Міністерство освіти і науки України
Криворізький національний університет
Кафедра моделювання та програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти магістра

зі спеціальності 121 – Інженерія програмного забезпечення

На тему: Розробка системи автоматизації проектування спеціалізованих цифрових моделей рельєфу місцевості

Засвідчую, що в цій кваліфікаційній роботі немає запозичень із праць інших авторів без відповідних посилань.

Студент гр. ІПЗ-23-2м

_____/Б. О. Лисенко /

Керівник
кваліфікаційної роботи _____ / І. А. Котов /

Завідувач кафедри _____ / А. М. Стрюк /

Кривий Ріг
2024

Криворізький національний університет

Факультет: Інформаційних технологій

Кафедра: Моделювання та програмного забезпечення

Ступінь вищої освіти: магістр

Спеціальність: 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:

Зав. кафедри

_____ А. М. Стрюк

«___» _____ 20__ р.

ЗАВДАННЯ

на кваліфікаційну роботу

студенту групи ІПЗ-23-2м Лисенку Богдану Олеговичу

1. Тема: Розробка системи автоматизації проектування спеціалізованих цифрових моделей рельєфу місцевості затверджено наказом по КНУ № 6 від «29» січня 2024 р.
2. Термін подання студентом закінченої роботи: «06» грудня 2024р.
3. Вихідні дані по роботі: результатом роботи є спеціалізована геоінформаційна система пошуку маршруту для прокладання трубопроводів.
4. Зміст пояснювальної записки (перелік питань, що їх треба розробити): провести критичний аналіз літературних джерел за темою та перевірити на актуальність, спроектувати та описати спеціалізовану систему прокладання оптимальних маршрутів для прокладання труб, реалізувати функціонал програмного забезпечення, провести тестування реалізованої системи та написати інструкцію користувача.
5. Перелік ілюстративного матеріалу: зображення функціональної схеми програмного забезпечення, блок-схеми алгоритмів роботи функціоналу, зображення концептів інтерфейсу програмного застосунку, зображення коду та зображення інструкції користувача.

Календарний план:

№	Найменування етапів кваліфікаційної роботи	Термін виконання етапів роботи
1	Дослідження та аналіз вимог до програмного забезпечення	05.02.2024 – 10.03.2024
2	Аналіз існуючих рішень	11.03.2024 – 02.04.2024
3	Розробка архітектури програмного забезпечення	03.04.2024 – 21.05.2024
4	Розробка алгоритмів	22.05.2024 – 23.07.2024
5	Реалізація програмного забезпечення	24.07.2024 – 04.11.2024
6	Тестування програмного забезпечення	05.11.2024 – 24.11.2024
8	Підготовка та оформлення пояснювальної записки	25.11.2024 – 02.12.2024

Дата видачі завдання: «29» січня 2024 р.

Студент _____ / Б. О. Лисенко /

Керівник роботи _____ / І. А. Котов /

РЕФЕРАТ

РОЗРОБКА, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, ВЕБ-ЗАСТОСУНОК, ОПТИМАЛЬНИЙ МАРШРУТ, ПРОКЛАДАННЯ ТРУБОПРОВОДІВ, ПОШУК МАРШРУТУ, АЛГОРИТМ ПОШУКУ, ІНТЕРАКТИВНА МАПА, МАРКЕРИ, ДОВЖИНА МАРШРУТУ, ВІЗУАЛІЗАЦІЯ МАПИ, ФУНКЦІОНАЛЬНІСТЬ.

Пояснювальна записка: 79 с., 33 рис., 1 дод., 10 джерел.

Метою кваліфікаційної роботи є розробка спеціалізованої геоінформаційної системи для прокладання трубопроводів на прикладі міста Київ.

Об'єктом розробки є спеціалізоване програмне забезпечення для пошуку оптимального маршруту прокладання труб, враховуючи інтерфейс, функціональність системи, вибір точок, прокладання маршрутів та візуалізація маршруту на мапі.

У роботі проаналізовано вимоги та функціональність програмного забезпечення прокладання маршруту, зокрема визначено основні функції, які повинні бути реалізовані в ПЗ, такі як вибір точок користувачем, створення графу вузлів на основі інфраструктури міста та рельєфу, алгоритм пошуку оптимального маршруту та візуалізація його на мапі веб частини ПЗ.

Розроблено інтерактивну геоінформаційну систему для прокладання трубопроводів на прикладі міста Київ. Для створення веб-інтерфейсу використано фреймворк Streamlit, який дозволяє швидко розробляти інтерактивні додатки на Python. Інтерфейс включає інтерактивну карту на основі бібліотеки Folium, що забезпечує можливість інтерактивного вибору точок маршруту та відображення оптимального шляху. Для обробки геопросторових даних використано бібліотеки scipy (KDTree), math та власні алгоритми маршрутизації на базі A* алгоритму.

Розроблена геоінформаційна система пройшла тестування на працездатність та належне функціонування. Реалізовано інтерактивну карту з використанням сучасних бібліотек обробки геопросторових даних, що забезпечує ефективне прокладання маршрутів трубопроводів з урахуванням інфраструктури міста Київ. Виконано відповідні налаштування та оптимізацію системи для забезпечення швидкості та продуктивності роботи з геопросторовими даними.

ABSTRACT

DEVELOPMENT, SPECIALIZED, SOFTWARE, WEB APPLICATION, OPTIMAL ROUTE, PIPELINE LAYING, ROUTE SEARCH, SEARCH ALGORITHM, INTERACTIVE MAP, A*, MARKERS, ROUTE LENGTH, MAP VISUALIZATION, FUNCTIONALITY, FRONTEND, BACKEND.

Thesis in: 79p., 33 fig., 1 app., 10 references.

The purpose of the qualification work is to develop a specialized geographic information system for laying pipelines on the example of Kyiv.

The object of development is specialized software for finding the optimal pipe laying route, including the interface, system functionality, point selection, route laying, and route visualization on the map.

The paper analyzes the requirements and functionality of route guidance software, in particular, identifies the main functions that should be implemented in the software, such as user selection of points, creation of a graph of nodes based on the city's infrastructure and terrain, an algorithm for finding the optimal route and visualizing it on the map of the software's web part.

An interactive geographic information system for laying pipelines has been developed on the example of the city of Kyiv. To create the web interface, the Streamlit framework was used, which allows for the rapid development of interactive Python applications. The interface includes an interactive map based on the Folium library, which provides the ability to interactively select route points and display the optimal path. To process geospatial data, the scipy (KDTree), math libraries and proprietary routing algorithms based on the A* algorithm are used.

The developed geographic information system has been tested for efficiency and proper functioning. An interactive map has been implemented using modern geospatial data processing libraries, which ensures efficient pipeline routing taking

into account the infrastructure of Kyiv. The system was configured and optimized to ensure the speed and performance of geospatial data processing.

ЗМІСТ

ВСТУП

Сучасний розвиток інформаційних технологій відкриває нові горизонти для автоматизації складних інженерних процесів. Це особливо важливо у сфері планування інфраструктури, оскільки правильне проектування об’єктів залежить від кількох факторів, таких як: рельєф місцевості, існуючі будівлі та інженерні комунікації. З використанням цифрових моделей рельєфу цей процес можна не тільки значно спростити, але й забезпечити високий рівень точності та адаптації до реальних умов.

Проектування трубопровідних систем є прикладом завдання, яке потребує врахування багатьох факторів: ухилів поверхні, міської інфраструктури, природних перешкод та економічної доцільності. Використовуючи геоінформаційні системи (ГІС), ви можете поєднувати просторові дані з математичними алгоритмами для розрахунку оптимальних маршрутів.

Тема даної роботи присвячена розробці спеціалізованої ГІС, що дозволяє автоматизувати планування трубопроводної мережі за допомогою цифрових моделей рельєфу.

Метою цієї кваліфікаційної роботи є розробка інтерактивного програмного забезпечення, яке дозволяє користувачам не тільки досліджувати місцевість на карті, а й планувати маршрути трубопроводів з урахуванням висот, схилів і існуючих будівель. Реалізація цього підходу планується на прикладі міста Київ.

РОЗДІЛ 1 СУЧАСНИЙ СТАН І АКТУАЛЬНІСТЬ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1.1 Сучасний стан проектування інженерних мереж

Планування інженерних мереж, особливо трубопроводів, є одним із найважливіших завдань у розвитку міської інфраструктури. Традиційно ці процеси здійснювалися за допомою топографічних карт, таблиць та ручних розрахунків, що не дозволяло врахувати всі особливості рельєфу та існуючу інфраструктуру. Крім того, такі підходи були трудомісткими та забирали багато часу на внесення змін до проектів.

На зміну застарілим методам прийшли сучасні геоінформаційні системи (ГІС), які дозволяють проводити просторовий аналіз з урахуванням цифрових моделей рельєфу (ЦМР). ГІС пропонує інтеграцію даних місцевості, таких як висоти, схили, перешкоди, і дозволяє розрахувати оптимальні маршрути для інженерних мереж.

Однак загальні програмні рішення не завжди пристосовані до конкретних вимог проектування трубопроводів. Відсутність спеціалізованих інструментів, що враховують не тільки рельєф місцевості, але й технічні вимоги до трубопровідних мереж, створює значні труднощі для проектувальників. Це підкреслює необхідність розробки спеціалізованого програмного забезпечення, яке може автоматизувати процес проектування з урахуванням усіх вимог трубопроводу.

1.2 Актуальність теми кваліфікаційної роботи

Питання автоматизації планування трубопроводних мереж стає особливо актуальним в умовах стрімкого розвитку міської інфраструктури. Щільність забудови, неоднорідність рельєфу та наявність існуючих інженерних комунікацій у великих містах, таких як Київ, створюють значні проблеми для планувальників.

Відсутність доступного спеціалізованого програмного забезпечення, яке інтегрує цифрові моделі рельєфу, алгоритми оптимізації шляху та сучасні веб-технології, значно уповільнює процес проектування та збільшує його вартість. Інструменти загального призначення, доступні на ринку, як правило, не враховують особливості конструкції трубопроводу, такі як вплив ухилу на гідравлічні властивості системи.

Розробка спеціалізованого програмного забезпечення, запропонованого в даній роботі, дозволяє:

а) забезпечити автоматизацію проектування з урахуванням особливостей рельєфу та інфраструктури;
б) інтегрувати просторові дані із сучасними методами оптимізації маршруту;
в) надати інженерам практичний інструмент для швидкого прийняття рішень.

Таким чином розробка такого програмного забезпечення важлива як для задач міської інфраструктури, так і для вирішення складних технічних завдань при проектуванні трубопровідних мереж.

1.3 Цілі та завдання кваліфікаційної роботи

Метою даної кваліфікаційної роботи є розробка спеціалізованого програмного забезпечення, що дозволяє автоматизувати проектування трубопроводних мереж на основі цифрових топографічних моделей та інтерактивного просторового аналізу. В роботі поставлені такі завдання:

а) проведення аналізу сучасних методів автоматизації проектування трубопроводних мереж і геоінформаційних систем;
б) розробка концепції спеціалізованого програмного забезпечення для планування трубопроводів;
в) реалізація цифрової моделі рельєфу досліджуваної території (на прикладі міста Києва);

г) розробка алгоритмів оптимізації маршруту з урахуванням параметрів рельєфу та обмежень міської інфраструктури;

д) створення веб-інтерфейсу для інтерактивної візуалізації карти та побудови маршруту;

е) інтеграція просторових даних з алгоритмами оптимізації в єдину систему;

ж) проведення тестів спеціалізованого програмного забезпечення для оцінки його ефективності та точності.

1.4 Аналіз існуючих рішень

У сучасному світі для автоматизації планування інженерних мереж активно використовується широкий спектр програмних рішень. Вони використовуються, зокрема, для ппроектування трубопроводів, транспортної інфраструктури та інших інженерних об'єктів. Однак більшість таких систем є універсальними і не враховують специфічних особливостей прокладання трубопровідних мереж на конкретних ділянках.

Найбільш поширеними категоріями програмних продуктів є:

а) Геоінформаційні системи (ГІС):

1) QGIS та ArcGIS – популярні інструменти для аналізу просторових даних. Вони дозволяють працювати з цифровими моделями рельєфу (ЦМР) і здійснювати базовий просторовий аналіз. Однак ці системи обмежені у виконанні специалізованих завдань, таких як оптимізація маршрутів трубопроводів з урахуванням інженерних параметрів;

2) Google Maps API та Mapbox – це інструменти для інтеграції картографічних даних у веб-додатки. Вони дозволяють базово візуалізувати маршрути, але не можуть враховувати рельєф місцевості або технічні параметри трубопроводів.

б) Інженерні програмні комплекси:

1) Autodesk InfraWorks та AutoCAD Civil 3D – ці інструменти орієнтовані на моделювання інженерних об'єктів, включаючи системи трубопроводів. Вони підтримують роботу з ЦМР і дозволяють проектувати мережі, але вимагають значних ресурсів і глибоких знань для ефективного використання;
2) Bentley Systems (OpenFlows) – спеціалізовані рішення для проектування мереж водопровідних та каналізаційних мереж. Вони пропонують гідравлічні розрахунки, але обмежені при роботі з рельєфом та маршрутизацією трубопроводів.

в) Програмне забезпечення для оптимізації маршрутів:
1) пакети Python (NetworkX, Geopy, OSRM): дозволяють реалізувати алгоритми оптимізації маршрутів, такі як A* або Дейкстри, але мають бути інтегровані з іншими інструментами для врахування рельєфу та візуалізації;
2) Komoot, RouteXL: сервіси для оптимізації транспортних маршрутів. Хоча вони враховують фізичні обмеження, їхні функції не пристосовані до потреб планування трубопровідних мереж.

г) Веб-орієнтовані рішення:
1) Інструменти на основі Leaflet.js або Mapbox GL дозволяють інтерактивно візуалізувати карти та маршрути. Однак їх використання для проектування інженерних мереж вимагає додаткової інтеграції алгоритмів та геопросторових даних.

Аналіз показує, що існуючі програмні рішення охоплюють лише деякі аспекти процесу проектування трубопроводів. Геоінформаційні системи дозволяють керувати просторовими даними, але не підходять для оптимізації маршрутів трубопроводів. Інженерні програмні комплекси пропонують потужні інструменти моделювання, але є занадто складними і дорогими для невеликих проектів. Інструменти оптимізації та веб-технології забезпечують

гнучкість, але для створення повноцінного продукту необхідні додаткові інтеграції.

