МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМ. В. ДАЛЯ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА ЕЛЕКТРОНІКИ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК ТА ІНЖЕНЕРІЇ

До захисту допускається
Т.в.о. завідувача кафедри
_____ Сафонова С.О.
«_____»_____ 2020р.

МАГІСТЕРСЬКА РОБОТА

НА ТЕМУ:

ДОСЛІДЖЕННЯ ТА УДОСКОНАЛЕННЯ МУРАШИНОГО АЛГОРИТМУ ДЛЯ ОПТИМІЗАЦІЇ ТУРИСТИЧНОГО МАРШРУТУ

Освітньо-кваліфікаційний рівень «Магістр»
Спеціальність 122 «Комп'ютерні науки»

Науковий керівник роботи:	_____ (підпис)	Є.В. Щербаков (ініціали, прізвище)
Консультант з охорони праці:	_____ (підпис)	Я.О. Критська (ініціали, прізвище)
Студент:	_____ (підпис)	В.В. Смородa (ініціали, прізвище)
Група:		КН-18дм

Сєвєродонецьк 2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Факультет Інформаційних технологій та електроніки
Кафедра Комп'ютерних наук та інженерії
Освітньо-кваліфікаційний рівень магістр
Напрям підготовки _____
(шифр і назва)
Спеціальність 122 Комп'ютерні науки
(шифр і назва)

ЗАТВЕРДЖУЮ:

Т.в.о завідувача
кафедри _____
_____ С.О. Сафонова
«_____» _____20____р.

ЗАВДАННЯ
НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Смороді Владиславу Володимировичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження та удосконалення мурашиного алгоритму для оптимізації туристичного маршруту
керівник проекту (роботи) Щербаков Євген Васильович, к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від " 11 " 10 2019 року № 135/15.15
2. Строк подання студентом проекту (роботи) 10.01.2020
3. Вихідні дані до роботи Матеріали науково-дослідної практики, мова програмування – Java, мова розмітки XML, середовище розробки – Android Studio

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
Теоретико-методологічні аспекти
Огляд сучасних робіт на тему оптимізації маршрутів
Дослідження алгоритмів прокладання туристичного маршруту
Аналіз та порівняння існуючих додатків з використанням навігації
Аналіз та удосконалення мурашиного алгоритму для оптимізації туристичного маршруту
Практична реалізація Android-додатку
Охорона праці та безпека в надзвичайних ситуаціях
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Електронні плакати

6. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці	Критська Я.О., асистент		

7. Дата видачі завдання 02.09.2019

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Аналітичний огляд літератури за темою роботи	02.09.19-29.09.19	
2	Огляд сучасних робіт на тему оптимізації маршрутів	30.10.19-13.10.19	
3	Дослідження алгоритмів прокладання туристичного маршруту	14.10.19-27.10.19	
4	Аналіз та порівняння існуючих додатків з використанням навігації	28.10.19-03.11.19	
5	Аналіз та удосконалення мурашиного алгоритму для оптимізації туристичного маршруту	04.11.19-11.11.19	
6	Практична реалізація Android-додатку	13.11.19-01.12.19	
7	Охорона праці та безпека в надзвичайних ситуаціях	02.12.19-22.12.19	
8	Оформлення пояснювальної записки	23.12.19-02.01.20	
9	Оформлення презентації роботи	03.01.20-10.01.20	

Студент _____ Смородa В.В.
(підпис) (прізвище та ініціали)

Науковий керівник _____ Щербаков Є.В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Смородa В. В. Дослідження та удосконалення мурашиного алгоритму для оптимізації туристичного маршруту.

Метою дослідження являється удосконалення мурашиного алгоритму та подальше його використання для розробки програмного додатка.

Проаналізовано методи розв'язання оптимізаційних задач дискретної математики.

По результатах дослідження був розроблений додаток для мобільних пристроїв під управлінням операційної системи Android, який дозволяє прокладати найкоротший маршрут через декілька точок на мапі.

Ключові слова: мурашиний алгоритм, задача комівояжера, java, android studio, android, мапи.

АННОТАЦИЯ

Смородa В. В. Исследование и совершенствование муравьиного алгоритма для оптимизации туристического маршрута.

Целью исследования является совершенствование муравьиного алгоритма и дальнейшее его использование для разработки программного приложения.

Проанализированы методы решения оптимизационных задач дискретной математики.

По результатам исследования было разработано приложение для мобильных устройств под управлением операционной системы Android, которое позволяет прокладывать кратчайший маршрут через несколько точек на карте.

Ключевые слова: муравьиный алгоритм задача коммивояжера, java, android studio, android, карты.

ABSTRACT

Smoroda V. V. Research and improvement of ant colony optimization algorithm to optimize the tourist route.

The purpose of the study is to improve the ant algorhythm and its further use for the development of software applications.

