НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено:

Завідувач кафедри

_____ Наталія АУШЕВА

«___»_____2022 р.

Дипломна робота

на здобуття ступеня бакалавра

спеціальності 122 «Комп'ютерні науки»

освітня програма «Комп'ютерний моніторинг та геометричне моделювання процесів і систем»

на тему: «Засоби пошуку першоджерел в літературі на основі методів теорії графів»

Виконала:
студентка IV курсу, групи ТР-81
Яковенко Катерина Олексіївна _____

Керівник:
доцент, кандидат технічних наук,
Кублій Лариса Іванівна _____

Рецензент:

_____ _____

Засвідчую, що в цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студентка _____

Київ — 2022

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Спеціальність 122 «Комп'ютерні науки»

освітня програма «Комп'ютерний моніторинг та геометричне моделювання процесів і систем»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Наталія АУШЕВА
(підпис)

«___»_____2022 р.

ЗАВДАННЯ

на дипломну роботу студенту

Яковенко Катерині Олексіївні
(прізвище, ім'я, по батькові)

1. Тема роботи «Засоби пошуку першоджерел в літературі на основі методів теорії графів»
керівник роботи доцент, кандидат технічних наук, Кублій Лариса Іванівна
(прізвище, ім'я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від "08" червня 2022 р.
№ 965-с

2. Строк подання студентом роботи 10 червня 2022 р.
3. Вихідні дані до роботи мова програмування JavaScript, середовище розробки WebStorm
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): дослідити існуючі програмні системи пошуку першоджерел; проаналізувати основні методи пошуку теорії графів; розробити програмну веб-систему пошуку першоджерел з використанням мультиграфів на основі теорії графів
5. Перелік ілюстративного матеріалу: Контекстна модель роботи сайту, Модель декомпозиції, Діаграма прецедентів, Зв'язки між основними таблицями бази даних, Інтерфейс адміністратора, Інтерфейс користувача.

Дата видачі завдання " 10 " вересня 2021 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи	25.05.2022	
2.	Вивчення та аналіз поставленої задачі	02-08.05.2022	
3.	Розробка архітектури та загальної структури системи	09-15.05.2022	
4.	Програмна реалізація системи	09-28.05.2022	
5.	Написання пояснювальної записки	28.05-02.06.2022	
6.	Захист програмного продукту	28.05.2022	
7.	Передзахист	09.06.2022	
8.	Захист		

Студент _____ Яковенко К. О.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Кублій Л. І.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Пояснювальна записка складається зі вступу, 5 розділів, висновку, списку використаних джерел із 20 найменувань та 1 додатку. Робота містить 74 сторінки, 33 рисунки, 1 таблицю.

У роботі розглянуто засоби пошуку першоджерел в літературі з використанням методів теорії графів.

Проведено аналіз методів пошуку на основі теорії графів, розглянуто особливості мультиграфів та операції над ними.

Результатом виконаної роботи є розробка алгоритму для розв’язання задачі пошуку першоджерел на основі методів теорії графів і створення програмної веб-системи з використанням мови програмування JavaScript і бібліотеки ThreeJs.

Ключові слова: теорія графів, 3D-graphs, JavaScript, алгоритм Дейкстри, силові алгоритми.

ABSTRACT

The explanatory note consists of an introduction, 4 chapters, a conclusion, a list of sources used with 20 titles and 1 appendix. The work contains 62 pages, 28 figures, 1 table.

The paper considers the means of searching for primary sources in the literature based on the methods of graph theory.

The analysis of search methods on the basis of the theory of graphs is carried out, features of multigraphs and operations on them are considered.

The result of this work is the development of an algorithm for solving the problem of finding primary sources based on methods of graph theory and the creation of a software web-system using the JavaScript programming language and the ThreeJs library.

Keywords: graph theory, 3D-graphs, JavaScript, Dijkstra algorithm, power algorithms.

ЗМІСТ

ВСТУП

Завдяки зростанню кількості користувачів і кількості інформації, яка створюється і є доступною для використання, кількість джерел інформації у світі зростає в геометричній прогресії. Внаслідок особливостей взаємодії людини і машини, з одного боку, та семантичної неоднорідності джерел інформації, з іншого, успішний пошук інформації стає дедалі складнішим. Розв'язання цієї проблеми полягає в персоналізації методів пошуку інформації, тобто в адаптації процесу пошуку до унікальних особливостей користувачів, що дає можливість швидко і з найменшими зусиллями знаходити відповідну інформацію.

Використання користувацьких моделей для розширення запиту під час адаптивного пошуку інформації значно зменшує тривалість інтерактивної взаємодії і витрати часу й зусиль користувача, оскільки уточнення запиту обробляється на боці клієнта. Такий підхід є найбільш перспективним з точки зору зниження часу і вартості пошуку.

Програмний дипломний проект полягає у створенні алгоритму на основі теорії графів для розв'язання задач пошуку першоджерел, а також реалізації класів для роботи зі структурами мультиграфів при виконанні процедури пошуку.

Дипломна записка містить 4 розділи. У першому розділі вказано мету дмпломного дослідження і зроблено постановку задачі, проаналізовано предметну область досліджуваної теми. У другому розділі проаналізовано алгоритми пошуку і вибрано силові алгоритмі візуалізації графів для графічного відображення графів у інтерфейсі. У третьому розділі обґрунтовано вибір засобів розробки системи. У четвертому розділі описано роботу користувача зі створеною системою і вказано її особливості.

Потенційними користувачами розробленої системи є користувачі з різних сфер діяльності, зокрема: науковці, студенти, викладачі, діячі культури, журналісти, публіцисти та інші.

1 ЗАДАЧА ПОШУКУ ІНФОРМАЦІЇ І ШЛЯХИ ЇЇ РОЗВ'ЯЗАННЯ

Перш ніж розробляти веб-систему пошуку інформації, зокрема першоджерел в літературі, треба чітко сформулювати завдання, які для цього мають бути ров'язані, дослідити переваги й недоліки існуючих пошукових систем і використані в них підходи й алгоритми, вибрати алгоритми, які можна напрямувикористати чи адаптувати для розв'язання саме задачі пошуку першоджерел, враховуючи особливості посилань в літературі.

1.1 Пошук першоджерел в літературі на основі методів теорії графів

Метою дипломної роботи є розробка алгоритму на основі методів теорії графів для розв'язання задачі пошуку першоджерел в літературі, а також створення програмної системи для роботи зі структурами мультиграфів при виконанні процедури пошуку.

Для досягнення мети треба провести аналіз сучасних шляхів побудови систем пошуку даних і визначити конкретні завдання, які треба розв'язати при розробці і створенні веб-сервісу з вбудованою системою пошуку першоджерел.

Для досягнення мети дипломної роботи необхідно виконати такі завдання:

— дослідити різноманітні існуючі системи для пошуку першоджерел;

— проаналізувати основні положення теорії графів та її похідних;

— провести обчислювальні експеременти з різноманітними алгоритмами побудови комплексних систем з використанням графів;

— оцінити якість результатів різних алгоритмів;

— розробити систему пошуку першоджерел з на основі теорії графів;

— для наочного подання результатів пошуку застосувати силові алгоритми візуалізації графів.

Створюваний програмний інструмент має забезпечити швидкий перегляд існуючої бази даних з джерелами або створення нової. Створена система пошуку першоджерел на основі веб-технологій повинна забезпечити користувачеві різноманітність і простоту використання на будь-якому пристрої без необхідності змінювати операційну систему.

Браузер і його віртуальна машина виступають як цільова універсальна операційна система і комп'ютер.

1.2 Існуючі системи пошуку

Існує досить велика кількість пошукових систем. Найпопулярнішими є Google, Yahoo, Baidu.

Пошукова система Google — це складна технологія, яка робить пошук інформації в Інтернеті не зовсім простим. Для одержання результатів пошуку Google використовує унікальний алгоритм. Але про цей алгоритм вона надає загальну інформацію, не розкриваючи деталей. Це зроблено для того, щоб залишатися конкурентоспроможною пошуковою системою [1].

Якщо розглядяти пошукову систему Yahoo, то на сторінці з результатами пошуку виводяться спочатку відповідні критерії пошуку категорії, а потім сайти. Біля кожної категорії в дужках стоїть кількість – це кількість сайтів у цій категорії. Якщо на Yahoo! немає результатів, одразу виводяться результати з Altavista. Вгорі та внизу сторінки виводиться маленька табличка, за допомогою якої можна одним натисканням кнопки миші зробити пошук у категоріях Yahoo!, на Altavista, у новинах та подіях. Кількість результатів пошуку на Yahoo!, звичайно, невелика, але більшість з них є релевантними. Можлива проблема з

відсутніми сторінками, оскільки веб-майстри зазвичай забувають видалити свої сайти з пошукових систем, а на Yahoo! немає механізму автоматичного оновлення. Для розширеного пошуку Yahoo! пропонує невеликий, але дуже корисний набір інструментів. Щоб потрапити на сторінку розширеного пошуку, потрібно перейти на посилання “options” з основної сторінки Yahoo!.

Якщо ж розглядати процес пошуку в системі Google, то можна зрозуміти, що він більш складний та при цьому зручніший у використанні.

Павуки, або сканери — це автоматизовані програми Google. Система Google, як і інші пошукові системи, має велику кількість проіндексованих термінів. Пошукова система Google ранжує результати пошуку, визначаючи цим порядок, у якому результати подаються на сторінці результатів пошуку. Система Google використовує алгоритм PageRank для визначення релевантності кожної веб-сторінки [2].

Для алгоритму пошуку в системі Google найважливішою є інформація про кількість інших сторінок, що містять посилання на сайти з інформацією, яка шукається.

Як приклад, можна розглянути словосполучення «планета Земля» під час пошуку. Рейтинг сторінки під назвою Discovery Earth, що містить шукану інформацію, покращується, оскільки на нього посилається більше веб-сторінок. Інші сайти відображаються нижче в результатах пошуку Google, так як Discovery займає вищий рейтинг [3].

У 2008 році Google почав експериментувати зі своєю пошуковою системою. Спочатку Google дозволяє невеликому набору бета-тестерів змінювати рейтинг результатів пошуку. Тестери бета-версії можуть рекламувати або мінімізувати результати пошуку в цій пробній версії, а також можуть змінити свій пошук, щоб зробити його більш релевантним [4].

Необхідно розглянути, яким чином Google знаходить інформацію, яка необхідна користувачу для відображення в результатах пошуку (рисунок 1.1).

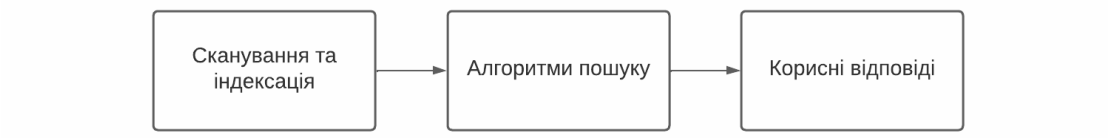

Рисунок 1.1 — Схема роботи пошукового сервісу

Щоб впорядкувати інформацію з онлайн-сторінок в індексі пошуку використовуються алгоритми веб-сканерів та павуки. Після цього запускаються алгоритми пошуку. Алгоритми рейтингу Google за лічені секунди “просіюють” мільярди веб-сторінок в індексі пошуку, щоб отримати корисні та релевантні результати.

Алгоритми пошуку є наступним етапом. Метод ранжування інформації, який Google використовує для надання інформації з Інтернету, зображений на рисунку 1.2.

Рисунок 1.2 — Схема процесу пошуку та отримання інформації в мережі Інтернет

Розуміння значення пошукового запиту має вирішальне значення для відповідних результатів пошуку. У результаті, перш ніж буде визначено сторінки з корисною інформацією, потрібно з'ясувати, що означають слова у пошуковому запиті. Мовні моделі створюються, щоб спробувати з'ясувати, які рядки слів шукати в індексах. Це охоплює такі речі, як розуміння орфографічних помилок, розпізнавання типу запитання та використання деяких з останніх досліджень розуміння природної мови.

Наприклад, система синонімів — це пошук кількох запитів, навіть якщо термін є значущим.

В Інтернеті існує багато спам-сайтів, які намагаються завершити результати пошуку, використовуючи такі стратегії, як повторення ключових слів або посилання, яке проходить через PageRank. Ці сайти пропонують поганий досвід користувачів і потенційно можуть завдати шкоди чи дезінформації [5].

Алгоритм використовує огляд контексту, а також таку інформацію, як місцезнаходження користувача, попередня історія пошуку та параметри пошуку, щоб змінити результати, щоб інформація була найбільш корисною та релевантною для користувача на даний момент.

Перш ніж алгоритм відобразить результати, виводяться найкращі результати пошуку. Алгоритм враховує, як поєднується весь релевантний матеріал: чи є одна тема серед результатів пошуку або їх велика кількість, а також чи занадто багато сторінок, присвячених одній обмеженій інтерпретації.

Алгоритм спрямований на надання різноманітного діапазону даних у формах, які є найбільш вигідними для певного типу пошуку.

Система Google має можливість пропонувати результати пошуку в широкій мережі форматів, щоб допомогти користувачеві швидко знайти потрібну інформацію, але це часто важко зробити через пошук релевантної інформації [6]. З більшою кількістю вмісту в розмові, ніж будь-коли раніше, Google має можливість пропонувати результати пошуку в широкій мережі форматів, щоб допомогти користувачеві швидко знайти потрібну інформацію, але це часто важко

зробити через пошук релевантної інформації [6]. Інтернет постійно змінюється, кожну секунду нові веб-сайти публікуються у великій кількості. Це відображається в результатах пошуку Google, оскільки компанія постійно аналізує дані в мережі, щоб індексувати свіжі матеріали.

Останнім етапом алгоритму є надання відповідних відповідей. Деякі сторінки результатів швидко змінюються у відповідь на запити користувачів, а інші стають більш надійними.

Щороку Google обробляє мільярди пошукових запитів. Щодня 15% запитів, які обробляються, є такими, яких користувачі ніколи не бачать. Розробка алгоритмів пошуку, які можуть забезпечити найбільш релевантні результати для всіх цих запитів, є складним процесом, який вимагає постійного тестування якості та інвестицій.

Основний сканер сканує всі онлайн-сторінки, які зустрічає, а інший, відомий як Orange, виконує швидке індексування, щоб гарантувати, що найактуальніші документи, включно з документами, що містяться в Інтернеті, або навіть за секунди до сканування, доступні в індекс пошукової системи. Веб-сайти, які необхідно проіндексувати, знаходяться в списках очікування обох сканерів. Нові посилання, виявлені сканерами на сторінках відвідувачів, регулярно додаються до списків. Після того, як власники сайтів використовують систему обслуговування для додавання своїх сторінок до індексу, нові посилання можуть з’являтися в списках очікування.

