

Лабораторна робота №1

Інструментальні засоби розробки програмного забезпечення

1. Тема, мета, короткий теоретичний вступ

Тема: Практичне застосування системи контролю версій Git та юніт-тестування у процесі розробки програмного забезпечення. Формування повного робочого циклу з GitHub — від створення репозиторію до Pull Request.

Мета роботи: Отримати практичні навички використання сучасних інструментів розробки ПЗ, а саме:

- роботи з системою контролю версій Git та сервісом GitHub;
- побудови історії комітів і гілок проєкту;
- написання юніт-тестів до існуючого коду;
- формування правильного процесу створення Pull Request та командної взаємодії у GitHub.

Короткий теоретичний вступ:

Система контролю версій Git — це розподілена система, що дозволяє розробникам відстежувати зміни у файлах проєкту, повертатися до попередніх станів, створювати паралельні гілки для розробки нового функціоналу та ефективно співпрацювати в команді. GitHub — це веб-сервіс, що надає хостинг для Git-репозиторіїв та інструменти для управління проєктом, такі як відстеження завдань, огляд коду та Pull Requests.

Юніт-тестування — це процес перевірки окремих компонентів (модулів, класів, функцій) програмного забезпечення на коректність їх роботи. Головна мета — ізолювати кожну частину програми та переконатися, що вона працює правильно за різних умов. Юніт-тести забезпечують надійність і підтримуваність коду, дозволяють швидко виявляти помилки на ранніх етапах розробки та полегшують подальший розвиток системи без ризику порушити існуючу функціональність.

2. Посилання на репозиторій GitHub

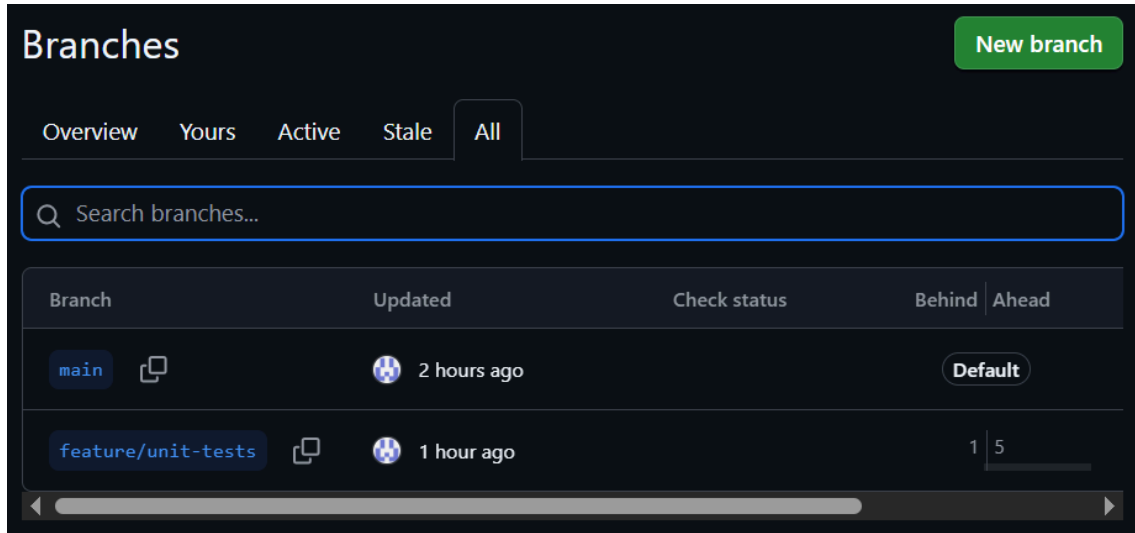
Створено новий репозиторій Software-Development-Tools. Лабораторна робота виконувалася на основі лабораторної роботи з ООП. У репозиторій було додано усі файли з лабораторної та README.md з коротким описом призначення програми.

