**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. Аналіз предметної області; 2. Аналіз програмних засобів; 3. Основні рішення з реалізації компонентів системи; 4. Керівництво програміста; 5. Керівництво користувача; 6. Додатки.   
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**

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

Об’єкт дослідження - програмне забезпечення для моделювання структур графів з використанням об'єктно-орієнтованого підходу.

Предмет дослідження - функціональність та особливості системи моделювання графових структур.

Метою роботи є розробити програмний продукт для візуалізації графів та можливості їх редагування використовуючи принципи ООП, який задовольняє вимоги до функціональності, продуктивності та зручності використання, який буде ефективним як для розробника, так і для кінцевого користувача.

Матеріали, методи та технічні засоби:

* Мова програмування: TypeScript
* Інструменти для веб-розробки: Vite, typescript lsp
* Інструмент для розробки: VSCode та neovim

Аналіз функцій системи. Розроблений продукт відзначається багатофункціональністю, орієнтованою на потреби розробників і користувачів. Широкий спектр функцій дозволяє ефективно моделювати графові структури в різних сценаріях.

Розробка структурної моделі системи. Структурна модель системи є гнучкою та розширюваною завдяки об'єктно-орієнтованому підходу. Архітектура дозволяє легко впроваджувати нові функції та модулі.

Розробка UML діаграм класів та опис класів. Використання UML діаграм класів стало ефективним засобом архітектурного аналізу та деталізації програмного комплексу.

Візуальне відображення графу. Механізм візуального представлення графів є інтуїтивно зрозумілим та динамічним, що сприяє комфортній та продуктивній роботі з системою.

Алгоритми обходу та аналізу графів. Реалізовані алгоритми обходу та аналізу графів виявилися ефективними та відповідають вимогам практичного використання.

Зміна ваги ребра та її вплив на систему. Механізм зміни ваги ребра та вивчення її впливу на систему є інтерактивним та практичним.

Висновки. Дослідження та розробка системи вказують на високий рівень професіоналізму та компетентності. Проект відображає інноваційний підхід та високі стандарти розробки програмного забезпечення.

У цілому, курсова робота свідчить про високий рівень знань, навичок та успішне впровадження сучасних підходів у розробці програмного забезпечення.

**ЗМІСТ**

[Вступ 4](#_Toc157530597)

[1 Огляд та аналіз сучасних методів та засобів проектування програмного забезпечення 6](#_Toc157530598)

[1.1 Сучасний стан питання 6](#_Toc157530599)

[1.2 Обґрунтування мети досліджень 7](#_Toc157530600)

[1.3 Вибір методів і підходів 8](#_Toc157530601)

[1.3.1 Аналітичний метод 8](#_Toc157530602)

[1.3.2 Експериментальний метод 9](#_Toc157530603)

[1.3.3 Об'єктно-орієнтований підхід 9](#_Toc157530604)

[1.4 Очікувані результати 10](#_Toc157530605)

[1.4.1 Розробка теоретичних основ моделювання структур графів з використанням об'єктно-орієнтованого програмування 11](#_Toc157530606)

[1.4.2 Розробка алгоритмів моделювання структур графів з використанням об'єктно-орієнтованого програмування 11](#_Toc157530607)

[1.4.3 Розробка програмного забезпечення для моделювання структур графів з використанням об'єктно-орієнтованого програмування 12](#_Toc157530608)

[1.4.4 Проведення експериментального дослідження розробленого програмного забезпечення 12](#_Toc157530609)

[1.5 Об'єктно-орієнтований підхід до моделювання структур графів 13](#_Toc157530610)

[2 Проектування програмного забезпечення системи, що розробляється 15](#_Toc157530611)

[2.1 Аналіз функцій системи 15](#_Toc157530612)

[2.1.1 Відображення графів 15](#_Toc157530613)

[2.1.2 Меню та елементи інтерфейсу 15](#_Toc157530614)

[2.1.3 Контекстне меню 16](#_Toc157530615)

[2.1.4 Додавання та видалення нод 16](#_Toc157530616)

[2.1.5 Зміна ваги ребра 16](#_Toc157530617)

[2.1.6 Візуальне підсвічування алгоритмів 16](#_Toc157530618)

[2.1.7 Керування відображенням 16](#_Toc157530619)

[2.1.8 Підтримка великих графів 17](#_Toc157530620)

[2.2 Розробка структурної моделі системи 17](#_Toc157530621)

[2.2.1 Компоненти системи 17](#_Toc157530622)

[2.2.2 Взаємозв'язки між компонентами 18](#_Toc157530623)

[2.2.3 Об'єктно-орієнтований підхід 18](#_Toc157530624)

[2.2.4 Динамічна взаємодія 18](#_Toc157530625)

[2.3 Розробка UML діаграм використання 19](#_Toc157530626)

[2.4 Розробка графічного інтерфейсу 20](#_Toc157530627)

[2.4.1 Основні елементи інтерфейсу 20](#_Toc157530628)

[2.4.2 Меню та кнопки 20](#_Toc157530629)

[2.4.3 Контекстне меню 21](#_Toc157530630)

[2.4.4 Відображення графів 21](#_Toc157530631)

[2.4.5 Динамічна зміна вигляду 21](#_Toc157530632)

[2.4.6 Реалізація графічного інтерфейсу 22](#_Toc157530633)

[2.4.7 Забезпечення взаємодії та зручності використання 22](#_Toc157530634)

[2.4.8 Підтримка різних операційних систем 22](#_Toc157530635)

[2.5 Розробка тестів для перевірки правильності програми 22](#_Toc157530636)

[2.5.1 Модульне тестування 23](#_Toc157530637)

[2.5.2 Інтеграційне тестування 23](#_Toc157530638)

[2.5.3 Тестування користувацького інтерфейсу 23](#_Toc157530639)

[2.5.4 Тестування в різних браузерах 23](#_Toc157530640)

[2.5.5 Відсутність помилок та резервне копіювання 24](#_Toc157530641)

[2.5.6 Визначення практичних сценаріїв тестування 24](#_Toc157530642)

[2.5.7 Валідація даних та безпека 24](#_Toc157530643)

[2.5.8 Автоматизоване тестування 24](#_Toc157530644)

[2.5.9 Висновки з тестування програмного забезпечення 24](#_Toc157530645)

[3 Розробка програмного забезпечення системи, що розробляється 26](#_Toc157530646)

[3.1 Розробка UML діаграм класів 26](#_Toc157530647)

[3.2 Опис класів програмного комплексу 26](#_Toc157530648)

[4 Аналіз ефективності програмного забезпечення 37](#_Toc157530649)

[4.1 Візуальне відображення графу 37](#_Toc157530650)

[4.1.1 Швидкість відображення 37](#_Toc157530651)

[4.1.2 Масштабованість 37](#_Toc157530652)

[4.1.3 Переключання напрямків та вісь ребер 38](#_Toc157530653)

[4.1.4 Гнучкість відображення 38](#_Toc157530654)

[4.2 Функціонал меню та контекстного меню 38](#_Toc157530655)

[4.2.1 Менюшка та роздільник менюшки 38](#_Toc157530656)

[4.2.2 Контекстне меню 39](#_Toc157530657)

[4.2.3 Інтуїтивність та ефективність використання 39](#_Toc157530658)

[4.2.4 Динамічність та анімації 39](#_Toc157530659)

[4.3 Реалізація базових алгоритмів 40](#_Toc157530660)

[4.3.1 Пошук в глибину (DFS) та Пошук в ширину (BFS) 40](#_Toc157530661)

[4.3.2 Алгоритм Дейкстри 40](#_Toc157530662)

[4.3.3 Алгоритм Беллмана-Форда 40](#_Toc157530663)

[4.3.4 Алгоритм Флойда-Воршелла 41](#_Toc157530664)

[4.3.5 Алгоритм взяття всіх нод в деякому радіусі 41](#_Toc157530665)

[4.3.6 Визначення типів ребер (Tree, Back, Edge та Cross Edges) у DFS графу 41](#_Toc157530666)

[4.4 Зміна ваги ребра та її вплив на систему 42](#_Toc157530667)

[5 Розробка документів на супроводження програмного забезпечення 44](#_Toc157530668)

[5.1 Інструкція програмісту 44](#_Toc157530669)

[5.2 Інструкція користувачеві 45](#_Toc157530670)

[Висновки 46](#_Toc157530671)

[Перелік джерел посилання 48](#_Toc157530672)

[ДОДАТОК А 49](#_Toc157530673)

[Додаток Б 106](#_Toc157530674)

# Вступ

Моделювання структур графів є однією з важливих галузей сучасної інформатики. Графи широко використовуються в різних сферах діяльності, таких як штучний інтелект, мережеві технології, логістика, біологія, економіка тощо.