Інша серверна програма видаляє всю зайву інформацію розмітки зі сторінки, залишаючи лише вміст. Потім він витягує інформацію про цей період часу та надсилає адміністратору всі слова в цьому документі. До наступного сканування вихідний документ також зберігається на сервері. Це дозволяє алгоритму надавати доступ до документів своїм користувачам, навіть якщо сайт зараз не працює. Алгоритм звільняє документ зі своїх серверів або замінює його більшою версією, якщо сайт закривається або документ знищено чи змінено.

Пошукова база створюється шляхом об’єднання пошукового індексу з копіями всіх індексованих документів, включаючи їх тип, код і мову (рисунок 1.4).

Рисунок 1.4 — Фрагмент бази даних індексів з копіями усіх проіндексованих документів [6]

Пошукова база даних має оновлюватися на регулярній основі, щоб не відставати від постійно мінливих матеріалів Інтернету та гарантувати, що пошукова система може знайти правильний баланс і найновішу інформацію у відповідь на певний пошуковий запит. Кожне оновлення бази даних спочатку керується на головному сервері пошуку, де може бути розташована перша пошукова система та результати використання результатів. Без дзеркал, дзеркальних сайтів або інших документів, які не є документами, базові пошукові сервери містять лише основну частину інформації. Це розділ пошукової бази даних, який відповідає на запити користувачів [7].

Кожні кілька днів «пакети» даних пошукової бази доставляються з серверів зберігання на головний пошуковий сервер.

Пошукова база даних має оновлюватися на регулярній основі, щоб гарантувати, що пошукова система зможе знайти належний баланс і отримати інформацію у відповідь на певний пошуковий запит. Кожне оновлення бази даних починається на головному сервері пошуку, де вона може бути розміщена як

початкова пошукова система та результати її використання. Базові пошукові системи мають лише більшість інформації без дзеркал, сайтів-дзеркал та інших документів, які не є документами. Це частина пошукової бази даних, яка відповідає на запити користувачів [8].

“Пакети” даних пошукової бази надсилаються з серверного сховища на головний пошуковий сервер кожні кілька днів.

Сканер Orange — це пристрій пошуку інформації в режимі реального часу. Його підхід, як і підхід Spider’s, налаштований на пошук останніх статей, а також вибір великої кількості сайтів, які були б цікавими для споживачів. Ці файли негайно аналізуються та пересилаються на основні пошукові сервери. Через невеликий розмір цих даних зміни можуть вноситися в режимі реального часу, навіть протягом дня, не викликаючи перевантаження сервера [8].

До того, як працює пошукова система, потрібно виконати два кроки. Сканування мережі, індексація сторінок і підготовка їх до пошуку на ранніх етапах розробки. Іншим етапом є пошук у раніше створеній пошуковій базі даних для відповіді на конкретний запит користувача [9].

Знайти цінну інформацію стає все важче через поширення інформації в Інтернеті. Основна складність пошукової системи полягає в тому, як ефективно отримувати та використовувати цю інформацію.

Система в першу чергу відповідає за збір даних, зберігання та оновлення інформації в Інтернеті як джерела інформації для всієї пошукової системи.

1.3 Елементи теорії графів

Нехай V — непорожня множина, $V^{(2)}$ — множина всіх невпорядкованих різних двоелементних підмножин множини V, а $MV^{(2)}$ — мультимножина $V^{(2)}$, тобто $MV^{(2)}$ може включати ті самі пари елементів із V, і цих пар може бути скільки завгодно. $V^{(2)}$ позначає декартів квадрат множини V.

Пара (V, E), де $E \subseteq MV^{(2)}$, є неорієнтованим мультиграфом G. Компоненти множини V — вершини, компоненти множини E — ребра. Пари (u, v) означають ребра, де u, v — вершини із множини V.

Якщо $E \subseteq V^{(2)}$, то мультиграф G = (V, E) називають неорієнтованим графом. Кожен мультиграф є графом, але не кожен граф є мультиграфом. Множина E може містити багато ребер, які зв'язують ті самі вершини u і v, якщо G = (V, E) є мультиграфом. Кілька ребер є різновидом ребра. У результаті граф є мультиграфом з кратністю одиниці для кожного ребра.

Якщо множини V і E скінченні, мультиграф називають скінченним. Для скінченного графа достатньо лише скінченності множин вершин V, оскільки скінченність V дорівнює скінченності $V^{(2)}$, тобто граф скінченний, якщо множина його вершин скінченна.

Граф n-го порядку — це скінченний граф з n вершинами. Іноді розглядають графи, які мають ребра (u, u) — петлі. Петля (u, u) — це тип ребра, а мультиграф із петлями відомий як псевдограф. Дві вершини u і v графа G = (V, E) суміжні, якщо $(u, v) \in E$, і несуміжні — в протилежному випадку.

Якщо $(u, v) \in E$, то вершини u, і v називають кінцями ребра (u, v). Множину вершин графа, суміжних з деякою вершиною u, позначають Cm(u).

Різниця між графом і мультиграфом полягає в тому, що дві вершини в графі можуть бути з'єднані лише одним ребром, відповідно до вищезгаданих визначень.

Якщо u — кінець ребра e, то вершина u і ребро e вважаються інцидентними; інакше їх називають неінцидентними.

Кількість ребер, інцидентних вершині в графі G, є його ступенем n (u). Вершина зі ступенем 0 — ізольована, тоді як вершина зі ступенем 1 — висяча, або кінцева.

Сума степенів всіх вершин графа є парним числом. Кожне ребро вносить у суму всіх вершин графа число 2 (1.1), тобто:

$$\sum_{v \in V} n(v) = 2|E| \qquad (1.1)$$

Графи можуть бути представлені в площині або просторі у вигляді діаграм, які складаються з точок і відрізків, що з'єднують точки (рисунок 1.5). Вершини ідентифікують точки, а ребра ідентифікують сегменти.

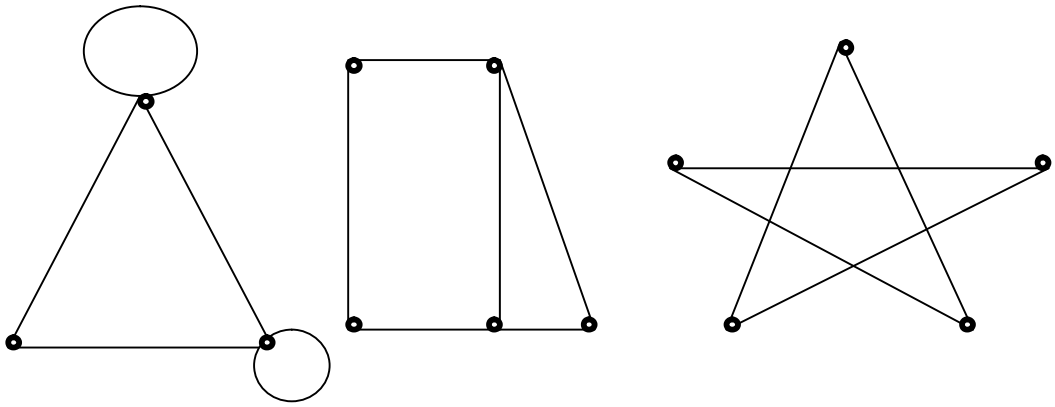

Рисунок 1.5 — Різновиди графів (псевдограф / мультиграф / граф)

Повний граф — це граф, у якому будь-які дві його вершини суміжні. У результаті, якщо G = (V, E) є повним графом. Позначення K_n позначає повний граф з n вершинами. Граф K_5 показано на рисунку 1.6.

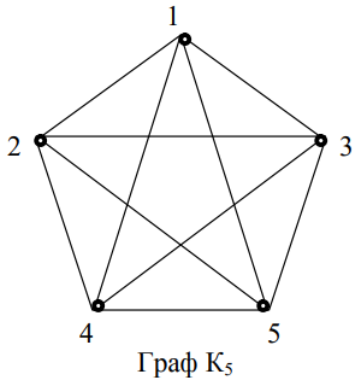

Рисунок 1.6 — Різновиди графів (повний граф)

Виявлено, що існують псевдографи, мультиграфи і звичайні графи, і вони доступні для використання при розробці веб-застосунку.

Більш загальними, ніж повні, є регулярні графи. Якщо всі вершини графа мають однаковий ступінь, його називають регулярним або однорідним. Граф називають регулярним графом ступеня k, якщо ступінь кожної вершини дорівнює k.

У результаті повний граф n-го порядку є регулярним графом ступеня n — 1. Кубічні або тривалентні графи — це регулярні графи третього ступеня. На рисунку 1.7 подано граф Петерсона.

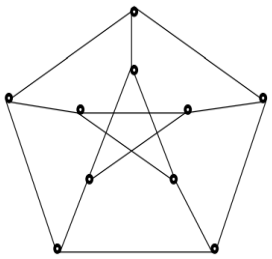

Рисунок 1.7 — Граф Петерсона

Якщо $E \subseteq V2$ — тобто вершини всіх його ребер впорядковані, то граф $G = (V, E)$ називають орієнтованим графом (орграфом). Якщо $(u, v) \in E$, то вершина u є початковою вершиною ребра, а v — кінцевою вершиною ребра. Орграфи малюються так само, як і графи, за винятком того, що їх ребра позначають стрілками, що йдуть від рочаткової до наступної вершини.

Якщо $E \subseteq MV2$ і кожне з його ребер упорядковані, тобто вказано, яка вершина перша, а яка друга, то граф $G = (V, E)$ називають орієнтованим мультиграфом. У результаті ребра (u, v) і (v, u) в орграфах відрізняються.

1.4 Операції над мультиграфами

Нехай $G = (V, E)$ — граф, де $e \in E$ позначає деяке його ребро. Якщо $G1 = (V, E \setminus \{e\})$, то граф G1 виводиться з графа G за допомогою процедури вилучення

ребра e. Множина V не включає кінцеві точки ребра e.

Легко довести, що для довільних ребер e і e1 графа G виконується тотожність: (G — e) — e1 = (G — e1) — e.

Оскільки виконується тотожність (A \ B) \ C = A \ (B ∪ C), то маємо

G1 = G — e = (V, E \ {e}),

(G — e) — e1 = G1 — e1 = (V, (E \ {e} \ {e1}) = (V, E \ ({e} ∪ {e1})) = (V, E\ ({e1} ∪ {e})) = (V, (E \ {e1}) \ {e}) = (G — e1) — e.

У результаті численних процедур вилучення ребер виконуються підряд (рисунок 1.8), але на результат не впливає порядок, у якому ребра вилучають.

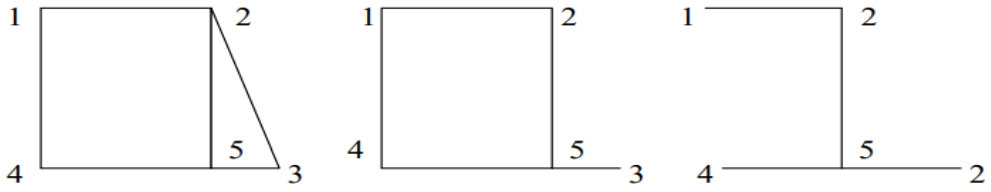

Рисунок 1.8 — Процедура вилучення ребра

Нехай G = (V, E) і v ∈ V. Якщо вершину v вилучити з V, а всі ребра, інцидентні вершині v, видалити з E, то в результаті операції вилучення вершини v з графа G утвориться граф G1 = G — v.

Послідовність, у якій вершини видаляються з графа, не має ніякого відношення до процедури вилучення вершин (рисунок 1.9).

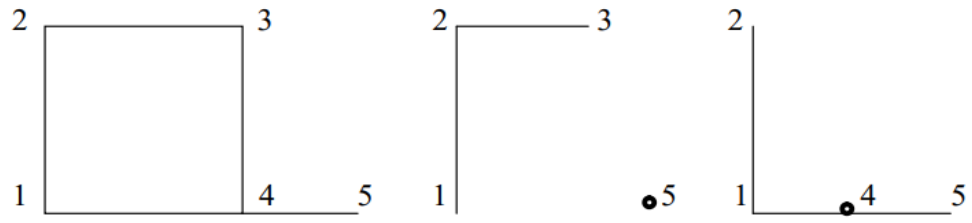

Рисунок 1.9 — Процедура вилучення вершини

Якщо u, v ∈ V і (u, v) ∉ E в графі G = (V, E), то граф G + e = (V, E ∪ ∪{e}), де e = (u, v).

Можна стверджувати, що послідовність операцій для введення ребер у граф G не залежить від порядку, в якому ці ребра вставляються в граф G, завдяки комутативній операції об'єднання множин.

Справедлива тотожність: ∀e, e1 ∈ E ((G + e) + e1 = (G + e1) + e).

Нехай (u, v) — деяке ребро графа G. Введенням вершини w в ребро (u, v) називають операцію, внаслідок якої буде одержано два ребра (u, w) і (w, v), а ребро (u, v) при цьому вилучається з графа G. Скінченна колекція X (члени якої називають вершинами) і множина U з двоелементних підмножин X (елементи множини U називають ребрами) утворюють граф G = (X, U).

Мультиграфи і псевдографи — це графи з багатьма неорієнтованими або орієнтованими ребрами (рисунок 1.10).

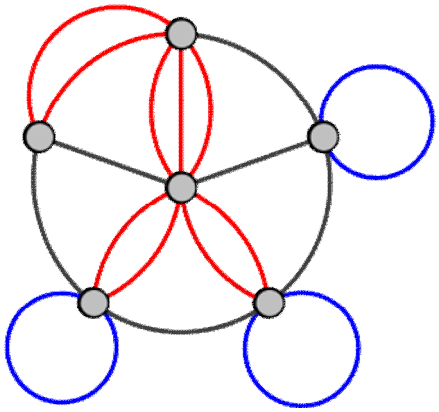

Рисунок 1.10 — Мультиграф з кратними ребрами (червоні) і петлями (сині)

Скінченна колекція X (члени якої називають вершинами) і множина U з двоелементних підмножин цього X (елементи множини U називають ребрами) утворююьб граф G = (X, U). Порівняний орієнтуючий граф (X, U) складається із скінченної множини X і множини U, що містить впорядковані пари компонентів множини X, відомі як орієнтовані ребра або дуги [10].

Наявність петель — ребер або дуг, обидві вершини яких збігаються — у цій ситуації (рисунок 1.11).

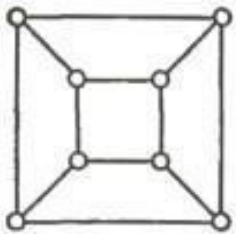

Рисунок 1.11 — Регулярний граф степеня 3

Матриця суміжності A мультиграфа G з множиною вершин $\{x_1, x_2, \ldots, x_n\}$ — це квадратна матриця порядку n, у якій значення a_{ij} елемента, розташованого на місці (i, j) дорівнює кількості ребер (дуг), що починаються у вершині x_i і закінчуються у вершині x_j.

Мультиграфи G=(X, U) і H=(Y, V) називають ізоморфними, якщо існує(1,1)-відображення y=φ(x) X на Y таке, що для будь-якої пари вершин x', x"∈X у H є стільки ж ребер, що прямують із y'=φ(x') до y"=φ(x"), скільки у G прямує із x' у x" (рисунок 1.12).