Це підтверджує потребу в розробці спеціалізованого програмного забезпечення, яке об’єднує роботу з цифровими моделями рельєфу, алгоритмами оптимізації та інтерактивними інструментами візуалізації з урахуванням особливостей проектування трубопровідних мереж.

1.5 Виклики сучасного проектування інженерних мереж

Сучасне проектування інженерних комунікацій, в тому числі трубопроводів, стикається з багатьма викликами, які значно ускладнюють процес створення ефективних, економічно вигідних та безпечних рішень. Одним з основних факторів є щільність міської забудови, яка створює значні бар'єри для прокладання нових мереж. У великих містах, таких як Київ, де вже існує розвинена інфраструктура, будь-який новий проект повинен враховувати існуючі дороги, будівлі, підземні комунікації та природні особливості місцевості. Це обмежує можливості проектувальників і вимагає рішень, які враховують ці численні фактори, щоб мінімізувати вплив на існуючу інфраструктуру та уникнути ризику пошкодження існуючих мереж.

Крім того, сучасні технології часто стикаються з проблемою інтеграції різних джерел даних. Наприклад, картографічні дані з географічних інформаційних систем (ГІС) мають бути поєднані з цифровими моделями рельєфу та інженерними специфікаціями. Цей процес ускладнюється різними форматами даних, а також неточностями або прогалинами в наявній інформації. Висока якість і точність даних мають вирішальне значення для проектування трубопроводу, оскільки навіть невелика помилка може призвести до значних витрат під час реалізації проекту або аварій у майбутньому.

Інша проблема полягає в тому, що існуюче програмне забезпечення, яке використовується для проектування розподільчих мереж, є недосконалим. Більшість доступних інструментів зосереджені на загальних

завданнях і не враховують специфіку трубопровідних систем. Наприклад, такі аспекти, як нахил місцевості, споживання енергії для транспортування води або газу та можливі природні перешкоди часто розглядаються окремо, що значно ускладнює процес проектування. Тому потрібні спеціалізовані рішення, які можуть автоматично враховувати всі ці фактори.

Часові та фінансові обмеження також відіграють важливу роль у створенні нових проектів. У сучасному середовищі все більше проектів реалізуються в стислі терміни, що вимагає автоматизації процесів, скорочення ручної роботи і швидкого отримання точних результатів. Крім того, необхідно максимізувати прибутковість проекту, щоб зменшити загальні витрати на побудову та експлуатацію мережі. Тому розробка спеціалізованих і специфічних інструментів для проектування трубопроводів є важливим напрямком, який може допомогти подолати всі ці виклики.

1.6 Тенденції автоматизації проектування

Автоматизація процесу проектування інженерних мереж - сучасний тренд, який активно розвивається у зв'язку з підвищенням вимог до точності, швидкості та економічної ефективності проектів. Все більше уваги приділяється розробці спеціалізованого програмного забезпечення, яке зменшує залежність від ручної праці та дозволяє уникнути людських помилок. Важливою тенденцією в цій сфері є використання географічних інформаційних систем (ГІС), які дозволяють працювати з просторовими даними та інтегрувати різні джерела інформації. ГІС-системи дозволяють управляти цифровими моделями місцевості, обробляти дані про інфраструктуру та природні об'єкти, а також автоматично аналізувати взаємодію між різними об'єктами місцевості.

Іншим важливим аспектом автоматизації є розробка алгоритмів оптимізації, які допомагають знайти найкращі рішення для прокладання труб. Такі алгоритми, як A* або алгоритм Дейкстри, активно використовуються для роботи з графами, що моделюють дорожні мережі. Їх

застосування дозволяє не тільки знаходити найкоротші маршрути, але й враховувати додаткові фактори, такі як нахил, обмежені ділянки або економічні параметри. Інтеграція цих алгоритмів у програмне забезпечення дозволяє значно скоротити час планування та підвищити точність розрахунків.

Розвиток технологій візуалізації заслуговує на особливу увагу. Інтерактивні карти, створені за допомогою таких інструментів, як Leaflet.js або Folium, дозволяють дизайнерам і клієнтам швидко оцінити результат роботи, внести корективи і вибрати найкраще рішення для реалізації. Такий підхід забезпечує високу зручність використання програмного забезпечення та робить процес проектування більш прозорим.

Нарешті, важливу роль в автоматизації процесу проектування відіграє врахування реальних умов на місцевості, в тому числі рельєфу. Тенденція інтеграції цифрових моделей рельєфу в програмне забезпечення дозволяє враховувати такі фактори, як градієнти, схили і нахили, що особливо важливо для проектування трубопроводів. Це допомагає мінімізувати витрати на будівництво та експлуатацію і уникнути майбутніх технічних проблем.

Таким чином, автоматизація проектування інженерних мереж - це багатогранний процес, який включає роботу з геопросторовими даними, розробку алгоритмів оптимізації, інтерактивну візуалізацію та врахування реальних польових умов. Ці тенденції дозволяють знаходити більш ефективні та економічно вигідні рішення для проектування трубопровідних систем.

РОЗДІЛ 2 РОЗРОБКА ТА АНАЛІЗ ФУНКЦІОНАЛЬНОЇ СХЕМИ І АЛГОРИТМУ

2.1 Розробка функціональної схеми спеціалізованого програмного забезпечення для прокладання оптимальних маршрутів трубопроводу

Важливим етапом при розробці програмного забезпечення є створення функціональної схеми. Функціональна схема відображає ключові функції та їх взаємодію, які будуть реалізовані для користувачів програмного забезпечення (див. рис. 2.1).

Рисунок 2.1 – Функціональна програмного забезпечення ????????

Основна мета цієї функціональної схеми – продемонструвати взаємозв'язок ключових компонентів програмного забезпечення, визначивши їх основні функції, входи і виходи, а також механізми управління та обслуговування. Це допомагає зрозуміти логіку роботи системи і переконатися, що вона відповідає поставленим вимогам. В функціональну схему входять наступні ключові компоненти:

а) цифрові моделі рельєфу. Для підвищення точності прокладання маршруту ПЗ використовує цифрому модель рельєфу будь-якого міста, наприклад, Київ, яка зберігається на сервері;

б) геопросторові дані. Для підвищення точності прокладання маршруту алгоритм ПЗ використовує файл, який містить інфраструктуру обраного міста та зберігається на сервері;

в) алгоритм Дейкстри. Для розрахування оптимального маршруту між точками використовується алгоритм Дейкстри;

г) обмеження рельєфу та забудови. Програмне забезпечення при прокладанні маршруту використовує файл з геопросторовими даними та використовує ці дані для аналізу місцевості;

д) початкова і кінцева точки. Користувач на веб сторінці повинен позначити на карті початкову та кінцеву точки між якими прокладається маршрут;

е) оптимальний маршрут. Після введення користувачем початкової і кінцевої точки алгоритм прокладає шлях з урахуванням рельєфу та інфраструктури міста;

ж) візуалізація маршруту на карті. Після того, як буде отриманий оптимальний маршрут ПЗ візуалізує його на карті веб сторінки.

2.2 Розробка алгоритмів роботи веб-застосунку

Розробка алгоритмів для програмного забезпечення для проектування трубопроводів включає кілька ключових етапів. Основна мета алгоритмів – забезпечити інтерактивне планування маршруту з урахуванням рельєфу місцевості, інфраструктури та специфічних обмежень.

2.2.1 Алгоритм завантаження та обробки даних

Метою цього алгоритму є забезпечення підготовки даних для подальшої роботи з маршрутизацією трубопроводів.

Нижче наведені кроки виконання та блок-схема алгоритму (див. рис. 2.2):

а) завантажити дані дорожньої інфраструктури з джерела (OSM або інший формат);

б) побудувати граф дорожньої мережі, де:

1) вузли графа – це точки з координатами;

2) ребра графа – це дороги, що з’єднують вузли, з атрибутами (довжина, тип дороги тощо).

в) завантажити цифрову модель рельєфу місцевості;

г) інтегрувати дані про висоти в граф дорожньої мережі:

1) для кожного вузла додати інформацію про висоту;

2) перерахувати вагу ребер із урахуванням ухилів (градієнтів).

Рисунок 2.2 – Алгоритм завнтаження та обробки даних

2.2.2 Алгоритм вибору точок на карті

Метою цього алгоритму є забезпечення взаємодії з користувачем для вибору початкової та кінцевої точок маршруту.

Нижче наведені кроки виконання та блок-схема алгоритму (див. рис. 2.3):

а) відобразити інтерактивну карту місцевості (завдяки API, наприклад, Leaflet.js або Folium);

б) надати користувачеві можливість вибрати початкову і кінцеву точки маршруту на карті;

в) знайти найближчі вузли графа до обраних точок:

1) використати алгоритм найближчих сусідів для пошуку.

г) зберегти ці вузли як стартову і кінцеву точки для подальшого прокладання маршруту.

Рисунок 2.3 – Алгоритм вибору точок на карті

2.2.3 Алгоритм побудови оптимального маршруту

Метою цього алгоритму є побудова оптимального маршруту з урахуванням рельєфу та дорожньої мережі.

Нижче наведені кроки виконання та блок-схема алгоритму (див. рис. 2.4):

а) використати граф дорожньої мережі з інтегрованими даними про висоти;

б) запустити алгоритм пошуку оптимального шляху:

1) використати алгоритм Дейкстри або A* для пошуку маршруту між вузлами графа;

2) врахувати вагу ребер (довжина + градієнт ухилу).

в) повернути список точок, які формують оптимальний маршрут.

Рисунок 2.4 – Алгоритм побудови оптимального маршруту

2.2.4 Алгоритм візуалізації маршруту

Метою цього алгоритму є надання користувачеві зрозуміле відображення побудованого маршруту.

Нижче наведені кроки виконання та блок-схема алгоритму (див. рис. 2.5):

а) відобразити базову карту місцевості;

б) нанести на карту:

1) оптимальний маршрут, виділеним кольором;

2) початкову та кінцеву точки (маркери).

в) вивести додаткову інформацію про маршрут:

1) довжина маршруту;

2) ухили вздовж маршруту;

3) потенційні перешкоди чи складні ділянки.

Рисунок 2.5 – Алгоритм візуалізації маршруту

2.2.5 Алгоритм інтерактивного оновлення

Мета цього алгоритму дозволити користувачеві змінювати точки маршруту і будувати новий маршрут.

Нижче наведені кроки виконання та блок-схема алгоритму (див. рис. 2.6):

а) додати можливість вибору нових точок на карті;

б) у разі зміни точок:

1) знайти нові найближчі вузли графа;

2) перезапустити алгоритм пошуку маршрута;

3) оновити карту з новим маршрутом.

Рисунок 2.6 – Алгоритм інтерактивного оновлення

Розроблені алгоритми визначають основні етапи роботи програмного забезпечення та забезпечують його ефективність та зручсність для користувача.

Було розроблено п’ять алгоритмів:

а) алгоритм завантаження та обробки даних дозволяє інтегрувати дорожню мережу та рельєф місцевості в граф для точного розрахунку маршрутів з урахуванням ухилів;

б) алгоритм вибору точок на карті забезпечує інтуїтивну взаємодію з користувачем для визначення початкових і кінцевих точок маршруту, автоматично прив’язуючи їх до графа;

в) алгоритм побудови оптимального маршруту виконує пошук найкоротшого шляху з урахуванням ухилів і довжин сегментів дороги, що підвищує точність проектування.

г) алгоритм візуалізації маршруту надає користувачеві зрозуміле графічне представлення розрахованого маршруту з додатковою інформацією, такою як довжина та характеристики рельєфу;

д) алгоритм інтерактивного оновлення додає гнучкість, дозволяючи в реальному часі змінювати точки маршруту та автоматично отримувати оновлені розрахунки.

Ці алгоритми є основою для створення спеціалізованого та ефективного програмного забезпечення, яке інтегрує сучасні технології просторового аналізу та врахування рельєфу.

2.3 Особливості роботи з рельєфом

Рельєф місцевості є критично важливим фактором при проектуванні інженерних комунікацій, особливо трубопроводів. Він впливає не лише на фінансові аспекти, такі як вартість будівництва, але й на технічні рішення, включаючи вибір матеріалів, методів монтажу та майбутню експлуатацію. Належне врахування рельєфу місцевості допомагає мінімізувати ризики, пов'язані з технічними труднощами та перевищенням бюджетних обмежень.

У програмному забезпеченні, що розробляється, рельєф інтегрується через цифрові моделі рельєфу (ЦМР), які є джерелом даних про висоту місцевості.

2.3.1 Підготовка даних рельєфу

Перш ніж інтегрувати місцевість у систему, дані про неї мають бути оброблені. Найважливішим джерелом таких даних є цифрові моделі рельєфу (ЦМР). ЦМР – це набір даних, що відображає висоту поверхні землі над рівнем моря. Ці дані зазвичай надаються у матричному форматі, де кожна комірка відповідає певній точці на земній поверхні. Для забезпечення точності використовуються високоякісні джерела, такі як SRTM NASA, OpenTopography або місцеві геодезичні вимірювання. Інтеграція рельєфу вимагає перетворення даних у відповідний робочий формат. У більшості випадків це формати GeoTIFF для матриць висот або таблиці CSV, що містять координати точок (широта, довгота) та їхні значення висот. Використання стандартних форматів спрощує обробку даних, скорочує час підготовки та дозволяє уникнути втрати точності.

При обробці даних про рельєф особливу увагу слід приділити нормалізації значень висот. Таким чином можна усунути відмінності в системах вимірювання, які могли бути використані у вихідних даних. Наприклад, якщо одні дані подано в метрах, а інші – у футах, необхідно їх перевести. Важливо також зазначити, що в даних ЦМР можуть бути прогалини. У деяких випадках певні ділянки місцевості можуть бути недоступними або містити помилкові значення. Для вирішення цієї проблеми використовується інтерполяція – метод математичної апроксимації – для заповнення відсутніх значень.

2.3.2 Інтеграція рельєфу в граф дорожньої мережі

Після того, як дані про місцевість підготовлені, наступним кроком є їх інтеграція в графік дорожньої мережі. Це важливий процес, який дозволяє врахувати рельєф місцевості при розрахунку маршрутів. Вузли на графіку,

які представляють географічні координати, отримують додатковий атрибут висоти. Для цього використовуються методи просторового аналізу, такі як пошук найближчого сусіда. Алгоритм KD-Tree є одним з найефективніших методів швидкого зіставлення вузлів графа з точками місцевості та знаходження відповідних значень висот. Якщо дані про висоту для деяких вузлів недоступні, використовується інтерполяція на основі найближчих точок рельєфу.

