodes.delete(node)
                              node = parent
                              break
                        }
                  }
            }

            path.unshift(node)
            return path
      }

      async dijkstra(start: Node, end: Node) {
            const unprocessedNodes = new Set<Node>()
            const timeToNodes = new Map<Node, number>()

            await this.initHashTables(start, unprocessedNodes, timeToNodes)
            await this.calculateTimeToEachNode(unprocessedNodes, timeToNodes)

            if (timeToNodes.get(end) === Infinity) return null
            return await this.getShortestPath(start, end, timeToNodes)
      }
}


src\algorithms\EdgesTypes.ts
import { setNodeStatus } from '../helpers'
import { Graph } from '../models/Graph'
import { Node } from '../models/Node'

export class EdgesTypes {
      graph: Graph
      render: () => void

      constructor(render: () => void, graph: Graph) {
            this.render = render
            this.graph = graph
      }

      async edgesTypes(id: number) {
            const visited: Node[] = []
            const startTime: Map<Node, number> = new Map()
            const endTime: Map<Node, number> = new Map()
            const state: { time: number } = { time: 0 }

            for (const item of this.graph.graph.values()) {
                  if (!visited.includes(item)) {
                        if (window.algorithmActiveId !== id) {
                              return
                        }

                        await  this.edgesTypesInner(item,  visited,  startTime,
endTime, state, id)
                  }
            }

            console.log(startTime.size, endTime.size)

            this.render()
      }

      async edgesTypesInner(
            node: Node,
            visited: Node[],
            startTime: Map<Node, number>,
```

```
            endTime: Map<Node, number>,
            state: { time: number },
            id: number
      ) {
            const jungle: Node[] = []

            if (window.algorithmActiveId !== id) {
                  return
            }

            if (!node) return null

            startTime.set(node, state.time)
            state.time++

            visited.push(node)
            jungle.push(node)

            for (const edge of [...node.edges].toReversed()) {
                  if (this.graph.mode === 'directed' && edge.status === 'no-
direction') {
                        continue
                  }

                  const adjacentNode = edge.adjacentNode

                  // if (item.value === 8) debugger

                  if (!visited.includes(adjacentNode)) {
                        console.log(
                              'Tree Edge: ' + node.value + '-->' +
adjacentNode.value + '<br>'
                        )

                        edge.type = 'default'

                        await this.edgesTypesInner(
                              adjacentNode,
                              visited,
                              startTime,
                              endTime,
                              state,
                              id
                        )
                  } else {
                        // if parent node is traversed after the neighbour node

                        const itemStartTime = startTime.get(node) ?? -1
                        const adjacentStartTime = startTime.get(adjacentNode) ??
-1
                        const itemEndTime = endTime.get(node) ?? -1
                        const adjacentEndTime = endTime.get(adjacentNode) ?? -1

                        console.table({
                              value: node.value,
                              // startTime: startTime,
                              // endTime: JSendTime,
                              adjacentNode: adjacentNode.value,
                              itemStartTime: itemStartTime,
                              adjacentStartTime: adjacentStartTime,
                              itemEndTime: itemEndTime,
                              adjacentEndTime: adjacentEndTime
                        })
```

```
                        if (itemStartTime >= adjacentStartTime &&
adjacentEndTime === -1) {
                              console.log(
                                    'Back Edge: ' + node.value + '-->' +
adjacentNode.value + '<br>'
                              )
                              edge.type = 'back'
                        }

                        // if the neighbour node is a  but not a part of the tree
                        else if (itemStartTime < adjacentStartTime &&
adjacentEndTime !== -1) {
                              console.log(
                                    'Forward Edge: ' + node.value + '-->' +
adjacentNode.value + '<br>'
                              )
                              edge.type = 'forward'
                        }

                        // if parent and neighbour node do not
                        // have any ancestor and descendant relationship between
them
                        else {
                              console.log(
                                    'Cross Edge: ' + node.value + '-->' +
adjacentNode.value + '<br>'
                              )
                              edge.type = 'cross'
                        }
                  }
            }

            endTime.set(node, state.time)
            state.time++

            await setNodeStatus(
                  node,
                  {
                        status: 'progress',
                        renderBeforeSleep: true
                  },
                  this.render.bind(this)
            )

            this.render()

            //  console.log('DFS:', jungle.map(item => item.value).join(',
      '))
      }
}


src\algorithms\FloydWarshall.ts
import { Node } from '../models/Node.ts'
import { Graph } from '../models/Graph.ts'

export class FloydWarshall {
      graph: Graph

      constructor(graph: Graph) {
            this.graph = graph
      }

      floydWarshall(nodes: Node[]) {
```

```
            const numNodes = nodes.length

            const distances = Array.from({ length: numNodes }, () =>
                  Array(numNodes).fill(Infinity)
            )

            nodes.forEach((node, i) => {
                  distances[i][i] = 0

                  node.edges.forEach(edge => {
                        if (this.graph.mode === 'directed' && edge.status ===
'no-direction') {
                              return
                        }