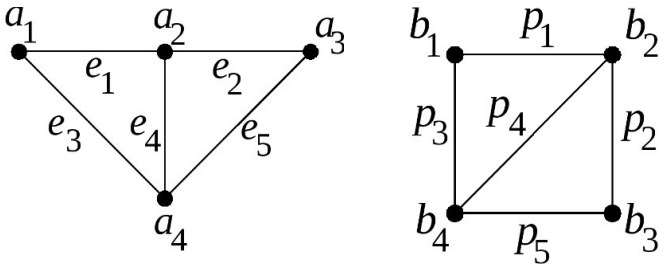

Рисунок 1.12 — Ізоморфні графи

Характеристичний поліном графа G складається з матриць суміжності A.

Власні значення і спектр матриці A називають власними значеннями і спектром графа G відповідно.

Якщо Y X і V U, то графік H = (Y, V) є підграфом графа G = (X, U). Якщо X = X, граф H відомий як каркасний підграф або частковий граф графа G. H є створеним підграфом, якщо множина V складається з усіх таких ребер множини U, які зв’язують вершини множини Y.

Повний граф на n вершинах — це граф, у якому будь-які дві різні вершини з’єднані одним ребром (рисунок 1.13).

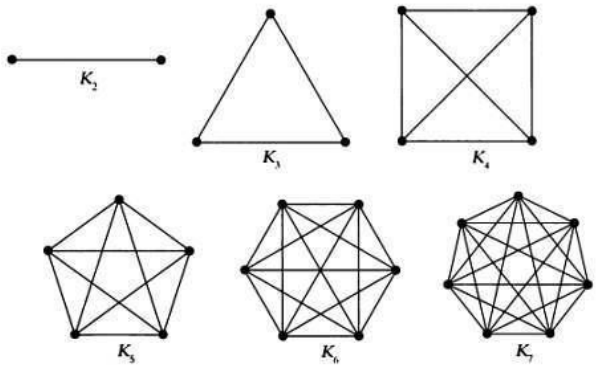

Рисунок 1.13 — Приклади повних графів

Маршрут — це сукупність послідовних ребер у мультиграфі. Кількість ребер на маршруті визначає його довжину. Один і той же край може з’являтися на шляху багато разів. Простий ланцюг $n_i n_i$ довжиною n — 1, n 2 — це мережа з n вершинами x1,..., $x_n x_n$ і n — 1 ребрами, з вершинами xi і xi + 1, I = 1,..., n — 1 з’єднані одним краєм.

Якщо граф має маршрут, що з’єднує вершини a1 і a2, то їх називають зв’язаними.

Графічні операції можна класифікувати на основі введення даних: бінарні операції, які будують нові графи з двох вхідних графів G1 (V1, E1) і G2 (V2, E2), є унарними операціями.

Операції, які не є поширеними:

— граф ребер: ребра неорієнтованого графа G зображуються реберним графом L (L);

— граф G′ — це граф, всі вершини якого збігаються з гранями планарного графа G (рисунок 1.14);

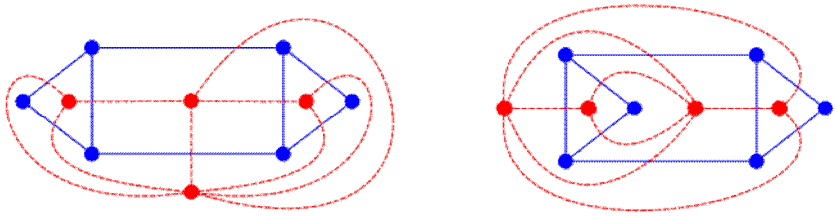

Рисунок 1.14 — Червоні графи є двоїстими до синього графа

— доповнення: граф H, який є доповненням графа G — граф на тих самих вершинах, поєднаних ребрами тоді і тільки тоді, коли вони несуміжні в G (рисунок 1.15) [4].

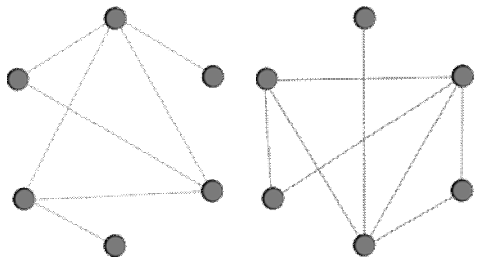

Рисунок 1.15 — Граф (зліва) і його доповнення

— невелика кількість: граф H — додатковий для даного графа G, який можна побудувати шляхом видалення ребер і вершин з G і стискання ребер;

— звуження ребра: операція, яка усуває ребро з графа та об’єднує вершини, які він раніше з’єднав, в одну;

— ступінь неорієнтованого графа (вектор): ступінь $G_k G_k$ неорієнтованого мультиграфа — це інший граф з тим же набором вершин, що й початковий граф G, і дві вершини цього графа знаходяться поблизу, якщо їх відстань у першому графі G не перевищує k [11];

До бінарних операцій належать:

— злиття графів: граф, який містить об’єднання множин вершин графа V1 і V2, а також наборів дуг [11].

— зв’язок: об’єднання двох графів, до якого додаються всі дуги, що зв’язують вершини обох графів [4, 10].

Висновки до розділу 1

Внаслідок особливостей взаємодії людини та машини, з одного боку, та семантичної неоднорідності джерел інформації з іншого, успішний пошук інформації стає дедалі складнішим. Розв’язання цієї проблеми полягає в персоналізації методів пошуку інформації, тобто в адаптації процесу пошуку до унікальних особливостей користувачів, що дозволяє швидко та з найменшими зусиллями знаходити відповідну інформацію.

У розділі розглянуто основі елементи теорії графів, які можуть бути використані при побудові системи пошуку першоджерел.

2 АНАЛІЗ ІСНУЮЧИХ АЛГОРИТМІВ ТЕОРІЇ ГРАФІВ

Пошук шляху — це процес використання комп'ютерної програми для виявлення найкоротшого маршруту між двома місцями. Для цього можна скористатися алгоритмами теорії графів.

2.1 Алгоритми пошуку найкоротших шляхів

Перед початком розробки веб-застосунку необхідно проаналізувати різноманітні існуючі алгоритми пошуку найкоротших шляхів для виявлення переваг та недоліків кожного з них і проведення подальшого аналізу.

2.1.1 Алгоритм Дейкстри

Алгоритм Дейкстри дає можливість визначити найбільш оптимальні шляхи в графі, коли перша вершина обрана спочатку, а шлях до інших невідомий. Основним недоліком цього підходу є те, що його можна використовувати лише на графах, усі ребра яких мають додатне значення. У житті бувають моменти, коли деякі бізнес-практики не приносять прибутку (загалом, від них слід відмовитися), але якщо ці практики є критичними, і немає іншого вибору, крім як їх використовувати, алгоритм Дейкстри не можна застосувати.

Процедура вимагає використання трьох наборів: V1 — набір вершин, для яких відстань була обчислена раніше, V2 — набір вершин, для яких на даний момент обчислюється відстань, і V3 — набір вершин, для яких відстань обчислюється. ще не розраховано. Алгоритм має вигляд:

1. Серед набору вершин вибирається вершина з найменшою вагою.

2. Порівнюються ваги сусідніх вершин, і для переходу вибирається сусідня вершина з найменшою вагою. Запам’ятовується, з якої вершини був здійснений перехід.

3. Крок 2 виконується для вершини, до якої було здійснено перенесення.

4. Процес зупиняється, коли навколишніх вершин більше немає, тобто набір вершин дорівнює 0.

Графічну візуалізацію алгоритму подано на рисунку 2.3.

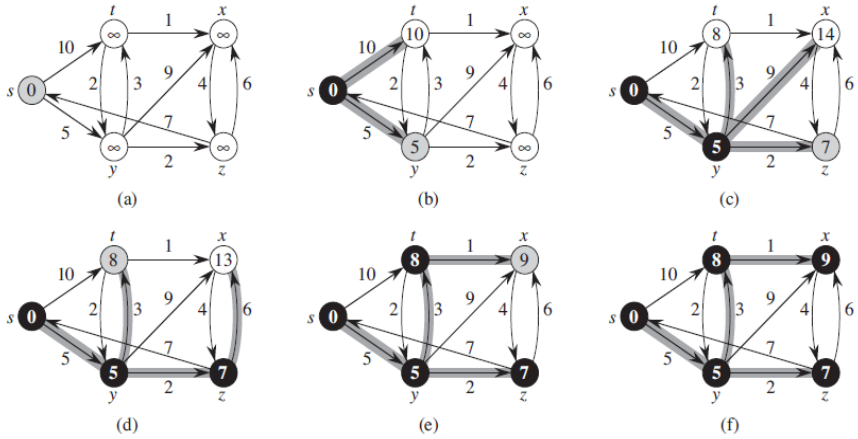

Рисунок 2.3 — Алгоритм Дейкстри

Техніка Дейкстри ідеально підходить для вирішення проблем з дорожнім покриттям, оскільки для неї потрібен графік, який як зважений (кожному краю призначається вага, наприклад, платні дороги), так і орієнтований (кожне ребро має напрямок) [12].

2.1.2 Алгоритм Беллмана-Форда

Алгоритм Беллмана-Форда подібний до алгоритму Дейкстри, але він має перевагу в тому, що він може впоратися з негативно зваженими ребрами.

Ключовою перевагою підходу динамічного програмування є його зрозумілість і комфорт (оскільки розробники звикли розбивати великі питання на менші). Опис алгоритму подано нижче:

1. Нумерація всіх вершин графу, щоб вказати, з якої вершини почнеться пошук.

2. Створюється двовимірний масив, який буде використовуватися для зберігання даних про найкоротші шляхи. Оскільки він містить інформацію про вагу ребра до вершини та номер цієї вершини, масив є двовимірним.

3. За винятком першого члена, усі члени двовимірного масиву дорівнюють нескінченності на початку процедури. Оскільки відстань між вибраним початковим джерелом і ним самим дорівнює нулю, перший рядок повністю складається з нулів.

4. Метод намагається послабити ребро на кожному кроці, що є зусиллям збільшити значення вершини за допомогою використання ребра з іншою вершиною.

5. Підраховано, що методу знадобиться n – 1 фаза, щоб визначити мінімальні ваги всіх ребер (рисунок 2.4).

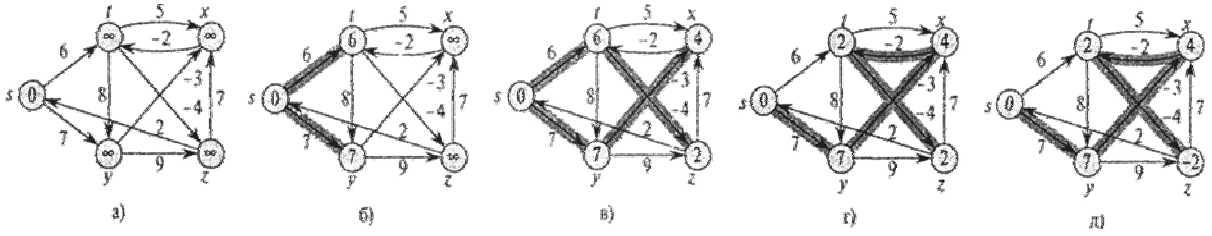

Рисунок 2.4 — Графічне подання алгоритму Беллмана-Форда

Алгоритм Беллмана-Форда використовує в своїй роботі підхід динамічного програмування (основна робота розбивається на підзадачі, для кожної підзадачі шукається рішення, а результат підзадачі застосовується до всього основного завдання).

2.1.3 Алгоритм Флойда-Уоршела

У зваженому орієнтованому графі використовується динамічний метод для визначення найбільш оптимальних шляхів серед усіх вершин. Він працює успішно, лише якщо на графіку немає циклів негативної ваги; якщо він все ще є, це дає змогу виявити принаймні один негативний цикл. Споживання пам'яті O(n^2), час роботи алгоритму O(n^3).

Розглянемо граф G = (V, E), де кожна вершина пронумерована від 1 до загальної кількості вершин.

Сформуємо матрицю суміжності D, розмір якої є квадратом розмірності вихідного графа G. Кожен член матриці суміжності позначений вагою ребра, а також сусідніми вершинами.

Ідеальна вага цього ребра постійно змінюється протягом усього алгоритму шляхом ослаблення елементів матриці суміжності: якщо $D_{ik} + D_{kj} < D_{ij}$, вага ребра D_{ij} змінюється на суму тимчасових ваг ребер $D_{ik} + D_{kj}$.

Алгоритм виконує своє завдання, коли вага всіх ребер мала і вимога $D_{ik} + D_{kj} < D_{ij}$ не виконується для кожного ребра (рисунок 2.5).

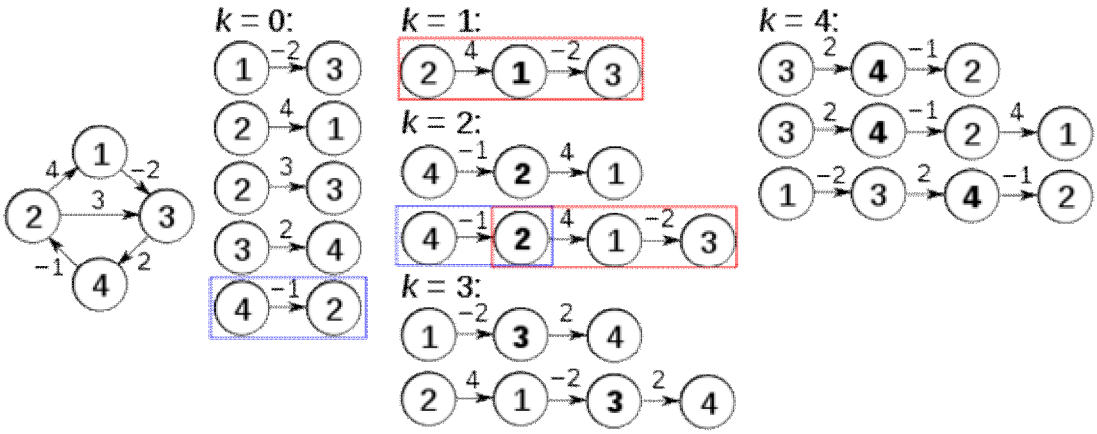

Рисунок 2.5 — Графічне представлення алгоритму Флойда-Уоршела

Отже, Метод Флойда-Уоршела є більш загальним, ніж алгоритм Дейкстри, оскільки він може працювати з негативними зацикленими мережами та виявити їх під час роботи.

2.1.4 Алгоритм A*

Фундаментальним критерієм алгоритму A* є те, що він може оцінити евристичну функцію, завжди може знайти найкращі рішення для обраної вершини, і що він не завищує відстань до цільової точки. Покроковий опис алгоритму подано нижче:

1. Необхідно розглянути всі потенційні шляхи в графі від початкової вершини до обраної вершини один за іншим, поки не буде визначено мінімум. Алгоритм спочатку оцінює найбільш ймовірні шляхи за допомогою евристичної функції.