Сучасний рівень розвитку науки та техніки у галузі моделювання структур графів характеризується наступними тенденціями:

* Розвиток теоретичних основ моделювання графів. Досліджуються нові типи графів, алгоритми їх побудови та обробки.
* Розробка нових методів і алгоритмів моделювання графів. Впроваджуються нові підходи до оптимізації алгоритмів, підвищення їхньої ефективності та надійності.
* Створення нових програмних засобів для моделювання графів. Розробляються зручні та ефективні інструменти, які дозволяють генерувати, редагувати, аналізувати та візуалізувати графи.

Метою курсової роботи є розробка програмного забезпечення моделювання структур графів з використанням об'єктно-орієнтованого програмування.

Для досягнення цієї мети необхідно вирішити наступні завдання:

* ознайомитися з теоретичними основами моделювання структур графів;
* вивчити основні методи і алгоритми моделювання графів;
* розробити програмне забезпечення для моделювання структур графів;
* провести експериментальне дослідження розробленого програмного забезпечення.

Актуальність курсової роботи обумовлена ​​тим, що розробка програмного забезпечення моделювання структур графів є актуальним завданням, оскільки дає можливість:

* створювати зручні та ефективні інструменти для моделювання графів;
* підвищити продуктивність і надійність алгоритмів моделювання графів;
* розширити можливості застосування графів у різних сферах діяльності.

Загальна постановка завдання полягає в розробці програмного забезпечення для моделювання структур графів, яке буде зручним як для користувача, так і для розробника.

Програмне забезпечення має забезпечувати наступні можливості:

* генерування графів різних типів;
* редагування графів;
* виконання операцій над графами;
* візуалізація графів.

Програмне забезпечення має бути реалізовано з використанням об'єктно-орієнтованого програмування [1].

У наступних розділах курсової роботи будуть розглянуті теоретичні основи моделювання структур графів, основні методи і алгоритми моделювання графів, буде розроблено програмне забезпечення для моделювання структур графів, проведено експериментальне дослідження розробленого програмного забезпечення.

Начало формы

# Огляд та аналіз сучасних методів та засобів проектування програмного забезпечення

## Сучасний стан питання

Моделювання структур графів є однією з важливих галузей сучасної інформатики. Графи широко використовуються в різних сферах діяльності, таких як штучний інтелект, мережеві технології, логістика, біологія, економіка тощо.

Сучасний рівень розвитку науки та техніки у галузі моделювання структур графів характеризується наступними тенденціями:

* розвиток теоретичних основ моделювання графів. Досліджуються нові типи графів, алгоритми їх побудови та обробки. Зокрема, активно розвивається теорія графів з вагами, графів з додатковими атрибутами, графів з топологічними обмеженнями тощо;
* розробка нових методів і алгоритмів моделювання графів. Впроваджуються нові підходи до оптимізації алгоритмів, підвищення їхньої ефективності та надійності. Зокрема, розробляються нові методи пошуку в графах, методи розв'язання задач оптимізації на графах, методи аналізу графів тощо;
* створення нових програмних засобів для моделювання графів. Розробляються зручні та ефективні інструменти, які дозволяють генерувати, редагувати, аналізувати та візуалізувати графи. Зокрема, розробляються спеціалізовані програмні пакети для моделювання графів у різних сферах діяльності.

Серед існуючих програмних засобів для моделювання графів можна виділити наступні:

* NetBeans Graph Editor - це безкоштовний та відкритий редактор графів, який дозволяє генерувати, редагувати, аналізувати та візуалізувати графи;
* Graphviz - це безкоштовний та відкритий інструмент для створення візуалізацій графів;
* GraphSAGE - це фреймворк для навчання графових моделей, який дозволяє генерувати, редагувати, аналізувати та візуалізувати графи.

Ці програми мають широкий спектр можливостей, але вони також мають ряд обмежень. Зокрема, вони не завжди є зручними для використання, можуть бути ненадійні або мати низьку продуктивність.

## Обґрунтування мети досліджень

Розробка програмного забезпечення для моделювання структур графів є актуальним завданням, оскільки дає можливість:

* зручно та ефективно використовувати застосунок для моделювання графів. Це дозволить користувачам швидко і легко створювати, редагувати та аналізувати графи;
* підвищити продуктивність і надійність алгоритмів моделювання графів. Це буде досягнуто за рахунок використання ефективних алгоритмів та структур даних;
* розширити можливості застосування графів у різних сферах діяльності. Зручне та ефективне програмне забезпечення для моделювання графів дозволить використовувати їх у нових сферах діяльності;

Метою курсової роботи є розробка програмного забезпечення для моделювання структур графів з використанням об'єктно-орієнтованого програмування.

Реалізація цієї мети передбачає вирішення наступних завдань:

* ознайомитися з теоретичними основами моделювання структур графів. Це необхідно для розуміння принципів роботи алгоритмів моделювання графів;
* вивчити основні методи і алгоритми моделювання структур графів. Це необхідно для розробки ефективних алгоритмів моделювання графів;
* розробити програмне забезпечення для моделювання структур графів. Це завдання передбачає реалізацію розроблених алгоритмів у вигляді програмного забезпечення;
* провести експериментальне дослідження розробленого програмного забезпечення. Це необхідно для оцінки ефективності розробленого програмного забезпечення.

## Вибір методів і підходів

Для досягнення поставленої мети курсової роботи будуть використані наступні методи і підходи:

* аналітичний метод буде використаний для вивчення теоретичних основ моделювання структур графів та розробки алгоритмів моделювання графів. Аналітичний метод передбачає вивчення наукової літератури, а також проведення власних досліджень;
* експериментальний метод буде використаний для перевірки ефективності розроблених алгоритмів та програмного забезпечення. Експериментальний метод передбачає проведення комп'ютерних експериментів;
* об'єктно-орієнтований підхід буде використаний для реалізації програмного забезпечення. Об'єктно-орієнтований підхід дозволяє розробляти програмне забезпечення, яке є гнучким, масштабованим і надійним.

### Аналітичний метод

Аналітичний метод буде використаний для вивчення теоретичних основ моделювання структур графів. Теоретичні основи моделювання структур графів включають в себе наступні розділи:

* теорія графів - це розділ математики, який вивчає властивості графів. Теорія графів надає теоретичні основи для розробки алгоритмів моделювання структур графів;
* алгоритми моделювання структур графів - це розділ інформатики, який вивчає алгоритми для створення, редагування, аналізу та візуалізації графів.

Для вивчення теоретичних основ моделювання структур графів буде використана наукова література, а також будуть проведені власні дослідження.

### Експериментальний метод

Експериментальний метод буде використаний для перевірки ефективності розроблених алгоритмів та програмного забезпечення. Для цього буде проведено комп'ютерні експерименти з різними типами графів.

Експериментальні дослідження будуть проводитися з використанням наступних метрик ефективності:

* час виконання - це час, який витрачається на виконання алгоритму;
* просторове споживання - це обсяг пам'яті, який витрачається на зберігання даних, необхідних для виконання алгоритму.

### Об'єктно-орієнтований підхід

Об'єктно-орієнтований підхід буде використаний для реалізації програмного забезпечення для моделювання структур графів.

При використанні об'єктно-орієнтованого підходу графи можуть бути представлені як набори об'єктів. Кожен об'єкт представляє одну вершину або ребро графа. Об'єкти, що представляють вершини, називаються вершинними об'єктами, а об'єкти, що представляють ребра, називаються ребровими об'єктами [2].

Вершинні об'єкти повинні мати такі атрибути:

* ідентифікатор вершини - це унікальний номер, який ідентифікує вершину;
* назва вершини - це символьний рядок, який може використовуватися для ідентифікації вершини;
* Значення вершини - це будь-яка інформація, яка може бути пов'язана з вершиною.

Реброві об'єкти повинні мати такі атрибути:

* ідентифікатор ребра - це унікальний номер, який ідентифікує ребро;
* назва ребра - це символьний рядок, який може використовуватися для ідентифікації ребра;
* відстань між вершинами - це відстань між вершинами, з'єднаними ребром.

Крім атрибутів, вершинні і реброві об'єкти можуть мати методи. Методи можуть використовуватися для виконання операцій над вершинами і ребрами.

