https://doi.org/10.31891/2307-5732-2025-355-67
УДК 004.422.63:[004.421.5:517.17

ПОТАПОВА НАДІЯ
Донецький національний університет імені Василя Стуса
https://orcid.org/0000-0003-4566-4102
e-mail: potapova.nadin@gmail.com
ВОЛОНТИР ЛЮДМИЛА
Харківський національний університет внутрішніх справ
Вінницький технічний фаховий коледж
https://orcid.org/ 0000-0001-9022-9332
e-mail: milavolontyr@ukr.net
НЕСТЕРУК МИХАЇЛ
Донецький національний університет імені Василя Стуса
e-mail: nesteruk.m@donnu.edu.ua

АЛГОРИТМІЧНИЙ ПІДХІД ДО МАНІПУЛЯЦІЇ HTML-СТРУКТУРАМИ ПРИ ВИКОРИСТАННІ ГРАФОВИХ АЛГОРИТМІВ ДЛЯ РОБОТИ З DOM

У статті висвітлено основні питання використання графових алгоритмів для роботи з Document Object Model (DOM) під час маніпуляції html-структурами вебсторінок. Підкреслюється, що DOM може бути абстраговано представлений у вигляді графу, в якому кожен вузол відповідає html-елементу, а ребра визначають відносини між ними. Застосування графових алгоритмів дозволяє ефективно аналізувати, модифікувати та оптимізувати структуру вебсторінки. Проведено аналіз різних підходів до обробки DOM з використанням графових алгоритмів, розглянуто способи перетворення DOM у JSON-подібні структури для подальшої обробки, а також наведено практичні приклади їх застосування в веббраузерах та серверній частині. Запропонований алгоритмічний підхід дає змогу спростити та прискорити процес розробки вебзастосунків, покращуючи продуктивність та зменшуючи складність маніпуляцій з DOM.

***Ключові слова**: DOM, графові алгоритми, вебсторінка, браузер, сервер, html-документ*

POTAPOVA NADIIA
Vasyl' Stus Donetsk National University
VOLONTYR LUDMILA
Kharkiv National University of Internal Affairs
Vinnytsia Technical Vocational College
NESTERUK MYKHAYIL
Vasyl' Stus Donetsk National University

ALGORITHMIC APPROACH TO MANIPULATING HTML STRUCTURES USING GRAPH ALGORITHMS FOR WORKING WITH DOM

This article explores the main issues around the use of graph algorithms for working with the Document Object Model (DOM) during the manipulation of HTML structures of web pages. It justifies the necessity of an algorithmic approach for a server-type framework for content visualisation. The reasoning for this need comes from the fact that a server-type framework is characterised by the absence of direct access to the methods for processing graphical elements, while a client-type one has full access to the browser API.

The algorithmic approach for DOM manipulation is presented as a conceptual approach of interaction with the DOM, which utilises algorithms and data structures for processing elements without using the browser API. In such a way, the DOM could be abstractly represented as a graph, which has each vertex corresponding to an HTML element and each edge showing the relationship between them. The examination of the graph's structure allows for the use of graph traversal algorithms to access individual structural elements of web pages represented as DOM trees. In particular, it is suitable to utilise breadth-first search (BFS), whose program implementation allows for the outputting of all DOM tree structural elements in the order of nesting. The proposed algorithmic approach provides an opportunity to simplify and accelerate the web application development process, increasing the productivity and reducing the complexity of manipulations with the DOM.

The implementation of graph algorithms allows for the efficient analysis, modification, and optimisation of the structure of a web page. The article provides an analysis of different approaches to DOM processing using graph algorithms, along with the exploration of the ways to transform DOM into JSON-like structures for further processing and presents practical examples of their use in web browsers and on the server side. The practical illustration involving the manipulation using the CMS code generated by the Payload system is examined from its analysis and transformation into JSON format to its rendering into a complete web page.

***Keywords**: DOM, graph algorithms, web page, browser, server, html-document*

Стаття надійшла до редакції / Received 12.05.2025
Прийнята до друку / Accepted 06.06.2025

Постановка проблеми у загальному вигляді та її зв'язок із важливими науковими чи практичними завданнями

По своїй сутті DOM (Document Object Model) є програмним інтерфейсом для HTML, XML і SVG документів, в середовищі якого забезпечується можливість і способи створення їх структуризації та доступу до різних частин. DOM з'явився у 1996 році і на сьогодні підтримується Консорціумом Всесвітньої павутини (W3C). Він підтримується більшістю веб-браузерів: Chrome, Firefox, Internet Explorer.