Визначивши висоту кожної вершини, врахуйте ці дані при зважуванні ребер графа. Кожне ребро являє собою відрізок шляху між двома вершинами. Вага ребра визначається як сума його довжини та нахилу, обчисленого з висот двох вершин, які ребро з'єднує. Нахил (градієнт) обчислюється як абсолютна різниця між висотами двох вершин, поділена на відстань між ними. Це дозволяє врахувати складність маршруту через підйоми та спуски. Якщо градієнт перевищує допустимі межі для проектування трубопроводу, край може бути виключений з розрахунків. Це дозволяє уникнути ділянок, де будівництво трубопроводу є технічно неможливим або економічно недоцільним.

2.3.3 Особливості обробки рельєфу для складних ділянок

Робота з місцевістю ускладнюється, якщо вона має значні перепади висот або містить природні перешкоди, такі як водойми, круті скелі чи яри. У таких випадках розрахунки маршруту повинні враховувати ці особливості. Наприклад, якщо нахил місцевості перевищує певне значення, цю ділянку маршруту можна визначити як критичну. Крім того, ділянки з надмірним нахилом можуть бути автоматично виключені з графіка, щоб забезпечити відповідність маршруту реальним умовам будівництва.

Природні перешкоди, такі як річки, можуть бути позначені як зони, де маршрут не можна перетнути без використання мостів або тунелів. У цих випадках додаток повинен попередити користувача і запропонувати альтернативні маршрути в обхід перешкоди.

2.3.4 Модульність роботи з рельєфом

Оскільки рельєф є динамічним параметром, який може змінюватися залежно від масштабу проекту, важливо забезпечити модульність обробки даних. Програмне забезпечення має підтримувати різні рівні деталізації рельєфу залежно від типу джерела даних. Для великих територій можна використовувати глобальні дані ЦМР, а для локальних проектів - більш детальні геодезичні вимірювання. Крім того, підтримка об'єднання декількох наборів даних дозволяє підвищити точність розрахунків за рахунок використання інформації, доступної з різних джерел.

Таким чином, робота з характеристиками рельєфу забезпечує гнучкість і точність проектування трубопроводу в складних умовах місцевості. Це дозволяє нам створювати оптимальні маршрути на основі фактичних умов і мінімізувати ризики, пов'язані з неврахуванням рельєфу місцевості.

2.4 Розрахунок оптимального маршруту

Процес розрахунку оптимального маршруту є важливим етапом роботи програмного забезпечення, який забезпечує визначення найбільш економічного і технічно вигідного маршруту прокладання труб. Цей процес базується на використанні графічних структур, що моделюють дорожню мережу, а також алгоритмів пошуку найкоротших шляхів, адаптованих до особливостей поставленої задачі. При розрахунку маршруту необхідно враховувати кілька важливих аспектів, таких як рельєф місцевості, протяжність маршруту, існуючі обмеження, такі як будівлі або природні перешкоди, а також економічні параметри.

Основою для розрахунків є граф дорожньої мережі, де вузлами є координати ключових точок (перехресть або кінців ділянок), а ребрами - відрізки доріг між ними. Кожне ребро має вагу, яка характеризує вартість або складність проїзду через цю ділянку. Вага відрізка може залежати від таких параметрів, як його довжина, нахил поверхні, перешкоди або тип дороги. Наприклад, нахил визначається як різниця висот між вузлами, що з'єднують

край, і є критично важливим фактором при проектуванні трубопроводу. Врахування нахилу допомагає забезпечити практичність прокладання маршруту, уникнути втрат матеріалу та енергії при підйомі або спуску по схилах, а також гарантує, що маршрут відповідає нормативним вимогам.

Розрахунок оптимального маршруту починається з вибору початкової та кінцевої точок на інтерактивній карті. Ці точки задаються користувачем вручну, а потім автоматично з'єднуються з найближчими вузлами на графіку. Це забезпечує точність розрахунків і враховує реальну дорожню інфраструктуру. Для зв'язування використовується просторовий аналіз, зокрема алгоритми пошуку близькості, такі як KD-Tree. Таким чином, початкова та кінцева точки стають вузлами на графі, між якими потрібно знайти оптимальний маршрут.

Для обчислення шляху використовуються сучасні графові алгоритми, такі як Дейкстри або A*. Алгоритм Дейкстри дозволяє знайти найкоротший шлях у графі з позитивними вагами, що гарантує оптимальність рішення. Алгоритм обходить всі вершини графа і поступово обчислює найкоротшу відстань від кожної вершини до початкової точки. Хоча цей алгоритм гарантує точний результат, його продуктивність може бути обмеженою для великих графів. Алгоритм A* є більш ефективним завдяки використанню евристичних функцій, які зменшують кількість вершин для аналізу. Евристичні функції в A* базуються на оцінці відстані до цільової вершини, наприклад, за допомогою евклідової або географічної відстані. Це значно прискорює процес обчислення, зберігаючи при цьому точність.

Особливістю використання цих алгоритмів при проектуванні трубопроводів є те, що вони враховують вагові компоненти ребер графа. На відміну від класичних задач пошуку найкоротшого шляху, де вага ребра зазвичай залежить тільки від його довжини, в цьому випадку додаються додаткові параметри, такі як градієнти та обмеження. Наприклад, якщо градієнт перевищує певне значення, вага ребра значно збільшується, щоб зменшити ймовірність вибору цього шляху. Якщо частина шляху заборонена

для прокладання труб через природні або технічні обмеження, вона виключається з графа перед початком розрахунку.

Після завершення обчислень алгоритм повертає список вузлів, які складають оптимальний маршрут. Цей маршрут – це набір координат, що описує шлях від початкової до кінцевої точки. Крім того, обчислюється інформація про характеристики маршруту, такі як довжина маршруту, його середній нахил і наявність складних ділянок. Ця інформація важлива для подальшого аналізу та прийняття рішень, наприклад, щодо використання додаткового обладнання чи матеріалів.

Окремо слід згадати питання обчислювальної потужності. Для великих графів з тисячами або навіть мільйонами вузлів час обчислень може бути критичним. Для підвищення продуктивності використовуються методи оптимізації, такі як паралельна обробка та попередня вибірка, що дозволяють зменшити розмір графа за рахунок видалення надлишкових областей. Також можна зберігати результати попередніх обчислень для повторного використання, що значно скорочує час обробки при оновленні шляху.

Розрахунок оптимального маршруту завершується передачею даних до модуля візуалізації, де маршрут відображається на інтерактивній карті. Це дозволяє користувачеві оцінити результат, за необхідності внести корективи та підготувати дані для подальшого планування. Таким чином, алгоритми розрахунку оптимального маршруту гарантують ефективність і надійність програмного забезпечення, дозволяючи проводити точні розрахунки навіть у складних умовах місцевості.

РОЗДІЛ 3 РОЗРОБКА СПЕЦІАЛІЗОВАНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Аналіз обраної середи програмування

Для реалізації програмного забезпечення було обрано мову програмування Python у поєднанні з інтегрованим середовищем розробки PyCharm. Це рішення ґрунтувалося на оцінці кількох ключових факторів, таких як масштабованість, підтримка бібліотек, продуктивність при роботі з зовнішніми даними, простота інтеграції та налагодження.

Python – це високорівнева інтерпретована мова програмування, яка набула популярності завдяки простоті використання, великій бібліотечній екосистемі та підтримці спільноти.

Особливістю реалізації стало використання багатопоточності для розподілу обчислень. У create_nodes.py розподіл завдань побудови графів між процесами значно скоротив час обробки великих наборів даних, таких як шляхи та перешкоди. Незважаючи на обмеження GIL, Python забезпечив швидкість і стабільність обчислень завдяки бібліотекам multiprocessing та tqdm.

Основні причини вибору Python для цього проекту:

а) простота синтаксису: Python має простий, зрозумілий синтаксис, який дозволяє швидко реалізовувати складні алгоритми. Це значно знижує поріг входу і скорочує час, необхідний для написання та тестування коду;

б) розгалужена екосистема бібліотек: Python має велику кількість бібліотек для різноманітних завдань. У цьому проекті були використані:

1) osmium для обробки даних OSM (OpenStreetMap) та яка дозволила обробляти великі набори геопросторових даних, витягуючи інформацію про вузли, дороги, будівлі та водні об’єкти;

2) shapely для обробки геометричних об'єктів, таких як багатокутники та лінії та яка дозволила створювати лінії та полігони та обчислювати перетини, які використовуються для перевірки перешкод під час побудови графа;
3) folium та streamlit забезпечують взаємодію з користувачем через графічний інтерфейс, дозволяють легко відображати маршрути, точки, а також інтегрувати їх із картами OpenStreetMap;
4) pandas для редагування таблиць даних;
5) numpy та scipy для наукових та математичних розрахунків;
6) tqdm для візуалізації прогресу обчислень у консольному режимі;
7) rtree для створення просторових індексів.

в) крос-платформенна переносимість та сумісність: Python сумісний з усіма основними операційними системами, що забезпечує легке розгортання додатків;
г) масштабованість: Python може інтегрувати модулі, написані іншими мовами програмування (наприклад, C/C++);
д) підтримка багатопотоковості: завдяки багатопоточним бібліотекам Python може ефективно обробляти великі обсяги даних у багатопотоковому режимі, що є важливим для обробки геоданих.

PyCharm – це професійне середовище розробки для Python, що надає інструменти для прискорення та спрощення роботи над програмними проектами.

Значною перевагою PyCharm є можливість запуску веб-інтерфейсу для візуалізації результатів. Файл map_and_route.py реалізує функціонал вибору точок і побудови маршрутів, який здійснюється через інтерактивний інтерфейс Streamlit. PyCharm дозволяє швидко запускати цей інтерфейс, налагоджувати його та інтегрувати функціонал з іншими елементами програми.

Найважливішими перевагами PyCharm є:

а) інструменти автоматичного завершення коду: PyCharm підтримує автоматичне завершення коду, що значно прискорює написання складних конструкцій і виключає можливість синтаксичних помилок;

б) можливості налагодження: вбудований відладчик дозволяє виявляти помилки в режимі реального часу. Це особливо важливо для складних багатопотокових завдань;

в) інтеграція з системами контролю версій: PyCharm має корисні інструменти для роботи з Git'ом, що дозволяє легко керувати версіями, створювати гілки та об'єднувати зміни;

г) інструменти тестування: є вбудована можливість написання та запуску модульних тестів, корисних для тестування продуктивності модулів;

д) інтеграція із зовнішніми бібліотеками: PyCharm використовує інтегрований менеджер пакетів (pip) для легкого керування залежностями проекту. Це дозволяє легко встановлювати та оновлювати потрібні бібліотеки;

е) простий у використанні графічний інтерфейс: PyCharm надає інтуїтивно зрозумілий інтерфейс для навігації по проекту, візуалізації структури коду та управління ресурсами;

ж) інтегрована підтримка Streamlit: PyCharm полегшує розробку веб-інтерфейсів таким чином, щоб логічна та візуальна частини програми могли працювати одночасно.

У PyCharm було інтегровано кілька складних бібліотек, які сильно залежать від деяких інших ресурсів:

а) osmium потребувала доступу до зовнішніх бібліотек C++ для обробки даних OSM. Завдяки PyCharm було реалізовано необхідне середовище для належного функціонування osmium;

б) shapely та rtree потребували інтеграції бібліотек системи GEOS, але їх можна було легко додати за допомогою менеджера пакетів;

в) streamlit було інтегровано у середовище розробки для швидкого запуску та тестування веб-інтерфейсу.

Обраний набір технологій дозволив досягти кількох ключових цілей:

а) реалізація ефективних алгоритмів обробки даних про рельєф, геометрію та інфраструктуру міста Києва;

б) створення інтерактивного інтерфейсу, що дозволяє користувачам обирати маршрути на карті в режимі реального часу;

в) підтримка масштабування та можливість додавання нового функціоналу (наприклад, інтеграція з іншими API);

г) поєднання обраного середовища програмування PyCharm та мови Python забезпечило продуктивний та практичний процес розробки програмного забезпечення та повністю відповідало поставленим завданням.

Обраний інструментарій дозволив нам виконати кілька важливих завдань. Наприклад, файл elevations_download.py реалізував автоматичне завантаження висот через OpenTopodata API, забезпечивши завантаження точних даних рельєфу для Києва. Ці дані були включені до файлу create_nodes.py, де вони стали основою для розрахунку ваги ребер графа з урахуванням градієнтів поверхні. Таким чином, ми змогли забезпечити точність розрахунків маршруту, що відображається в реальному часі у файлі map_and_route.py.

Python у поєднанні з PyCharm створили ідеальні умови для розробки спеціалізованого програмного забезпечення. Простий синтаксис, багата екосистема бібліотек, зручне середовище розробки та висока продуктивність при роботі з геопросторовими даними дозволили створити ефективне програмне забезпечення, яке відповідає всім вимогам.

3.2 Розробка алгоритму пошуку маршруту

Процес пошуку маршруту є ключовим елементом програмного забезпечення для проектування трубопроводу. Він базується на графічній моделі дорожньої мережі, де вузли представляють ключові точки на місцевості (наприклад, перехрестя або унікальні координати), а ребра – зв'язки між вузлами з певними атрибутами, такими як довжина, нахил, тип поверхні або перешкоди. Алгоритм пошуку маршруту повинен враховувати всі ці атрибути, щоб знайти оптимальний маршрут.

3.2.1 Основні завдання алгоритму

Алгоритм пошуку маршруту повинен виконувати такі функції:

а) знаходити найкоротший маршрут між початковою та кінцевою точками, заданими користувачем на карті;

б) враховувати вагові коефіцієнти, пов'язанів з рельєфом, довжиною маршруту та іншими факторами;

в) забезпечувати обхід перешкод (якщо можна), таких як забудовані території та водні об'єкти;

г) працювати швидко й ефективно навіть на великих наборах даних.

3.2.2 Постановка задачі алгоритму

Граф дорожньої мережі було створюється на основі даних OpenStreetMap (OSM) та цифрової моделі рельєфу (ЦМР). Кожен вузол графа містить координати (широта і довгота) та висоту. Ребра графіка містять інформацію про довжину сегменту та ухил між вузлами. Ухил обчислюється шляхом ділення різниці висот між двома вузлами на довжину ребра.