<https://github.com/anzhelates/Software-Development-Tools.git>

3. Скріншоти основних етапів роботи (створення гілки, коміти, Pull Request)

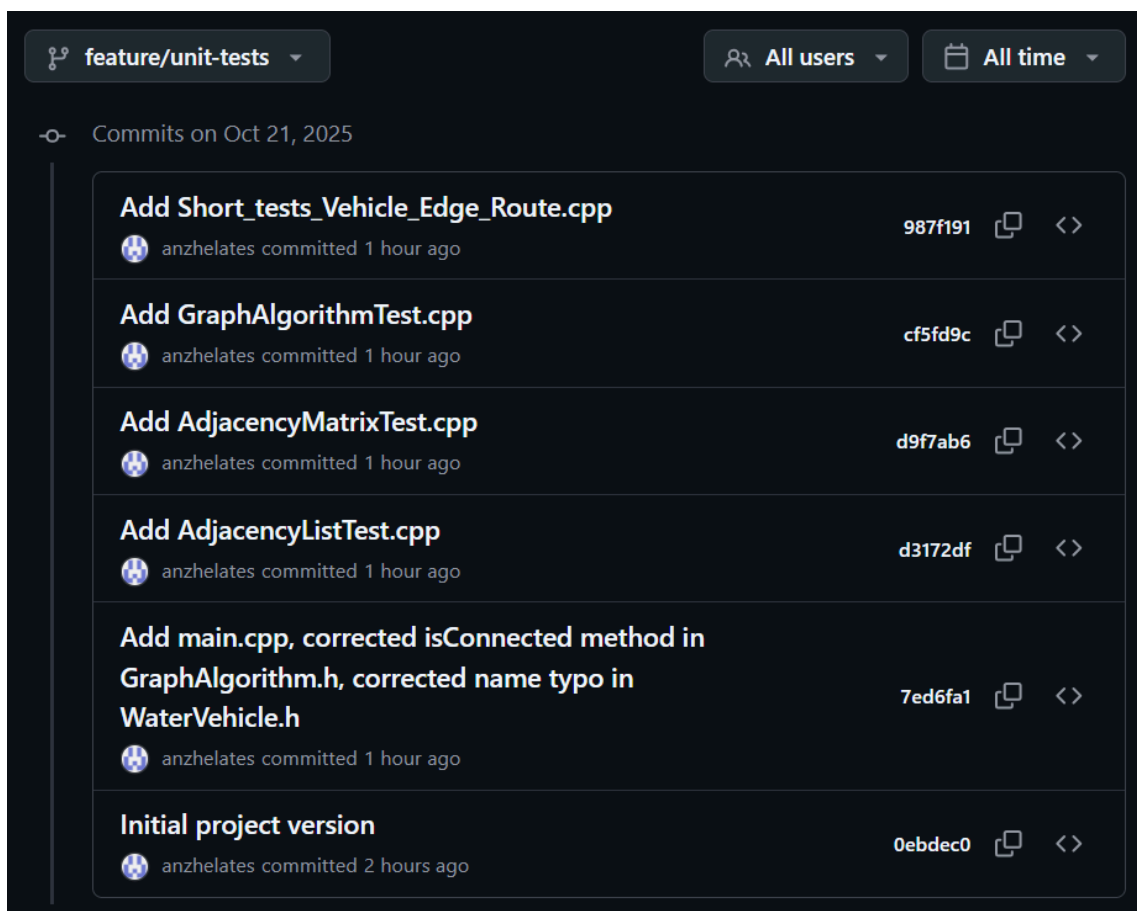
3.1. Створення гілки

Для роботи з юніт-тестами було створено нову гілку feature/unit-tests.



3.2. Історія комітів

Оскільки юніт-тести вже були створені в межах лабораторної роботи з ООП, я послідовно додала вже готові тести з невеликими змінами.



3.3. Pull Request

<https://github.com/anzhelates/Software-Development-Tools/pull/1>

Feature/unit tests #1

[Edit](#) [Code](#) [Jump to bottom](#)


[Open](#) anzhelates wants to merge 5 commits into `main` from `feature/unit-tests`

Conversation 0

Commits 5

Checks 0

Files changed 7

 **anzhelates** commented 48 minutes ago Owner ...

Pull Request


- Юніт-тести розроблялися за допомогою фреймворку **doctest**.
- Було створено окрему гілку для роботи з юніт-тестами `feature/unit-tests`.



Що додано чи змінено:



- Додано юніт-тести: `AdjacencyListTest`, `AdjacencyMatrixTest`, `GraphAlgorithmTest` (BFS, DFS, Dijkstra, `isConnected`), `Short_tests_Vehicle_Edge_Route`.
- Змінено метод `isConnected` у `GraphAlgorithm.h` у ході виправлення помилок. Також проведено невеликий рефакторинг BFS та DFS для усунення плутанини зі змінними.