## Очікувані результати

Очікувані результати курсової роботи включають:

* розробку теоретичних основ моделювання структур графів з використанням об'єктно-орієнтованого програмування. Цей результат буде досягнуто шляхом вивчення теоретичних основ моделювання структур графів та розробки об'єктно-орієнтованих моделей графів;
* розробку алгоритмів моделювання структур графів з використанням об'єктно-орієнтованого програмування. Цей результат буде досягнуто шляхом розробки об'єктно-орієнтованих алгоритмів для створення, редагування, аналізу та візуалізації графів;
* розробку програмного забезпечення для моделювання структур графів з використанням об'єктно-орієнтованого програмування. Цей результат буде досягнуто шляхом реалізації розроблених алгоритмів у вигляді програмного забезпечення;
* проведення експериментального дослідження розробленого програмного забезпечення. Цей результат буде досягнуто шляхом проведення комп'ютерних експериментів з метою оцінки ефективності розробленого програмного забезпечення.

### Розробка теоретичних основ моделювання структур графів з використанням об'єктно-орієнтованого програмування

Розробка теоретичних основ моделювання структур графів з використанням об'єктно-орієнтованого програмування передбачає вивчення наступних питань:

* Абстрактні типи даних для представлення графів. Які абстрактні типи даних можуть бути використані для представлення вершин і ребер графів?
* Методи оперування вершинами і ребрами графів. Які методи можуть бути використані для виконання операцій над вершинами і ребрами графів?
* Особливості об'єктно-орієнтованого підходу до моделювання структур графів. Які переваги і недоліки має об'єктно-орієнтований підхід до моделювання структур графів?

### Розробка алгоритмів моделювання структур графів з використанням об'єктно-орієнтованого програмування

Розробка алгоритмів моделювання структур графів з використанням об'єктно-орієнтованого програмування передбачає розробку наступних алгоритмів:

* Алгоритм створення графів. Який алгоритм можна використовувати для створення графів?
* Алгоритм редагування графів. Які алгоритми можна використовувати для редагування графів?
* Алгоритм аналізу графів. Які алгоритми можна використовувати для аналізу графів?
* Алгоритм візуалізації графів. Які алгоритми можна використовувати для візуалізації графів?

### Розробка програмного забезпечення для моделювання структур графів з використанням об'єктно-орієнтованого програмування

Розробка програмного забезпечення для моделювання структур графів з використанням об'єктно-орієнтованого програмування передбачає реалізацію наступних функцій:

* Створення графів. Програма повинна дозволяти користувачеві створювати графи різних типів.
* Редагування графів. Програма повинна дозволяти користувачеві редагувати графи.
* Аналіз графів. Програма повинна надавати користувачеві інструменти для аналізу графів.
* Візуалізація графів. Програма повинна надавати користувачеві інструменти для візуалізації графів.

### Проведення експериментального дослідження розробленого програмного забезпечення

Проведення експериментального дослідження розробленого програмного забезпечення передбачає проведення комп'ютерних експериментів з метою оцінки ефективності розробленого програмного забезпечення.

У експериментах будуть використовуватися наступні метрики ефективності:

* Час виконання - це час, який витрачається на виконання алгоритму.
* Просторове споживання - це обсяг пам'яті, який витрачається на зберігання даних, необхідних для виконання алгоритму.

## Об'єктно-орієнтований підхід до моделювання структур графів

Об'єктно-орієнтований підхід є ефективним підходом до моделювання структур графів, оскільки він дозволяє:

* Чітко розмежовувати структуру і поведінку об'єктів. Це дозволяє забезпечити гнучкість і масштабованість моделі.
* Використовувати спадкування для повторного використання коду. Це дозволяє скоротити обсяг коду і полегшити його підтримку.
* Реалізувати поліморфізм для забезпечення гнучкості моделі. Це дозволяє використовувати один і той же код для різних типів графів.

При використанні об'єктно-орієнтованого підходу до моделювання структур графів графи можуть бути представлені як набори об'єктів. Кожен об'єкт представляє одну вершину або ребро графа. Об'єкти, що представляють вершини, називаються вершинними об'єктами, а об'єкти, що представляють ребра, називаються ребровими об'єктами [3].

Вершинні об'єкти повинні мати такі атрибути:

* Ідентифікатор вершини. Цей атрибут є унікальним для кожної вершини.
* Назва вершини. Цей атрибут може використовуватися для ідентифікації вершини.
* Значення вершини. Цей атрибут може використовуватися для зберігання інформації про вершину.

Реброві об'єкти повинні мати такі атрибути:

* Ідентифікатор ребра. Цей атрибут є унікальним для кожного ребра.
* Назва ребра. Цей атрибут може використовуватися для ідентифікації ребра.
* Відстань між вершинами. Цей атрибут може використовуватися для зберігання інформації про відстань між вершинами, з'єднаними ребром.

Крім атрибутів, вершинні і реброві об'єкти можуть мати методи. Методи можуть використовуватися для виконання операцій над вершинами і ребрами.

# Проектування програмного забезпечення системи, що розробляється

В даному розділі розглядається процес проектування програмного забезпечення системи моделювання структур графів з використанням об'єктно-орієнтованого програмування (ООП). Система призначена для зручного візуального моделювання графів, що може бути використана як користувачами, так і розробниками для відображення роботи різних алгоритмів.

## Аналіз функцій системи

Аналіз функцій системи є ключовим етапом у розробці програмного забезпечення для моделювання структур графів. Визначення основних функціональних вимог є необхідним для подальшого проектування та реалізації системи. Детальний розгляд функцій дозволяє забезпечити повноту та ефективність розроблюваного програмного продукту.

### Відображення графів

Однією з основних функцій системи є відображення графів в графічному інтерфейсі. Графічна репрезентація включає в себе візуальне відображення вузлів (нод) та ребер графу. Користувач повинен мати можливість легко розпізнати та взаємодіяти з елементами графу. Надається можливість зміни масштабу та переміщення по графу для поліпшення зручності спостереження за великими графами.

### Меню та елементи інтерфейсу

Система має меню, яке дозволяє користувачам здійснювати різноманітні операції. Елементи меню повинні бути легкодоступними та інтуїтивно зрозумілими. Меню містить опції для додавання нових нод, видалення нод, зміни ваги ребра та налаштування відображення графу.

### Контекстне меню

Контекстне меню визивається правим кліком мишею. Це важливий елемент інтерфейсу, який дозволяє користувачам швидко та зручно виконувати операції на окремих елементах графу. Наприклад, додавання нової ноди або видалення існуючої.

### Додавання та видалення нод

Система повинна дозволяти користувачам динамічно додавати нові ноди та видаляти існуючі. Це забезпечує гнучкість у роботі з графами та можливість швидко змінювати їхню структуру.

### Зміна ваги ребра

Важливою функцією є можливість коригувати вагу ребра. Це особливо актуально для використання алгоритмів, які враховують ваги ребер. Користувач може клікнути на ребро та ввести необхідне значення.

### Візуальне підсвічування алгоритмів

Система має базові алгоритми, які візуально підсвічують ноди та ребра, які взяли участь у виконанні певного алгоритму. Це допомагає користувачам швидко розуміти результати виконання алгоритмів та їх вплив на граф.

### Керування відображенням

Система повинна надавати можливості керування відображенням, такими як включення/виключення напрямків ребер, показ вісів ребер та інші параметри, які дозволяють користувачам налаштовувати вигляд графу за своїми уподобаннями.

Також система повинна мати можливість працювати між вікнами, відображаючи сайт однаково добре та автоматично синхронізуючись

### Підтримка великих графів

Система повинна бути оптимізована для роботи з великими графами, забезпечуючи швидку відповідь та зручність взаємодії навіть при великій кількості елементів.

## Розробка структурної моделі системи

Структурна модель системи визначає основні компоненти та їх взаємозв'язки. Цей розділ включає детальний опис структурної моделі для системи моделювання структур графів з використанням об'єктно-орієнтованого програмування (ООП) [4].