                        const j = nodes.indexOf(edge.adjacentNode)

                        distances[i][j] = edge.weight
                  })
            })

            for (let i = 0; i < numNodes; i++) {
                  for (let j = 0; j < numNodes; j++) {
                        for (let k = 0; k < numNodes; k++) {
                              if    (distances[j][i]    +    distances[i][k]    <
distances[j][k]) {
                                    distances[j][k]    =    distances[j][i]    +
distances[i][k]
                              }
                        }
                  }
            }

            nodes.forEach((nodeTop, i) => {
                  nodes.forEach((nodeBottom, j) => {
                        const result = distances[i][j]

                        if (result === Infinity) return

                        console.log(`${nodeTop.value}  =>  ${nodeBottom.value}:
${result}`)
                  })
            })

            return distances
      }
}


src\algorithms\NodesByDeepLevel.ts
import { Node } from '../models/Node.ts'
import { sleep } from '../helpers'
import { DELAY } from '../config/delay.ts'
import { Graph } from '../models/Graph.ts'

export class NodesByDeepLevel {
      graph: Graph
      render: () => void

      constructor(graph: Graph, render: () => void) {
            this.graph = graph
            this.render = render
      }
```

```
	async getNodesByDeepLevel(
		node: Node,
		visited: Node[],
		id: number,
		maxLevel = 2,
		level = 0
	) {
		if (level > maxLevel) return []

		visited.push(node)

		node.status = 'progress'

		this.render()

		await sleep(DELAY)

		const result: (string | null)[] = []

		for (const edge of node.edges) {
			if (this.graph.mode === 'directed' && edge.status === 'no-
direction') {
				continue
			}

			if (visited.includes(edge.adjacentNode)) {
				continue
			}

			if (window.algorithmActiveId !== id) {
				return
			}

			const nodes = await this.getNodesByDeepLevel(
				edge.adjacentNode,
				visited,
				id,
				maxLevel,
				level + 1
			)

			if (nodes === undefined) {
				continue
			}

			result.push(...nodes)
		}

		return [node.value, ...result].flat()
	}
}


src\config\defineWindowVariables.ts
export {}

declare global {
	interface Window {
		DEBUG: boolean
		algorithmActiveId: number
		render: () => void
	}
}
```

```
window.DEBUG = false
window.algorithmActiveId = -1
window.render = () => {}


src\config\delay.ts
export const DELAY = 200


src\config\nodeColors.ts
export const NODE_COLORS = {
      default: 'white',
      checked: 'var(--color-red-300)',
      progress: 'var(--color-yellow-300)',
      done: 'var(--color-green-300)',
      passed: 'var(--color-slate-300)'
}


src\helpers\index.ts
export { sleep } from './sleep'
export { setNodeStatus } from './setNodeStatus'
export { resetActiveId } from './resetActiveId'


src\helpers\resetActiveId.ts
export const resetActiveId = () => {
      const activeId = new Date().getTime()
      window.algorithmActiveId = activeId
}


src\helpers\setNodeStatus.ts
import { Node } from '../models/Node.ts'
import { sleep } from './sleep.ts'
import { DELAY } from '../config/delay.ts'

export async function setNodeStatus(
      node: Node,
      params: {
            status?: Node['status']
            sleep?: boolean
            render?: boolean
            statusForChange?: Node['status'] | 'any'
            renderBeforeSleep?: boolean
      } = {},
      render: () => void
) {
      const status = params.status ?? 'default'
      const sleepFlag = params.sleep ?? true
      const renderProp = params.render ?? true
      const statusForChange = params.statusForChange ?? node.status
      const renderBeforeSleep = params.renderBeforeSleep ?? false

      if (statusForChange && statusForChange === node.status) {
            node.status = status
      }

      if (renderProp && renderBeforeSleep) {
            render()
      }

      if (sleepFlag) {
            await sleep(DELAY)
```

```
      }

      if (renderProp && !renderBeforeSleep) {
            render()
      }
}


src\helpers\sleep.ts
export async function sleep(time: number) {
  await new Promise(resolve => {
    setTimeout(() => {
      resolve(null)
    }, time)
  })
}


src\models\Edge.ts
import { Node } from './Node'

class Edge {
      adjacentNode: Node
      weight: number
      status: 'standart' | 'no-direction' = 'standart'
      type: 'default' | 'forward' | 'cross' | 'back' = 'default'

      constructor(
            adjacentNode: Node,
            weight: number,
            status: 'standart' | 'no-direction' = 'standart'
      ) {
            this.adjacentNode = adjacentNode
            this.weight = weight
            this.status = status
      }
}

export { Edge }


src\models\Graph.ts
import { Edge } from './Edge'
import { Node } from './Node'

export type GraphValue = string

export class Graph {
      graph = new Map<GraphValue, Node>()
      mode: 'directed' | 'undirected' = 'directed'
      weights: boolean = false

      addOrGetNode(
            graph: Map<GraphValue, Node>,
            value: GraphValue,
            x?: number,
            y?: number
      ) {
            if (value.length === 0) return null
            if (graph.has(value)) return graph.get(value) as Node