Після визначення вузлів і ребер кожному ребру призначається вага, яка враховує:

а) довжину сегменту;

б) ухил поверхні;

в) додаткові параметри, такі як складність прокладання трубопроводу на певному типі ґрунту чи через перешкоди.

Вага ребра визначається за формулою:

$$\text{weight} = \text{length} + k \cdot \text{slope}$$

Рисунок 3.1 – Формула розрахунку ваги ребра

В цій формулі length – довжина сегменту, k – коефіцієнт, що враховує вплив ухилу на складність прокладання, а slope – ухил поверхні.

3.2.3 Вибір алгоритму

Для наших цілей було виділено два алгоритми, які підходять для пошуку маршруту та задачі оптимізації в графах:

а) алгоритм Дейкстри використовується для пошуку найкоротшого шляху між вершинами позитивно зваженого графа. Він гарантує точні результати, але може бути повільнішим для великих наборів даних;

б) алгоритм A* (A-star) дозволяє використовувати евристичні функції для прискорення процесу. У цьому випадку евристична функція є оцінкою відстані між поточним вузлом і цільовою точкою. A* швидший за алгоритм Дейкстри для великих графів і зберігає його точність.

3.2.4 Алгоритм A*

Для розробки геоінформаційної системи було обрано алгоритм A*.

Алгоритм A* (A-star) – один з найефективніших алгоритмів пошуку найкоротшого шляху на графі, що поєднує евристичні та точні методи маршрутизації.

Основні характеристики A*:

а) використовує додаткову евристичну функцію для оптимізації пошуку шляху;

б) обчислює найкоротший шлях з мінімальними витратами;

в) має кращу продуктивність порівняно з класичним алгоритмом Дейкстри.

Переваги A* перед Дейкстрою:

а) враховує напрямок руху завдяки евристичній функції;

б) знаходить оптимальний маршрут набагато швидше;

в) більш ефективний пошук маршрутів у великих графах;

г) дозволяє значно зменшити кількість вузлів, що досліджуються;

д) поки Дейкстра досліджує всі можливі шляхи без урахування напрямку, A* спрямовано досліджує простір пошуку, суттєво скорочуючи час обчислень.

Функція A* складається з двох компонентів:

а) g(n) – реальна вартість руху від початкової точки до поточного вузла;

б) h(n) – евристична оцінка відстані від поточного вузла до кінцевої точки.

Причини вибору A* для геоінформаційної системи:

а) оптимізація пошуку маршруту трубопроводу;

б) врахування географічних особливостей місцевості;

в) ефективність обробки великої кількості вузлів;

г) можливість швидкого знаходження найкоротшого шляху.

У коді ПЗ використано гаверсинусну відстань як евристичну функцію, яка обчислює відстань між географічними координатами.

Гаверсинусна відстань (формула гаверсинусів) – це метод обчислення відстані між двома точками на сфері (зокрема на земній кулі) з урахуванням її кривизни.

Основні характеристики:

а) математична формула, яка враховує сферичну природу Землі;

б) дозволяє обчислити найкоротшу відстань між двома географічними точками;

в) вона є більш точною, ніж прості евклідові обчислення відстані.

Формула враховує:

а) широту точок;

б) довготу точок;

в) радіус сфери.

Переваги використання гаверсинусної відстані:

а) висока точність розрахунків для великих відстаней;

б) компенсація викривлення земної поверхні.

в) врахування сферичної геометрії

г) мінімальна похибка обчислень для великих площ.

У контексті геоінформаційних систем гаверсинусна відстань дозволяє:

а) точно вимірювати відстані між географічними точками;

б) прокладати оптимальні маршрути з урахуванням кривизни Землі;

в) забезпечувати високу точність геопросторового аналізу.

Через всі ці переваги при розробці даної геоінформаційної системи прокладання трубопроводів використовується алгоритм A* та гаверсинусна відстань як евристична функція.

3.2.4 Реалізація алгоритму

Процес пошуку маршруту починається з вибору користувачем початкової та кінцевої точок на інтерактивній карті. Ці точки, задані у вигляді географічних координат (широта і довгота), прив’язуються до найближчих вузлів графа. Прив’язка здійснюється за допомогою алгоритму KD-tree, який забезпечує швидкий пошук найближчих точок у великому наборі даних. Після того, як початковий та цільовий вузли визначені, запускається алгоритм A*.

Під час роботи алгоритму A* пріоритет віддається вузлам, які мають найнижчу суму значень функцій вартості та евристики. Це дозволяє

алгоритму швидко звузити простір пошуку та ігнорувати вузли, які не наближають до цілі.

3.2.5 Особливості врахування рельєфу

Рельєф місцевості враховується за допомогою ухилів між вузлами графа. Ця інформація, отримана з цифрової моделі рельєфу (ЦМР), дозволяє алгоритму автоматично адаптуви алгоритм до рельєфу місцевості. Наприклад, на ділянках зі стрімкими підйомами вага ребер збільшується настільки, що алгоритм ігнорує ці ділянки. Щоб врахувати складність різних типів поверхонь, додаються коефіцієнти, які змінюють вагу ребер в залежності від типу поверхні.

3.2.6 Результат роботи алгоритму

Результатом роботи алгоритму є оптимальний маршрут, представлений у вигляді послідовності вузлів графа. Цей маршрут зберігається у вигляді списку координат, який надсилається до модуля візуалізації. Візуалізація малює на карті маршрут між двома точками. Крім того, розраховується загальна довжина маршруту.

3.3 Програмна реалізація основних функцій

У цьому підрозділі описано реалізацію основних функцій розробленого програмного забезпечення. Кожна функція була створена для виконання певного завдання в аналізі, обробці та візуалізації геопросторових даних. Програмна реалізація базується на використанні бібліотек Python, які забезпечують ефективність та гнучкість.

Основні функції програми:

a) обробка даних OSM;

b) створення та обробка геометричних об'єктів;

c) побудова оптимального маршруту;

d) візуалізація на карті;

e) інтерактивний інтерфейс користувача.

3.3.1 Підготовка OSM файла

Для роботи розробленого програмного забезпечення необхідно мати актуальний файл OSM, який містить геопросторові дані про територію, що аналізується. У даному проекті як джерело геопросторових даних обрано OpenStreetMap через його відкритість, велику спільноту користувачів і деталізованість інформації.

Першим кроком є завантаження файлу OSM у форматі .pbf (Protocolbuffer Binary Format), який є стандартним форматом для збереження великих наборів даних OSM. Для цього використовується ресурс Geofabrik, який надає актуальні геопросторові дані для різних регіонів світу. Зокрема, для даного проекту файл, що охоплює територію України, завантажується за посиланням https://download.geofabrik.de/europe/ukraine-latest.osm.pbf.

Завантажений файл містить усю дорожню мережу, будівлі, водні об’єкти та інші елементи, але для проекту необхідно зосередитися на конкретному регіоні – Києві. Тому наступним етапом є обрізка файлу до меж Києва. Це дозволяє суттєво зменшити обсяг даних, що обробляються, і прискорити роботу програмного забезпечення. Для обрізки використовується утиліта osmconvert.

Процес обрізки здійснюється за межами міста Києва, які задаються у вигляді координат.

Наприклад, межі можуть бути визначені як:

а) 30.2394 – західна довгота;

б) 50.2132 – південна широта;

в) 30.7683 – східна довгота;

г) 50.5901 – північна широта.

Команда для обрізки файлу до меж Києва виглядає так:

```
osmconvert ukraine-latest.osm.pbf -b=30.2394,50.2132,30.7683,50.5901 -
o=kiev.osm.pbf
```

Після обрізки дані необхідно конвертувати у формат .osm, який краще підходить для подальшої обробки в Python за допомогою бібліотеки osmium. Конвертація здійснюється через інструмент osmconvert. Наприклад, команда для конвертації може виглядати так:

```
osmconvert kiev.osm.pbf -o=kiev.osm
```

Цей файл (kiev.osm) використовується як вхідний для функцій, що відповідають за створення графа дорожньої мережі, інтеграцію висотних даних та розрахунок маршрутів. Він містить усю необхідну інформацію для побудови моделі дорожньої мережі, зокрема:

а) геометрію доріг у вигляді вузлів та ребер;

б) типи доріг (автостради, міські дороги, пішохідні доріжки тощо);

в) інші об’єкти інфраструктури, які можуть впливати на проектування (мости, тунелі, річки).

Таким чином, підготовка OSM-файла є важливим етапом у роботі програмного забезпечення, оскільки вона забезпечує надійну основу для подальшої обробки даних та побудови графа дорожньої мережі. Цей етап є обов'язковим, оскільки від точності та актуальності даних залежить ефективність і точність результатів проекту.

3.3.2 Опис головного файлу

Файл main.py є головним модулем програми, який координує виконання всіх основних етапів роботи. Він забезпечує перевірку наявності необхідних даних, автоматичне завантаження або створення відсутніх файлів, а також запуск інтерфейсу для користувачів. Завдяки цьому модуль є точкою входу для програми, спрощуючи її використання та автоматизуючи основні дії.

Функція main() складається з трьох основних етапів:

а) перевірка наявності даних про висоти (kyiv_elevations.csv). Якщо файл, що містить висоти місцевості, відсутній, програма викликає

elevations_download.py, який завантажує ці дані через API і зберігає їх у форматі CSV;

б) перевірка наявності графа дорожньої мережі (graph.json) і вузлів (nodes.csv). Якщо граф і вузли не створені, викликається create_nodes.py, який будує ці файли на основі даних OSM та висот;

в) запуск інтерфейсу для візуалізації та побудови маршруту. Після підготовки всіх необхідних даних викликається файл map_and_route.py через Streamlit, що відкриває веб-інтерфейс для взаємодії з користувачем.

Файл забезпечує повний цикл роботи програмного забезпечення: від перевірки та підготовки даних до запуску кінцевого інтерфейсу. Завдяки цьому користувачеві не потрібно запускати кожен компонент окремо – всі дії автоматизовано в одному модулі. Це значно спрощує використання системи та забезпечує стабільність її роботи.

3.3.3 Опис файлу завантаження рельєфу заданої області

Файл elevations_download.py відповідає за завантаження даних рельєфу для заданої області (Києва) з використанням API OpenTopodata. Цей модуль є основою для інтеграції даних про висоти у граф дорожньої мережі, оскільки дозволяє отримувати точні значення висот кожної точки місцевості.

Основне завдання цього модуля — автоматичне отримання висотних даних у межах Києва та збереження їх у форматі CSV для подальшого використання іншими частинами програмного забезпечення. Завдяки використанню API процес завантаження даних повністю автоматизовано, що забезпечує точність і швидкість отримання рельєфної інформації.

Основна логіка роботи:

а) задання області завантаження. В коді прописуються параметри для визначення області, рельєф якої буде завантажено. Ці параметри

визначають сітку точок з інтервалом 0.003 градуси, що забезпечує високу деталізацію даних (див. рис. 3.2);

б) формування списку точок. На основі заданих меж і кроку створюється список координат у вигляді пар широти та довготи (див. рис. 3.3);

в) розбиття запитів на частини. API OpenTopodata дозволяє обробляти обмежену кількість точок за один запит. Тому список координат ділиться на частини по 100 точок (див. рис. 3.4) та відправка кожної частини з інтервалом в 5 секунд;

г) відправлення запитів до API. Для кожного набору координат відправляється запит до API, і отримані дані зберігаються (див. рис. 3.5). Якщо запит успішний, результати додаються до загального списку (див. рис. 3.6);

д) збереження даних у CSV. Отримані дані зберігаються у файл kyiv_elevations.csv, де кожен рядок містить широту, довготу та висоту точки (див. рис. 3.7).

```
min_lat, max_lat = 50.3, 50.6
min_lng, max_lng = 30.3, 30.8
step = 0.003
```

Рисунок 3.2 – Параметри задання області

```
lats = np.arange(min_lat, max_lat, step)
lngs = np.arange(min_lng, max_lng, step)
locations = [f"{lat},{lng}" for lat in lats for lng in lngs]
```

Рисунок 3.3 – Формування списку координат

```
chunks = [locations[i:i + 100] for i in range(0, len(locations), 100)]
```

Рисунок 3.4 – Розбиття запитів на частини

```
url = "https://api.opentopodata.org/v1/srtm30m"
for i, chunk in enumerate(chunks):
    params = {
        "locations": "|".join(chunk),
        "interpolation": "cubic",
    }
    response = requests.get(url, params=params)
```

Рисунок 3.5 – Відправлення запитів до API

```
if response.status_code == 200:
    results = response.json().get("results", [])
    all_results.extend(results)
    print(f"Запит {i + 1}/{len(chunks)}: успіх, отримано {len(results)} точок.")
```

Рисунок 3.6 – Додавання результатів запиту до загального списку

```
output_file = "kyiv_elevations.csv"
with open(output_file, "w", newline="") as f:
    writer = csv.writer(f)
    writer.writerow(["Latitude", "Longitude", "Elevation"])
    for result in all_results:
        lat = result["location"]["lat"]
        lng = result["location"]["lng"]
        elevation = result["elevation"]
        writer.writerow([lat, lng, elevation])
```

Рисунок 3.7 – Збереження даних у CSV

3.3.4 Опис файлу створення графа дорожньої мережі та вузлів

Файл create_nodes.py відповідає за створення графа дорожньої мережі та вузлів на основі даних з OSM і висот місцевості. Цей файл є ключовим для побудови моделі, що враховує рельєф і перешкоди (будівлі, водойми). Результати зберігаються у вигляді вузлів (nodes.csv) і графа (graph.json), які використовуються для пошуку маршрутів.

Файл є ключовим для побудови моделі дорожньої мережі. Завдяки інтеграції даних рельєфу та обліку перешкод створюється точна основа для пошуку оптимальних маршрутів. Цей модуль забезпечує гнучкість, дозволяючи враховувати географічні, рельєфні та інфраструктурні обмеження.