2. Спочатку вибираються найближчі вузли з найменшим значенням функції F (n), а потім вершина, яку відвідують.

3. Метод діє на вершини, які ще не були відвідані на кожному кроці, тому вершини називаються сукупністю часткових рішень, і зазвичай вони ставлять першими в черзі.

4. Алгоритм продовжує працювати до тих пір, поки цільова функція F (n) не поверне найменше значення в черзі, або алгоритм не обходить весь графік.

Серед різноманітних варіантів вибирається рішення з найменшою вартістю. Графічна візуалізація алгоритму зображена на рисунку 2.6.

Алгоритм A* використовує модель пошуку першого найкращого збігу. Це означає, що алгоритм пошуку графа розширює найбільш перспективні вузли, визначені правилом.

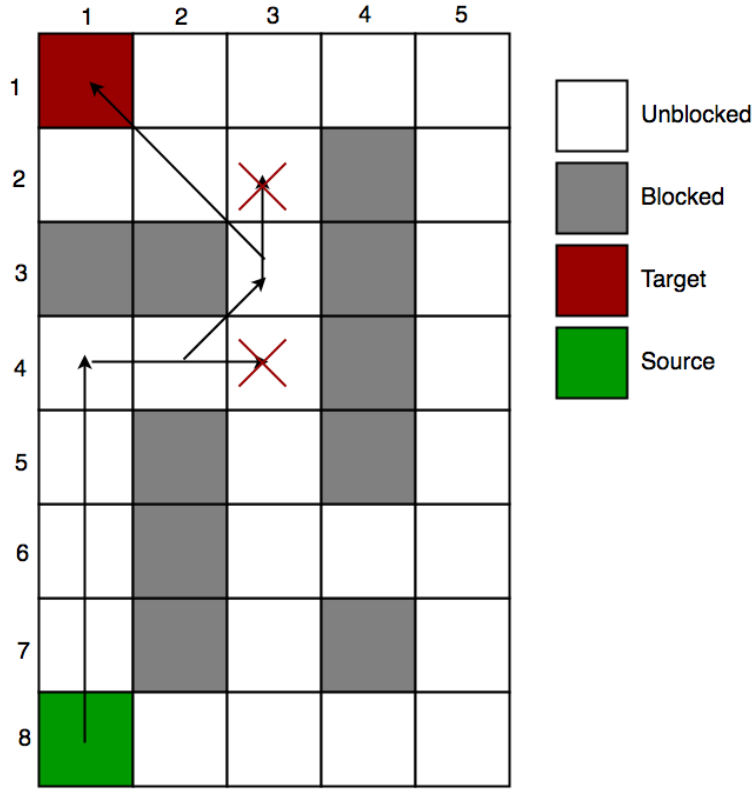

Рисунок 2.6 — Графічне представлення алгоритму A*

На кожному етапі методу відоме значення поточного шляху та оцінене значення розширення маршруту для цільової точки повинні використовуватися, щоб вибрати, який із шляхів розширювати.

2.1.5 Алгоритм D*

Алгоритм D* — це назва алгоритму, оскільки він подібний до A*, за винятком того, що витрати на дугу можуть змінюватися, оскільки елемент рухається за вказаним маршрутом. Вихід гарантовано буде оптимальним, якщо перехід між вузлами відповідним чином пов’язаний з реконструкцією оптимального маршруту [12].

Для формалізації даних використовуються наступні позначення та поняття. Набір уявлень, що вказують вузли, пов’язані спрямованими дугами, кожна з яких має відповідне значення, може бути представлена як проблемний простір.

Елемент починається з одного вузла і рухається по дугах до інших вузлів (враховуючи вартість переходу), поки не досягне вузла призначення, позначеного E. За винятком останнього вузла, кожен вузол N має зворотний вказівник до наступного вузла M, який позначається наступним чином:

$$b(N) = Mb(N) = M \tag{2.1}$$

Зворотні вказівники використовуються в алгоритмі D* для представлення шляхів до кінцевого вузла. Вартість переходу від вузла M до вузла N є цілим додатним числом, що визначається функцією значення дуги:

$$c(N) = M\, c(N) = M \tag{2.2}$$

c(N, M) не визначено, якщо M не має дуги до N. Якщо вказано c(N, M) або c(M, N), два вузли N і M є сусідніми просторами.

Метод D*, як і алгоритм A*, дозволяє відкрити список вузлів. Відкритий список використовується для повідомлення про зміни функції вартості дуги, а також для обчислення вартості маршруту до вузлів у просторі. Кожен вузол N має відповідний йому стан $t(N)$. Якщо N ніколи не було в списку відкритих вузлів, це новий вузол; якщо N є в списку відкритих вузлів, це відкритий вузол; і якщо N більше немає у списку відкритих вузлів, це закритий вузол.

Метод D* зберігає оцінку загальної вартості маршруту від N до E, визначеної функцією вартості шляху, для кожного вузла N: $h(E,N)$.

Ця оцінка порівнянна з оптимальною вартістю маршруту від вузла N до E за відповідних обставин. Для кожного вузла N у списку відкритих вузлів є ключова функція: $k(E,N)$.

Оскільки N було включено до списку відкритих вузлів, цю функцію вирішено дорівнювати мінімальному $h(E,N)$ до наступних змін, і всі значення вважаються прийнятними h(E, N). Ключова функція ділить відкриті вузли N на

два типи: стани високого значення $k(E,N) < h(E,N)$ і стани низького значення $k(E,N) = h(E,N)$. Вище значення у відкритому списку використовується алгоритмом D* для поширення інформації про зростання вартості шляху, тоді як стан нижчого значення використовується для поширення інформації про зниження вартості шляху.

Інформація поширюється шляхом постійного видалення вузлів зі списку відкритих вузлів. Коли вузол видаляється зі списку, він збільшується, щоб дозволити надсилання змін значень сусідам. Потім ці сусіди додаються до списку відкритих вузлів, що дозволяє продовжити процес.

Значення функції ключа використовується для впорядкування вузлів у списку відкритих вузлів. Параметр k_{min} описується наступним чином: $min(K(N))$.

Формула справедлива для всіх N елементів у відкритому списку.

У методі D* параметр k_{min} позначає істотну межу: якщо маршрут менший або дорівнює k_{min}, він є оптимальним; якщо значення перевищує kmmin, воно може бути неоптимальним. Перш ніж остаточний вузол буде видалено зі списку відкритих вузлів, kmin визначає параметр k_{old}. k_{old} не вказується, якщо жоден вузол не було стерто.

Якщо вузол упорядковано в ряді, його називають упорядкованим якщо:

$$b(N_i + 1) = N_i b(N_i + 1) = N_i \quad (2.3)$$

Формула (2.3) справедлива для всіх і таких, що $1 < i < N$.

Таким чином, послідовність визначає шлях зворотних покажчиків від X_n до X_1. Алгоритм D* будує та підтримує монотонну послідовність $\{E,N\}$, яка представляє зменшення поточних або нижчих витрат на шлях. для кожного вузла N, який є або був у списку відкритих вузлів.

Нижче подано детальний опис алгоритму D*:

Крок 1. Усі вузли в сітці позначаються як нові, що дозволяє обробляти кожен вузол, як тільки він стає доступним. Вартість кінцевого вузла встановлюється на нуль, а кінцевий вузол додається до відкритого списку.

Крок 2. Перший можливий маршрут від початкового до кінцевого вузла створюється циклом. Тут використовується функція пошуку шляху, яка починається з вузла з найменшим значенням функції ключа.

Крок 3. Кожен із сусідів поточного вузла перевіряється, і вибирається наступний прийнятний вузол.

Крок 4. Створіть маршрут за допомогою перевернутих покажчиків, щоб зв'язати оброблені прийнятні вузли. Якщо маршрут знаходить необхідну перешкоду, ця стратегія завжди дає змогу повернутися до будь-якого попереднього вузла.

Крок 5. Після побудови першого правильного маршруту предмет починає подорожувати. Під час руху цикл регулярно перевіряє поточну вартість перевезення через сусідів. Якщо під час цієї перевірки буде виявлено перешкоду або оптимальний маршрут руху, вихідний створений шлях буде відкоригований відповідною функцією.

Крок 6: початкова вартість переміщення між вузлами с (N, M) вказується для запису зміни вартості маршруту, що дозволяє завжди мати найновішу достовірну інформацію про вартість переходу між вузлами.

2.1.6 Алгоритм Theta*

Метод Theta* є дійсним (знаходить лише шляхи від початкової вершини до цільової вершини, які не мають повного блокування шляху) і повним (знаходить лише шляхи від початкової вершини до вершини призначення, які не мають повного блокування шляху). Існує безперешкодний маршрут між двома вершинами, якщо є незаблокований шлях між тими самими двома вершинами вздовж сітки. Якщо між тими самими двома вершинами існує незаблокований маршрут [$s_0 s_0$,...,$s_n s_n$], між ними є шлях без перешкод.

Розглянемо будь-який відрізок $s_k s_{k+1}$ ділянки цього маршруту. Розглянемо безперешкодний маршрут сітки від вершини s_k до вершини s_{k+1}, який відповідає сегменту шляху, незалежно від того, є сегмент шляху горизонтальним чи вертикальним.

Також, розглянемо послідовність розблокованих вузлів (b_0,...,b_m), через які проходить частина маршруту. Оскільки вузли розділені ребром або розташовані по діагоналі, будь-які дві послідовні клітинки b_j і b_{j+1} мають принаймні одну вершину s_{j+1}. (Необхідно вибрати будь-яку з вершин, якщо у них більше однієї). Розглянемо маршрут сітки [$s_0 = s_k$, s_1,..., s_{m+1}= s_{k+1}].

Оскільки будь-які дві послідовні вершини на ньому є кутами однієї і тієї ж незаблокованої клітинки і, отже, видимими сусідами, цей маршрут сітки від вершини s_k до вершини s_{k+1} не перешкоджає.

Отримуємо безперешкодний маршрут до сітки від вершини s_0 до вершини s_n, повторюючи цю техніку для кожної ділянки шляху та агрегуючи результати. (Усі, крім однієї вершини в маршруті сітки, можуть бути видалені, якщо багато послідовних вершин подібні) [13].

Під час виконання методу Theta* безперешкодний маршрут від початкової вершини до цієї вершини в протилежному напрямку може бути знайдений шляхом проходження раніше зв’язаних вузлів шляху від будь-якої вершини у відкритих або закритих списках до вихідної вершини. В результаті цього правила кожна попередня вершина у відкритих або закритих списках також присутня у відкритих або закритих списках.

Оскільки перша вершина є єдиною вершиною у відкритих або закритих списках, це твердження спочатку вірне. Однак, коли вершина змінює свою попередню вершину або належність до відкритих або закритих списків, оператор продовжує виконуватися. Вершина стає членом відкритого або закритого списку, щойно вона додається до неї.

Тільки коли метод Theta* оновлює деякі вершини s, а також значення значення маршруту g і попередній елемент неоновленого видимого сусіда s, вершина може стати членом відкритих або закритих списків. Відповідно до припущення, вершина s знаходиться в закритому списку, а її попередній елемент — у відкритому або закритому списку. Слідом за попередніми елементами від вершини s (або її попередніх елементів) до вихідної вершини ми отримуємо безперешкодний маршрут у зворотному напрямку від початкової вершини до вершини s (або її попередньої відповідно).

Якщо метод Theta* оновлює вершину s уздовж одного маршруту, твердження залишається істинним, оскільки вершина s та її сусід по шляху видимі один одному, і тому сегмент шляху між цими вершинами розблокований.

Твердження виконується, навіть якщо алгоритм Theta* оновлює вершину s іншим маршрутом, оскільки метод Theta* спеціально гарантує, що шлях від попередньої вершини до вершини s не блокується. Немає додаткових опцій для зміни попередньої вершини.

Кожна ітерація алгоритму Theta* змінює одну вершину у відкритому списку, якщо алгоритм закінчується. У процесі він видаляє вершину з відкритого списку, і її ніколи не можна буде повторно вставити у відкритий список. Відкритий список в кінцевому підсумку стає порожнім, оскільки кількість вершин скінченна, і метод Theta* повинен зупинитися, якщо він цього ще не зробив.

Оскільки попередня вершина має бути або видимим сусідом вершини, або попередником видимого сусіда, що не обов’язково має місце для фактичних найкоротших шляхів, метод Theta* не є оптимальним (тобто він не гарантує місцезнаходження справжнього найкоротшого шляхи).

Деталізований алгоритм Theta* представлений нижче:

Крок 1. Спочатку формуються два списки, один відкритий і один закритий, як в алгоритмі A*. Закритий список зберігає всі вузли, які недоступні для переміщення, тоді як відкритий список зберігає всі вузли, які є. У закритому

списку спочатку немає жодних елементів. Перший вузол додається до списку відкритих вузлів.

Крок 2: Визначаємо довжину маршруту f від першого до останнього елемента. Оскільки елемент не може повернутися до свого початкового розташування після початку руху, значення початкового вузла встановлюється на нуль.

Крок 3: Запускається цикл, який обробляє всі елементи відкритого списку, доки він не стане порожнім. Пункт відкритого списку з найнижчою вартістю перетворення є першим контрольним пунктом у циклі. Оскільки додаткові елементи не були перевірені під час першої ітерації циклу, таким елементом є початковий вузол.

Крок 4: Під час циклу перевіряється, чи досягнуто кінцевого вузла. У цій перевірці поточний вузол у циклі порівнюється з кінцевим. Якщо вузли збігаються, то алгоритм перестає працювати і виконується функція створення всього маршруту. Кожен вузол, пройдений від кінцевого до початкового, зберігається функцією побудови всього маршруту, що дозволяє відновити весь пройдений шлях.

Крок 5: Якщо кінцевий вузол не досягнуто, поточний перевірений вузол видаляється зі списку відкритих вузлів і додається до списку закритих вузлів.

Ця дія дозволяє алгоритму продовжувати рухатися вперед без необхідності повторно перевіряти вузли, які він раніше пройшов.

2.1.7 Обґрунтування вибору алгоритму пошуку для реалізації

Для реалізації алгоритмів та використання їх у практичних завданнях насамперед необхідно виявити їхні сильні та слабкі сторони. Для цього були проаналізовані алгоритми побудови найкоротших маршрутів та виділені основні

переваги та недоліки розуміння того, яке завдання використовувати той чи інший алгоритм.

Наприклад, при розробці системи для пошуку першоджерел найкращий алгоритм вже буде відомий. Результати проведеного аналізу подано в таблиці 2.1.

