Звіт-ретроспектива

1. Які конкретні задачі планували вирішувати за допомогою цієї бібліотеки?

За допомогою підключених бібліотек (наприклад, js-data-structures, graphology, collections) планувалося вирішити одну ключову задачу: об'єктивне порівняння продуктивності (бенчмаркінг) власних реалізацій структур даних з високооптимізованими, перевіреними стандартами.

2. Чому було обрано саме цю бібліотеку, а не аналоги?

Вибір цих бібліотек обумовлений наступним:

- ***benchmark.js*: це **галузевий стандарт** для вимірювання продуктивності JavaScript у Node.js та браузерах. Він не просто вимірює час виконання, а й виконує багато повторень, враховує JIT-оптимізацію (Just-In-Time Compilation) та використовує статистичні методи для визначення похибки (margin of error). Це забезпечує **об'єктивність** результатів, що критично для академічного порівняння.
- ***graphology***: є однією з найшвидших і найсучасніших бібліотек для роботи з графами в JavaScript. Вона оптимізована для швидких операцій додавання/видалення вузлів та ребер. Її модульна архітектура також дозволяє легко підключати додаткові алгоритми (graphology-traversal, graphology-shortest-path), що спрощує розширення бенчмарк-тестів.
- ***js-data-structures**:* ця бібліотека була обрана, оскільки вона надає **чіткі, окремі класи** для класичних структур, таких як BinarySearchTree та AVLTree. Це дозволяє провести пряме, "один до одного" порівняння з вашими власними реалізаціями, не покладаючись на складніші чи багатоцільові бібліотеки.
- ***collections:*** бібліотека містить ефективну реалізацію **двозв'язного списку** (DoublyLinkedList), де операції вставки/видалення з початку (як у SimpleQueue.dequeue) мають **O(1)** складність. Її було обрано, щоб чітко продемонструвати різницю у продуктивності порівняно з вашими O(N) реалізаціями на базі масивів.
- ***js-graph-algorithms:*** бібліотека була обрана як автономний еталон, оскільки не існує поширеного, офіційного плагіна graphology

конкретно для Беллмана-Форда, який є складнішим і рідше використовується. js-graph-algorithms надає надійну, перевірену реалізацію цього алгоритму, що дозволяє провести чисте порівняння коду без необхідності інтегрувати його в екосистему graphology.
- **Доступність та документація**: ці бібліотеки мають велику спільноту, активну підтримку та готові реалізації саме тих структур, які були потрібні для прямого порівняння.

3. Наскільки просто та зрозуміло було отримати, встановити, налаштувати та почати використовувати цю бібліотеку?

Дуже просто. Завдяки **npm**, процес зводиться до однієї команди npm install <назва бібліотеки>. Це сучасний стандарт, який вимагає мінімальних зусиль. Налаштування мінімальне. Головне — правильно імпортувати класи (require).

4. Наскільки зрозумілою та корисною була документація бібліотеки?

- benchmark.js: Документація корисна, але досить складна. Вона детально описує методи, але вимагає розуміння асинхронного виконання та "циклів".
- Всі інші бібліотеки як правило, мають чітку документацію з прикладами і поясненнями, які допомагають зрозуміти, які методи використовувати (insert, addNode, bfs).

5. Наскільки було зрозуміло, як саме використовувати бібліотеку, які класи/методи/функції використовувати для вирішення поставлених задач?

Оскільки бібліотеки мають імена, що відповідають структурам даних (наприклад, BinarySearchTree з js-data-structures), було очевидно, які класи потрібні. Для benchmark.js ключові функції — **Benchmark.Suite** (для групи тестів) та **.add()** (для додавання окремого тесту).

6. Наскільки зручно було використовувати бібліотеку, чи не треба було писати багато надлишкового коду?

Було дуже зручно, бо бібліотеки істотно зменшують кількість коду. Наприклад, замість написання власного алгоритму Дейкстри з нуля, ви просто викликаєте один метод dijkstra(startNode) з graphology-shortest-path.

Надлишкового коду небагато, він потрібен лише для підготовки тестових даних (генерація 10 000 випадкових чисел) та обгортання ваших та бібліотечних методів у формат, який розуміє benchmark.js.

7. Наскільки зрозумілою була поведінка класів/методів/функцій з бібліотеки?

В цілому була зрозумілою, оскільки поведінка відповідає стандартній теорії структур даних (наприклад, AVLTree.insert повинен підтримувати баланс). Хоча, за винятком, можуть бути невеликі відмінності в іменуванні методів (наприклад, у вашому коді deleteNode, а в бібліотеці remove).

8. Наскільки зрозумілою була взаємодія між різними класами/методами цієї бібліотеки, а також взаємодія між бібліотекою та власним кодом?

Взаємодія між різними класами і методами бібліотек була дещо складною. Наприклад, довелося окремо імпортувати graphology для структури та graphology-shortest-path для алгоритму.

Проте взаємодія бібліотек з моїм власним кодом була дуже зрозомілою, бо вона зводилася до:

1. Створення екземпляра вашого класу (const myTree = new SimpleBST();).
2. Створення екземпляра бібліотечного класу (const libTree = new BinarySearchTree();).
3. Виклику однакових операцій для порівняння.

9. Чи виникали якісь проблеми з використанням бібліотеки?

Так, виникали деякі проблеми:

- Пошук відповідників для ваших методів (наприклад, deleteNode vs remove).
- На початкових етапах було незрозуміло, як правильно запускати suite.run({ 'async': true }) та обробляти результати у колбеках.
- У чистому JS використання бібліотек без явних типів викликало помилки, пов'язані з передачею неправильних аргументів.

10. Що хорошого можна сказати про цю бібліотеку?

- **Об'єктивність:** гарантія того, що вимірювання продуктивності є точними та статистично значущими (benchmark.js).

- **Якість еталону:** бібліотеки надають високооптимізований код, що є чесним еталоном для порівняння.
- **Швидкість розробки:** швидкий доступ до складних структур та алгоритмів.

11. Що поганого можна сказати про цю бібліотеку?

- **Розмір node_modules:** встановлення багатьох бібліотек, особливо graphology та її залежностей, значно збільшує розмір папки проєкту.
- **Складність налаштування:** потреба підключати кілька окремих пакетів для графів (один для структури, інший для Дейкстри, третій для Беллмана-Форда).

12. Якби довелось вирішувати аналогічну задачу, але вже враховуючи досвід використання в цій лабораторній роботі, що варто було б робити так само, а що змінити? Можливо, використати інші бібліотеки, чи використати інші можливості цієї бібліотеки, чи інакше організувати код, чи ще щось?

- **Організацію коду.** Можливо, варто було б реалізувати власні структури через інтерфейси або наслідування, щоб забезпечити абсолютну відповідність іменування методів між вашим кодом і бібліотечними аналогами.
- **Алгоритми Графів.** Можливо, варто було б використовувати тільки одну бібліотеку для графів (наприклад, graphology), і самостійно реалізувати лише відсутній алгоритм Беллмана-Форда для порівняння, замість підключення третьої бібліотеки (js-graph-algorithms). Це зробило б проєкт чистішим.

Посилання на ресурси, які були використані під час роботи з бібліотекою

https://benchmarkjs.com/

https://dev.to/harperdb/performance-testing-javascript-node-with-benchmark-js-4g1f

https://jsbenchmark.com/

https://codesandbox.io/examples/package/graphology

https://dev.to/adarshmadrecha/getting-started-with-graphology-2214

https://stackoverflow.com/questions/5909452/javascript-data-structures-library

https://www.npmjs.com/package/js-graph-algorithms