Основна логіка роботи:

а) завантаження даних з OSM. За допомогою класу OsmHandler обробляються дані OpenStreetMap, витягуються вузли, дороги, будівлі та водойми. Ці елементи зберігаються у вигляді списків, які формуються з координат;

б) інтеграція висот. Дані висот завантажуються з файлу kyiv_elevations.csv за допомогою pandas і інтегруються через просторовий KDTree. Це забезпечує швидкий доступ до висоти для кожної точки;

в) формування полігонів перешкод. Будівлі та водойми перетворюються на геометричні об’єкти (полігони або лінії) за допомогою бібліотеки Shapely. Для прискорення перевірки перетинів використовується Rtree;

г) побудова графа. Унікальні вузли формуються з координат доріг, після чого обчислюються ребра графа. Вага ребер залежить від довжини сегмента та різниці висот між вузлами (див. рис. 3.8);

д) збереження результатів. Отримані вузли (nodes.csv) і граф (graph.json) зберігаються для подальшого використання.

```
weight = dist * (1 + elevation_diff / 100.0)

local_graph[u].append((v, weight))
local_graph[v].append((u, weight))
```

Рисунок 3.8 – Визначення ваги ребра

Основні функції:

а) OsmHandler. Клас для обробки OSM-файлів. Збирає вузли, дороги, будівлі та водойми (див. рис. 3.9);

б) функція build_graph. Формує граф дорожньої мережі, враховуючи перешкоди та рельєф (див. рис. 3.10);

в) функція intersects_obstacles_optimized. Перевіряє, чи перетинає лінія дороги будівлі чи водойми, використовуючи індекси (див. рис. 3.11);

г) функції save_nodes та save_graph. Вузли та граф зберігаються у вигляді файлів для подальшого використання (див. рис. 3.12 та рис. 3.13).

```
class OsmHandler(osmium.SimpleHandler):
    def node(self, n):
        self.nodes[n.id] = (n.location.lat, n.location.lon)

    def way(self, w):
        if 'highway' in w.tags:
            self.highways.append(coords)
        elif 'building' in w.tags:
            self.buildings.append(coords)
        elif w.tags.get('natural') == 'water':
            self.waters.append(coords)
```

Рисунок 3.9 – Фрагмент класу OsmHandler

```
def build_graph(handler, elev_tree, elev_values, elev_coords):
    for road in handler.highways:
        for i in range(len(road) - 1):
            line = LineString([(p1[1], p1[0]), (p2[1], p2[0])])
            if intersects_obstacles_optimized(line, building_polygons, water_polygons,
                continue
            dist = haversine(p1, p2)
            elevation_diff = abs(elev_p2 - elev_p1)
            weight = dist * (1 + elevation_diff / 100.0)
            graph[u].append((v, weight))
```

Рисунок 3.10 – Фрагмент функції build_graph

```
def intersects_obstacles_optimized(line, building_polygons, water_polygons, ...):
    for i in building_index.intersection(line.bounds):
        if line.intersects(building_polygons[i]):
            return True
    ...
    return False
```

Рисунок 3.11 – Фрагмент функції intersects_obstacles_optimized

```python
def save_nodes(nodes, filename="nodes.csv"):
    print(f"Збереження вузлів у файл {filename}...")
    with open(filename, "w", newline="", encoding="utf-8") as f:
        writer = csv.writer(f)
        writer.writerow(["index", "lat", "lon"])
        for i, (lat, lon) in enumerate(tqdm(nodes, desc="Збереження вузлів")):
            writer.writerow([i, lat, lon])
    print("Вузли збережено.")
```

Рисунок 3.12 – Функція save_nodes

```python
def save_graph(graph, filename="graph.json"):
    print(f"Збереження графа у файл {filename}...")
    json_graph = []
    for adj in tqdm(graph, desc="Збереження ребер графа"):
        json_graph.append([{"node": v, "weight": w} for v, w in adj])

    with open(filename, "w", encoding="utf-8") as f:
        json.dump(json_graph, f, indent=2)
    print("Граф збережено.")
```

Рисунок 3.12 – Функція save_graph

3.3.5 Опис файлу пошуку маршруту та створення інтерфейсу

Файл map_and_route.py відповідає за створення інтерактивного інтерфейсу для роботи з картою, вибору точок на ній, побудови маршруту та відображення результатів. Він використовує бібліотеки Streamlit та Folium, що забезпечують візуалізацію даних та інтерактивність у веб-інтерфейсі.

Файл забезпечує інтерактивність програмного забезпечення, дозволяючи користувачам вибирати точки на карті, будувати маршрути та оцінювати їхню довжину. Завдяки використанню A* алгоритму та інтеграції з графами забезпечується точність і швидкість обчислень. Цей модуль є завершальною частиною системи, яка безпосередньо взаємодіє з користувачем.

Основні функції:

а) завантаження вузлів та графа. Для роботи з даними програма завантажує вузли (nodes.csv) та граф (graph.json). Це реалізується функціями load_nodes та load_graph, які перетворюють дані у зручний формат (див. рис. 3.13);

б) вибір точок на карті. Користувачі можуть обирати початкову та кінцеву точки маршруту безпосередньо на інтерактивній карті. Для цього використовується Folium із обробкою кліків (див. рис. 3.14);

в) пошук маршруту. Алгоритм A* використовується для знаходження найкоротшого маршруту між обраними точками, враховуючи ваги ребер графа. Для цього спочатку обчислюється евристика на основі гаверсинусної відстані, а потім будуються найкоротші шляхи (див. рис. 3.15);

г) візуалізація маршруту. Побудований маршрут відображається на карті червоною лінією. Маркери вказують початкову та кінцеву точки (див. рис. 3.16);

д) інформація про маршрут. Додатково розраховується довжина маршруту та виводяться деталі у боковій панелі (див. рис. 3.17).

```
def load_nodes(filename="nodes.csv"):
    print(f"Завантаження вузлів з файлу {filename}...")
    nodes = []
    with open(filename, "r", encoding="utf-8") as f:
        reader = csv.DictReader(f)
        for row in reader:
            nodes.append((float(row["lat"]), float(row["lon"])))
    print(f"Завантажено {len(nodes)} вузлів.")
    return nodes


@st.cache_data
def load_graph(filename="graph.json"):
    print(f"Завантаження графу з файлу {filename}...")
    with open(filename, "r", encoding="utf-8") as f:
        graph = json.load(f)
    print(f"Граф завантажено.")
    return graph
```

Рисунок 3.13 – Завантаження вузлів та графа

```python
if map_data and map_data.get("last_clicked"):
    clicked_coords = map_data["last_clicked"]
    if len(st.session_state["markers"]) < 2:
        st.session_state["markers"].append((clicked_coords["lat"], clicked_coords["lng"]))
```

Рисунок 3.14 – Вибір точок на карті

```python
def a_star(graph, nodes, start, end):

    def heuristic(u, end):
        return haversine(nodes[u], nodes[end])

    dist = [math.inf] * len(graph)
    dist[start] = 0
    prev = [-1] * len(graph)
    heap = [(heuristic(start, end), start)]

    while heap:
        current_f, u = heapq.heappop(heap)
        if u == end:
            break
        if current_f > dist[u] + heuristic(u, end):
            continue
        for neighbor in graph[u]:
            v, w = neighbor["node"], neighbor["weight"]
            new_dist = dist[u] + w
            if new_dist < dist[v]:
                dist[v] = new_dist
                prev[v] = u
                heapq.heappush(heap, (new_dist + heuristic(v, end), v))

    if dist[end] == math.inf:
        return []

    path = []
    cur = end
    while cur != -1:
        path.append(cur)
        cur = prev[cur]
    path.reverse()
    return path
```

Рисунок 3.15 – Функція a_star

```python
if st.session_state["route"] is not None:
    folium.PolyLine(st.session_state["route"], color="red", weight=5).add_to(m)
```

Рисунок 3.16 – Візуалізація маршруту

```
route_length = sum(
    haversine(nodes[path_nodes[i]], nodes[path_nodes[i+1]])
    for i in range(len(path_nodes) - 1)
)

st.session_state["route_length"] = route_length
st.session_state["details"] = "Маршрут може потребувати корегування."
```

Рисунок 3.17 – Інформація про маршрут

3.4 Розробка інтерфейсу

Інтерфейс є ключовим компонентом програмного забезпечення, оскільки він дозволяє користувачеві взаємодіяти з системою. У цьому проекті розроблено інтуїтивно зрозумілий веб-інтерфейс, який дозволяє виконувати основні дії: вибирати точки на карті, будувати маршрут і відображати результати. Інтерфейс був реалізований з використанням бібліотек Streamlit та Folium, які забезпечують гнучкість відображення даних та зручність взаємодії.

Основні вимоги до інтерфейсу:

а) простота використання. Інтерфейс має бути зрозумілим для користувача без необхідності технічних знань. Усі основні функції (вибір точок, побудова маршруту, очищення даних) мають бути доступними через прості кнопки чи інтерактивні елементи;

б) інтерактивність. Взаємодія з картою має бути динамічною: користувачі повинні мати можливість обирати точки, бачити маркери та маршрути в реальному часі;

в) інформативність. Інтерфейс повинен надавати користувачеві інформацію про обрані точки, побудований маршрут (довжину, основні характеристики) та інші деталі;

г) кросплатформеність. Веб-інтерфейс повинен забезпечувати доступ з будь-якого пристрою, де є веб-браузер.

Інтерфейс складається з двох основних частин:

а) головна панель. Відображає інтерактивну карту з можливістю вибору точок та перегляду маршруту;

б) бокова панель. Надає додаткову інформацію про маршрут, опції очищення даних та управління процесами.

3.5 Тестування програмного забезпечення

Тестування програмного забезпечення - важливий етап розробки, який дозволяє перевірити правильність роботи всіх компонентів системи, виявити потенційні помилки і забезпечити стабільність роботи програми. У цьому проекті тестування проводилося на реальній місцевості Києва та даних дорожньої мережі. Процес тестування охоплював функціональні та операційні аспекти системи.

Цілі тестування:

а) перевірка коректності алгоритмів. Забезпечити правильність обробки даних, побудови графа, розрахунку ваг ребер і знаходження найкоротших маршрутів за допомогою алгоритму A*;

б) оцінка інтеграції даних. Перевірити, чи правильно система поєднує дані OSM і висот, а також чи коректно враховуються перешкоди (будівлі, водойми);

в) тестування інтерфейсу. Перевірити інтерактивну взаємодію з картою, правильність відображення маркерів, маршрутів і виведення інформації про результати.

г) оцінка продуктивності. Визначити, як швидко система обробляє великі обсяги даних і знаходить маршрути;

д) тестування стабільності. Перевірити роботу програми в різних сценаріях, включаючи некоректні дані чи дії користувача.

Методологія тестування:

а) функціональне тестування. Для перевірки кожного модуля використовувалися тестові сценарії, що охоплюють основні функції:

1) завантаження даних OSM і висот;

2) побудова графа дорожньої мережі;

3) знаходження маршрутів за заданими точками;
4) відображення результатів на карті.

б) інтеграційне тестування. Перевірялося, як модулі працюють разом, зокрема інтеграція графа та висот, а також передача даних між компонентами (наприклад, між create_nodes.py і map_and_route.py);

в) тестування продуктивності. Було проведено серію тестів для оцінки швидкості виконання алгоритмів, зокрема обробки графа та пошуку маршруту. Для цього використовувалися великі набори даних з OSM;

г) тестування інтерфейсу. Інтерактивність веб-інтерфейсу перевірялася вручну, а також за допомогою автоматичних інструментів (наприклад, Selenium).

Результати тестування:

а) коректність роботи алгоритмів:
1) алгоритм A* правильно обчислював маршрути для тестових сценаріїв, навіть у випадках із значними ухилами;
2) вага ребер графа коректно враховувала різницю висот і довжину сегментів.

б) інтеграція даних:
1) дані про висоти та OSM інтегрувалися без помилок, а перешкоди (будівлі, водойми) враховувалися під час побудови графа;
2) проблем із обробкою геометричних об'єктів не виявлено.

в) інтерактивність інтерфейсу:
1) інтерфейс стабільно реагував на вибір точок та побудову маршрутів;
2) відображення маркерів і маршрутів відповідало заданим координатам.

г) продуктивність:

1) обробка графа з приблизно 50 000 вузлів і 100 000 ребер займала приблизно 5 хвилин;
2) розрахунок маршруту між довільними точками займав менше 2 секунд, навіть для довгих маршрутів.

д) стабільність:

1) програма коректно працювала з некоректними або відсутніми даними (наприклад, виводила попередження про помилку);
2) випадкові помилки користувача (вибір однієї точки, очищення міток тощо) не призводили до збоїв.

Результати тестування підтвердили, що розроблене програмне забезпечення відповідає поставленим вимогам. Всі модулі працюють коректно, алгоритми забезпечують точні розрахунки, а веб-інтерфейс є надійним і простим у використанні. Система демонструє високу продуктивність навіть з великими наборами даних, що дозволяє використовувати її для реальних задач проектування трубопроводів.

3.6 Інструкція користувача

Перед початком роботи з програмою необхідно виконати кілька підготовчих кроків. Спочатку слід встановити всі необхідні залежності та підготувати вхідні дані. Це забезпечить коректну роботу системи та уникнення помилок під час виконання.

Після завершення підготовчого етапу користувач може приступити до запуску програми. Для цього потрібно виконати головний файл main.py. Цей файл ініціює основні процеси. При запуску програми система розпочне процес отримання даних про висоти рельєфу для вказаної області.

Візуалізація процесу збору даних про висоти представлена на рисунку 3.18. Це дозволяє користувачу спостерігати за прогресом виконання завдання та оцінювати час, необхідний для завершення аналізу обраного діапазону координат. Це може зайняти багато часу, оскільки все залежить від площі обраної території та обмежень API для отримання даних.

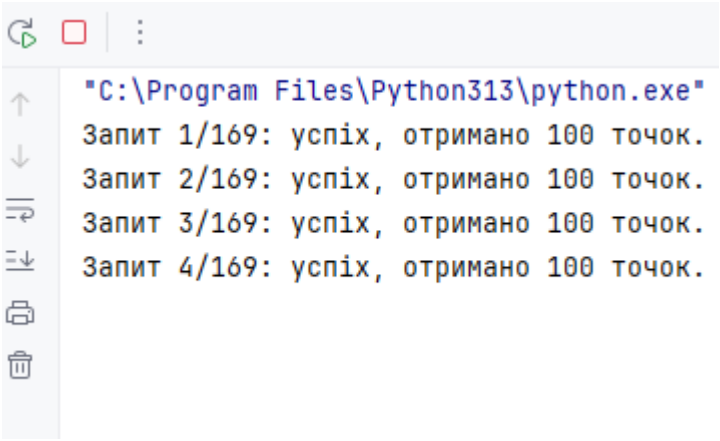

Рисунок 3.18 – Візуалізація процесу збору даних про висоти

Після успішного завершення користувач побачить відповідне повідомлення, а також новий файл, який буде створено в корені проєкту (див. рис. 3.19).