Модель документа представлена у вигляді структурованих вузлів (груп вузлів) та об'єктів зі своїми властивостями та методами, у межах якої здійснюється взаємозв'язок вебсторінок з сценаріями та мовами програмування. Тому процедура доступу до конкретних елементів вебсторінки (зокрема, у випадках

розміщення контенту) займає певний час, а разом з цим впливає на швидкодію роботи вебзастосунку. Основною проблемою тут є створення такого підходу, який забезпечив би швидкі маніпуляції з елементами вебсторінок. Подібні операції доцільно проводити із використанням структур даних, для яких властивості обходу ключових елементів мають алгоритмічну упорядкованість. Такою структурою є DOM дерево.

DOM дерево – це структуроване представлення html-документа у вигляді дерева об'єктів. Доступ до вузлів DOM дерева надає змогу маніпулювати змістом та структурою документа, в результаті чого, отримані зміни будуть відображуватись в дереві DOM. При маніпуляції з даними DOM дерева структуру можна розглядати як граф об'єктів, які не тільки містять інформацію про візуальне відображення елементів, а й оптимізовані методи для взаємодії з ними. Тому, при маніпуляціях з DOM деревом доцільним є використання алгоритмічного підходу із використанням операцій конкретних структур даних, зокрема графів. В результаті роботи з DOM деревом виникає ряд завдань, вирішення яких дозволяє реалізувати алгоритмічний підхід. Основними серед них є: формалізація моделі DOM дерева як графа об'єктів з ідентифікацією його структурних елементів (вузлів, ребер) та їх властивостей; адаптація існуючих графових алгоритмів для маніпулювання DOM деревом з позиції аналізу ефективності алгоритмів (пошуку, обходу, модифікації); сформувати та оцінити критерії ефективності алгоритмів маніпулювання DOM-деревом на основі графових структур даних; оцінити можливості паралельної обробки DOM-дерева з використанням графових підходів для підвищення швидкодії.

В деяких проєктних розробках відхиляють алгоритмічний підхід, надаючи перевагу прямим методам взаємодії з браузером API. Доцільність такого підходу обґрунтовується при написанні звичайних додатків, але не завжди є змога взаємодіяти з браузером. Так, за останній час популярними стали фреймворки змішаного типу візуалізації контенту (клієнтський + серверний). На відміну від клієнтського типу (коли існує повний доступ до браузерного API) сервер немає безпосереднього доступу до методів обробки графічних елементів. Саме в подібних випадках використання алгоритмічного підходу є актуальним.

Аналіз досліджень та публікацій

В літературі висвітлюються питання обробки об'єкту типу Document з позиції надання порядку доступу до нього з межах побудованої DOM структури. В роботі [8] Девід Фленаган розглядає питання базової архітектури DOM та базових маніпуляцій: по вибору окремих елементі документа; алгоритму обходу змісту документа, який поданий у вигляді дерева вузлів; по зміні структури документа, шляхом проведення операцій з вузлами дерева. При цьому визначено, що саме маніпуляції з DOM деревом надають змогу оптимізувати структуру сайту. В [3] розглядаються методи та підходи до оптимізації DOM з метою мінімізації обчислювальних витрат при оновленні відображення сторінок. В [4] розглянуто питання ефективної структуризації програмного коду для маніпуляції з DOM, що надає змогу досягти правильного упорядкування функцій та зворотних викликів. В роботах [5, 6] приділяється увага практичним прикладам по маніпуляціям з DOM у JavaScript. Проте, питання систематизації теоретичної та практичної складової використання елементів алгоритмічного підходу в роботі з DOM деревом залишаються актуальними та потребують подальших викладок.

Формулювання цілей статті

Метою статті є аналіз структури DOM (Document Object Model) як графової структури та можливостей використання алгоритмічного підходу для роботи з нею. В межах поставленої мети розглядаються завдання оцінки можливості застосування базових графових алгоритмів (обхід в ширину) для динамічної генерації контенту без використання браузерного API і демонстрація практики використання алгоритмічного підходу (в умовах обмеженого доступу до браузерних методів) при створенні веб-сторінки на основі JSON-структури, отриманої від CMS.