Таблиця 2.1 — Переваги та недоліки алгоритмів побудови найкоротших шляхів

Алгоритм	Переваги	Недоліки
Алгоритм Дейкстри	Використовується в багатьох картографічних сервісах.	Ребра тільки з позитивним значенням. Не завжди знаходить оптимальний маршрут.
Алгоритм Белмана-Форда	Можливість роботи з ребрами з негативним значенням.	Не працює з графами, які мають цикли негативного значення. Повільніший за алгоритм Дейкстри.
Алгоритм Флойда-Уоршела	За один крок знаходить відстань найкоротшого шляху між усіма парами по вертикалі.	Не може працювати з графами з негативними циклами. Повільний час виконання.
Алгоритм A*	Знаходить відповідь, якщо вона існує. Головна мета дістатися до обраної вершини, а не до усіх інших.	Не є оптимальним при використанні неоптимальної евристичної функції

У результаті проведення порівняльння було виявлено, що алгоритм Дейсктри має переваги. Крім того, він використовується в більшості картографічних інтернет-сервісів, отже, може бути використаний при розробці веб-застосунку.

2.2 Силові алгоритми візуалізації графів

Естетично красиві методи візуалізації графіків відомі як алгоритми візуалізації графа потужності. Їхня мета — розмістити вузли графа в двовимірному або тривимірному просторі так, щоб усі ребра були приблизно однакової довжини, і зменшити кількість перетину ребер, прикладаючи зусилля до кількох ребер і вузлів на основі їх взаємного розташування, а потім використання цих сил для моделювання руху ребер і вузлів або для зменшення їх енергії.

Хоча візуалізація графів може бути складною, силові алгоритми, як і фізичні моделі, зазвичай не потребують глибокого розуміння теорії графів, наприклад планарності графа.

Для візуалізації графа потужні алгоритми додають сили до ребер і вузлів графа. Сили тяжіння, подібні до пружини, часто використовуються для приписування сил парам ребер графа на основі закону Гука. Використовуйте сили відштовхування, аналогічні відштовхуванню електричних зарядів на основі закону Кулона, щоб розділити всі пари вузлів.

Ребра мають тенденцію набувати правильну довжину (завдяки дії пружин), щоб досягти стану рівноваги цієї системи сил, тоді як вузли, які не з’єднані ребром, мають тенденцію віддалятися один від одного (завдяки дії сили відштовхування).

Використовуючи функції, які не залежать від реальної поведінки пружин і частинок, можна розрахувати сили притягання (ребра) і сили відштовхування

(вузли). Деякі системи живлення, наприклад, використовують пружини, зусилля яких змінюються логарифмічно, а не лінійно.

Інші сили, крім механічних пружин і сил відштовхування заряду, можуть використовуватися в графіку потужності.

Силу, порівнянну з гравітацією, можна застосувати, щоб витягнути вершини в просторі для малювання графа до фіксованої точки. Це може бути використано для об'єднання багатьох компонентів зв'язку некогерентного графа в одне ціле, які в іншому випадку розійшлися б один від одного через сили відштовхування. Це також полегшує орієнтування вузлів на діаграмі [14].

Відстань між вершинами одного компонента з'єднання також може вплинути.

Для орієнтованих графіків можна використовувати аналоги магнітного поля. І краї, і вузли можна відштовхувати, щоб запобігти перекриття або майже перекриття в остаточному компонуванні.

Сили можна розташувати в контрольних точках вигнутих країв, таких як дуги кіл або сплайни, щоб збільшити кутову роздільну здатність на зображеннях із вигнутими краями, наприклад, дуги кіл або сплайнів.

На рисунку 2.7 подано візуалізацію соціальної мережі за допомогою силового алгоритму візуалізації графа.

На рисунку 2.8 подано візуалізацію візуаліцію зв'язків сторінок у Вікіпедії за допомогою силового алгоритму візуалізації розміщення.

Істотними перевагами є вказані нижче якості силових алгоритмів.

Високоякісні результати. Результати, як правило, мають надзвичайно прийнятні малюнки графів за такими критеріями: однорідність довжин ребер, рівномірний розподіл вершин та симетрія, принаймні для графів середнього розміру (до 50-500 вершин). В інших алгоритмах остання умова є найбільш важливою і її важче задовольнити.

Адаптивність. Додаткові естетичні критерії можна просто змінити та покращити за допомогою силових алгоритмів. В результаті алгоритми стають

більш адаптованими. Прикладами поточних розширень є алгоритми орієнтованих графів, відображення тривимірних графіків, кластерна візуалізація графіків, візуалізація графіків з обмеженнями та динамічна візуалізація графіків.

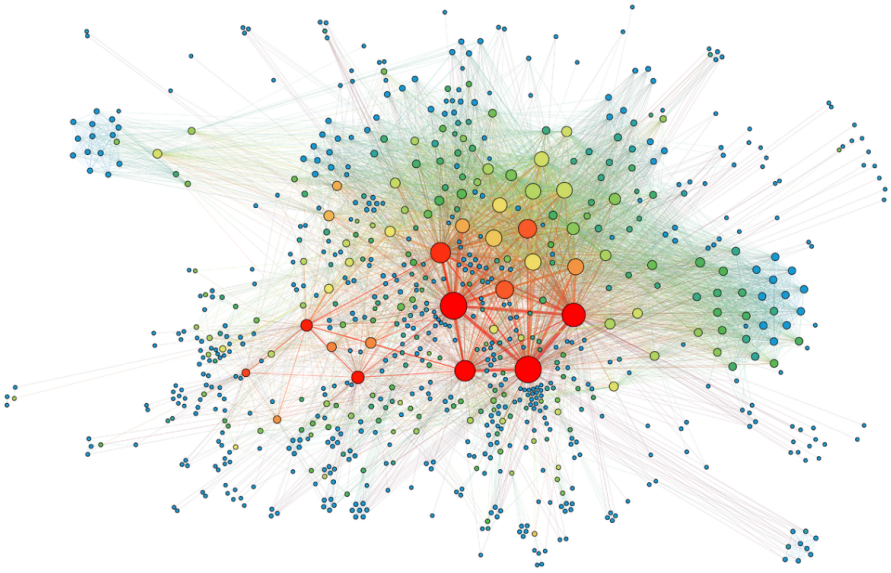

Рисунок 2.7 — Візуалізація соціальної мережі з моделюванням графів [15]

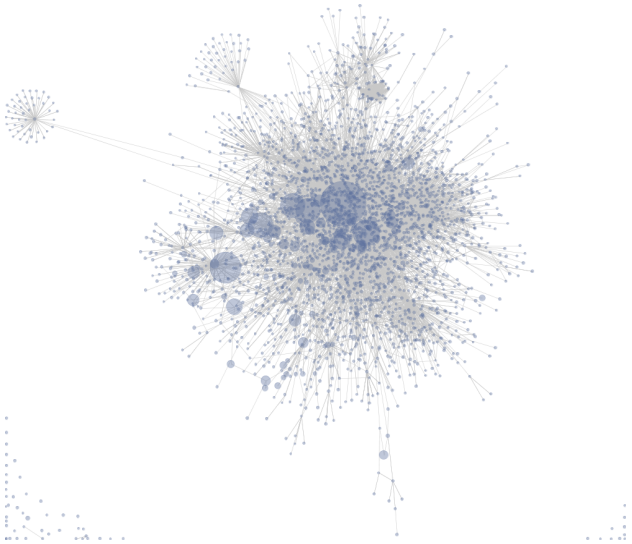

Рисунок 2.8 — Візуалізація зв’язків сторінок у Вікіпедії за допомогою силового алгоритму візуалізації розміщення [15]

Чутливість. Поведінку алгоритму дуже легко передбачити та зрозуміти, оскільки вони засновані на фізичних аналогах звичайних елементів, таких як пружини. Інші методи малювання графіків не мають цієї проблеми.

Простота використання. Алгоритми потужності часто прості і можуть бути реалізовані в кілька рядків коду.

Інші типи алгоритмів візуалізації, такі як алгоритми ортогонального позиціонування, потребують набагато більше зусиль.

Висновки до розділу 2

Для правильної реалізації алгоритмів та використання їх у практичних завданнях насамперед необхідно виявити їх сильні та слабкі сторони. Для цього були проаналізовані алгоритми побудови найкоротших маршрутів та виділені основні переваги та недоліки розуміння того, яке завдання використовувати той чи інший алгоритм.

У розділі були розглянуті основні алгоритми теорії графів, які можна використати при побудові системи пошуку першоджерел.

3 ЗАСОБИ РОЗРОБКИ СИСТЕМИ

Для розробки системи обрано такі інструменти розробки, як мова програмування JavaScript, мова розмітки гіпертексту HTML та ряд бібліотек, переваги яких описано нижче.

3.1 Основі засоби розробки системної архітектури

Клієнтський JavaScript, багаторазовий API та попередньо встановлена розмітка становлять основу сучасної архітектури веб-розробки [16]. Отриманий дизайн поділяє функціональні обов'язки на три основні категорії:

— API (Application Program Interface) використовуються для операцій із стороною сервера, наприклад сторонніх служб або спеціальних функцій;

— JavaScript обробляє всі динамічні функції;

— розмітка веб-сайту подається у вигляді файлів HTML (мова розмітки гіпертексту), які можуть бути створені з використанням генераторів статичних сайтів із вихідних файлів.

Веб-системи, розроблені за допомогою JAMStack отримують переваги від методології у таких напрямках:

— швидше виконання за рахунок попередньо позначеної розмітки та активів, переданих через CDN (мережа доставки контенту);

— покращена безпека, оскільки прямі підключення до серверів або баз даних більше не використовуються;

— краща масштабованість у разі поширення веб-системи за рахунок компенсації CDN;

— нижча вартість веб-сховища. Розміщення файлів статичної розмітки коштує дуже недорого; веб-сайти також можна обробляти без централізованого

адміністрування за допомогою системи керування вмістом, зазвичай відомої як CMS (система керування вмістом).

На рисунку 3.1 зображено абстрактний процес JAMStack.

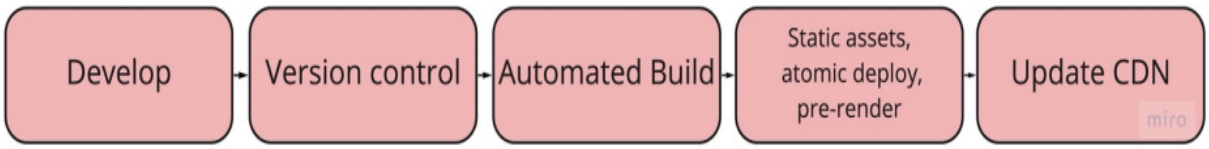

Рисунок 3.1 — Архітектура JAMStack для створення веб-додатків [17]

Мова розмітки гіпертексту HTML є основою будь-якого веб-сайту. Мова HTML — це мова розмітки, яка використовується для опису та організації вмісту веб-сайту.

Каскадні таблиці стилів (CSS) — це простий підхід до створення стилів веб-сайту (наприклад, додавання шрифтів, кольорів, макетів тощо).

Під час розробки використовувався процес розробки інтерфейсу BEM (Block Element Modifier).

При додаванны функціональності до веб-сайту мова програмування JavaScript має вирішальне значення. Онлайн-інтерфейс користувача системи рекомендацій створено за допомогою HTML, CSS та JavaScript з використанням фрейморку ReactJS:

— ReactJS — JavaScript бібліотека для побудови веб-інтерфейсів;

— StyledComponents — Бібліотека для створення та редагування графічних елементів веб-інтерфейсів;

— Three.js — кроссбраузерна бібліотека JavaScript, що використовується для створення і відображення анімованої комп’ютерної 3D графіки при розробці веб-додатків. Three.js скрипти можуть використовуватись спільно з елементом HTML5 CANVAS, SVG або WebGL.

3.2 Переваги використання React.js при створенні додатків

Програмна платформа React.js є прикладом передової технології веб-розробки [7].

На відміну від PHP, React.js не є мовою програмування; натомість це середовище з 16 виконаннями для написання додатків на стороні сервера за допомогою JavaScript. Нижче наведено деякі з основних переваг використання React.js [8]:

— швидке серверне рішення середовище React.js дає можливість використовувати чергу подій JavaScript для розробки неблокуючих програм вводу-виводу, які можуть обробляти численні запити. водночас, що є критичним для майбутнього REST-сервера, який отримає дуже велику кількість клієнтських запитів;

— фронт-енд і серверний код написані однією мовою. Ви можете отримати всі переваги мови програмування JavaScript як на стороні клієнта, так і на стороні сервера, використовуючи React.js на сервері;

— адаптивність. React.js не має жорстких обмежень чи вимог, що дозволяє створювати широкий спектр додатків. Архітектуру залежностей вибирають самі розробники;

— доступ до величезної кількості сторонніх бібліотек і готових до використання модулів. Менеджер пакетів npm у програмній платформі React.js є багатофункціональним і простим;

— необхідно використовувати формат JSON.

Це найбільш поширений і найзручніший формат для обміну даними через Інтернет. Проте у цього налаштування є ряд недоліків, зокрема:

— високий вхідний бар’єр;

— незручно використовувати для великих обчислень;

— виникають проблема з хостингом.

3.3 Переваги використання бібліотеки Three.js

Служба розробки матиме окреме вікно, де користувач може оглянути завантажену модель, щоб переглянути її ергономіку та дизайн, а також оцінити структури мультиграфів для пошуку першоджерел.

Бібліотека Three.js [11], пакет JavaScript, який дає змогу працювати з 3D-візуалами під час розробки онлайн-додатків, робить цю роботу можливою.

Нижче подано основні переваги Three.js:

— доступність і простота використання;

— можливість використовувати JavaScript для відображення та зміни тривимірних візуальних зображень на веб-сайтах;

— додавання та видалення 3D-об’єктів у режимі реального часу;

— це багатоплатформна бібліотека;

— 3D-модель можна побачити на будь-якій платформі, оскільки вона об’єднує їх при перегляді;

— можливість роботи з елементом інтерфейсу canvas, що дозволяє побачити 3D-модель на різноманітних мобільних пристроях;

— дає можливість створити тривимірний об’єкт для відображення в браузері повністю з нуля — тільки використовуючи аналітичну геометрію.

Бібліотека Three.js включає такі компоненти:

— Scene — своєрідна платформа, де розміщуються всі створені користувачем об'єкти;

— Camera — знімає і відображає об’єкти, розташовані на сцені;

— Renderer – візуалізатор, який дає можлмвість показувати сцену камери.

Недоліками бібліотеки Three.js є:

— щоб показати тривимірні об’єкти в браузері, необхідно спочатку розробити сцену, камеру та візуалізатор, які не є очевидними для звичайного розробника;

— підтримуються лише сучасні браузери.

Архітектурна структура Three.Js зображена на рисунку 3.2.