Methods for solving optimization problems of discrete mathematics are analyzed.

As a result of the research, a software application was developed for mobile devices running the Android operating system, which allows you to build a path between several points on the map in order to build the shortest route.

Key words: ant algorithm for the traveling salesman problem, java, android studio, android, maps.

ЗМІСТ

ВСТУП

Актуальність теми: Розвиток туристичних шляхів визначається зростанням міських, регіональних територій, зростом кількості населення, розміщенням різних туристичних функціональних локацій. Розвиток туризму залежить від якості організації і технічних можливостей сучасних транспортних систем. Побудова ефективної для міста та міжрегіональних сполучень транспортної системи – це дуже складна проблема для України, яка містить в собі ряд різних завдань, котрі різняться за значимістю, складністю та трудомісткістю, серед яких встановлення маршрутів до туристичних місць, розподіл маршрутів поміж перевізників, створення розкладу та оптимізація режимів руху на маршруті. Становлення мережі туристичних маршрутів є важливим кроком у розвитку ефективної транспортної системи туристичної галузі. Наскільки добре розвинена маршрутна мережа, наскільки добре та гармонійно вона інтегрована у транспортну мережу відвідувань туристичних центрів, залежить задоволеність туристів транспортом та ефективність діяльності туристичних компаній.

Одним із методів вирішення задач пошуку оптимальних маршрутів на графіках є алгоритм оптимізації колонії мурашок (ACO), який вирішує задачу оптимізації дискретної математики, а саме задачу комівояжера.

Оптимізаційні задачі дискретної математики находять широке застосування на практиці, при укладанні розкладу робіт, будівництві транспортних мереж, у логістиці при укладанні оптимальних планів перевозу вантажів, у електроніці під час розробки оптимальної стратегії виготовлення інтегральних мікросхем тощо. Проблемою є те, що для точного знаходження оптимального рішення дискретних задач існує лише один метод – метод повного перебору варіантів та його модифікації, зокрема метод гілок і границь. У реальних задачах високої розмірності кількість таких варіантів є дуже великою, і повний перебір варіантів за розумний час становиться неможливим навіть використовуючи сучасні суперкомп’ютери. Тому розробка наближених методів для розв’язання оптимізаційних задач дискретної математики, які дозволяють отримувати приємні рішення за приємний час обчислень, є актуальною проблемою.

На теперішній час у сучасній кібернетиці з успіхом використовується біологічний підхід, який полягає у використанні виявлених механізмів і методів, за допомогою яких Природа вміє розв’язувати складні задачі еволюції і діяльності живих організмів. В результаті

такого підходу розроблені такі відомі і популярні методи, як генетичні алгоритми та нейронні мережі, що широко використовуються у сучасних інтелектуальних системах.

При розробці біологічно-орієнтованих алгоритмів увагу дослідників останніми роками привертає так званий мурашиний алгоритм, заснований на моделюванні поведінки колонії мурашок. Але до теперішнього часу відсутні у широкому вжитку комп'ютерні програми, які можна використовувати для навчального процесу і проведення широких досліджень. Тому розробка таких програм, вивчення їх ефективності є актуальною задачею.

Мета і задачі дослідження: удосконалення мурашиного алгоритму та розробка програмного додатка, який використовує мурашиний алгоритм для оптимізації туристичного маршруту.

Об'єкт досліджень: методи дискретної математики.

Методологічна та теоретична основа дослідження: використання методів біоніки для розв'язання задач прикладної математики.

Методи дослідження: методи біоніки, програмування, комп'ютерного експерименту.

Практичне значення отриманих результатів: результати, що отримані, можуть бути використані для розв'язання практичних задач, що пов'язані із задачею комівояжера, зокрема оптимізації туристичного маршруту.

Структура та обсяг роботи. Магістерська робота складається зі вступу, 4 розділів, висновків на 67 сторінках, переліку використаних джерел з 45 найменувань на 4 сторінках, двох додатків на 25 сторінках. Загальний обсяг роботи складає 105 сторінок. В магістерській роботі міститься 2 таблиці, 22 рисунка.

1 ОГЛЯД ЛІТЕРАТУРИ

1.1 Аналіз літератури та огляд сучасних робіт на тему оптимізації маршрутів

У нашій країні розвитком маршрутного транспорту та маршрутів почали цікавитись наприкінці XIX ст. Деякі перші роботи в цій галузі належать Ларіонову В.С. та Полякову А.А. [2, 1]. Узагальнюючи роботи того часу, можна отримати висновок, що головна увага дослідників приділяється формуванню набору вимог до схем маршрутів. Методи побудови маршрутів руху та рекомендації щодо формування маршрутних мереж у роботах того періоду розглядаються поверхово, були сформульовані тільки загальні положення про їх проектування. Але вже тоді автори мали сформульоване припущення, що для реалізації потреб населення міста у перевезеннях, мережа маршрутів повинна базуватися на даних про фактичні переміщення населення в межах міста або поміж населених пунктів, тобто матриці переписки пасажирів. У роботах того часу вперше було сформульовано критерії за якими слід оптимізувати міські маршрутні мережі, такі як найкоротший маршрут між початковою та кінцевою точками маршруту, мінімальний час, витрачений на рух усіма пасажирами.