### Компоненти системи

Система складається з основних компонентів, що взаємодіють між собою:

* Графічний інтерфейс (GUI). Основний компонент, що відповідає за візуалізацію графів та взаємодію з користувачем. Включає у себе елементи інтерфейсу, такі як відображення нод та ребер, кнопки, меню та інші.
* **Контролер (Controller).** Відповідає за обробку подій, що надходять від користувача через GUI. Забезпечує виклик відповідних методів для взаємодії з різними компонентами системи.
* **Алгоритми обробки даних.** Цей компонент включає в себе реалізацію алгоритмів, які застосовуються до графів. Наприклад, алгоритми пошуку шляху, визначення компонентів зв'язності, тощо [5].
* **Сховище даних.** В якості сховища даних використовується браузер

### Взаємозв'язки між компонентами

* **GUI та Controller:** Графічний інтерфейс взаємодіє з контролером через події, такі як кліки мишею, натискання кнопок тощо. Контролер обробляє ці події та викликає відповідні методи для оновлення відображення графу та інші дії.
* **Controller та Алгоритми обробки даних:** Контролер взаємодіє з алгоритмами обробки даних при виклику різних функцій, таких як виконання алгоритмів чи зміна ваги ребра. Алгоритми повинні отримувати дані від контролера та здійснювати відповідні операції.
* **Controller та База даних:** У випадку використання бази даних, контролер взаємодіє з нею для збереження та отримання інформації про графи. Це може бути важливо для збереження стану графів між сесіями роботи з програмою.

### Об'єктно-орієнтований підхід

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.

### Динамічна взаємодія

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.

## Розробка UML діаграм 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 3.1 показана UML діаграма використання.

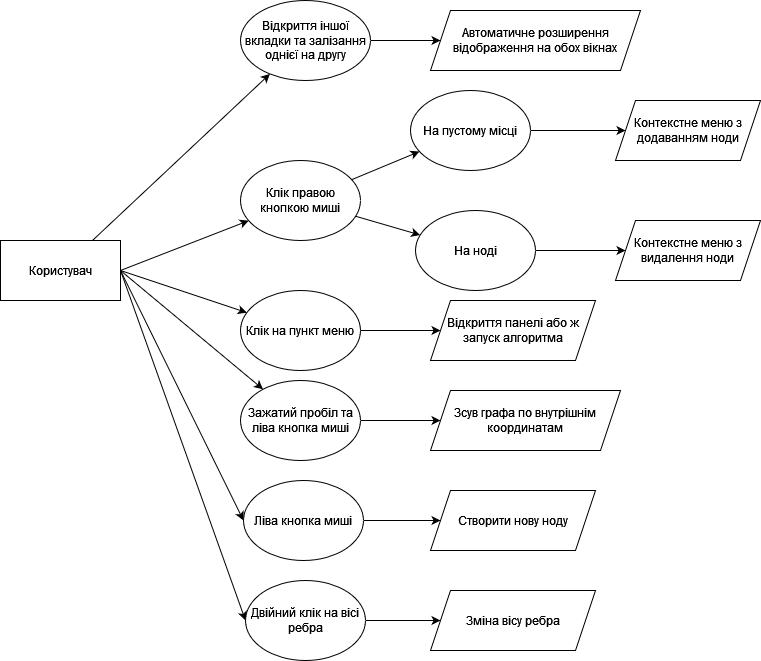


Рисунок 3.1 – UML діаграма використання

## Розробка графічного інтерфейсу

Розробка графічного інтерфейсу (GUI) є ключовим етапом у створенні програмного забезпечення для моделювання структур графів з використанням об'єктно-орієнтованого програмування (ООП). Інтерфейс повинен бути зручним, інтуїтивно зрозумілим та забезпечувати зручність взаємодії користувача з системою [6].

### Основні елементи інтерфейсу

* **Відображення графів:** Основний візуальний елемент, що представляє собою графічне відображення нод (вузлів) та ребер графу. Кожна нода повинна мати свій унікальний вигляд, а ребра можуть бути забарвлені або відмічені для кращого розрізнення.
* **Меню та кнопки:** Меню містить основні опції та кнопки, що дозволяють користувачу виконувати різні дії, такі як створення нового графу, збереження чи завантаження, виклик контекстних функцій тощо.
* **Контекстне меню:** Взаємодія з окремими елементами графу можлива через контекстне меню, яке виводиться за правим кліком мишею. Воно містить опції для швидкого виклику функцій, таких як видалення нод чи зміна їхньої ваги.

### Меню та кнопки

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.

* **Створення нового графу:** Можливість почати новий проект, очищаючи екран та готуючи його до створення нової моделі.
* **Збереження та завантаження:** Опції для збереження поточного стану графу в файл та подальшого завантаження його для продовження роботи.
* **Відображення напрямків ребер:** Включення або виключення відображення напрямку ребер, що корисно при роботі з орієнтованими графами.
* **Відображення ваги ребер:** Можливість ввімкнення або вимкнення відображення ваги ребер на графі для наглядності та аналізу.

### Контекстне меню

Контекстне меню взаємодії з окремими елементами графу, які можна виділити або обрати. Воно динамічно змінюється відповідно до контексту та містить такі опції:

* **Додати нову ноду:** Використання контекстного меню для додавання нового вузла в ту точку, де був здійснений правий клік мишею.
* **Видалити ноду чи ребро:** Видалення обраного елемента за допомогою контекстного меню, що спрощує управління графом.

### Відображення графів

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.

### Динамічна зміна вигляду

Графічний інтерфейс дозволяє користувачеві динамічно змінювати вигляд графу. Зокрема, можливість масштабування, переміщення та повороту графу для забезпечення оптимального перегляду великих та складних структур.

### Реалізація графічного інтерфейсу

Для реалізації графічного інтерфейсу було використано HTML та CSS.

### Забезпечення взаємодії та зручності використання

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.

### Підтримка різних операційних систем

Розроблений графічний інтерфейс повинен бути сумісний з різними операційними системами (Windows, Linux, MacOS). Це забезпечить універсальність програмного забезпечення та розширить коло його користувачів.

## Розробка тестів для перевірки правильності програми

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.

### Модульне тестування

Модульне тестування полягає в перевірці окремих функціональних модулів програми. Для TypeScript коду використовуються єдиниці тестування (unit tests), написані за допомогою тестових фреймворків, таких як Jest чи Mocha. Кожна функція та клас перевіряється на правильність виконання своїх завдань.

### Інтеграційне тестування

Інтеграційне тестування спрямоване на перевірку взаємодії між різними компонентами системи. У випадку браузерного застосунку, це означає тестування взаємодії між JavaScript-модулями, які відповідають за різні частини інтерфейсу, логіку обробки подій та взаємодію з сервером.

### Тестування користувацького інтерфейсу

Тестування користувацького інтерфейсу включає в себе перевірку його зручності, інтуїтивності та відповідності дизайну. Використання інструментів, таких як Selenium або Puppeteer, дозволяє автоматизувати тестування елементів інтерфейсу та їх взаємодії з користувачем.

### Тестування в різних браузерах

Оскільки браузери можуть по-різному трактувати та виконувати JavaScript, важливо забезпечити, що програмне забезпечення працює стабільно в різних браузерах. Тестування ведеться в популярних браузерах, таких як Google Chrome, Mozilla Firefox, та Microsoft Edge.

### Відсутність помилок та резервне копіювання

Тестування також включає перевірку на виявлення та виправлення помилок. Важливо переконатися, що програмне забезпечення відповідає вимогам та не має критичних помилок, які можуть вплинути на коректну роботу.

### Визначення практичних сценаріїв тестування

Розробка тестових сценаріїв для визначених вимог та функціональності. Тестування повинно охоплювати різні сценарії взаємодії користувача з програмним забезпеченням, а також враховувати можливі варіанти введення даних та взаємодії з елементами інтерфейсу.

### Валідація даних та безпека

Тестування повинно включати валідацію даних, щоб переконатися, що система вірно обробляє різні типи введених даних та попереджає про можливі помилки. Також важливо перевірити відповідність системи вимогам щодо безпеки [8].

### Автоматизоване тестування

Використання інструментів для автоматизації тестування дозволяє ефективно виконувати тести та забезпечити швидке виявлення помилок під час розробки. Це включає в себе запуск тестів під час збирання коду та інтеграції в системі контролю версій.

### Висновки з тестування програмного забезпечення

Тестування програмного забезпечення є необхідним етапом для забезпечення його якості та надійності. Використання різних видів тестів дозволяє охопити різні аспекти функціональності, продуктивності та безпеки браузерного застосунку для моделювання структур графів.

# Розробка програмного забезпечення системи, що розробляється

## Розробка UML діаграм класів

На рисунку 4.1 представлена UML діаграма класів.