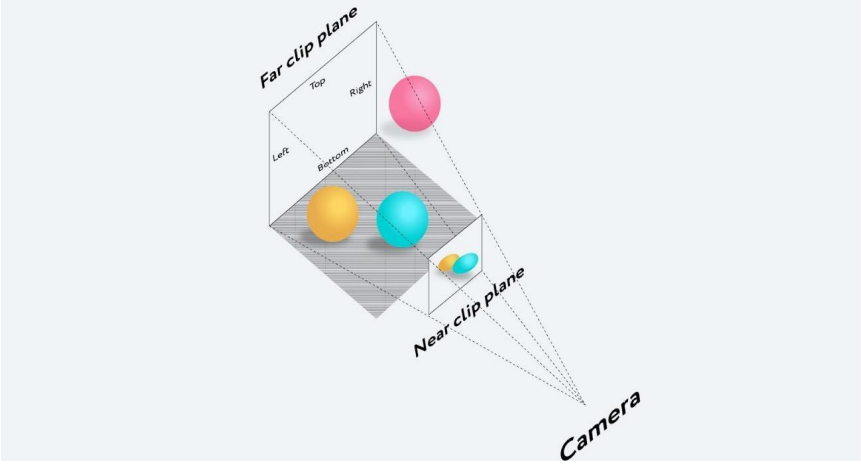

Рисунок 3.2 — Архітектурна структура Three.Js

Бібліотека Three.js дає можливість створювати прискорену на GPU 3D-графіку, використовуючи мову JavaScript як частину сайту без підключення пропрієтарних плагінів для браузера. Це можливо завдяки використанню технології WebGL.

Висновки до розділу 3

При додаванні функціональності до веб-сайту мова програмування JavaScript має вирішальне значення. Онлайн-інтерфейс користувача системи рекомендацій створено за допомогою HTML, CSS та JavaScript з використанням фреймворку ReactJS.

Бібліотека Three.js — кроссбраузерна бібліотека JavaScript, що використовується для створення і відображення анімованої комп'ютерної 3D графіки при розробці веб-додатків. Three.js скрипти можуть використовуватись спільно з елементом HTML5 CANVAS, SVG або WebGL.

У розділі розглянуто основні засоби програмної реалізації, які повинні бути використані при побудові системи пошуку першоджерел.

4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

Для пошуку на основі методів теорії графів першоджерел в літературі шляхом побудови веб-системи використовуються вибрані раніше інструменти розробки. Програмна реалізація веб-системи пошуку інформації потребує ретельно продуманої архітектури, розділеної на окремі компоненти, між якими встановлюються зв’язки.

Діаграма класів демонструє зв’язок між модулями розробленої системи. Вона зображена на рисунку 4.1.

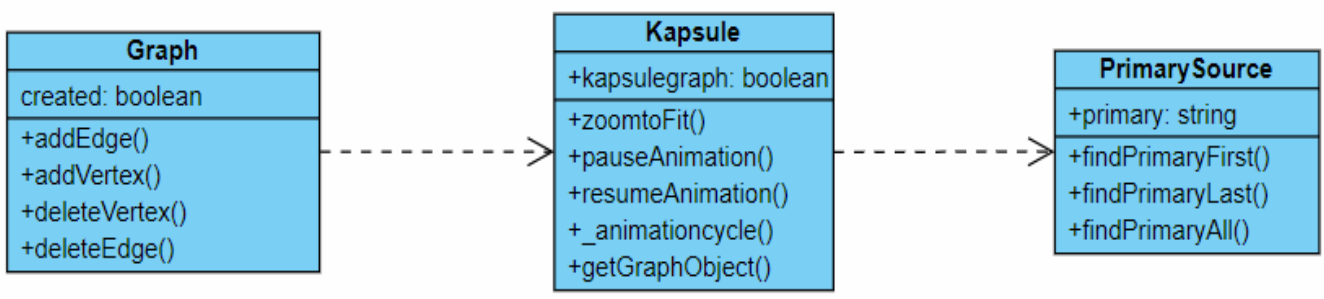

Рисунок 4.1 — Діаграма класів розробленої системи

Складовими системи є:

— Graph — клас, який відповідає за побудову структур графів, включаючи функції додавання вершин та ребер;

— Kapsule — клас, який відповідає за побудову структури першоджерел у вигляді графів;

— PrimarySource — клас, який відповідає за функції знаходження першоджерел.

Послідовність роботи створеної системи з пошуку першоджерел подано на рисунку 4.2.

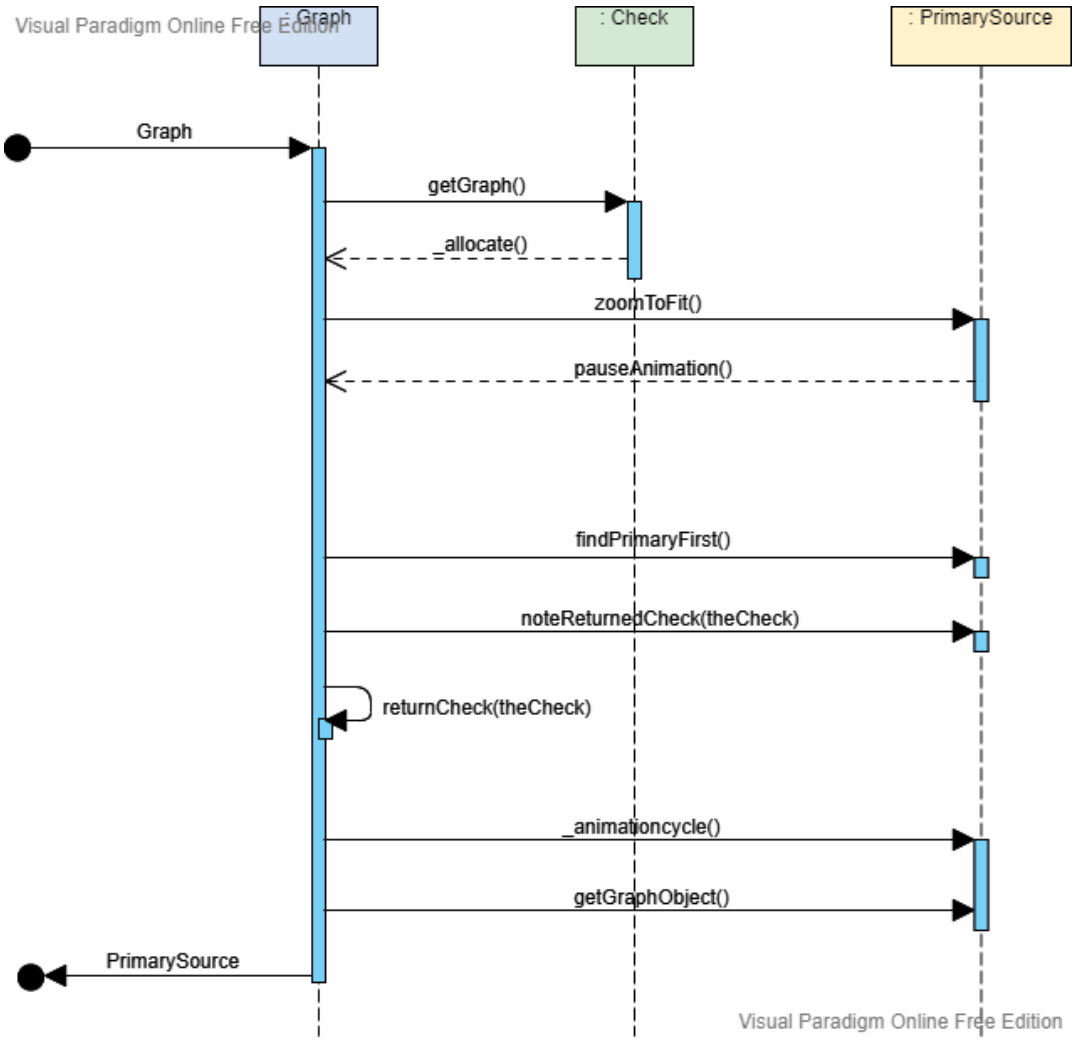

Рисунок 4.2 — Діаграма послідовності роботи системи

З рисунка 4.2 видно, що система отримує вхідний граф з вершинами, які відповідають деяким джерелам з бази даних або з файлу формату .json, обробляє вхідні дані, формуючи 3D-представлення.

Після цього, система знаходить існуючі першоджерела в базі даних і формує додаткові вершини до існуючих графів.

На рисунку 4.3 подано код функції обробки анімації відображення графів (зупинення анімації, продовження анімації).

На рисунку 4.4 подано код функції додавання першоджерел (childs) до бази даних, а також рендеринг доданих об’єктів.

```
147   pauseAnimation: function(state) {
148     if (state.animationFrameRequestId !== null) {
149       cancelAnimationFrame(state.animationFrameRequestId);
150       state.animationFrameRequestId = null;
151     }
152     return this;
153   },
154
155   resumeAnimation: function(state) {
156     if (state.animationFrameRequestId === null) {
157       this._animationCycle();
158     }
159     return this;
160   },
161   _animationCycle(state) {
162     if (state.enablePointerInteraction) {
163       // reset canvas cursor (override dragControls cursor)
164       this.renderer().domElement.style.cursor = null;
165     }
```

Рисунок 4.3 — Обробка анімацій відображення першоджерел

```
194   // Add relative container
195   domNode.appendChild(state.container = document.createElement('div'));
196   state.container.style.position = 'relative';
197
198   // Add renderObjs
199   const roDomNode = document.createElement('div');
200   state.container.appendChild(roDomNode);
201   state.renderObjs(roDomNode);
202   const camera = state.renderObjs.camera();
203   const renderer = state.renderObjs.renderer();
204   const controls = state.renderObjs.controls();
205   controls.enabled = !!state.enableNavigationControls;
206   state.lastSetCameraZ = camera.position.z;
207
208   // Add info space
209   let infoElem;
210   state.container.appendChild(infoElem = document.createElement('div'));
211   infoElem.className = 'graph-info-msg';
212   infoElem.textContent = '';
```

Рисунок 4.4 — Додавання першоджерел і рендеринг об’єктів

Код функції формування зв’язків між батьками і нащадками подано на рисунку 4.5.

```
// config forcegraph
state.forceGraph
  .onLoading(() => { infoElem.textContent = 'Loading...' })
  .onFinishLoading(() => { infoElem.textContent = '' })
  .onUpdate(() => {
    // sync graph data structures
    state.graphData = state.forceGraph.graphData();

    // re-aim camera, if still in default position (not user modified)
    if (camera.position.x === 0 && camera.position.y === 0 && camera.position.z === state.lastSetCameraZ && st
      camera.lookAt(state.forceGraph.position);
      state.lastSetCameraZ = camera.position.z = Math.cbrt(state.graphData.nodes.length) * CAMERA_DISTANCE2NOD
    }
  })
  .onFinishUpdate(() => {
    // Setup node drag interaction
    if (state._dragControls) {
      const curNodeDrag = state.graphData.nodes.find(node => node.__initialFixedPos && !node.__disposeControls
      if (curNodeDrag) {
        curNodeDrag.__disposeControlsAfterDrag = true; // postpone previous controls disposal until drag ends
      } else {
        state._dragControls.dispose(); // cancel previous drag controls
      }

      state._dragControls = undefined;
    }
```

Рисунок 4.5 — Функція формування зв’язків між батьками та нащадками

При розробці модулів системи було враховано основні положення теорії графів і використано методи теорії графів, зокрема, алгоритм Дейкстри пошуку найкоротшого шляху, а також силовий алгоритм візуалізації розміщення графів у просторі.

Висновки до розділу 4

У розділі розглянуто діаграму класів розробленої системи, діаграма послідовності роботи системи, основні використані функції під час розробки системи з пошуку першоджерел.

5 РОБОТА КОРИСТУВАЧА З СИСТЕМОЮ ПОШУКУ ПЕРШОДЖЕРЕЛ

У розділі вказано системні й апаратні вимоги для встановлення розробленої веб-системи пошуку першоджерел, наведено приклади роботи користувача з системою.

5.1 Встановлення програмного забезпечення

Для використання розробленого програмного забезпечення необхідно провести ряд налаштувань в операційній системі користувача. Насамперед необхідно перевірити, чи відповідає пристрій вимогам для запуску та оптимальної роботи системи розпізнавання жестів.

Вимоги до програмного забезпечення такі:

— мова програмування JavaScript;

— провідник Google Chrome 85.0 і вище;

— фреймворк React.Js версії 18.2.0 і вище;

— пакетний менеджер NPM версії 8.7.0 і вище;

— операційна система Windows, Linux, MacOS.

Вимоги до апаратної частини:

— комп’ютер на базі процесора Intel Pentium і вище;

— оперативна пам’ять — не менше 512 Мбайт;

— вільний простір жорсткого диска не менше 200 Мбайт;

— доступ до Інтернету.

Оскільки в ході розробки програмного продукту використано тільки засоби з відкритим кодом, то для встановлення необхідних компонентів достатньо

перейти за посиланнями на відповідні сайти [18-20] зазначених вище програмних продуктів. Після встановлення компонентів потрібно скористатися кодом, наведеним у додатку А, і запустити його.

5.2 Робота користувача з програмним забезпеченням

Обидва актори системи як звичайний користувач, так і адміністратор, повинні дотримуватися всіх вимог роботи з користувацьким інтерфейсом додатку для коректної роботи всіх модулів системи.

На рисунку 5.1. подано діаграму варіантів використання застосунку користувачем.

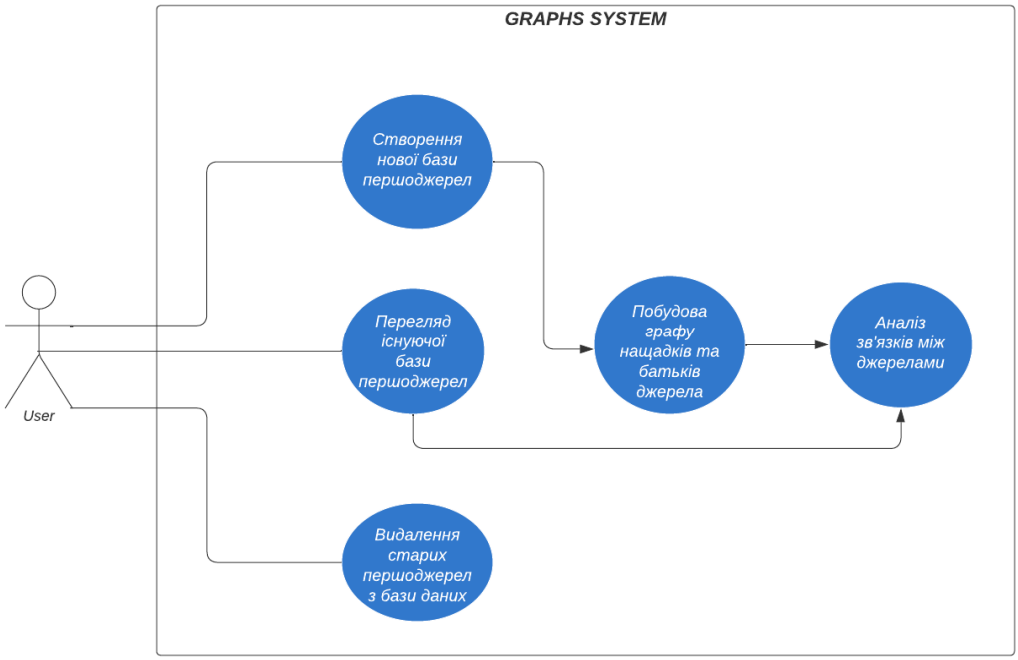

Рисунок 5.1 — Можливі варіанти використання застосунку користувачем

Веб-інтерфейс реалізовано таким чином, що на початковому екрані (рисунок 5.2) користувач може побачити форму для додавання першоджерела.