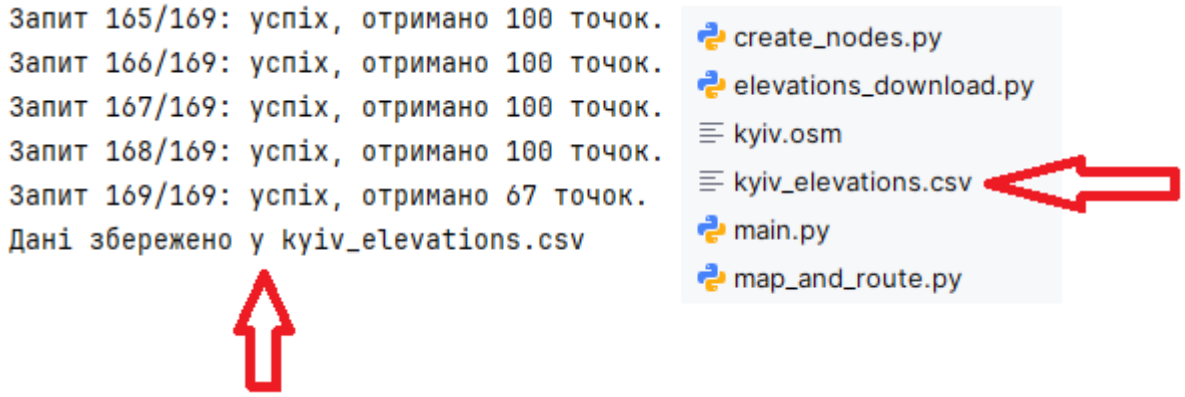

Рисунок 3.19 – Успішне створення файлу з висотами

Одразу після успішного створення файлу з даними про висоти, починається процес обробки OSM-файлу. Цей процес включає: формування полігонів перешкод, обробку доріг і побудову графа. Після успішного внутрішнього опрацювання, буде створено ще 2 нових файли - nodes.csv і graph.json (див. рис. 3.20). Кожен із цих процесів досить ресурсовитратний, тому рекомендується закрити всі програми перед запуском. Користувач може

бачити поточний прогрес відповідного процесу і приблизний час до його завершення. Пояснення позначень прогрес-бара (див. рис. 3.21):

1 – час, який уже минув;

2 – приблизний час до кінця обробки;

3 – скільки за секунду виконується ітерацій.

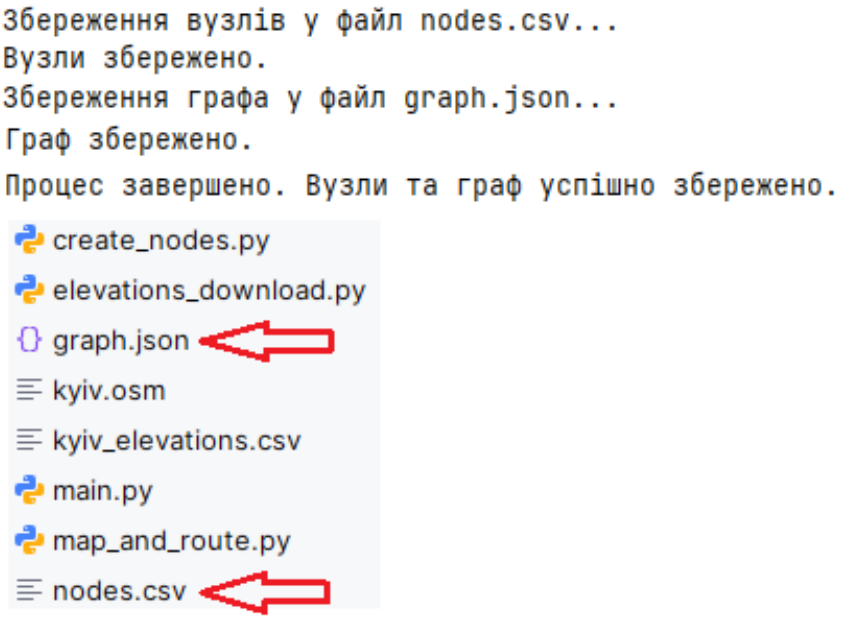

Рисунок 3.20 – Успішне створення файлів nodes.csv та graph.json

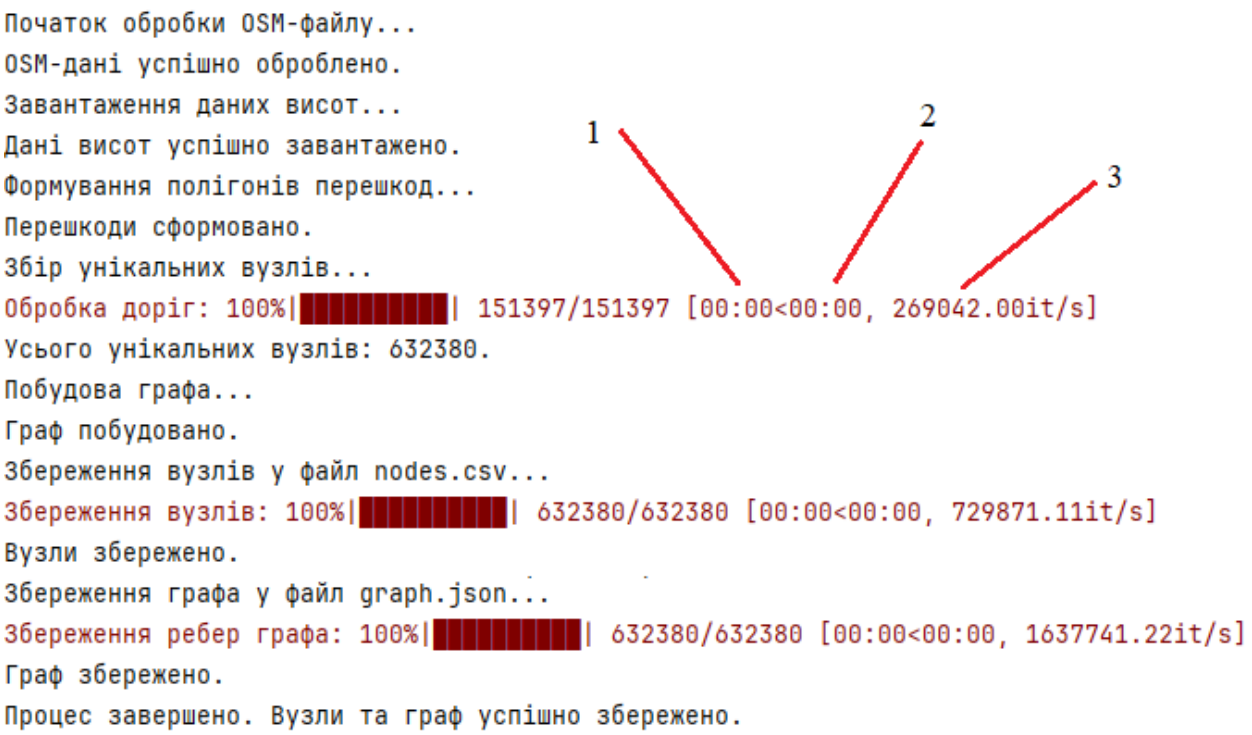

Рисунок 3.21 – Процес обробки OSM-файлу

Після завершення підготовки файлів дані буде оброблено, і веб-сайт (див. рис. 3.22) автоматично відкриється в браузері за замовчуванням. Якщо цього не сталося, введіть в адресний рядок браузера одну з таких адрес:

а) локальний URL: http://localhost:8501;

б) мережева адреса: http://192.168.xx.xxx:8501.

Зверніть увагу, що значення 8501 - це порт, який використовується за замовчуванням.

У консолі додатка буде налагоджувальна інформація, яка допоможе зрозуміти, чи успішно працює додаток. Інформація зображена на рисунку 3.23.

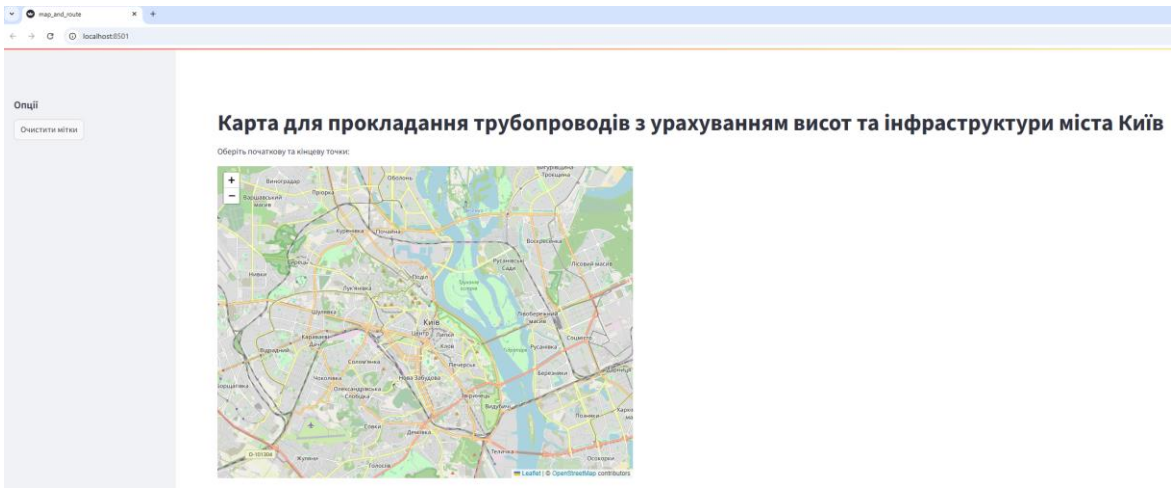

Рисунок 3.22 – Головна сторінка додатку

```
You can now view your Streamlit app in your browser.

Local URL: http://localhost:8501
Network URL: http://192.168.[redacted]:8501

Завантаження вузлів з файлу nodes.csv...
Завантажено 632380 вузлів.
Завантаження графу з файлу graph.json...
Граф завантажено.
```

Рисунок 3.23 – Інформація про те, що додаток успішно запущено

На інтерактивній карті ви можете досліджувати регіони, які вас цікавлять, змінювати масштаб для детального перегляду і переміщатися по карті в будь-якому напрямку. Керування картою інтуїтивно зрозуміле: використовуйте кнопки, мишу або сенсорний екран для зручної взаємодії.

Інструкція щодо користування картою:

а) масштабування карти:

1) для збільшення масштабу натисніть кнопку « + », розташовану на карті (див. рис. 3.24);

2) для зменшення масштабу натисніть кнопку « - » (див. рис. 3.24);

3) ви також можете використовувати колесо миші для плавного наближення або віддалення.

б) переміщення по карті:

1) щоб перемістити карту, затисніть ліву кнопку миші та перетягніть карту в потрібне місце.

в) скидання масштабу або місця розташування:

1) якщо ви загубили потрібну область, натисніть кнопку перезавантаження сторінки або поверніться до вихідного положення вручну.

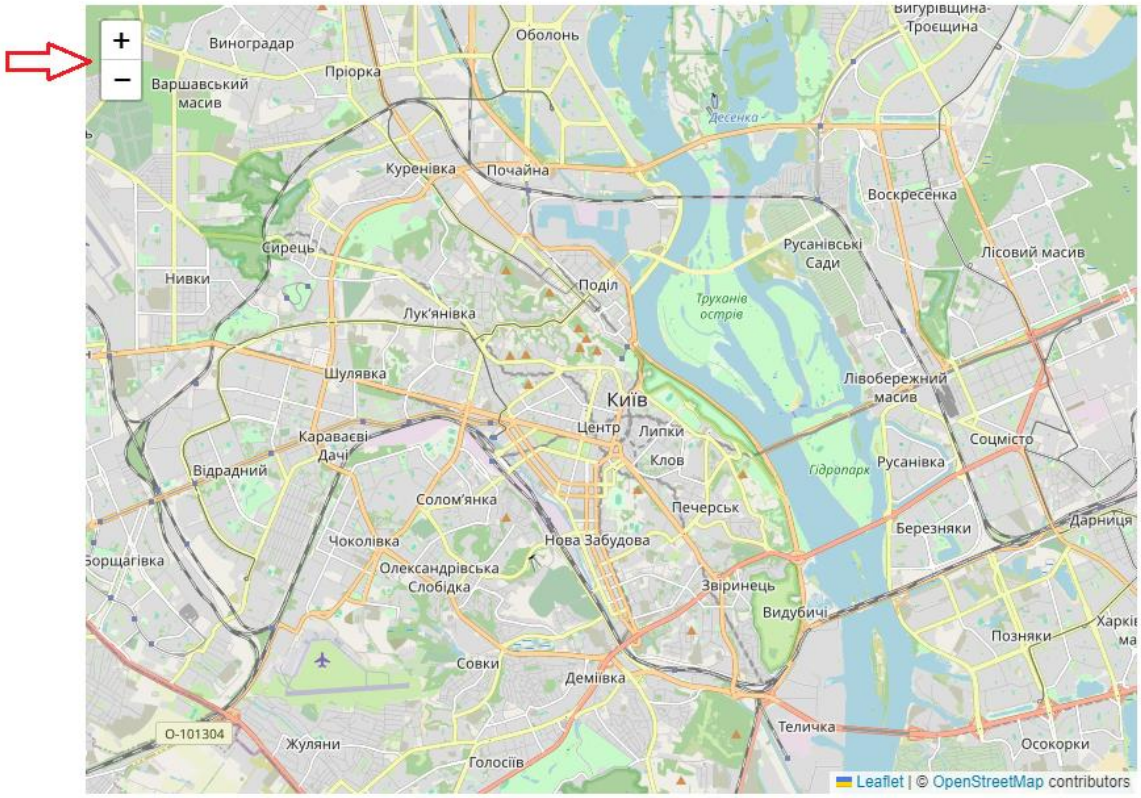

Рисунок 3.24 – Розташування кнопок зміни масштабу на карті

Щойно ви вибрали місце, яке вас цікавить, поставте на ньому початковий маркер (точка 1), натиснувши на карті лівою кнопкою миші. Вигляд маркеру зображено на рисунку 3.25.