            const node: Node = new Node(value, x, y)
            graph.set(value, node)
```

```
            return node
      }

      toggleEdge(fromNode: Node, toNode: Node, weight: number = 1) {
            const findedEdgeFromTo = [...fromNode.edges].find(edge => {
                  return edge.adjacentNode === toNode
            })

            const findedEdgeToFrom = [...toNode.edges].find(edge => {
                  return edge.adjacentNode === fromNode
            })

            if (findedEdgeFromTo) {
                  if (findedEdgeFromTo.status === 'no-direction') {
                        findedEdgeFromTo.status = 'standart'
                        return
                  }

                  fromNode.edges.delete(findedEdgeFromTo)
                  toNode.parents.delete(fromNode)

                  if (findedEdgeToFrom) {
                        if (this.mode === 'directed') {
                              if (findedEdgeToFrom.status === 'no-direction') {
                                    toNode.edges.delete(findedEdgeToFrom)
                                    fromNode.parents.delete(toNode)
                              }

                              if (findedEdgeToFrom.status === 'standart') {
                                    const newEdge = new Edge(toNode, 1, 'no-
direction')
                                    fromNode.edges.add(newEdge)
                                    toNode.parents.set(fromNode, newEdge)
                              }
                        }

                        if (this.mode === 'undirected') {
                              toNode.edges.delete(findedEdgeToFrom)
                              fromNode.parents.delete(toNode)
                        }
                  }
            } else {
                  const newEdge = new Edge(toNode, weight, 'standart')
                  fromNode.edges.add(newEdge)
                  toNode.parents.set(fromNode, newEdge)

                  if (!findedEdgeToFrom) {
                        if (this.mode === 'directed') {
                              const newEdge = new Edge(fromNode, weight, 'no-
direction')
                              toNode.edges.add(newEdge)
                              fromNode.parents.set(toNode, newEdge)
                        }

                        if (this.mode === 'undirected') {
                              const newEdge = new Edge(fromNode, weight,
'standart')
                              toNode.edges.add(newEdge)
                              fromNode.parents.set(toNode, newEdge)
                        }
                  }
            }
      }
```

```
	createGraph(newGraphData: {
		nodes: {
			value: GraphValue
			x: number
			y: number
		}[]
		edges: {
			from: GraphValue
			to: GraphValue
			weight: number
		}[]
	}) {
		const newGraph = new Map<GraphValue, Node>()

		for (const nodeData of newGraphData.nodes) {
			this.addOrGetNode(newGraph,    nodeData.value,    nodeData.x,
nodeData.y)
		}

		for (const row of newGraphData.edges) {
			const node = this.addOrGetNode(newGraph, row.from)

			if (!node) continue

			const adjuacentNode = this.addOrGetNode(newGraph, row.to)

			if (adjuacentNode === null) continue

			this.toggleEdge(node, adjuacentNode, row.weight)
		}

		return newGraph
	}
}


src\models\Node.ts
import { Edge } from './Edge'

import { GraphValue } from './Graph'

class Node {
	value: GraphValue
	edges = new Set<Edge>()
	parents = new Map<Node, Edge>()

	x: number | null = null
	y: number | null = null
	status: 'default' | 'progress' | 'done' | 'passed' = 'default'

	constructor(
		value: GraphValue,
		x: number | null = null,
		y: number | null = null
	) {
		this.value = value

		this.x = x
		this.y = y
	}

	toString() {
		return JSON.stringify({
			from: this.value,
```

```
                to: -1,
                weight: 1,
                x: this.x,
                y: this.y
            })
        }
}

export { Node }


src\modules\ContextMenu.ts
export class ContextMenu {
      root: Element
      items: (ContextItem<unknown> | ContextDivider)[] = []

      private _open = false

      get open() {
            return this._open
      }

      set open(value: boolean) {
            this._open = value
      }

      close() {
            this.root.innerHTML = ''
            this.items = []
            this.root.setAttribute('style', '')

            this.open = false
      }

      constructor() {
            const root = document.querySelector('#contextMenu')

            if (!root) {
                  throw new Error('Error in context root')
            }

            this.root = root

            this.root.addEventListener('mousedown', e => {
                  e.stopPropagation()
            })
      }

      renderItems() {
            this.root.innerHTML = ''

            this.items.map(item => {
                  if (item instanceof ContextDivider) {
                        const divider = document.createElement('div')
                        divider.className = 'context-menu__divider'
                        this.root.append(divider)
                        return
                  }

                  const button = document.createElement('button')
                  button.className = 'context-menu__button'

                  button.innerHTML = `
                  <div class="context-menu__button_key">${item.key?.name}</div>
```

```
                        <div class="content-menu__button_name">${item.text}</div>
                        `

                        button.addEventListener('click', e => {
                              item.callback(e.target as HTMLButtonElement, item.other)
                        })