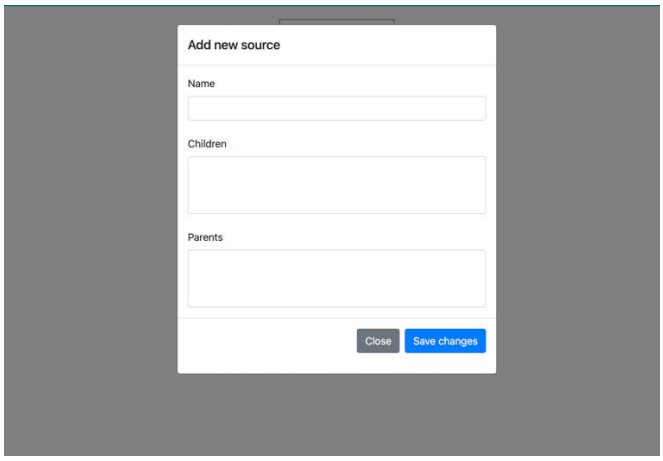

Рисунок 5.2 — Модальна форма для додавання джерела

Елементами форми для додавання джерела є:

— Name — назва джерела;

— Children — нащадки досліджуваного джерела;

— Parents — батьки досліджуваного джерела.

Для тестування роботи системи використано тестову вибірку: Book1, Book2, Book3, додану до бази даних системи через модальну форму (рисунок 5.3).

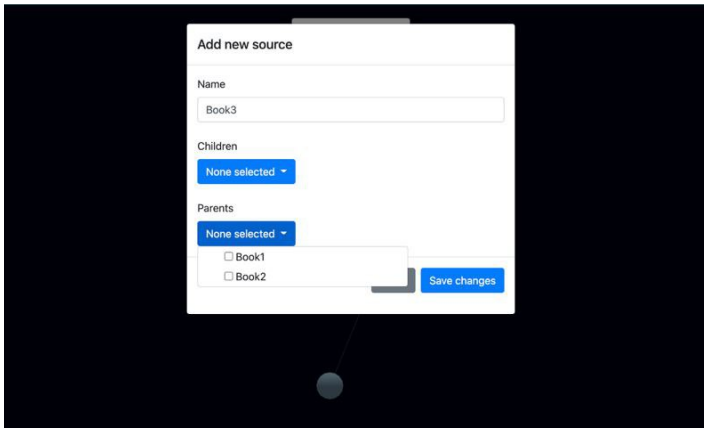

Рисунок 5.3 — Додавання джерел до бази даних системи

Після додавання трьох джерел до бази даних, система формує деревовидну структуру для графічної візуалізації даних (рисунок 5.4):

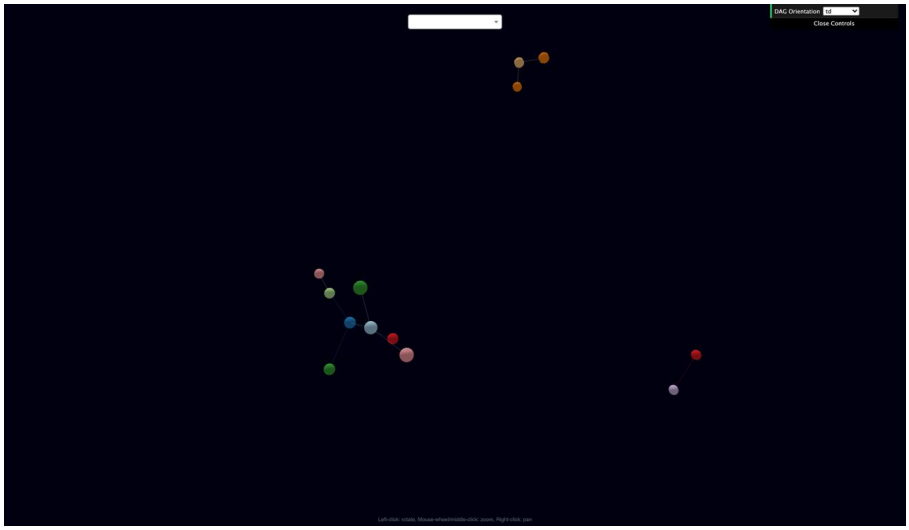

Рисунок 5.4 — Сформований шаблон подання графів

Після цього за допомогою меню користувач має можливість змінювати обране джерело та розглянути деталізовані графічні структури для виявлення нащадків та батьків, тобто використовує процедуру пошуку першоджерела (рисунки 5.5 і 5.6).

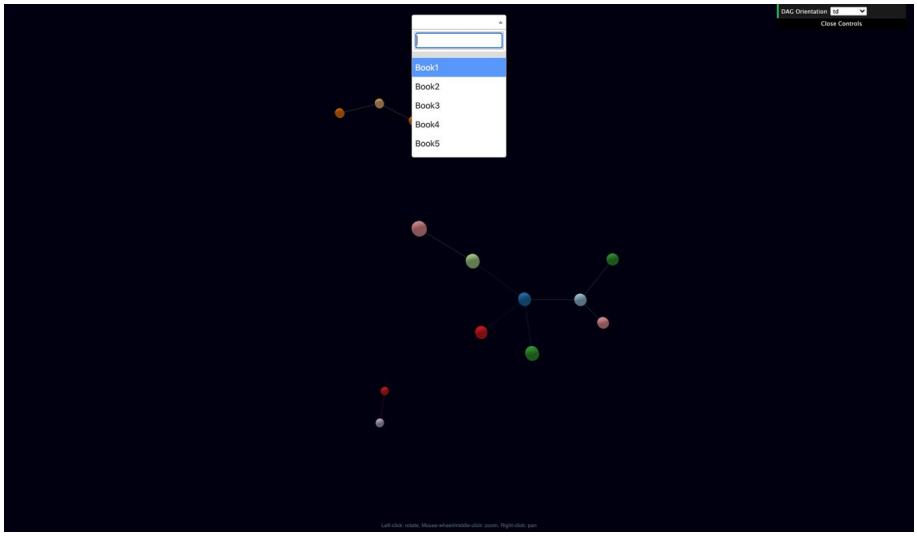

Рисунок 5.5 — Меню для зміни вибраного джерела для дослідження

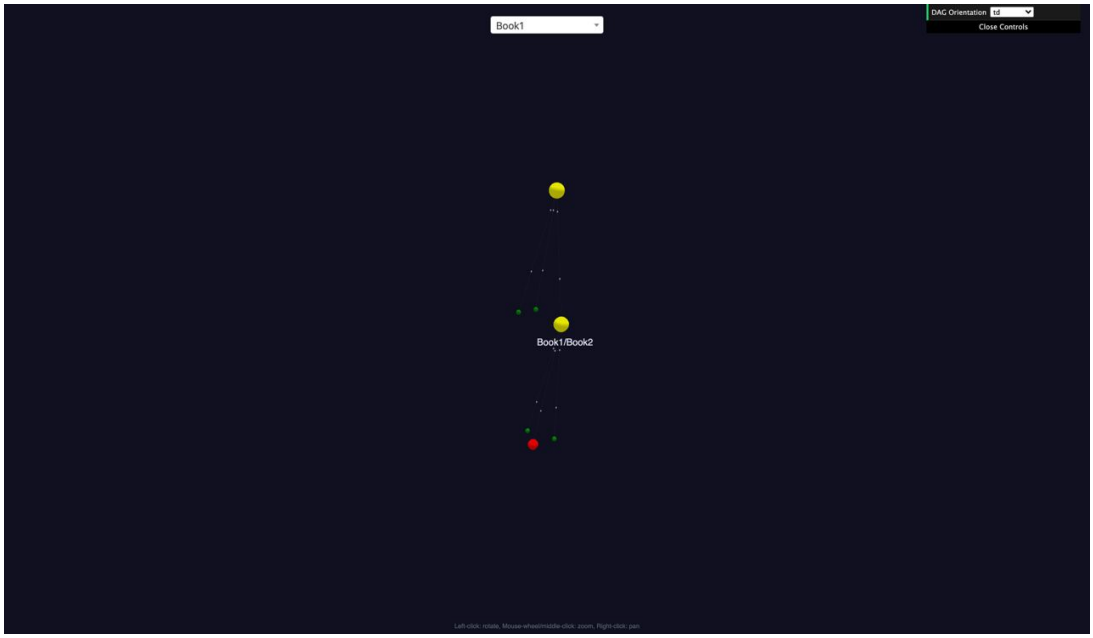

Рисунок 5.6 — Зображення нащадків для Book1/Book2

Як видно з рисунку 6.6, сформовано 3D-граф, який візуалізує першоджерело, а також його нащадків у вигляді вершин.

Висновки до розділу 5

Обидва актори системи як звичайний користувач, так і адміністратор, повинні дотримуватися всіх вимог роботи з користувацьким інтерфейсом додатку для коректної роботи всіх модулів системи.

У розділі розглянуто роботу користувача з розробленою системою пошуку першоджерел на основі теорії графів, продемонстровано розроблений веб-інтерфейс.

ВИСНОВКИ

Створено веб-систему для пошуку та аналізу першоджерел в літературі на основі методів теорії графів. Реалізовано важливу задачу — візуалізацію графів для відображення взаємозв’язків на множинах, що є додатковою перевагою з точки зору наочності всього процесу.

При створенні веб-системи виконано такі завдання:

— досліджено різноманітні існуючі системи для пошуку першоджерел на основі силових алгоритмів візуалізації графів;

— проаналізовано основні положення теорії графів і її похідних;

— проведено експерименти з різноманітними алгоритмами побудови комплексних систем з використанням графів;

— оцінено якість результатів використання різноманітних алгоритмів;

— розроблено систему пошуку першоджерел з на основі теорії графів.

У роботі проаналізовано предметну область, розглянуто різні підходи до розв’язання поставленого завдання, розглянуто існуючі аналоги.

Розроблена веб-система може бути використана в різних сферах діяльності людей, зокрема: науковцями, студентами, викладачами, діячами культури, журналістами, публіцистами та іншими.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Кучеренко Є. І., Творошенко І. С., Анопрієнко Т. В. Моделювання та оцінювання станів складних об’єктів із застосуванням формальної логіки. Системи обробки інформації. 2016. No 1. С. 76-82.

2. Labeling Algorithm for Shortest Paths on Road Networks. / Abraham I., Delling D., Goldberg A., Werneck R. Symposium on Experimental Algorithms Philadelphia. 2020. P. 230-241.

3. Dijkstra E. A note on two problems in connection with graphs. Numerische Mathematik. 2017. V. 1. P. 269-271.

4. Ford L., Fulkerson D. Flows in Networks. Princeton: Princeton University Press. 2016. 253 p.

5. MJT — Contraction hierarchies path finding algorithm. URL: https://www.mjt.me.uk/posts/contraction-hierarchies/ (дата звернення: 01.06.2022)

6. Modeling the Structure of Intellectual Means of DecisionMaking Using a System-Oriented NFO Approach / M. Ayaz Ahmad, Irina Tvoroshenko, Jalal Hasan Baker, Vyacheslav Lyashenko. International Journal of Emerging Trends in Engineering Research. 2019. No 11. P. 460-465.

7. Methods of Using Fuzzy Interval Logic During Processing of Space States of Complex Biophysical Objects / Lyashenko V., Mustafa S.K., Tvoroshenko I., Ahmad M.A. International Journal of Emerging Trends in Engineering Research. 2020. Vol. 8, No2. P. 372-377.

8. Daradkeh Y.I., Tvoroshenko I. Technologies for Making Reliable Decisions on a Variety of Effective Factors using Fuzzy Logic. International Journal of Advanced Computer Science and Applications. 2020. Vol. 11, No 5. P. 43-50.

9. Tvoroshenko I.S., Gorokhovatsky V.O. Effective tuning of membership function parameters in fuzzy systems based on multi-valued interval logic. Telecommunications and Radio Engineering. 2020. Vol. 79, No 2. P. 149-163.

10. Gorokhovatskyi V.O., Tvoroshenko I.S., Peredrii O.O. Image classification method modification based on model of logic processing of bit description weights vector. Telecommunications and Radio Engineering. Vol. 79, No 1. 2020. P. 59-69.

11. Gorokhovatskyi V., Tvoroshenko I. Image Classification Based on the Kohonen Network and the Data Space Modification. In CEUR Workshop Proceedings: Computer Modeling and Intelligent Systems (CMIS-2020). 2020. No 1. P. 1013-1026.

12. Gorokhovatskyi V.O., Tvoroshenko I.S., Vlasenko N.V. Using fuzzy clustering in structural methods of image classification. Telecommunications and Radio Engineering. Vol. 79, No 9. 2020. P. 781-791.

13. The application of non-parametric statistics methods in image classifiers based on structural description components. Kobylin O., Gorokhovatskyi V., Tvoroshenko I., Peredrii O. Telecommunications and Radio Engineering. 2020. No 1. P. 855-863.

14. Nesbit J., Adesope O. Learning With Concept and Knowledge Maps: A Meta-Analysis. *Review of Educational Research.* 2020. № 1. P. 413-448.

15. Peter Eades. A heuristic for graph drawing. *Congressus numerantium*. 2020. Vol. 42. P. 149 — 160.

16. Immerman N. Relational Queries Computable in Polynomial Time. *Proceedings of the fourteenth annual ACM symposium on Theory of computing.* 2020. Vol. 82. P. 86-104.

17. What is REST? URL: http://www.restapitutorial.com/lessons/whatisrest.html (дата звернення: 04.06.2022)

18. Markus Egger — MVVM Survival Guide for Enterprise Architectures in Silverlight and WPF. URL: https://www.packtpub.com/application-development/mvvm-survival-guide-enterprisearchitectures-silverlight-and-wpf. (дата звернення: 04.06.2022)

19. L. Boratto, S. Carta. State-of-the-art in group recommendation and new approaches for automatic identification of groups. *In Information Retrieval and Mining in Distributed Environments*. 2011. Vol. 324. P. 1–20.

20. Guha S. Rock: A robust clustering algorithm for categorical attributes. *Information Systems*. 2000. Vol. 25. № 5. P. 345–366.