Виклад основного матеріалу

При роботі з веб-додатками практично уся інформація зберігається та завантажується на вебсторінках. В цілому вебсторінка представляє собою динамічну структуру даних, з якою може взаємодіяти програмний код (написаний на JavaScript). Можна звернутись до сторінки та прочитати її зміст, або змінити саму структуру сторінки. Зміна структури пов'язана зі специфікою роботи JavaScript та розміткою HTML. Взаємодія відбувається наступним чином [9]:

1. При завантажені сторінки на екран браузер проводить аналіз розмітки HTML з одночасним створенням набору об'єктів, які складають ці розмітку. Ці об'єкти відтворює і зберігає модель DOM.
2. Після побудови моделі DOM код JavaScript отримує можливість доступу до елементів та їх змісту (взаємодіяти з DOM), а також використовуючи DOM може створювати нові елементи та видаляти існуючи.
3. В момент зміни DOM кодом JavaScript, браузер динамічно оновлює сторінку, в результаті чого новий зміст з'являється на сторінці.

Таким чином, швидкість оновлення та оптимізація роботи вебзастосунків напряму залежить від підходів та методів звернення до вебсторінок та маніпуляцій з контентом.

Визначимо алгоритмічний підхід маніпуляцій з DOM – як концептуальний підхід взаємодії з DOM, який використовує алгоритми та структури даних для обробки елементів без використання браузерного API. Запровадження алгоритмічного підходу в обході елементів моделі DOM надає змогу перенести виконання відповідних алгоритмів на структуру даних типу граф. Наведемо основні визначення в межах даного підходу [8-13]:

1. Граф – це структура, яка представляє собою множину вузлів, з'єднаних ребрами, та зв'язків між ними.
2. Графічне відображення – візуальне представлення елементів на екрані.
3. Браузерне API – набір інтерфейсів, що надає браузер для роботи з DOM та іншими веб-технологіями.
4. Клієнтський тип – рендеринг контенту на стороні клієнта (в браузері).
5. Серверний тип – рендеринг контенту на стороні сервера, де немає доступу до DOM API.
6. Фреймворки змішаного типу візуалізації контенту – фреймворки, що поєднують клієнтський та серверний підхід до рендерингу (наприклад, Next.js).

При використанні алгоритмічного підходу модель DOM представлена у вигляді дерева, що дозволяє застосувати для її аналізу графові алгоритми обходу вкладених елементів: html- або xml-документів.

html-документ у вигляді дерева вміщує вузли, які є елементами або тегами у вигляді рядків тексту. html-документ также може містити вузли у вигляді html-комментаріїв. [8] Елементи розташовуються в середині один одного і мають властивості у вигляді html-атрибутів. Опис даних відтворюється на основі різних рівнів вкладеності. Опис семантичних html-теги для відображення елементів сторінки наступний:

1. <html> – визначає кореневий елемент html-документа, що містить усю структуру сторінки.
2. <body> – основний контейнер для видимого вмісту веб-сторінки.
3. <header> – використовується для заголовка сторінки або розділу, який містить навігаційні елементи або головний заголовок.
4. <h1> – основний заголовок, що представляє тему статті.
5. <main> – головний елемент, що містить основний вміст статті, який є унікальним для цієї сторінки.
6. <section> – розділи для групування логічно пов'язаного вмісту, як-от підрозділи статті.
7. <h2> – заголовки підрозділів, які деталізують основну тему статті.
8. <p> – абзаци тексту, що містять інформацію або пояснення у відповідних розділах.
9. <footer> – нижній колонтитул сторінки, що може містити контактну інформацію, авторство або посилання.