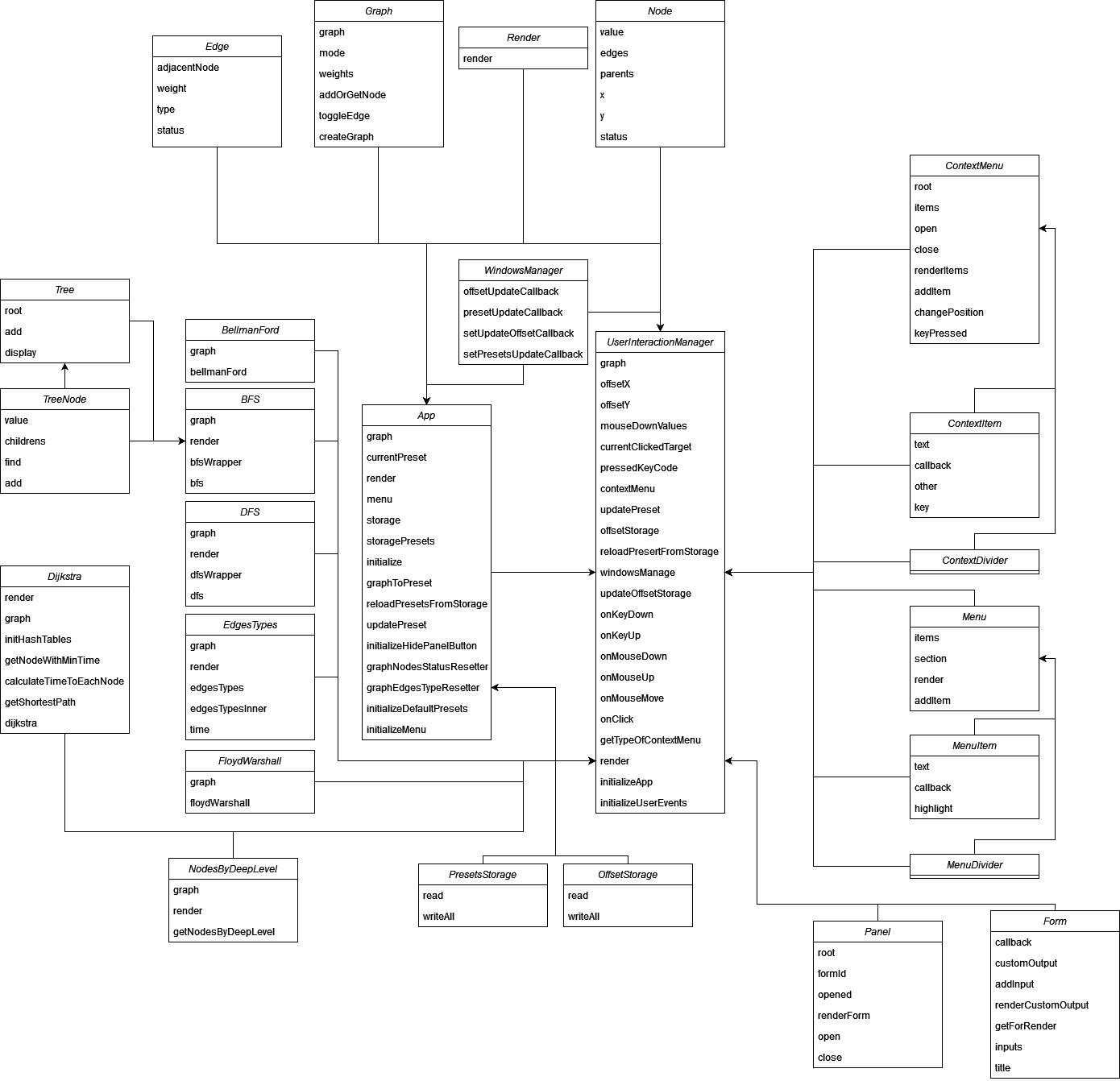


Рисунок 4.1 – UML діаграма класів

## 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.

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 користвувача спеціальне поле для виводу інформації алгоритма
* 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 алгоритма.

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 викликі контекстного меню
* render() – Метод рендера графа
* initializeApp() – Метод інітіалізації який викликається з класу App
* initializeUserEvents() – Метод інітіалізації користувацьких ефектів, таких як клік, переміщення миші та інше

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

Першими базовими алгоритмами є алгоритми DFS та BFS. DFS та BFS є стандартними алгоритмами 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 алгоритма
* 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 алгоритма
* 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 реалізованний в класі BellmanFord та має наступні властивості та методи:

* graph – Приватне поле графа. Потрібне для використання в алгоритмі.
* bellmanFord() – Метод запуску алгоритма Белламана-Форда\

Алгоритм Дейкстри 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 алгоритма
* initHashTables()
* getNodeWithMinTime()
* calculateTimeToEachNode()
* getShortestPath()
* dijkstra() – Метод запуску алгорима Дейкстри

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 алгоритма
* edgesTypes() – Метод запуску алгоритма визначення типів ребер
* edgesTypesInner() – Приватний внутрішній метод для рекурсії

Алгоритм Флойда-Воршелла реалізованний в класі FloydWarshall та має наступні властивості та методи:

* graph – Приватне поле графа. Потрібне для використання в алгоритмі.
* floydWarshall() – Метод для запуску алгоритма Флойда-Воршелла

Алгоритм взяття нод за рівнем вкладеності 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 алгоритма
* getNodesByDeepLevel() – Метод для запуску отримання всіх нод в радіусі

# Аналіз ефективності програмного забезпечення

Аналіз ефективності програмного забезпечення є важливим етапом у розгляді роботи створеного застосунку для моделювання структур графів. У цьому розділі роботи проведемо оцінку ефективності та продуктивності програми, розглянемо важливі аспекти взаємодії користувача та системи, а також виявимо можливості для подальшого вдосконалення.

## Візуальне відображення графу

Важливим аспектом аналізу є ефективність візуального відображення графу в розробленому застосунку для моделювання структур графів. Цей елемент взаємодії з користувачем визначає його зручність та сприйняття інформації.

У зв'язку з тим, що в програмі використовується графічний інтерфейс для візуалізації графів, важливо оцінити продуктивність при роботі з великою кількістю вузлів та ребер. Вплив розміру графу на швидкість відображення повинен бути мінімізований для забезпечення плавності роботи застосунку.

Розглянемо основні аспекти, що впливають на ефективність візуального відображення графу:

### Швидкість відображення

Перш за все, важливо забезпечити швидке та ефективне відображення графічної інформації на екрані. Застосунок повинен бути оптимізований таким чином, щоб при додаванні нових елементів або зміні параметрів графу затримок відображення було мінімізовано.

### Масштабованість

Система повинна бути адаптована для ефективної роботи з графами різної складності та розміру. Масштабування графічних елементів повинно бути плавним та не впливати на продуктивність, навіть при великій кількості вузлів та ребер.

### Переключання напрямків та вісь ребер

Наявність опцій для переключання напрямку ребер та відображення вісів ребер є важливою для зручності сприйняття графічної структури. Важливо, щоб ці функції були ефективними та миттєвими, щоб користувач міг динамічно налаштовувати відображення графу під свої потреби.

### Гнучкість відображення

Забезпечення гнучкості відображення графу дає можливість користувачеві налаштовувати вигляд графу згідно з власними вимогами. Переміщення та масштабування повинні бути інтуїтивно зрозумілими та беззатримковими.

## Функціонал меню та контекстного меню

Функціонал меню та контекстного меню є ключовими елементами для забезпечення зручності та ефективності взаємодії користувача з розробленим застосунком. Ці елементи інтерфейсу повинні бути інтуїтивно зрозумілими, легкими у використанні та відповідати потребам розробника та звичайного користувача.

### Менюшка та роздільник менюшки

Менюшка розташована внизу екрану та може бути легко прихована за допомогою кнопки в правому верхньому куті. Це забезпечує оптимальне використання простору та дозволяє користувачам миттєво отримати доступ до функціоналу.

Важливим елементом є наявність роздільника менюшки, який дозволяє групувати та структурувати функції, забезпечуючи логічну організацію опцій та зручний доступ до них.

### Контекстне меню

Контекстне меню активується при правому кліку на пустій області або на окремому вузлі графу. Для користувача це зручний спосіб виклику необхідних опцій в залежності від контексту.

Основні функції контекстного меню включають можливість створення нової ноди при кліку на пустій області та видалення ноди при кліку на існуючу ноду. Це надає користувачеві простий та швидкий спосіб взаємодії з елементами графу.

### Інтуїтивність та ефективність використання

Важливо, щоб меню та контекстне меню були легкими у використанні та інтуїтивно зрозумілими. Іконки та підказки повинні чітко передавати призначення кожної опції, щоб користувачі могли швидко зорієнтуватися в функціоналі.

### Динамічність та анімації

Використання динамічних анімацій при відкритті/закритті меню та контекстного меню може зробити взаємодію більш привабливою та приємною для користувача.