ДОДАТОК А

Засоби пошуку першоджерел в літературі на основі методів теорії графів

Текст програмного модуля

УКР.НТУУ“КПІ ім. Ігоря Сікорського”.ТР81350_22Б 12-1

Аркушів 13

Київ — 2022

Вихідний код для формування структури графу для першоджерела

```
import { AmbientLight, DirectionalLight, Vector3 } from 'three';

const three = window.THREE
  ? window.THREE // Prefer consumption from global THREE, if exists
  : { AmbientLight, DirectionalLight, Vector3 };

import { DragControls as ThreeDragControls } from
'three/examples/jsm/controls/DragControls.js';

import ThreeForceGraph from 'three-forcegraph';
import ThreeRenderObjects from 'three-render-objects';

import accessorFn from 'accessor-fn';
import Kapsule from 'kapsule';

import linkKapsule from './kapsule-link.js';

//

const CAMERA_DISTANCE2NODES_FACTOR = 170;

//

// Expose config from forceGraph
const bindFG = linkKapsule('forceGraph', ThreeForceGraph);
const linkedFGProps = Object.assign(...[
  'jsonUrl',
  'graphData',
  'numDimensions',
  'dagMode',
```

’dagLevelDistance’,
’dagNodeFilter’,
’onDagError’,
’nodeRelSize’,
’nodeId’,
’nodeVal’,
’nodeResolution’,
’nodeColor’,
’nodeAutoColorBy’,
’nodeOpacity’,
’nodeVisibility’,
’nodeThreeObject’,
’nodeThreeObjectExtend’,
’linkSource’,
’linkTarget’,
’linkVisibility’,
’linkColor’,
’linkAutoColorBy’,
’linkOpacity’,
’linkWidth’,
’linkResolution’,
’linkCurvature’,
’linkCurveRotation’,
’linkMaterial’,
’linkThreeObject’,
’linkThreeObjectExtend’,
’linkPositionUpdate’,
’linkDirectionalArrowLength’,
’linkDirectionalArrowColor’,
’linkDirectionalArrowRelPos’,
’linkDirectionalArrowResolution’,
’linkDirectionalParticles’,
’linkDirectionalParticleSpeed’,
’linkDirectionalParticleWidth’,

```
  'linkDirectionalParticleColor',
  'linkDirectionalParticleResolution',
  'forceEngine',
  'd3AlphaDecay',
  'd3VelocityDecay',
  'd3AlphaMin',
  'ngraphPhysics',
  'warmupTicks',
  'cooldownTicks',
  'cooldownTime',
  'onEngineTick',
  'onEngineStop'
].map(p => ({ [p]: bindFG.linkProp(p)})));
const linkedFGMethods = Object.assign(...[
  'refresh',
  'getGraphBbox',
  'd3Force',
  'd3ReheatSimulation',
  'emitParticle'
].map(p => ({ [p]: bindFG.linkMethod(p)})));

// Expose config from renderObjs
const bindRenderObjs = linkKapsule('renderObjs', ThreeRenderObjects);
const linkedRenderObjsProps = Object.assign(...[
  'width',
  'height',
  'backgroundColor',
  'showNavInfo',
  'enablePointerInteraction'
].map(p => ({ [p]: bindRenderObjs.linkProp(p)})));
const linkedRenderObjsMethods = Object.assign(
  ...[
    'cameraPosition',
    'postProcessingComposer'
```

```
  ].map(p => ({ [p]: bindRenderObjs.linkMethod(p)})),
  {
   graph2ScreenCoords: bindRenderObjs.linkMethod('getScreenCoords'),
   screen2GraphCoords: bindRenderObjs.linkMethod('getSceneCoords')
  }
);

//

export default Kapsule({

  props: {
   nodeLabel: { default: 'name', triggerUpdate: false },
   linkLabel: { default: 'name', triggerUpdate: false },
   linkHoverPrecision: { default: 1, onChange: (p, state) =>
state.renderObjs.lineHoverPrecision(p), triggerUpdate: false },
   enableNavigationControls: {
    default: true,
    onChange(enable, state) {
     const controls = state.renderObjs.controls();
     if (controls) {
      controls.enabled = enable;
     }
    },
    triggerUpdate: false
   },
   enableNodeDrag: { default: true, triggerUpdate: false },
   onNodeDrag: { default: () => {}, triggerUpdate: false },
   onNodeDragEnd: { default: () => {}, triggerUpdate: false },
   onNodeClick: { triggerUpdate: false },
   onNodeRightClick: { triggerUpdate: false },
   onNodeHover: { triggerUpdate: false },
   onLinkClick: { triggerUpdate: false },
   onLinkRightClick: { triggerUpdate: false },
```

```
  onLinkHover: { triggerUpdate: false },
  onBackgroundClick: { triggerUpdate: false },
  onBackgroundRightClick: { triggerUpdate: false },
  ...linkedFGProps,
  ...linkedRenderObjsProps
},

methods: {
  zoomToFit: function(state, transitionDuration, padding, ...bboxArgs) {
    state.renderObjs.fitToBbox(
      state.forceGraph.getGraphBbox(...bboxArgs),
      transitionDuration,
      padding
    );
    return this;
  },
  pauseAnimation: function(state) {
    if (state.animationFrameRequestId !== null) {
      cancelAnimationFrame(state.animationFrameRequestId);
      state.animationFrameRequestId = null;
    }
    return this;
  },

  resumeAnimation: function(state) {
    if (state.animationFrameRequestId === null) {
      this._animationCycle();
    }
    return this;
  },
  _animationCycle(state) {
    if (state.enablePointerInteraction) {
      // reset canvas cursor (override dragControls cursor)
      this.renderer().domElement.style.cursor = null;
```

```
    }

    // Frame cycle
    state.forceGraph.tickFrame();
    state.renderObjs.tick();
    state.animationFrameRequestId = requestAnimationFrame(this._animationCycle);
  },
  scene: state => state.renderObjs.scene(), // Expose scene
  camera: state => state.renderObjs.camera(), // Expose camera
  renderer: state => state.renderObjs.renderer(), // Expose renderer
  controls: state => state.renderObjs.controls(), // Expose controls
  tbControls: state => state.renderObjs.tbControls(), // To be deprecated
  _destructor: function() {
    this.pauseAnimation();
    this.graphData({ nodes: [], links: []});
  },
  ...linkedFGMethods,
  ...linkedRenderObjsMethods
},

stateInit: ({ controlType, rendererConfig, extraRenderers }) => ({
  forceGraph: new ThreeForceGraph(),
  renderObjs: ThreeRenderObjects({ controlType, rendererConfig, extraRenderers })
}),

init: function(domNode, state) {
  // Wipe DOM
  domNode.innerHTML = '';

  // Add relative container
  domNode.appendChild(state.container = document.createElement('div'));
  state.container.style.position = 'relative';

  // Add renderObjs
```

```
        const roDomNode = document.createElement('div');
        state.container.appendChild(roDomNode);
        state.renderObjs(roDomNode);
        const camera = state.renderObjs.camera();
        const renderer = state.renderObjs.renderer();
        const controls = state.renderObjs.controls();
        controls.enabled = !!state.enableNavigationControls;
        state.lastSetCameraZ = camera.position.z;

        // Add info space
        let infoElem;
        state.container.appendChild(infoElem = document.createElement('div'));
        infoElem.className = 'graph-info-msg';
        infoElem.textContent = '';

        // config forcegraph
        state.forceGraph
          .onLoading(() => { infoElem.textContent = 'Loading...' })
          .onFinishLoading(() => { infoElem.textContent = '' })
          .onUpdate(() => {
            // sync graph data structures
            state.graphData = state.forceGraph.graphData();

            // re-aim camera, if still in default position (not user modified)
            if (camera.position.x === 0 && camera.position.y === 0 && camera.position.z ===
state.lastSetCameraZ && state.graphData.nodes.length) {
              camera.lookAt(state.forceGraph.position);
              state.lastSetCameraZ = camera.position.z = Math.cbrt(state.graphData.nodes.length) *
CAMERA_DISTANCE2NODES_FACTOR;
            }
          })
          .onFinishUpdate(() => {
            // Setup node drag interaction
            if (state._dragControls) {
```

```
            const curNodeDrag = state.graphData.nodes.find(node => node.__initialFixedPos &&
!node.__disposeControlsAfterDrag); // detect if there's a node being dragged using the existing drag
controls
            if (curNodeDrag) {
              curNodeDrag.__disposeControlsAfterDrag = true; // postpone previous controls
disposal until drag ends
            } else {
              state._dragControls.dispose(); // cancel previous drag controls
            }

            state._dragControls = undefined;
          }

          if (state.enableNodeDrag && state.enablePointerInteraction && state.forceEngine ===
'd3') { // Can't access node positions programatically in ngraph
            const dragControls = state._dragControls = new ThreeDragControls(
              state.graphData.nodes.map(node => node.__threeObj).filter(obj => obj),
              camera,
              renderer.domElement
            );

            dragControls.addEventListener('dragstart', function (event) {
              controls.enabled = false; // Disable controls while dragging

              // track drag object movement
              event.object.__initialPos = event.object.position.clone();
              event.object.__prevPos = event.object.position.clone();

              const node = getGraphObj(event.object).__data;
              !node.__initialFixedPos && (node.__initialFixedPos = {fx: node.fx, fy: node.fy, fz:
node.fz});
              !node.__initialPos && (node.__initialPos = {x: node.x, y: node.y, z: node.z});

              // lock node
```

```
    ['x', 'y', 'z'].forEach(c => node[`f${c}`] = node[c]);

    // drag cursor
    renderer.domElement.classList.add('grabbable');
  });

  dragControls.addEventListener('drag', function (event) {
    const nodeObj = getGraphObj(event.object);

    if (!event.object.hasOwnProperty('__graphObjType')) {
      // If dragging a child of the node, update the node object instead
      const initPos = event.object.__initialPos;
      const prevPos = event.object.__prevPos;
      const newPos = event.object.position;

      nodeObj.position.add(newPos.clone().sub(prevPos)); // translate node object by the motion delta

      prevPos.copy(newPos);
      newPos.copy(initPos); // reset child back to its initial position
    }

    const node = nodeObj.__data;
    const newPos = nodeObj.position;
    const translate = {x: newPos.x — node.x, y: newPos.y — node.y, z: newPos.z — node.z};
    // Move fx/fy/fz (and x/y/z) of nodes based on object new position
    ['x', 'y', 'z'].forEach(c => node[`f${c}`] = node[c] = newPos[c]);

    state.forceGraph
      .d3AlphaTarget(0.3) // keep engine running at low intensity throughout drag
      .resetCountdown();  // prevent freeze while dragging

    node.__dragged = true;
    state.onNodeDrag(node, translate);
```

```
});

dragControls.addEventListener('dragend', function (event) {
  delete(event.object.__initialPos); // remove tracking attributes
  delete(event.object.__prevPos);

  const node = getGraphObj(event.object).__data;

  // dispose previous controls if needed
  if (node.__disposeControlsAfterDrag) {
    dragControls.dispose();
    delete(node.__disposeControlsAfterDrag);
  }

  const initFixedPos = node.__initialFixedPos;
  const initPos = node.__initialPos;
  const translate = {x: initPos.x — node.x, y: initPos.y — node.y, z: initPos.z — node.z};
  if (initFixedPos) {
    ['x', 'y', 'z'].forEach(c => {
      const fc = `f${c}`;
      if (initFixedPos[fc] === undefined) {
        delete(node[fc])
      }
    });
    delete(node.__initialFixedPos);
    delete(node.__initialPos);
    if (node.__dragged) {
      delete(node.__dragged);
      state.onNodeDragEnd(node, translate);
    }
  }

  state.forceGraph
    .d3AlphaTarget(0)   // release engine low intensity
```

```
            .resetCountdown();  // let the engine readjust after releasing fixed nodes

          if (state.enableNavigationControls) {
            controls.enabled = true; // Re-enable controls
            controls.domElement && controls.domElement.ownerDocument && controls.domElement.ownerDocument.dispatchEvent(
              // simulate mouseup to ensure the controls don't take over after dragend
              new PointerEvent('pointerup', { pointerType: 'touch' })
            );
          }

          // clear cursor
          renderer.domElement.classList.remove('grabbable');
        });
      }
    });

  // config renderObjs
  state.renderObjs
    .objects([ // Populate scene
      new three.AmbientLight(0xbbbbbb),
      new three.DirectionalLight(0xffffff, 0.6),
      state.forceGraph
    ])
    .hoverOrderComparator((a, b) => {
      // Prioritize graph objects
      const aObj = getGraphObj(a);
      if (!aObj) return 1;
      const bObj = getGraphObj(b);
      if (!bObj) return -1;

      // Prioritize nodes over links
      const isNode = o => o.__graphObjType === 'node';
      return isNode(bObj) — isNode(aObj);
```

```
    })
    .tooltipContent(obj => {
      const graphObj = getGraphObj(obj);
      return graphObj ? accessorFn(state[`${graphObj.__graphObjType}Label`])(graphObj.__data) || '' : '';
    })
    .hoverDuringDrag(false)
    .onHover(obj => {
      // Update tooltip and trigger onHover events
      const hoverObj = getGraphObj(obj);

      if (hoverObj !== state.hoverObj) {
        const prevObjType = state.hoverObj ? state.hoverObj.__graphObjType : null;
        const prevObjData = state.hoverObj ? state.hoverObj.__data : null;
        const objType = hoverObj ? hoverObj.__graphObjType : null;
        const objData = hoverObj ? hoverObj.__data : null;
        if (prevObjType && prevObjType !== objType) {
          // Hover out
          const fn = state[`on${prevObjType === 'node' ? 'Node' : 'Link'}Hover`];
          fn && fn(null, prevObjData);
        }
        if (objType) {
          // Hover in
          const fn = state[`on${objType === 'node' ? 'Node' : 'Link'}Hover`];
          fn && fn(objData, prevObjType === objType ? prevObjData : null);
        }

        // set pointer if hovered object is clickable
        renderer.domElement.classList[
          ((hoverObj && state[`on${objType === 'node' ? 'Node' : 'Link'}Click`]) || (!hoverObj && state.onBackgroundClick)) ? 'add' : 'remove'
        ]('clickable');

        state.hoverObj = hoverObj;
```

```
        }
      })
      .clickAfterDrag(false)
      .onClick((obj, ev) => {
        const graphObj = getGraphObj(obj);
        if (graphObj) {
          const fn = state[`on${graphObj.__graphObjType === 'node' ? 'Node' : 'Link'}Click`];
          fn && fn(graphObj.__data, ev);
        } else {
          state.onBackgroundClick && state.onBackgroundClick(ev);
        }
      })
      .onRightClick((obj, ev) => {
        // Handle right-click events
        const graphObj = getGraphObj(obj);
        if (graphObj) {
          const fn = state[`on${graphObj.__graphObjType === 'node' ? 'Node' : 'Link'}RightClick`];
          fn && fn(graphObj.__data, ev);
        } else {
          state.onBackgroundRightClick && state.onBackgroundRightClick(ev);
        }
      });

    //

    // Kick-off renderer
    this._animationCycle();
  }
});

//

function getGraphObj(object) {
```

```
  let obj = object;
  // recurse up object chain until finding the graph object
  while (obj && !obj.hasOwnProperty('__graphObjType')) {
    obj = obj.parent;
  }
  return obj;
}
```