                        this.root.append(button)
                  })

                  this.open = true
            }

            addItem(item: ContextItem<unknown> | ContextDivider) {
                  this.items.push(item)
            }

            changePosition(x: number, y: number) {
                  this.root.setAttribute('style', `top: ${y}px; left: ${x}px`)
            }

            keyPressed(key: string) {
                  const itemByPressedKey = this.items.find(item => {
                        if (!(item instanceof ContextItem)) return false

                        return item.key?.code === key
                  }) as ContextItem<unknown>

                  console.log(key, itemByPressedKey)

                  if (!itemByPressedKey) return

                  itemByPressedKey.callback(
                        document.createElement('button'),
                        itemByPressedKey.other
                  )
            }
      }

      interface ContextItemKey {
            name: string
            code: string
      }

      export class ContextItem<T> {
            text: string
            callback: (target: HTMLButtonElement, other: T) => Promise<void>
            other?: T
            key?: ContextItemKey

            constructor(
                  text: string,
                  callback: (target: HTMLButtonElement, other: T) => Promise<void>,
                  other?: T,
                  key?: ContextItemKey
            ) {
                  this.text = text
                  this.callback = callback
                  this.other = other
                  this.key = key
            }
      }

      export class ContextDivider {}
```

src\modules\Form.ts

```
function getRandomInt(max: number) {
	return Math.floor(Math.random() * max)
}

export class Form {
	inputs: HTMLElement[] = []
	title: string
	callback: (value: Record<string, string>) => void
	customOutput: HTMLElement | null = null

	constructor(
		title: string,
		callback: (value: Record<string, string>) => void
	) {
		this.title = title
		this.callback = callback
	}

	addInput(key: string, title: string) {
		const id = `${getRandomInt(1000)}-${new Date().getTime()}`

		const label = document.createElement('label')
		label.className = 'panel__form-label'
		label.htmlFor = id

		const div = document.createElement('div')
		div.textContent = title

		const input = document.createElement('input')
		input.className = 'panel__form-input panel__form-input--from'
		input.id = id
		input.dataset.key = key
		input.type = 'text'

		label.append(div, input)

		this.inputs.push(label)
	}

	renderCustomOutput = (child: HTMLElement | string) => {
		if (!this.customOutput) return

		if (typeof child === 'string') {
			this.customOutput.innerHTML = child
			return
		}

		this.customOutput.innerHTML = ''
		this.customOutput.append(child)
	}

	getForRender() {
		const form = document.createElement('form')
		form.className = 'panel__form'

		const h2 = document.createElement('h2')
		h2.className = 'panel__form-heading'
		h2.textContent = this.title

		form.append(h2)
		form.append(...this.inputs)
```

```
            const button = document.createElement('button')
            button.textContent = 'Run'
            button.className = 'panel__form-button'

            const customOutput = document.createElement('pre')
            customOutput.className = 'div'

            this.customOutput = customOutput

            form.append(button)
            form.append(customOutput)

            form.onsubmit = e => {
                  e.preventDefault()

                  const allInputs = [
                        ...(e.target                                          as
HTMLFormElement).querySelectorAll('input')
                  ]

                  const result = allInputs.reduce((acc, input) => {
                        const key = input.dataset.key

                        if (!key) {
                              return acc
                        }

                        return {
                              ...acc,
                              [key]: input.value
                        }
                  }, {})

                  this.callback(result)
            }

            return form
      }
}


src\modules\Menu.ts
type TextColors =
      | 'pink'
      | 'green'
      | 'sky'
      | 'amber'
      | 'lime'
      | 'teal'
      | 'indigo'
      | 'purple'
      | 'rose'

export class Menu {
      items: [MenuItem | MenuDivider, TextColors][] = []

      section: Element

      constructor() {
            const section = document.querySelector('#menu')

            if (!section) {
                  throw new Error('section not found')
```

```
            }

            this.section = section
      }

      render() {
            const nav = document.createElement('nav')
            nav.className = 'menu__nav'

            const ul = document.createElement('ul')
            ul.className = 'menu__list'

            ul.append(
                  ...this.items.map(([item, color]) => {
                        const listElement = document.createElement('li')
                        listElement.className = 'menu__item'

                        if (item instanceof MenuItem) {
                              const button = document.createElement('button')
                              button.className   =   `menu__link   menu__link--
${color}`

                              button.textContent = item.text
                              button.onclick = async e => {
                                    if (item.highlight) {
                                          button.classList.add('menu__link--
active')
                                    }

                                    await       item.callback(e.target       as
HTMLButtonElement)

                                    if (item.highlight) {
                                          button.classList.remove('menu__link--
active')
                                    }
                              }

                              listElement.append(button)
                        }

                        if (item instanceof MenuDivider) {
                              const divider = document.createElement('div')
                              divider.className = `menu__divider`

                              listElement.append(divider)
                        }

                        return listElement
                  })
            )

            nav.append(ul)
            nav.onclick = e => {
                  e.stopPropagation()
            }

            this.section.innerHTML = ''
            this.section.append(nav)
      }

      addItem(item: MenuItem | MenuDivider, color: TextColors = 'pink') {
            this.items.push([item, color])
      }
```