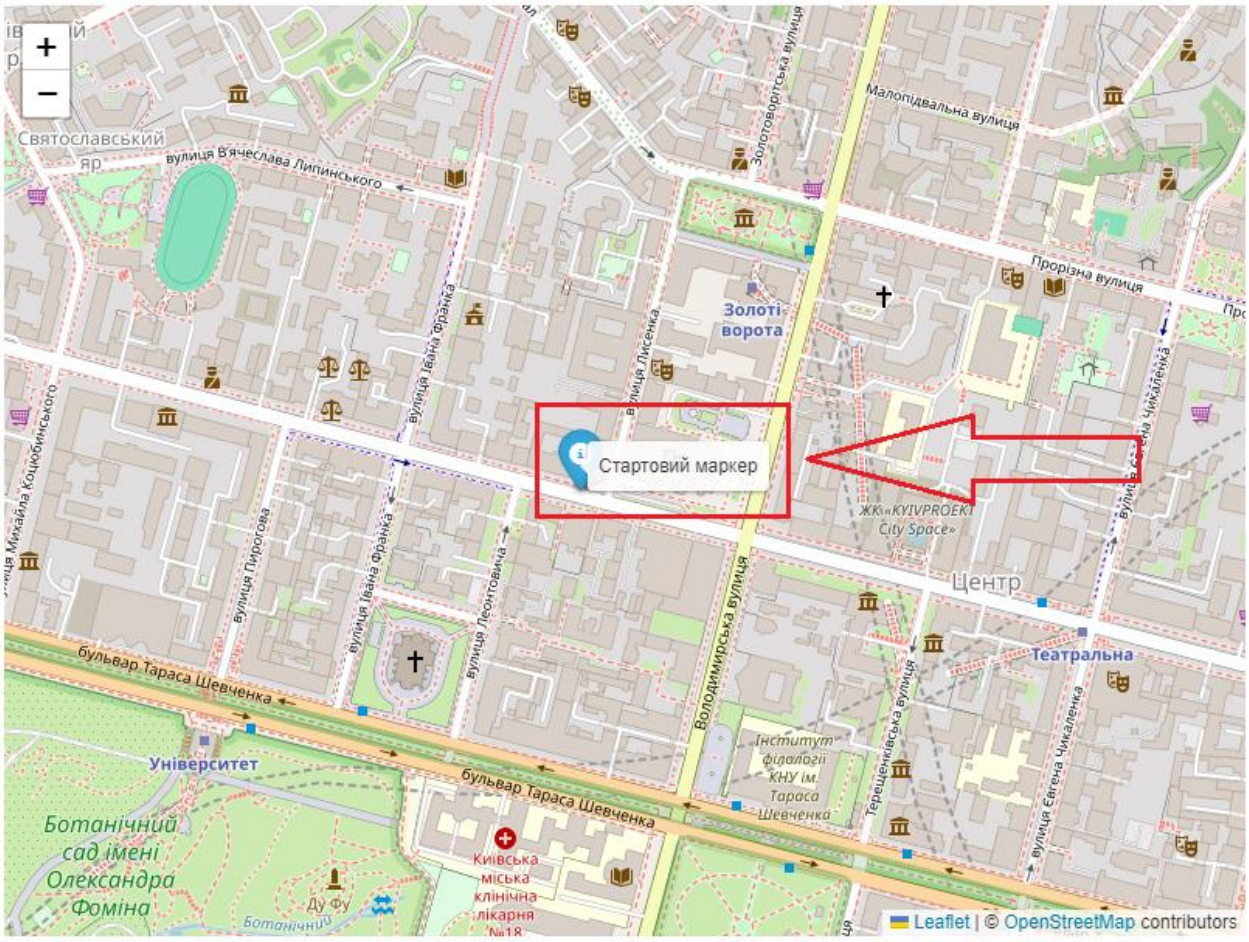

Рисунок 3.25 – Вигляд початкового маркеру

Так само встановіть і кінцевий маркер (точка 2). Після успішного встановлення обох точок, на лівій панелі можна побачити подробиці про їхнє точне розташування (див. рис. 3.26). Синім кольором зображується початкова точка, а червоним - кінцева.

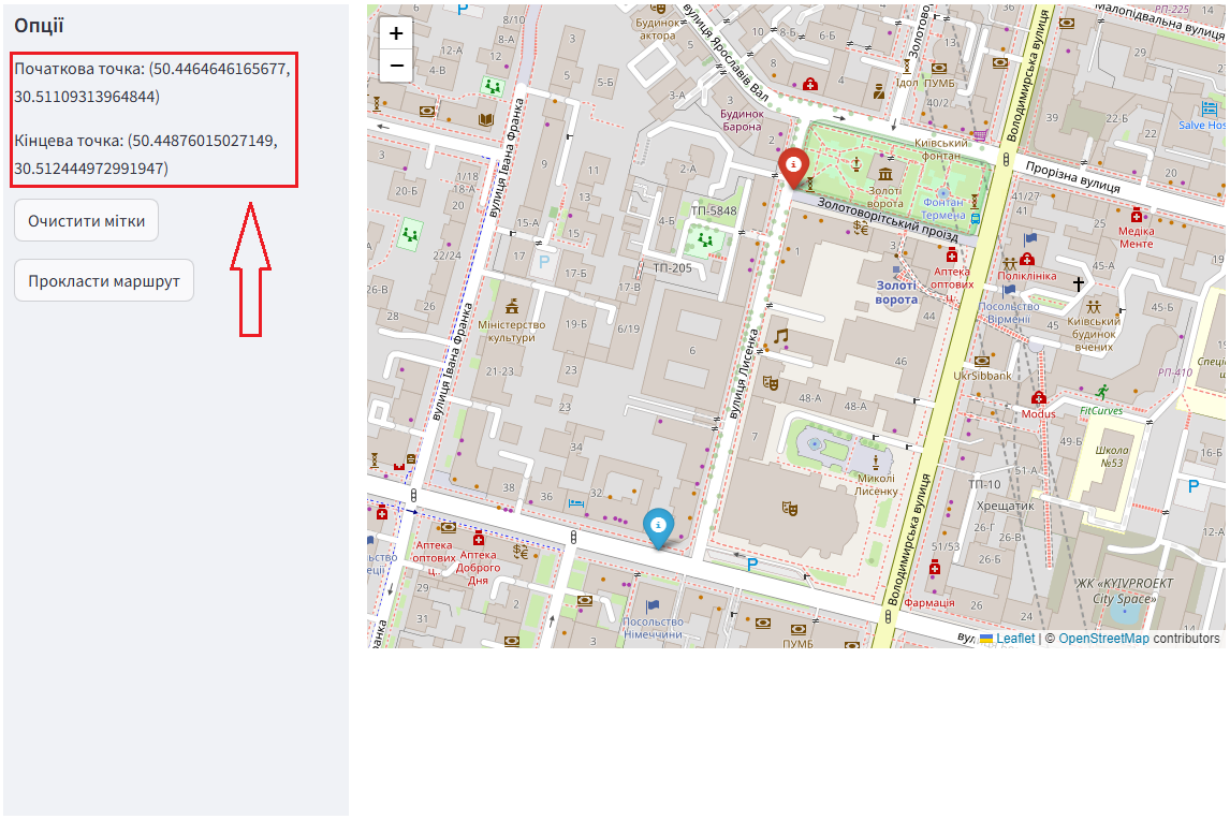

Рисунок 3.26 – Детальна інформація про розташування точок

Щоб побудувати оптимальний маршрут між двома визначеними точками, натисніть кнопку «Прокласти маршрут». Після цього на карті буде відображено криву червоного кольору та деякі деталі (див. рис. 3.27).

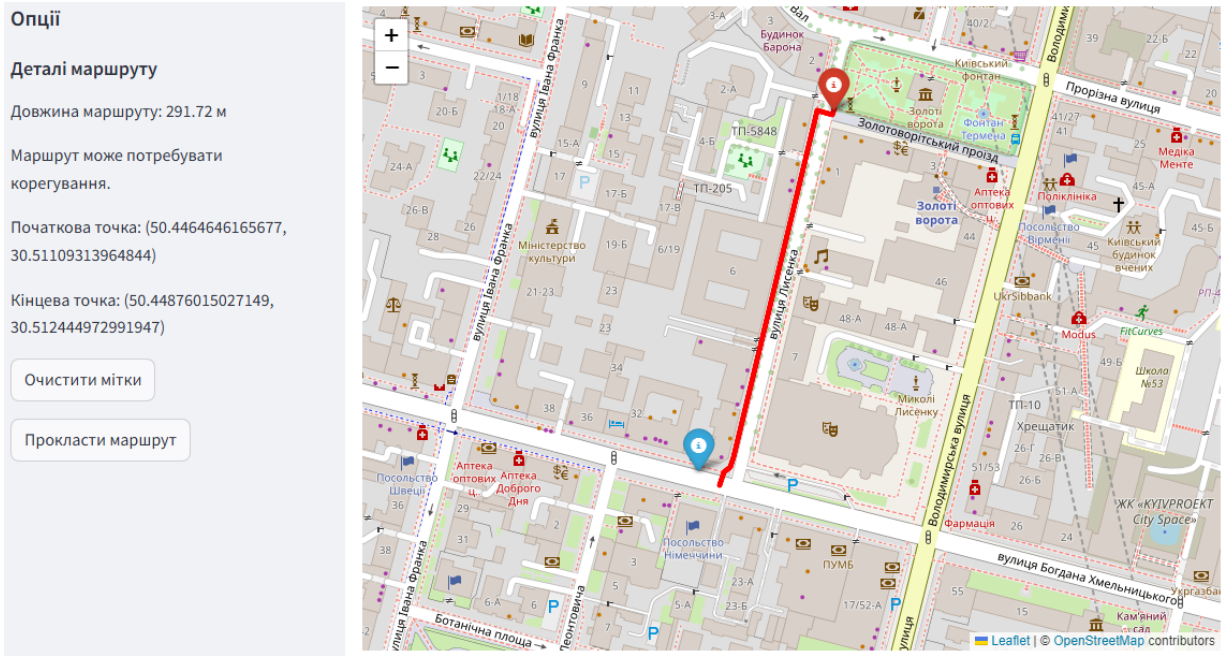

Рисунок 3.27 – Карта з маршрутом і детальною інформацією про нього

Якщо користувач хоче поставити нові точки, варто скористатися кнопкою «Очистити мітки», або оновити сторінку.

ВИСНОВОК

У процесі виконання кваліфікаційної роботи на тему "Розробка системи автоматизації проектування спеціалізованих цифрових моделей рельєфу місцевості" було створено геоінформаційну систему, яка автоматизує ключові етапи проектування трубопровідних мереж з урахуванням рельєфу місцевості та міської інфраструктури. Розроблене програмне забезпечення поєднує сучасні геоінформаційні технології, алгоритмічні підходи та інтерактивний веб-інтерфейс.

У ході виконання роботи досягнуто наступних результатів:

а) обґрунтовано актуальність та поставлено завдання. Проведено аналіз сучасних методів автоматизації проектування інженерних мереж і визначено ключові проблеми, пов’язані з врахуванням рельєфу та інфраструктури. Було сформульовано основні вимоги до розроблюваної геоінформаційної системи, зокрема: інтеграція даних про рельєф, використання картографічної інформації та забезпечення простого й функціонального інтерфейсу;

б) розроблено спеціалізовані алгоритми обробки геопросторових даних. Реалізовано алгоритми для побудови графа дорожньої мережі на основі даних OpenStreetMap та цифрових моделей рельєфу. Особливу увагу приділено врахуванню рельєфу: різниця висот між вузлами враховується при розрахунку ваг ребер графа, що дозволяє будувати оптимальні маршрути для прокладання трубопроводів;

в) реалізовано автоматизовану систему з використанням ГІС-технологій. Розроблене програмне забезпечення інтегрує геоінформаційні дані та алгоритми оптимізації. Для завантаження даних про рельєф використано API OpenTopodata, а для обробки картографічних даних — бібліотеку Osmium. Інтерактивний

інтерфейс створено за допомогою Streamlit і Folium, що забезпечує можливість роботи з картою в режимі реального часу;

г) здійснено тестування програмного забезпечення. Проведено функціональне, інтеграційне та продуктивне тестування програми з використанням реальних даних рельєфу та інфраструктури Києва. Результати підтвердили, що система коректно обробляє дані, враховує перешкоди (будівлі, водойми) та швидко знаходить оптимальні маршрути для прокладання інженерних мереж.

д) створено веб-інтерфейс із підтримкою інтерактивної взаємодії. Веб-інтерфейс забезпечує зручність використання: користувачі можуть обирати точки на карті, будувати маршрути та отримувати інформацію про довжину шляху.

Розроблена геоінформаційна система забезпечує автоматизацію проектування спеціалізованих цифрових моделей місцевості, що є важливим етапом проектування інженерних мереж. Система інтегрує сучасні алгоритми та ГІС-технології, забезпечуючи точність і надійність розрахунків та зручність використання для кінцевих користувачів.

Таким чином, виконана робота підтверджує актуальність обраної теми та демонструє ефективність запропонованого підходу до автоматизації проектування інженерних мереж. Розроблена система може бути використана в практичній діяльності, а її алгоритмічна основа та архітектура відкривають подальші можливості для вдосконалення.

ПЕРЕЛІК ПОСИЛАНЬ

1. Python Geospatial Development / Evans A., Pereira R., 2018. – 494 p. – (Packt Publishing).
2. The Art of Computer Programming, Volume 1: Fundamental Algorithms / Knuth D. E., 1997. – 650 p. – (Addison-Wesley).
3. HTML and CSS: Design and Build Websites / Duckett J., 2011. – 514 p. – (Wiley).
4. Osmium Library [Електронний ресурс] – Режим доступу до ресурсу: https://osmcode.org/libosmium/.
5. Folium Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://python-visualization.github.io/folium/.
6. Shapely Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://shapely.readthedocs.io/.
7. OpenTopodata API [Електронний ресурс] – Режим доступу до ресурсу: https://www.opentopodata.org/.
8. PostGIS Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://postgis.net/documentation/.
9. OpenStreetMap Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://wiki.openstreetmap.org/.
10. JavaScript and jQuery: Interactive Front-End Web Development / Duckett J., 2014. – 640 p. – (Wiley).