## Реалізація базових алгоритмів

У цьому застосунку впроваджено ряд ключових алгоритмів, що забезпечують різноманітні функціональність та можливості взаємодії з графами. Давайте розглянемо кожен з цих алгоритмів для глибшого розуміння їхнього впливу та ролі у вашому застосунку.

### Пошук в глибину (DFS) та Пошук в ширину (BFS)

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

### Алгоритм Дейкстри

Алгоритм Дейкстри використовується для знаходження найкоротших шляхів в графі з невід'ємними вагами ребер. Цей алгоритм дозволяє вам визначати оптимальні шляхи між вершинами, що є важливим для візуалізації та аналізу графів.

### Алгоритм Беллмана-Форда

Цей алгоритм використовується для знаходження найкоротших шляхів в графі, навіть якщо ребра можуть мати від'ємні ваги. Його впровадження дозволяє враховувати різні сценарії та умови у вашому застосунку.

### Алгоритм Флойда-Воршелла

Алгоритм Флойда-Воршелла використовується для знаходження всіх можливих шляхів між усіма парами вершин у графі. Це дозволяє аналізувати структуру графу та визначати можливі шляхи для будь-якої пари вершин.

### Алгоритм взяття всіх нод в деякому радіусі

Цей алгоритм дозволяє вам визначати всі вершини, які знаходяться в певному радіусі від обраної вершини. Це можливість важлива для аналізу навколишніх зв'язків та структури графу.

### Визначення типів ребер (Tree, Back, Edge та Cross Edges) у DFS графу

В рамках розробленого алгоритму DFS в нашому застосунку проведено детальний 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.

Враховуючи вищезазначені типи ребер у DFS алгоритмі, ми отримуємо додатковий інструментарій для глибшого розуміння структури графу та визначення його ключових характеристик.

## Зміна ваги ребра та її вплив на систему

У вашому застосунку реалізована можливість зміни ваги ребра, що дозволяє користувачу динамічно модифікувати ваги та переглядати вплив цих змін на візуальне представлення графу. Ця функціональність є ключовою для адаптації графу під нові умови та вивчення впливу змін на алгоритми та структуру графу.

Зміна ваги ребра: Користувач може взаємодіяти з ребрами графу, клікаючи на них та вводячи нове значення ваги. Це реалізовано для кожного ребра окремо, дозволяючи точно налаштовувати параметри графу за необхідності.

Вплив на систему:

* Перевизначення найкоротших шляхів: Зміна ваги ребра може впливати на найкоротші шляхи в графі, зокрема у випадках, коли використовуються алгоритми Дейкстри, Беллмана-Форда чи інші. Зміна ваги може призвести до оновлення шляхів та переоцінки оптимальних маршрутів.
* Реакція алгоритмів: Вага ребра може вплинути на реакцію інших алгоритмів, таких як DFS, BFS та інші, які базуються на структурі та вагах графу. Зміна ваги може впливати на порядок обходу, визначення циклів чи інших характеристик графу.
* Візуалізація змін: Система візуалізації графу має враховувати зміни ваг ребер та динамічно оновлювати відображення графу під час введення нових значень. Це допомагає користувачеві в реальному часі оцінювати вплив змін та взаємозв'язки між вершинами.
* Адаптація до нових умов: Зміна ваги ребра може служити інструментом адаптації графу до нових умов чи сценаріїв. Користувач може досліджувати, як вага ребра впливає на ефективність алгоритмів та структуру графу у відповідь на зміни у вихідних умовах.

Ця функціональність дозволяє користувачам взаємодіяти з графом та досліджувати його динаміку за допомогою зміни ваги ребер.

# Розробка документів на супроводження програмного забезпечення

## Інструкція програмісту

Для зручності розробки та подальшого вдосконалення застосунку моделювання структур графів з використанням ООП, слід дотримуватися певних кроків та рекомендацій. Нижче наведено детальну інструкцію для програмістів, яка полегшить робочий процес та сприятиме ефективній розробці.

* Клонування репозиторію. Почніть з клонування репозиторію застосунку на вашому робочому пристрої. Використовуйте команду git clone та вказуйте URL 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.
* 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.
* 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.
* Розробка та тестування. Під час розробки, використовуйте доступні елементи графічного інтерфейсу для створення, видалення та зміни елементів графу. Тестуйте алгоритми на різних конфігураціях та переконайтеся в їхній коректності.
* Документація. Не забувайте докладно описувати функції та методи у вигляді коментарів, щоб інші розробники могли швидше зорієнтуватися в коді. Важливо також оновлювати документацію при змінах.

## Інструкція користувачеві

Для користувачів, які мають намір використовувати ваш застосунок для моделювання структур графів, процес старту та користування максимально простий та інтуїтивно зрозумілий.

* Захід на веб-сайт. Відкрийте веб-браузер та перейдіть за адресою вашого веб-сайту, на якому розміщений застосунок. Це може бути доменна адреса чи IP-адреса, яку ви вказали при розгортанні.
* Знайомство з інтерфейсом. Після відкриття сайту, ви опинитесь перед візуально привабливим та інтуїтивно зрозумілим інтерфейсом. Ви побачите візуалізацію графу, ноди, ребра та інші елементи.
* Меню та елементи управління. Ознайомтеся з доступними функціями у меню, яке може бути розташоване знизу екрану. Тут ви знайдете опції для створення та видалення нод, зміни ваги ребер, а також можливості відображення напрямків та вісів ребер.
* Додавання та видалення елементів. Для додавання нової ноди натискайте лівою кнопкою миші на порожній області. При ПКМ на ноді відкриється контекстне меню для видалення. Змінюйте вагу ребра, клікнувши по ньому та вводячи нове значення.
* Експерименти та дослідження. Користуйтеся базовими алгоритмами, які візуально підсвічують ноди та грані за певними критеріями. Експериментуйте з різними конфігураціями графу та спостерігайте за змінами.
* Збереження та завершення роботи. Ваша робота автоматично зберігається. Якщо ви хочете продовжити працювати над тим же графом пізніше, просто поверніться на веб-сайт та продовжте з місця, де ви останнього разу припинили.

# 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.

Розробка структурної моделі системи. Структурна модель системи, визначена в рамках роботи, є ефективною та легко розширюваною. За допомогою об'єктно-орієнтованого підходу до проектування системи, була створена гнучка архітектура, що дозволяє легко впроваджувати нові функції та модулі.

Використання UML діаграм класів стало ефективним засобом архітектурного аналізу та деталізації програмного комплексу. Опис класів надає чітку картину структури системи та взаємодій між її компонентами.

Механізм візуального представлення графів виявився дуже ефективним. Інтуїтивний інтерфейс, здатний динамічно адаптуватися до потреб користувача, робить роботу з застосунком приємною та продуктивною.

Реалізовані алгоритми обходу та аналізу графів, такі як DFS, BFS, алгоритми Дейкстри, Беллмана-Форда, Флойда-Воршелла, виявилися досить ефективними та відповідають вимогам практичного використання. Унікальні алгоритми, які враховують радіус та тип граней, розширюють можливості системи.

Механізм зміни ваги ребра та вивчення її впливу на систему виявився практичним та інтерактивним. Це дозволяє користувачам визначати та адаптувати вагу ребра в режимі реального часу, спостерігаючи за змінами в графі.

Незважаючи на досягнутий успіх, варто розглянути деякі аспекти для подальшого вдосконалення. Зокрема, розширення функціоналу алгоритмів, оптимізація візуалізації та можливість інтеграції з іншими системами можуть покращити конкурентоспроможність продукту.

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.