```
}

export class MenuDivider {}

export class MenuItem {
	text: string
	callback: (target: HTMLButtonElement) => Promise<void>
	highlight: boolean

	constructor(
		text: string,
		callback: (target: HTMLButtonElement) => Promise<void>,
		highlight: boolean = false
	) {
		this.text = text
		this.callback = callback
		this.highlight = highlight
	}
}
```

src\modules\Panel.ts

```
export class SidePanel {
	root: HTMLElement
	public formId: string = ''
	private _opened: boolean = false

	get opened() {
		return this._opened
	}

	set opened(newValue: boolean) {
		this._opened = newValue
		if (newValue === false) {
			this.root.classList.remove('panel--opened')
		} else {
			this.root.classList.add('panel--opened')
		}
	}

	constructor(id: string) {
		const newRoot = document.querySelector(`.${id}`)

		if (!newRoot) {
			throw new Error('Error with SidePanel id')
		}

		this.root = newRoot as HTMLElement
	}

	renderForm(form: HTMLElement, formId: string) {
		this.root.innerHTML = ''
		this.root.append(form)
		this.formId = formId
	}

	open() {
		this.opened = true
	}

	close() {
		this.opened = false
	}
}
```

src\modules\Render.ts

```
import { NODE_COLORS } from '../config/nodeColors.ts'
import { Node } from '../models/Node.ts'
import { Graph } from '../models/Graph.ts'

export class Render {
	#getNodeStatusForRender(
		node: Node,
		currentClickedTarget: HTMLElement | null
	) {
		return currentClickedTarget &&
			currentClickedTarget.dataset.elementid === node.value
			? 'checked'
			: node.status
	}

	#getRenderedCircles(
		graph: Graph,
		offsetX: number,
		offsetY: number,
		currentClickedTarget: HTMLElement | null
	) {
		return [...graph.graph.entries()].map(([_, node]) => {
			const status = this.#getNodeStatusForRender(node,
currentClickedTarget)
			const x = (node.x ?? 0) + offsetX
			const y = (node.y ?? 0) + offsetY

			return `<g
					fixed="false"
					style="cursor: pointer;"
				>
					<circle
						class="content--circle"
						stroke-width="2"
						fill="${NODE_COLORS[status]}"
						stroke="black"
						r="19"
						data-elementId="${node.value}"
						cx="${x}"
						cy="${y}"
					></circle>
					<text
						class="content--text"
						font-size="14"
						dy=".35em"
						text-anchor="middle"
						stroke-width="1"
						fill="black"
						stroke="black"
						data-elementId="${node.value}"
						x="${x}"
						y="${y}"
						style="user-select: none"
					>
						${node.value}
					</text>
				</g>`
		})
	}

	#getLinesForRender(graph: Graph, offsetX: number, offsetY: number) {
```

```
            return (
                  [...graph.graph.entries()]
                        // eslint-disable-next-line
                        .map(([_, node]) => {
                              return [...node.edges].map(edge => {
                                    const adjacentNodeX = edge.adjacentNode.x ??
0
                                    const adjacentNodeY = edge.adjacentNode.y ??
0
                                    const nodeX = node.x ?? 0
                                    const nodeY = node.y ?? 0

                                    const  vectorOne  =  [adjacentNodeX  -  nodeX,
adjacentNodeY - nodeY]
                                    const        vectorOneProtectionToX        =
[Math.abs(adjacentNodeX - nodeX), 0]

                                    const top =
                                          vectorOne[0]                          *
vectorOneProtectionToX[0] +
                                          vectorOne[1]                          *
vectorOneProtectionToX[1]
                                    const bottom =
                                          Math.sqrt(vectorOne[0]    **    2    +
vectorOne[1] ** 2) *
                                          Math.sqrt(
                                                vectorOneProtectionToX[0] ** 2 +
vectorOneProtectionToX[1] ** 2
                                          )

                                    const  arrowRotateDeg  =  (Math.acos(top  /
bottom) * 180) / Math.PI
                                    const arrowRotateDegWithReflection =
                                          vectorOne[1] < 0 ? 360 - arrowRotateDeg
: arrowRotateDeg

                                    const distanceFromCenter = [
                                          19                                    *
Math.cos((arrowRotateDegWithReflection * Math.PI) / 180),
                                          19                                    *
Math.sin((arrowRotateDegWithReflection * Math.PI) / 180)
                                    ]

                                    if   (edge.status   ===   'no-direction'   &&
graph.mode !== 'undirected')
                                          return

                                    const color =
                                          edge.status   ===   'no-direction'   &&
window.DEBUG
                                                ? 'green'
                                                : edge.type === 'default'
                                                  ? 'black'
                                                  : edge.type === 'back'
                                                    ? 'lightblue'
                                                    : edge.type === 'cross'
                                                      ? 'lightgreen'
                                                      : 'lightpink'