Додаток А – Код програми

Файл main.py:

```
import os
import subprocess


def main():
    if not os.path.exists("kyiv_elevations.csv"):
        print("Файл kyiv_elevations.csv не знайдено. Завантаження
даних...")
        subprocess.run(["python", "elevations_download.py"])


    if not (os.path.exists("graph.json") and
os.path.exists("nodes.csv")):
        print("Файли graph.json та nodes.csv не знайдено. Створення
вузлів...")
        subprocess.run(["python", "create_nodes.py"])


    print("Запуск інтерфейсу для побудови маршруту...")
    subprocess.run(["streamlit", "run", "map_and_route.py"])


if __name__ == "__main__":
    main()
```

Файл elevations_download.py:

```
import numpy as np
import requests
import time
import csv

min_lat, max_lat = 50.3, 50.6
min_lng, max_lng = 30.3, 30.8
step = 0.003
```

```
lats = np.arange(min_lat, max_lat, step)
lngs = np.arange(min_lng, max_lng, step)
locations = [f"{lat},{lng}" for lat in lats for lng in lngs]

chunks = [locations[i:i + 100] for i in range(0, len(locations), 100)]

url = "https://api.opentopodata.org/v1/srtm30m"

all_results = []

for i, chunk in enumerate(chunks):
    params = {
        "locations": "|".join(chunk),
        "interpolation": "cubic",
    }
    response = requests.get(url, params=params)

    if response.status_code == 200:
        results = response.json().get("results", [])
        all_results.extend(results)
        print(f"Запит {i + 1}/{len(chunks)}: успіх, отримано
{len(results)} точок.")
    else:
        print(f"Запит {i + 1}/{len(chunks)}: помилка
{response.status_code}, {response.text}")

    time.sleep(5)

output_file = "kyiv_elevations.csv"
with open(output_file, "w", newline="") as f:
    writer = csv.writer(f)
```

```
    writer.writerow(["Latitude", "Longitude", "Elevation"])
    for result in all_results:
        lat = result["location"]["lat"]
        lng = result["location"]["lng"]
        elevation = result["elevation"]
        writer.writerow([lat, lng, elevation])

print(f"Дані збережено у {output_file}")
```

Файл create_nodes.py:

```
import osmium
import json
import csv
import math
from shapely.geometry import Polygon, LineString
from scipy.spatial import KDTree
import pandas as pd
from tqdm import tqdm
from multiprocessing import Pool
from rtree import index

class OsmHandler(osmium.SimpleHandler):
    def __init__(self):
        super(OsmHandler, self).__init__()
        self.nodes = {}
        self.highways = []
        self.buildings = []
        self.waters = []

    def node(self, n):
        self.nodes[n.id] = (n.location.lat, n.location.lon)
```

```
    def way(self, w):
        if len(w.nodes) < 2:
            return
        tags = dict(w.tags)

        coords = []
        for nd in w.nodes:
            if nd.ref in self.nodes:
                coords.append(self.nodes[nd.ref])

        if 'highway' in tags and len(coords) > 1:
            self.highways.append(coords)
        elif 'building' in tags and len(coords) > 2 and coords[0] ==
coords[-1]:
            self.buildings.append(coords)
        elif tags.get('natural') == 'water':
            if coords[0] == coords[-1] and len(coords) > 2:
                self.waters.append(('polygon', coords))
            else:
                self.waters.append(('line', coords))

def load_elevation_data(file="kyiv_elevations.csv"):
    print("Завантаження даних висот...")
    try:
        df = pd.read_csv(file)
        coords = df[["Latitude", "Longitude"]].values
        elev = df["Elevation"].values
        tree = KDTree(coords)
        print("Дані висот успішно завантажено.")
        return tree, elev, coords
    except FileNotFoundError:
```

```
        print("Файл з даними висот не знайдено.")
        return None, None, None


def get_elevation_for_point(point, elev_tree, elev_values,
elev_coords):
    if elev_tree is None:
        return 0
    dist, idx = elev_tree.query([point[0], point[1]])
    return elev_values[idx]


def haversine(p1, p2):
    R = 6371e3
    lat1, lon1 = math.radians(p1[0]), math.radians(p1[1])
    lat2, lon2 = math.radians(p2[0]), math.radians(p2[1])
    dlat = lat2 - lat1
    dlon = p2[1] - p1[1]
    a = math.sin(dlat / 2) ** 2 + math.cos(lat1) * math.cos(lat2) *
math.sin(dlon / 2) ** 2
    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))
    return R * c


def build_spatial_index(polygons):
    idx = index.Index()
    for i, poly in enumerate(polygons):
        idx.insert(i, poly.bounds)
    return idx


def intersects_obstacles_optimized(line, building_polygons,
water_polygons, water_lines, building_index, water_index,
water_lines_index):
    for i in building_index.intersection(line.bounds):
        if line.intersects(building_polygons[i]):
            return True
```

```
    for i in water_index.intersection(line.bounds):
        if line.intersects(water_polygons[i]):
            return True

    for i in water_lines_index.intersection(line.bounds):
        if line.intersects(water_lines[i]):
            return True

    return False

def process_road_chunk(roads_chunk, unique_nodes, building_polygons,
water_polygons, water_lines, building_index, water_index,
water_lines_index, elev_tree, elev_values, elev_coords):
    local_graph = [[] for _ in range(len(unique_nodes))]
    for road in roads_chunk:
        for i in range(len(road) - 1):
            p1 = road[i]
            p2 = road[i + 1]
            u = unique_nodes[p1]
            v = unique_nodes[p2]

            line = LineString([(p1[1], p1[0]), (p2[1], p2[0])])
            if intersects_obstacles_optimized(line, building_polygons,
water_polygons, water_lines, building_index, water_index,
water_lines_index):
                continue

            dist = haversine(p1, p2)
            elev_p1 = get_elevation_for_point(p1, elev_tree,
elev_values, elev_coords)
            elev_p2 = get_elevation_for_point(p2, elev_tree,
elev_values, elev_coords)
```

```
            elevation_diff = abs(elev_p2 - elev_p1)

            weight = dist * (1 + elevation_diff / 100.0)

            local_graph[u].append((v, weight))
            local_graph[v].append((u, weight))
    return local_graph

def build_graph(handler, elev_tree, elev_values, elev_coords):
    print("Формування полігонів перешкод...")
    building_polygons = [Polygon([(lon, lat) for (lat, lon) in c]) for
c in handler.buildings]

    water_polygons = []
    water_lines = []
    for wt, coords in handler.waters:
        if wt == 'polygon':
            water_polygons.append(Polygon([(lon, lat) for (lat, lon)
in coords]))
        else:
            water_lines.append(LineString([(lon, lat) for (lat, lon)
in coords]))

    print("Перешкоди сформовано.")

    print("Збір унікальних вузлів...")
    unique_nodes = {}
    current_index = 0

    for road in tqdm(handler.highways, desc="Обробка доріг"):
        for (lat, lon) in road:
            if (lat, lon) not in unique_nodes:
```

```
                unique_nodes[(lat, lon)] = current_index
                current_index += 1

    print(f"Усього унікальних вузлів: {len(unique_nodes)}.")

    nodes = [None] * len(unique_nodes)
    for (lat, lon), idx in unique_nodes.items():
        nodes[idx] = (lat, lon)

    print("Побудова графа...")
    graph = [[] for _ in range(len(nodes))]

    building_index = build_spatial_index(building_polygons)
    water_index = build_spatial_index(water_polygons)
    water_lines_index = build_spatial_index(water_lines)

    with Pool() as pool:
        chunk_size = len(handler.highways) // pool._processes
        road_chunks = [handler.highways[i:i + chunk_size] for i in
range(0, len(handler.highways), chunk_size)]
        results = pool.starmap(process_road_chunk, [
            (chunk, unique_nodes, building_polygons, water_polygons,
water_lines, building_index, water_index, water_lines_index,
elev_tree, elev_values, elev_coords)
            for chunk in road_chunks
        ])

        for local_graph in results:
            for i, adj in enumerate(local_graph):
                graph[i].extend(adj)

    print("Граф побудовано.")
```

```
    return nodes, graph

def save_nodes(nodes, filename="nodes.csv"):
    print(f"Збереження вузлів у файл {filename}...")
    with open(filename, "w", newline="", encoding="utf-8") as f:
        writer = csv.writer(f)
        writer.writerow(["index", "lat", "lon"])
        for i, (lat, lon) in enumerate(tqdm(nodes, desc="Збереження
вузлів")):
            writer.writerow([i, lat, lon])
    print("Вузли збережено.")

def save_graph(graph, filename="graph.json"):
    print(f"Збереження графа у файл {filename}...")
    json_graph = []
    for adj in tqdm(graph, desc="Збереження ребер графа"):
        json_graph.append([{"node": v, "weight": w} for v, w in adj])

    with open(filename, "w", encoding="utf-8") as f:
        json.dump(json_graph, f, indent=2)
    print("Граф збережено.")

if __name__ == "__main__":
    print("Початок обробки OSM-файлу...")
    handler = OsmHandler()
    handler.apply_file("kyiv.osm", locations=True)

    print("OSM-дані успішно оброблено.")

    elev_tree, elev_values, elev_coords =
load_elevation_data("kyiv_elevations.csv")
```

```
    nodes, graph = build_graph(handler, elev_tree, elev_values,
elev_coords)


    save_nodes(nodes, "nodes.csv")
    save_graph(graph, "graph.json")


    print("Процес завершено. Вузли та граф успішно збережено.")
```

Файл map_androute.py:

```
import math
import heapq
import json
import csv
import streamlit as st
import folium
from streamlit_folium import st_folium
from scipy.spatial import KDTree

def load_nodes(filename="nodes.csv"):
    print(f"Завантаження вузлів з файлу {filename}...")
    nodes = []
    with open(filename, "r", encoding="utf-8") as f:
        reader = csv.DictReader(f)
        for row in reader:
            nodes.append((float(row["lat"]), float(row["lon"])))
    print(f"Завантажено {len(nodes)} вузлів.")
    return nodes

@st.cache_data
def load_graph(filename="graph.json"):
    print(f"Завантаження графу з файлу {filename}...")
    with open(filename, "r", encoding="utf-8") as f:
```

```
        graph = json.load(f)
    print(f"Граф завантажено.")
    return graph

def haversine(p1, p2):
    R = 6371e3
    lat1, lon1 = math.radians(p1[0]), math.radians(p1[1])
    lat2, lon2 = math.radians(p2[0]), math.radians(p2[1])
    dlat = lat2 - lat1
    dlon = dlon = lon2 - lon1
    a = math.sin(dlat / 2)**2 +
math.cos(lat1)*math.cos(lat2)*math.sin(dlon / 2)**2
    c = 2*math.atan2(math.sqrt(a), math.sqrt(1 - a))
    return R * c

def get_nearest_node(lat, lon, nodes):
    coords = [(n[0], n[1]) for n in nodes]
    tree = KDTree(coords)
    dist, idx = tree.query([lat, lon])
    return idx

def a_star(graph, nodes, start, end):
    def heuristic(u, end):
        return haversine(nodes[u], nodes[end])

    dist = [math.inf] * len(graph)
    dist[start] = 0
    prev = [-1] * len(graph)
    heap = [(heuristic(start, end), start)]

    while heap:
        current_f, u = heapq.heappop(heap)
```

```
            if u == end:
                break
            if current_f > dist[u] + heuristic(u, end):
                continue
            for neighbor in graph[u]:
                v, w = neighbor["node"], neighbor["weight"]
                new_dist = dist[u] + w
                if new_dist < dist[v]:
                    dist[v] = new_dist
                    prev[v] = u
                    heapq.heappush(heap, (new_dist + heuristic(v, end),
v))

    if dist[end] == math.inf:
        return []

    path = []
    cur = end
    while cur != -1:
        path.append(cur)
        cur = prev[cur]
    path.reverse()
    return path

st.set_page_config(layout="wide")
st.title("Карта для прокладання трубопроводів з урахуванням висот та
інфраструктури міста Київ")
st.write("Оберіть початкову та кінцеву точки:")

nodes = load_nodes("nodes.csv")
graph = load_graph("graph.json")
```

```
if "markers" not in st.session_state:
    st.session_state["markers"] = []
if "route" not in st.session_state:
    st.session_state["route"] = None

m = folium.Map(location=[50.45, 30.52], zoom_start=12)

if len(st.session_state["markers"]) > 0:
    folium.Marker(location=st.session_state["markers"][0],
tooltip="Стартовий маркер", icon=folium.Icon(color="blue")).add_to(m)
if len(st.session_state["markers"]) > 1:
    folium.Marker(location=st.session_state["markers"][1],
tooltip="Кінцевий маркер", icon=folium.Icon(color="red")).add_to(m)

if st.session_state["route"] is not None:
    folium.PolyLine(st.session_state["route"], color="red",
weight=5).add_to(m)

map_data = st_folium(m, width=800, height=600)

if map_data and map_data.get("last_clicked"):
    clicked_coords = map_data["last_clicked"]
    if len(st.session_state["markers"]) < 2:
        st.session_state["markers"].append((clicked_coords["lat"],
clicked_coords["lng"]))

with st.sidebar:
    st.header("Опції")
    if st.session_state.get("route_length") is not None:
        st.subheader("Деталі маршруту")
        st.write(f"Довжина маршруту:
{st.session_state['route_length']:.2f} м")
        st.write(st.session_state.get("details"))
```

```
    if len(st.session_state["markers"]) > 0:
        st.write(f"Початкова точка: {st.session_state['markers'][0]}")
    if len(st.session_state["markers"]) > 1:
        st.write(f"Кінцева точка: {st.session_state['markers'][1]}")
    if st.button("Очистити мітки"):
        st.session_state["markers"] = []
        st.session_state["route"] = None

    if len(st.session_state["markers"]) == 2:
        if st.button("Прокласти маршрут") and
st.session_state.get("route") is None:
            start, end = st.session_state["markers"]
            st.write(f"Початкова точка: {start}")
            st.write(f"Кінцева точка: {end}")

            start_node = get_nearest_node(start[0], start[1], nodes)
            end_node = get_nearest_node(end[0], end[1], nodes)
            path_nodes = a_star(graph, nodes, start_node, end_node)
            if path_nodes:
                route_coords = [nodes[i] for i in path_nodes]
                st.session_state["route"] = route_coords

                route_length = sum(
                    haversine(nodes[path_nodes[i]],
nodes[path_nodes[i+1]])
                    for i in range(len(path_nodes) - 1)
                )

                st.session_state["route_length"] = route_length
                st.session_state["details"] = "Маршрут може
потребувати корегування."
            else:
```

```
            st.error("Маршрут не знайдено.")
```