Що покрито тестами:



- Були створені допоміжні функції `makeCity`, `makeEdge`, `buildSimpleGraph`.
- `Short_tests_Vehicle_Edge_Route`:



 **anzhelates** added 5 commits 1 hour ago

  [Add main.cpp, corrected isConnected method in GraphAlgorithm.h, corre...](#) [7ed6fa1](#)

  [Add AdjacencyListTest.cpp](#) [d3172df](#)

  [Add AdjacencyMatrixTest.cpp](#) [d9f7ab6](#)

  [Add GraphAlgorithmTest.cpp](#) [cf5fd9c](#)

  [Add Short_tests_Vehicle_Edge_Route.cpp](#) [987f191](#)

4. Розробка та опис усіх юніт-тестів:

- Спочатку були створені тести для класів Vehicle, Edge, Route, аби переконатися в коректності базових розрахунків (час, швидкість, паливо).
- Потім - тести для структур суміжності AdjacencyList та AdjacencyMatrix, перевірялися додавання, отримання, видалення вершин/ребер для орієнтованих та неорієнтованих графів.
- І наостанок тести для алгоритмів BFS, DFS, Dijkstra та перевірки isConnected.

4.1. Детальний опис:

1. Клас Vehicle:

- getSpeed: перевірка модифікаторів швидкості залежно від характеристик дороги.
- canUse: перевірка сумісності транспорту з типом дороги (ROAD, RAIL, AIR, WATER).
- calculateFuel: тестування розрахунку витрати пального.

2. Клас Edge:

- getDistance, getSource, getDestination: тестування базових властивостей ребра.
- isActive, markActive, markInactive: перевірка активації/деактивації ребра.
- addObstacle: перевірка додавання перешкод.
- calculateTravelTime: перевірка розрахунку часу без перешкод, розрахунку часу з урахуванням перешкод, повернення нескінченності для несумісного транспорту.

3. Клас Route:

- totalDistance: коректне сумування загальної відстані маршруту.
- totalTime: коректне сумування часу, якщо хоча б одне ребро недосяжне - нескінченність.
- totalFuel: коректне сумування витрат пального.

4. Клас AdjacencyList:

- Операції з вершинами: addVertex, getVertexById, getNumberOfVertices, basic vertex operations (присвоєння ID та отримання даних).
- Операції з ребрами: addEdge, getEdges, edge distances sum (коректне обчислення суми відстаней активних ребер).

- Неорієнтований граф: getNeighbors - перевірка коректності списку сусідів, getEdge - перевірка симетричності ребер, getEdgesFrom - перевірка списку вихідних з вершини ребер.
- Орієнтований граф: directed edge (перевірка односпрямованості).
- Видалення: removeEdge та removeVertex (перевірка деактивації інцидентних ребер).
- Граничні випадки: getEdge correctness and invalid IDs (повернення nullptr для неіснуючих ребер/невалідних ID).

5. Клас AdjacencyMatrix:

- Операції з вершинами та ребрами: addVertex, addEdge, removeEdge, removeVertex.
- Орієнтований/Неорієнтований: перевірка коректного додавання та симетричності зв'язків.
- Отримання даних: getNeighbors, getEdge, getEdgesFrom.
- edges sum - перевірка обчислення суми відстаней.
- Граничні випадки: обробка невалідних ID.

6. Методи BFS та DFS:

- Граничні умови: обхід порожнього графа та графа з однією вершиною.
- Перевірка досяжності всіх вершин у простому неорієнтованому графі.

7. Dijkstra:

- Недосяжність: unreachable vertices (відстань дорівнює нескінченності).
- Знаходження найкоротшого шляху: shortest path undirected.
- Вплив перешкод: obstacle impact (збільшення часу подорожі через одну/кілька перешкод).
- Орієнтований граф: directed graph unreachable (перевірка недосяжності в односпрямованому зв'язку).
- Транспорт: bike slower than car (порівняння часу подорожі для різних транспортних засобів).

8. isConnected:

- Неорієнтований граф: Перевірка зв'язаного та незв'язаного стану.
- Орієнтований граф: перевірка сильної зв'язності (очікує true).
- Орієнтований граф: перевірка слабкої зв'язності (очікує false).

5. Висновки

У ході виконання лабораторної роботи я отримала практичні навички роботи з системами контролю версій та розробки юніт-тестів. Я навчилася вести роботу в окремих гілках, фіксувати зміни за допомогою комітів та синхронізувати їх з віддаленим репозиторієм. Освоїла процес створення Pull Request. Також, я здобула практичний досвід написання юніт-тестів з використанням фреймворку doctest. Навчилася ізолювати компоненти системи та перевіряти їх функціональність у різних сценаріях, включаючи як стандартну поведінку, так і граничні випадки.