                                    const   arrow   =   `<path   stroke="${color}"
fill="${color}" d="M -15 5.5 L 0 0 L -15 -5.5 Z" transform="translate (${
                                          adjacentNodeX     +     offsetX     -
distanceFromCenter[0]
                                    } ${
```

```
                                                adjacentNodeY     +     offsetY     -
distanceFromCenter[1]
                                          })
rotate(${arrowRotateDegWithReflection})"></path>`

                                          const textPosition = {
                                                x:  (nodeX  +  adjacentNodeX)  /  2  +
distanceFromCenter[0] + offsetX,
                                                y:  (nodeY  +  adjacentNodeY)  /  2  +
distanceFromCenter[1] + offsetY
                                          }

                                          const text = `
                                                <text            x="${textPosition.x}"
y="${textPosition.y}"   data-edge-text-id-back="${edge}"   style="stroke:white;
stroke-width:0.6em">${edge.weight}</text>
                                                <text            x="${textPosition.x}"
y="${textPosition.y}"            data-edge-text-id="${edge.adjacentNode.value}-
${edge.weight}-${edge.status}" style="fill:black">${edge.weight}</text> `

                                          return `<g>
                                                <path
                                                class="content--edge"
                                                      d="M ${nodeX + offsetX} ${nodeY
+ offsetY} L ${adjacentNodeX + offsetX} ${
                                                            adjacentNodeY + offsetY
                                                      }"
                                                      fill="none"
                                                      stroke-width="2"
                                                      stroke="${color}"
                                                ></path>
                                                ${graph.mode    ===    'directed'    ||
window.DEBUG ? arrow : ''}
                                                ${graph.weights ? text : ''}
                                          </g>`
                                    })
                              })
                              .flat()
                  )
            }

            render(
                  graph: Graph,
                  offsetX: number,
                  offsetY: number,
                  pressedKeyCode: string | null,
                  currentClickedTarget: HTMLElement | null
            ) {
                  const ourNodes = this.#getRenderedCircles(
                        graph,
                        offsetX,
                        offsetY,
                        currentClickedTarget
                  )
                  const ourEdges = this.#getLinesForRender(graph, offsetX, offsetY)

                  const content = document.querySelector<HTMLDivElement>('#content')

                  if (!content) {
                        throw new Error('No content')
                  }

                  content.innerHTML = `
                        <div class="graph__wrapper">
```

```
                        <svg
                        width="100%"
                        height="100%"
                        preserveAspectRatio="none"
                        cursor="${pressedKeyCode === 'Space' ? 'grabbing' :
'default'}"
                        >
                              <g>
                                    <g>
                                          ${ourEdges.join(' ')}
                                    </g>
                                    <g>
                                          ${ourNodes.join(' ')}
                                    </g>
                              </g>
                        </svg>
                  </div>
            `
      }
}


src\presets\default.ts
import { Preset } from '../types/PresetItem.ts'

export const DEFAULT_PRESET: Preset = {
      nodes: [
            { value: '1', x: 751, y: 189 },
            { value: '2', x: 516, y: 335 },
            { value: '3', x: 691, y: 372 },
            { value: '4', x: 1020, y: 336 },
            { value: '5', x: 390, y: 508 },
            { value: '6', x: 564, y: 511 },
            { value: '7', x: 681, y: 502 },
            { value: '8', x: 855, y: 494 },
            { value: '9', x: 961, y: 492 },
            { value: '10', x: 1136, y: 484 },
            { value: '11', x: 622, y: 657 },
            { value: '12', x: 1002, y: 646 },
            { value: '13', x: 813, y: 649 }
      ],
      edges: [
            { from: '1', to: '9', weight: 1 },
            { from: '1', to: '2', weight: 1 },
            { from: '1', to: '3', weight: 1 },
            { from: '1', to: '4', weight: 1 },
            { from: '2', to: '5', weight: 1 },
            { from: '2', to: '6', weight: 1 },
            { from: '3', to: '7', weight: 1 },
            { from: '3', to: '8', weight: 1 },
            { from: '4', to: '9', weight: 1 },
            { from: '4', to: '10', weight: 1 },
            { from: '5', to: '', weight: 1 },
            { from: '6', to: '11', weight: 1 },
            { from: '7', to: '11', weight: 1 },
            { from: '8', to: '1', weight: 1 },
            { from: '9', to: '12', weight: 1 },
            { from: '10', to: '', weight: 1 },
            { from: '11', to: '13', weight: 1 },
            { from: '12', to: '', weight: 1 },
            { from: '13', to: '12', weight: 1 }
      ]
}
```

src\presets\index.ts
```
export { WEIGHTS_PRESET } from './weights.ts'
export { MINUS_WEIGHTS_PRESET } from './minusWeights.ts'
export { DEFAULT_PRESET } from './default.ts'
```

src\presets\minusWeights.ts
```
import { Preset } from '../types/PresetItem'