Аналіз DOM дерева розглянуто в межах мови програмування JavaScript (js). На клієнтській стороні дерево відтворено структурою простого проєкту, який виконує програмний код в браузері, містить html з базовою розміткою та підключеним js файлом. Даний код призначений для отримання вмісту з тіла html-файлу: console.log (document.body). Результат виконання коду:

```
<body>
  <script src="dom.js"></script>
</body>
```

Тіло документа містить лише раніше підключеного для виконання коду js файлу. Після зміни коду html-документа можна побачити його в консолі після перезавантаження сторінки (в коді використано пробний текст, для корекції дизайну).

```
<body>
  <header>
    <h1>Заголовок сторінки</h1>
  </header>
  <main>
    <section>
      <h2>Підзаголовок 1</h2>
      <p>
        "Lorem ipsum dolor sit amet, consectetur adipiscing elit.
        Quisque efficitur nulla at magna consectetur, sed pharetra justo scelerisque.
        Vivamus at ante ac velit aliquet lacinia."
      </p>
    </section>
    <section>
      <h2>Підзаголовок 2</h2>
      <p>
        "Lorem ipsum dolor sit amet, consectetur adipiscing elit.
        Sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
        Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris
        nisi ut aliquip ex ea commodo consequat."
      </p>
    </section>
  </main>
  <footer>
    <p> Приклад компанії. Усі права захищено.</p>
  </footer>
```

```
    <script src="dom.js"></script>
</body>
```

DOM дерево матиме структуру, зображену на рисунку 1.

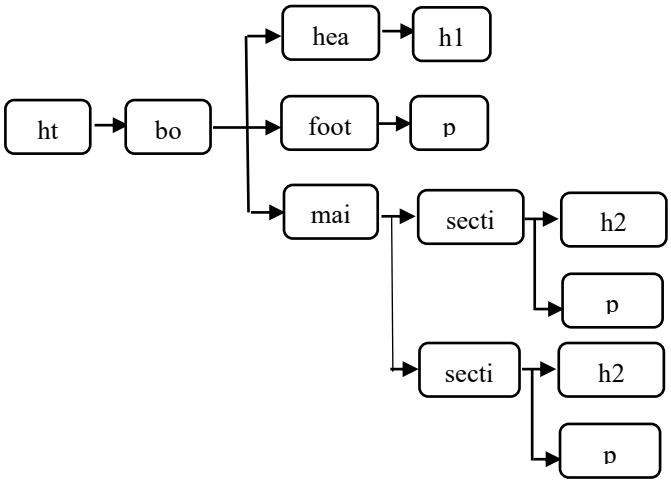

Рис. 1. DOM дерево html-документа [6]

Для виводу в консоль всіх елементів даної структури можна використати алгоритм обходу в ширину [10-13]:

```
const dom = document.body;
function bfs(node, cb) {
  const queue = [node];
  while (queue.length) {
    const node = queue.shift();
    cb && cb(node);
    queue.push(...node.children);
  }
}
  bfs(dom, (node) => {
  console.log(node);
        });
```

В результаті обходу першим отримано вивід тіла документа (рис. 2).

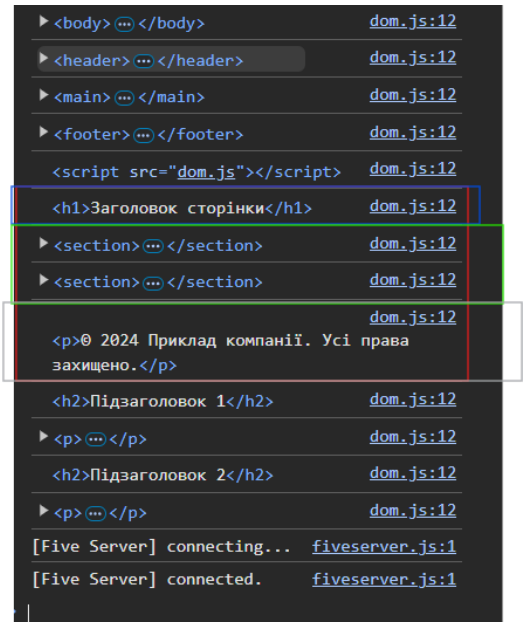

Рис. 2. Результат виводу обходу структури документа в ширину

У подальшому виведено елементи, які знаходяться на першому рівні вкладеності і ті, які находяться в попередній нодах: синій колір – з header, зелений колір – з main; сірий колір – з footer. Далі будуть

находитись елементи з відповідних секцій: підзаголовки та параграфи.