# 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

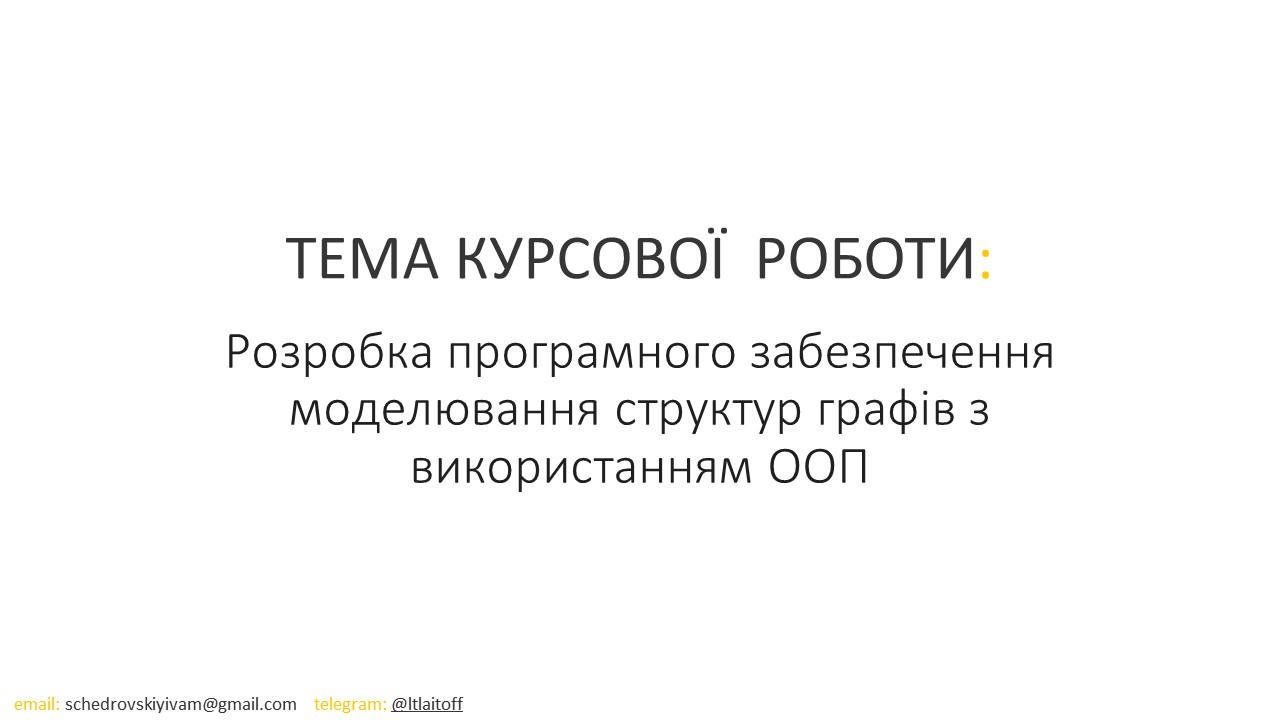
}

}

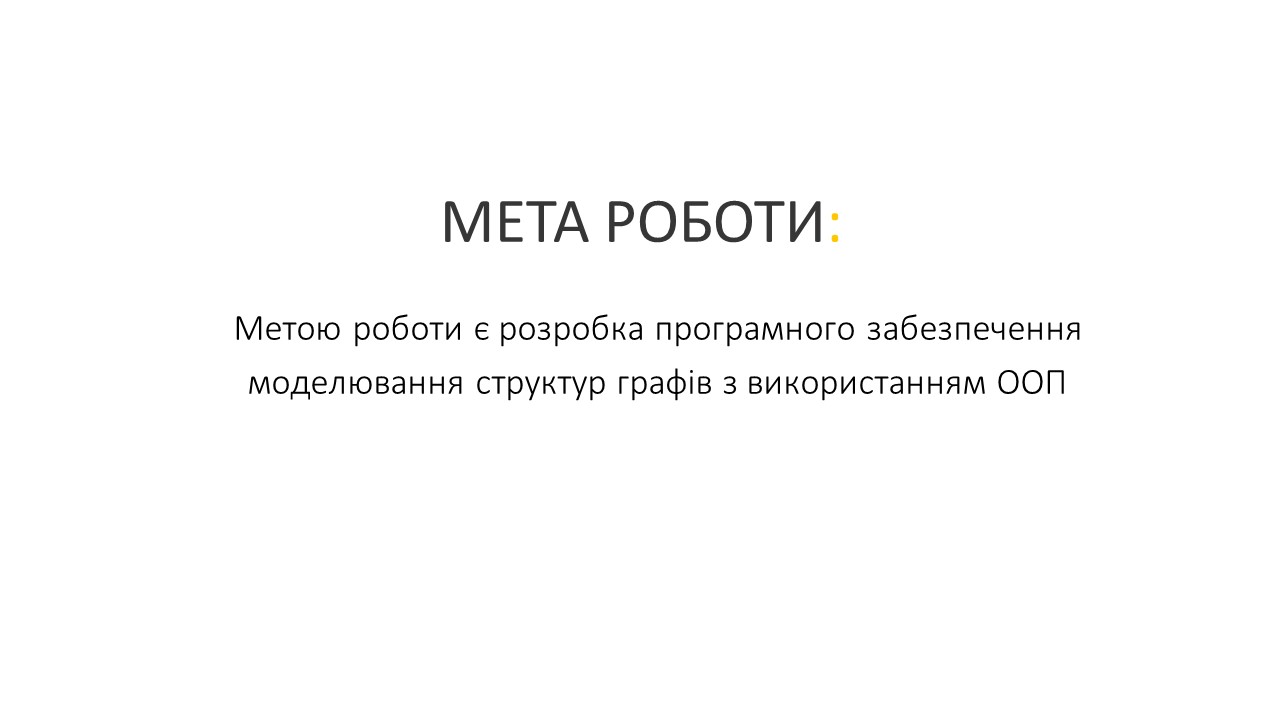
export default WindowManager

# Додаток Б

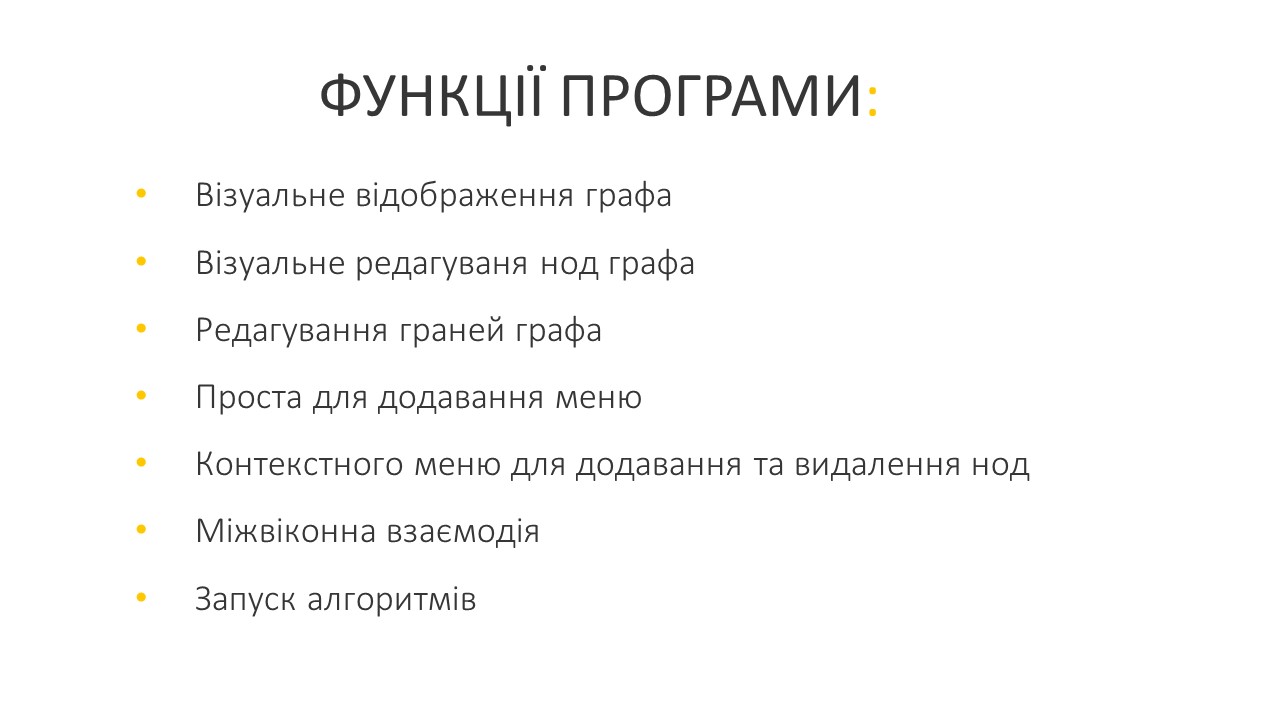
Слайди презентації:



Слайд 1



Слайд 2



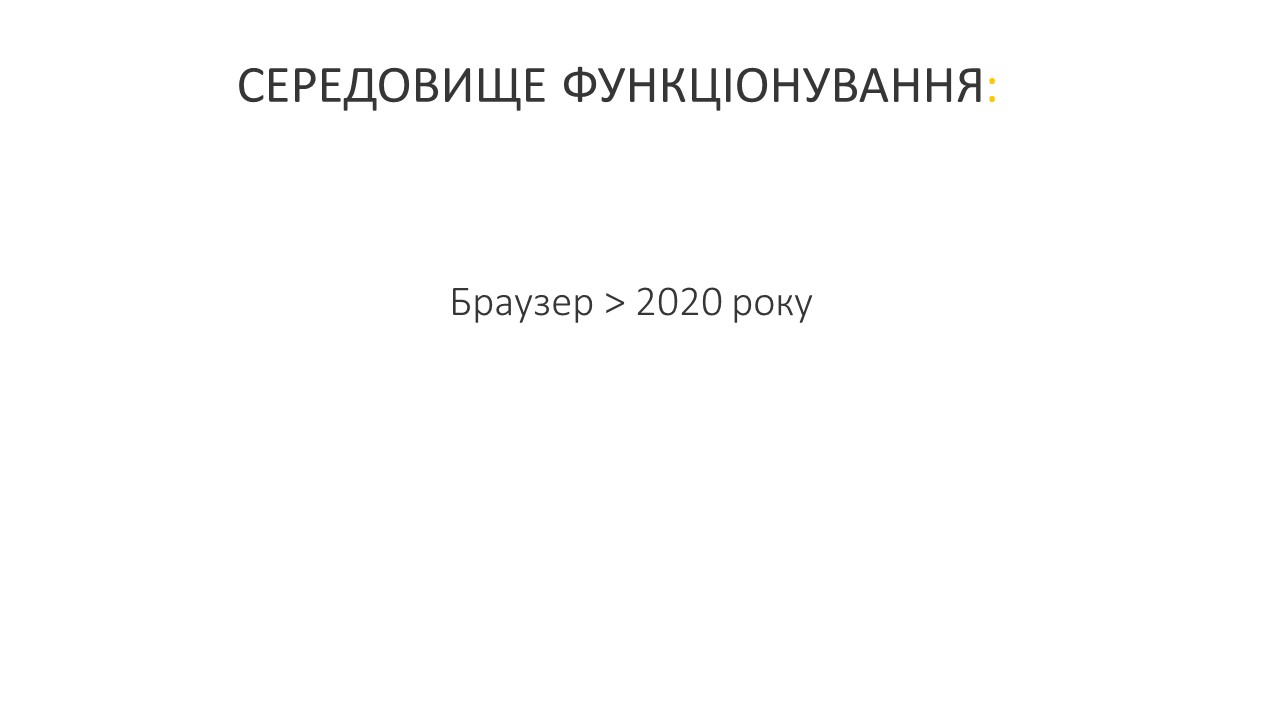
Слайд 3



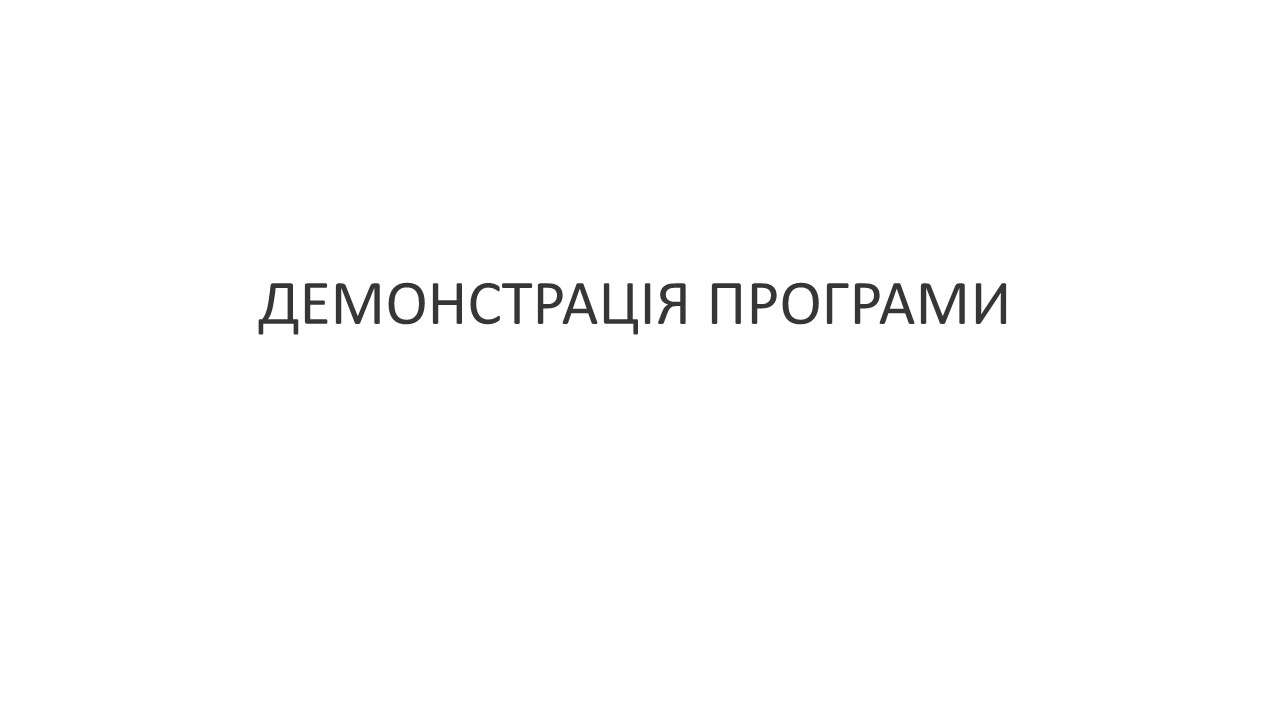
Слайд 4



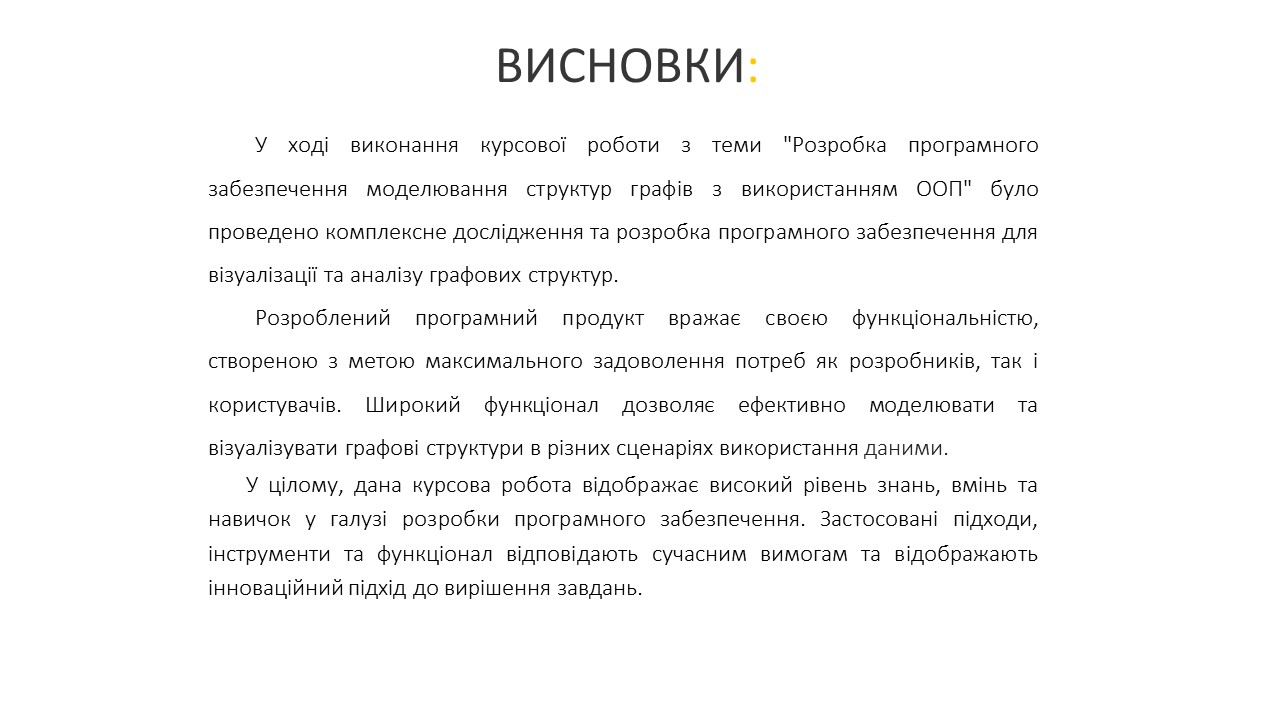
Слайд 5



Слайд 6



Слайд 7



Слайд 8



Слайд 9