export const MINUS_WEIGHTS_PRESET: Preset = {
      nodes: [
            { value: '1', x: 500, y: 500 },
            { value: '1', x: 500, y: 500 },
            { value: '1', x: 500, y: 500 },
            { value: '1', x: 500, y: 500 },
            { value: '2', x: 494, y: 216 },
            { value: '3', x: 984, y: 336 },
            { value: '3', x: 984, y: 336 },
            { value: '3', x: 984, y: 336 },
            { value: '4', x: 668, y: 346 },
            { value: '5', x: 123, y: 333 },
            { value: '5', x: 123, y: 333 },
            { value: '6', x: 396, y: 338 },
            { value: '6', x: 396, y: 338 },
            { value: '7', x: 633, y: 6 },
            { value: '8', x: 888, y: 70 },
            { value: '9', x: 800, y: 214 }
      ],
      edges: [
            { from: '1', to: '6', weight: 22 },
            { from: '1', to: '5', weight: -31 },
            { from: '1', to: '4', weight: -31 },
            { from: '1', to: '3', weight: 52 },
            { from: '2', to: '9', weight: 37 },
            { from: '3', to: '2', weight: -52 },
            { from: '3', to: '9', weight: 89 },
            { from: '3', to: '4', weight: -52 },
            { from: '4', to: '2', weight: 13 },
            { from: '5', to: '6', weight: -65 },
            { from: '5', to: '2', weight: 73 },
            { from: '6', to: '4', weight: 68 },
            { from: '6', to: '2', weight: -40 }
      ]
}
```

src\presets\weights.ts
```
import { Preset } from '../types/PresetItem'

export const WEIGHTS_PRESET: Preset = {
      nodes: [
            { value: '1', x: 500, y: 500 },
            { value: '1', x: 500, y: 500 },
            { value: '1', x: 500, y: 500 },
            { value: '1', x: 500, y: 500 },
            { value: '2', x: 494, y: 216 },
            { value: '3', x: 984, y: 336 },
            { value: '3', x: 984, y: 336 },
            { value: '3', x: 984, y: 336 },
            { value: '4', x: 668, y: 346 },
            { value: '5', x: 123, y: 333 },
            { value: '5', x: 123, y: 333 },
            { value: '6', x: 396, y: 338 },
```

```
            { value: '6', x: 396, y: 338 },
            { value: '7', x: 633, y: 6 },
            { value: '8', x: 888, y: 70 },
            { value: '9', x: 800, y: 214 }
      ],
      edges: [
            { from: '1', to: '6', weight: 22 },
            { from: '1', to: '5', weight: 31 },
            { from: '1', to: '4', weight: 31 },
            { from: '1', to: '3', weight: 52 },
            { from: '2', to: '9', weight: 37 },
            { from: '3', to: '2', weight: 52 },
            { from: '3', to: '9', weight: 89 },
            { from: '3', to: '4', weight: 52 },
            { from: '4', to: '2', weight: 13 },
            { from: '5', to: '6', weight: 65 },
            { from: '5', to: '2', weight: 73 },
            { from: '6', to: '4', weight: 68 },
            { from: '6', to: '2', weight: 40 },
            { from: '7', to: '', weight: 18 },
            { from: '8', to: '', weight: 81 },
            { from: '9', to: '', weight: 60 }
      ]
}


src\types\Offset.ts
export interface Offset {
      offsetX: number
      offsetY: number
}


src\types\PresetItem.ts
export interface PresetNode {
      value: string
      x: number
      y: number
}

export interface PresetEdge {
      from: string
      to: string
      weight: number
}

export interface Preset {
      nodes: PresetNode[]
      edges: PresetEdge[]
}


src\types\StoragePresets.ts
import { PresetEdge, PresetNode } from './PresetItem.ts'

export type StoragePresets = Record<
      string,
      {
            nodes: PresetNode[]
            edges: PresetEdge[]
      }
>
```

src\utils\OffsetStorage.ts

```
import { Offset } from '../types/Offset.ts'

export class OffsetStorage {
	key = 'offsets'

	writeAll(data: Offset) {
		localStorage.setItem(this.key, JSON.stringify(data))
	}

	read() {
		const value = localStorage.getItem(this.key)

		if (!value) {
			return null
		}

		return JSON.parse(value)
	}
}
```

src\utils\PresetStorage.ts

```
import { StoragePresets } from '../types/StoragePresets.ts'

export class PresetStorage {
	key = 'presets'

	writeAll(data: StoragePresets) {
		localStorage.setItem(this.key, JSON.stringify(data))
	}

	read() {
		const value = localStorage.getItem(this.key)