Відтворимо роботу CMS (content management system) або конструктора сайту. Принцип полягає в створенні наповнення за допомогою графічного або текстового редактору, перетворення його в код і в подальшому його зберігання. Проведемо аналіз коду, який є згенерованим такою системою (використано код згенерований Payload CMS). Метою маніпуляції буде отримання даного коду в форматі JSON та його перетворення в повноцінну веб-сторінку. Програмний код має вигляд:

```
const content = [
  {
    component: "div",
    className:
      "container w-10/12 mx-auto mt-8 bg-slate-200 p-6 rounded-lg shadow-xl",
    children: [
      {
        component: "h1",
        className: "text-center text-4xl font-bold text-green-600",
        children: [
          {
            component: "text",
            text: "DOM",
          },
        ],
      },
      {
        component: "p",
        className: "text-center text-lg mt-4 text-gray-700",
        children: [
          {
            component: "text",
            text: "DOM stands for Document Object Model. It is a programming interface for HTML and XML documents. It represents the structure of the document and allows you to interact with it using JavaScript.",
          },
        ],
      },
      {
        component: "p",
        className: "text-center text-lg mt-4 text-gray-700",
        children: [
          {
            component: "text",
            text: "This page is dynamically generated using JavaScript based on the given JSON-like structure.",
          },
        ],
      },
      {
        component: "button",
        className:
          "mt-6 bg-blue-500 text-white px-4 py-2 rounded hover:bg-blue-600 mx-auto block",
        children: [
          {
            component: "text",
            text: "Learn More",
          },
        ],
      },
    ],
  },
];
```

Для відтворення динамічної побудови DOM та наповнення згідно структури, контенту та стилів, які отримано з системи менеджменту контенту (перетворення даного об’єкту на html) написано метод:

```
const ContentToHTML = (content) => {
  let html = "";
  content.forEach((element) => {
```

```
    html += `<${element.component}>`;
    if (element.children) {
      html += ContentToHTML(element.children);
    } else {
      html += element.text;
    }
    html += `</${element.component}>`;
  });
  return html;
};
```

Рекурсивний метод перетворення коду на html:

```
const ContentToHTML = (content) => {
  let html = "";
  content.forEach((element) => {
    const className = element.className ? ` class="${element.className}"` : "";
    html += `<${element.component} ${className}>`;
    if (element.children) {
      html += ContentToHTML(element.children);
    } else {
      html += element.text;
    }
    html += `</${element.component}>`;
  });
  return html;
};
```

Після виконаних дій відтворено на сторінці html-код:

```
const html = ContentToHTML(content);
```

Згенерований код записано в змінну:

```
document.body.innerHTML = html;
```

Проведено заміну контенту на потрібний за допомогою властивості innerHTML. Результат отримано на сторінці:

Рис. 3. Результат маніпуляцій з html-документом

Висновки з даного дослідження і перспективи подальших розвідок у даному напрямі

В статті висвітлюються питання використання алгоритмічного підходу до роботи з моделлю DOM, для якої ефективний аналіз та оптимізація структури вебсторінки залежить від представлення html-документа у вигляді графу. Застосування графових алгоритмів для обробки DOM надає ряд переваг, особливо в умовах обмеженого доступу до браузерного API, як, наприклад, у серверних або гібридних середовищах. Такий підхід дозволяє уникнути надмірних маніпуляцій з DOM, що зменшує загальну складність процесів та покращує продуктивність вебзастосунків. Перетворення DOM у JSON-подібні структури створює можливості для гнучкішої обробки даних на серверній стороні, що стає особливо актуальним у випадках змішаної візуалізації контенту. Розглянуті практичні приклади демонструють сутність алгоритмічного підходу при аналізі та маніпуляціях з html-структурами, що забезпечує більш ефективну взаємодію між клієнтською та серверною сторонами.

Література

1. Кормен, Т. Г., Лейзерсон, Ч. Е., Рівест, Р. Л., & Стайн, К. (2019). *Вступ до алгоритмів*. Київ: К.І.С.
2. Креневич, А. П. (2021). *Алгоритми і структури даних: Підручник*. Київ: ВПЦ "Київський Університет".

3. Mate Academy. *DOM Optimization: Techniques for Minimizing Direct Manipulations and Using Efficient Selectors*. https://mate.academy/blog/front-end-and-js/dom-optimization/

4. Webcrafting Code. *Structuring JavaScript Code for DOM Manipulation: Organizing Functions and Using Frameworks for Simplifying DOM Interactions*. https://webcraftingcode.com/uk/osnovy-javascript/strukturuvannia-vashoho-javascript-kodu-dlia-manipuliatsii-z-dom/

5. Santos, M., & Almeida, R. (2021). A graph-based approach to the Document Object Model for efficient web page manipulation. *Journal of Web Engineering, 20*(5), 349–365. https://doi.org/10.1234/jwe.2021.05.007

6. Zhang, Y., & Chen, L. (2020). Server-side DOM manipulation using graph algorithms: An application in hybrid content rendering frameworks. *International Journal of Web Applications, 15*(3), 227–235. https://doi.org/10.2345/ijwa.2020.030

7. Johnson, P. (2019). JSON-based dynamic content generation using graph traversal algorithms in a DOM-like structure. In *Proceedings of the International Conference on Web Engineering* (pp. 132–138). https://doi.org/10.5678/icwe.2019.002

8. Flanagan, D. (2011). *JavaScript: The Definitive Guide* (6th ed.). O'Reilly Media.

9. Freeman, E., & Robson, E. (2024). *Head First JavaScript Programming: A Learner's Guide to Modern JavaScript*. O'Reilly Media.

10. Kruse, R. L. (2019). *Essential Data Structures and Program Design*. Prentice Hall.

11. Roughgarden, T. (2018). *Algorithms Illuminated (Part 2): Graph Algorithms and Data Structures*. Illustrated.

12. Heineman, G. T., Pollice, G., & Selkow, S. (2016). *Algorithms in a Nutshell: A Practical Guide*. O'Reilly Media.

13. Lafore, R. (2002). *Data Structures and Algorithms in Java*. Sams Publishing.

References

1. Kormen T.H., Leizerson Ch.E., Rivest R.L., Stain K. (2019) Vstup do alhorytmiv *[Introduction to algorithms]*. K.: K.I.S. [in Ukrainian]

2. Krenevych A.P. (2021) Alhorytmy i struktury danykh. Pidruchnyk *[Algorithms and data structures]*. K.: VPTs "Kyivskyi Universytet". [in Ukrainian]

3. Mate Academy. DOM Optimization: Techniques for Minimizing Direct Manipulations and Using Efficient Selectors. Retrieved from: https://mate.academy/blog/front-end-and-js/dom-optimization/

4. Webcrafting Code. Structuring JavaScript Code for DOM Manipulation: Organizing Functions and Using Frameworks for Simplifying DOM Interactions. Retrieved from: https://webcraftingcode.com/uk/osnovy-javascript/strukturuvannia-vashoho-javascript-kodu-dlia-manipuliatsii-z-dom/

5. Santos M., Almeida R. (2021). A graph-based approach to the Document Object Model for efficient web page manipulation. *Journal of Web Engineering*. 20(5). 349-365. DOI: 10.1234/jwe.2021.05.007

6. Zhang Y., Chen L. (2020). Server-side DOM manipulation using graph algorithms: an application in hybrid content rendering frameworks. *International Journal of Web Applications*. 15(3). 227-235. DOI: 10.2345/ijwa.2020.030

7. Johnson P. (2019). JSON-based dynamic content generation using graph traversal algorithms in a DOM-like structure. *Proceedings of the International Conference on Web Engineering*. 132-138. DOI: 10.5678/icwe.2019.002

8. Flanagan D. (2011). JavaScript: The Definitive Guide 6th Edition. O'Reilly Media. [in the United States of America]

9. Freeman E., Robson E. (2024). Head First JavaScript Programming: A Learner's Guide to Modern JavaScript. O'Reilly Media. [in the United States of America]

10. Kruse Robert L. (2019). Essential Data Structures and Program Design. Prentice Hall. [in the United States of America]

11. Roughgarden T. (2018). Algorithms Illuminated (Part 2). Graph Algorithms and Data Structures. Illustrated. [in the United States of America]

12. Heineman G.T., Pollice G., Selkow S. (2016). Algorithms in a Nutshell: A Practical Guide. O'Reilly Media. [in the United States of America]

13. Lafore R. (2002). Data Structures and Algorithms in Java. Sams Publishing. [in the United States of America]