		if (!value) {
			return null
		}

		return JSON.parse(value)
	}
}
```

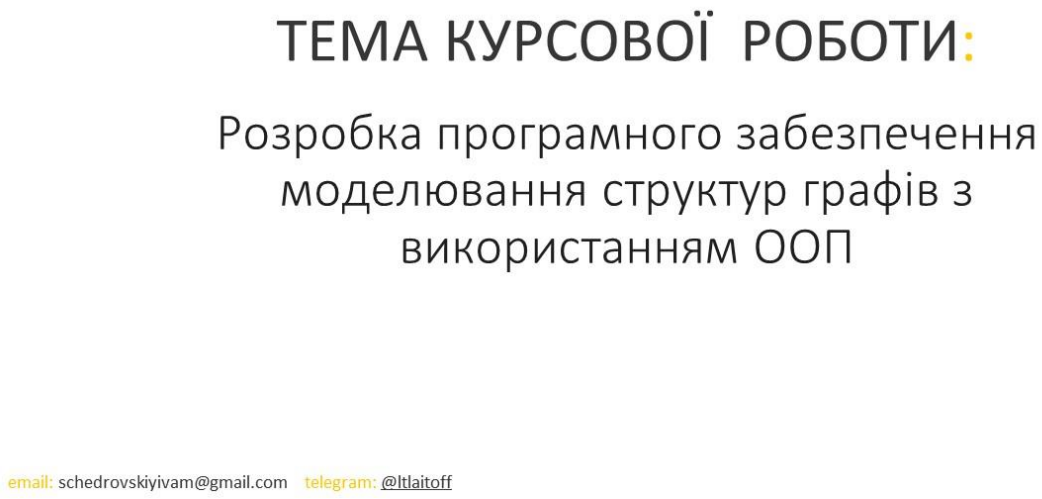
src\utils\WindowManager.ts

```
/* eslint-disable */
/* Remove eslint disable on usage :) */

class WindowManager {
	offsetUpdateCallback: (() => void) | null = null
	presetsUpdateCallback: (() => void) | null = null

	constructor() {
		addEventListener('storage', event => {
			if (event.key === 'offsets') {
				if (this.offsetUpdateCallback) {
					this.offsetUpdateCallback()
				}
			}

			if (event.key === 'presets') {
				if (this.presetsUpdateCallback) {
					this.presetsUpdateCallback()
				}
```

```
                  }
            })

            window.addEventListener('beforeunload', e => {
                  alert('beforeunload' + JSON.stringify(e))
            })
      }

      setUpdateOffsetCallback(callback: () => void) {
            this.offsetUpdateCallback = callback
      }

      setPresetsUpdateCallback(callback: () => void) {
            this.presetsUpdateCallback = callback
      }
}

export default WindowManager
```

ДОДАТОК Б

Слайди презентації:



Слайд 1

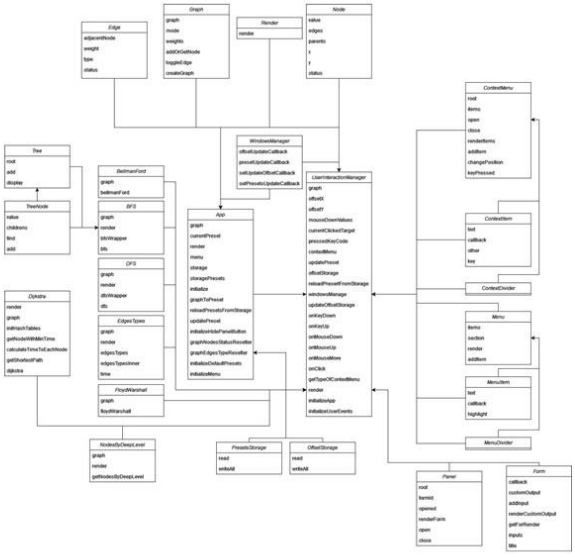
Слайд 2

ФУНКЦІЇ ПРОГРАМИ:

- Візуальне відображення графа
- Візуальне редагуваня нод графа
- Редагування граней графа
- Проста для додавання меню
- Контекстного меню для додавання та видалення нод
- Міжвіконна взаємодія
- Запуск алгоритмів

Слайд 3

СПРОЄКТОВАНА ДІАГРАМА КЛАСІВ:



Слайд 4

ВИКОРИСТАНІ ТЕХНОЛОГІЇ:

- JavaScript, TypeScript
- NodeJS, pnpm, asdf, vite
- Eslint, prettier

Слайд 5

СЕРЕДОВИЩЕ ФУНКЦІОНУВАННЯ:

Браузер > 2020 року

Слайд 6

ДЕМОНСТРАЦІЯ ПРОГРАМИ

Слайд 7

ВИСНОВКИ:

У ході виконання курсової роботи з теми "Розробка програмного забезпечення моделювання структур графів з використанням ООП" було проведено комплексне дослідження та розробка програмного забезпечення для візуалізації та аналізу графових структур.

Розроблений програмний продукт вражає своєю функціональністю, створеною з метою максимального задоволення потреб як розробників, так і користувачів. Широкий функціонал дозволяє ефективно моделювати та візуалізувати графові структури в різних сценаріях використання даними.

У цілому, дана курсова робота відображає високий рівень знань, вмінь та навичок у галузі розробки програмного забезпечення. Застосовані підходи, інструменти та функціонал відповідають сучасним вимогам та відображають інноваційний підхід до вирішення завдань.

Слайд 8

ДОПОВІДЬ ЗАВЕРШЕНО

Слайд 9