У середині XX століття під час розвитку економіко-математичних методів розпочався новий період у формуванні наукових пізнань про роботу систем пасажирського транспорту. Найвідомішими авторами того періоду є Ольховський С.Ю. та Яворський В. тощо [3 - 5].

Підходи до проектування раціональних туристичних шляхів пасажирського транспорту діляться на три групи:

а) автоматизоване проектування туристичних маршрутів використовуючи формалізовані математичні моделі;

б) експертна оцінка отриманих результатів фахівцем і часткова автоматизація процесу побудови туристичних маршрутів;

в) прийняття рішень використовуючи досвід і неформалізований аналіз експертів.

1.1.1 Планування туристичних маршрутів з використанням методу пошуку табу

У 2019 році Міністерство туризму Індонезії знаходиться в процесі вдосконалення програми Go Digital для індустріальної епохи 4.0, де інтернет став одним із способів визначення туристичних напрямків. Проте в даний час туристи все ще відчувають труднощі з отриманням детальної та повної інформації про туристичні напрямки, відвідуючи кілька

напрямків в одній поїздці. Туристи все ще відчувають труднощі з оцінкою відстані і часу, необхідного для туризму самостійно, без необхідності залежати від туристичних агентів. Ці проблеми часто називають проблемами комівояжера (TSP). Тому було запропоновано рішення даної задачі TSP у вигляді системи планування та пошуку маршруту туристського руху з використанням методу табу-пошуку, який дозволяє туристам знайти оптимальне рішення, виходячи з часу в дорозі, експлуатаційних годин туристичної пам'ятки, а також ліміту часу відвідування в добу [41]. Розрахунки в методі пошуку табу об'єднуються з концепцією MAUT (Multi-Attribute Utility Theory) для визначення оптимального туру на основі декількох критеріїв: популярності, вартості та кількості відвідуваних пам'яток.

Результати тестування методу пошуку табу були порівняні з методом Світлячка(Firefly method). Результат показує, що метод пошуку табу краще, ніж метод firefly, де спостерігається збільшення точності на 48% у розрахунку значень пристосованості, 47% у середньому часу виконання і 27% в кількості турів, які необхідно відвідати протягом 3 днів відвідування туру.

1.1.2 Оптимізований алгоритм маршрутизації для мереж спеціальних засобів пересування

У цій статті було представлено двоякий підхід, що передбачає розробку нової метрики маршруту для зв'язку мереж спеціальних транспортних засобів (МСТЗ), яка враховує такі важливі параметри, як сила сигналу; потужність передачі, частота і втрати шляху [42]. Далі було представлено вдосконалений метод оптимізації маршруту на основі генетичного алгоритму (IGAROT), який гарантує кращу маршрутизацію в МСТЗ. Було використано IGAROT для визначення оптимальних маршрутів, необхідних для ефективного зв'язку дорожніх аномалій між транспортними засобами в МСТЗ. Ефективність запропонованого ними алгоритму порівнювалася з добре відомим традиційним генетичним алгоритмом (ГА) оптимізації маршруту в тих же умовах моделювання. Ґрунтуючись на отриманих середніх результатів маршруту, вони показують, що IGAROT забезпечив приріст на 4,24%, 75,7% і 420% порівняно із звичайним ГА в сценаріях низької, середньої і високої щільності автомобілів відповідно. Отримані результати свідчать про те, що IGAROT покращує зв'язок серед транспортних засобів в зв'язку з дорожніми аномаліями, що дозволяє водіям краще орієнтуватися на аномальних дорогах з метою зменшення кількості дорожньо-аномальних аварій. Додаткові переваги їхньої системи можуть включати в себе оперативне повідомлення дорожньо-експлуатаційних служб про збереження дорожніх умовах через транспортний засіб для зв'язку з інфраструктурою.

1.1.3 Планування малоефективних туристичних маршрутів в межах об'єкта суспільного значення шляхом оптимізації біологічних і логістичних критеріїв

Головною метою в управлінні охоронюваними територіями є сприяння збалансованому співіснуванню дикої природи і людей. Хоча ця мета може бути досягнута шляхом обмеження людської діяльності через буферні зони та обмежені зони, більш строгий підхід може бути заснований на моделях оптимізації, заснованих на кількісних і прозорих наукових методологіях.

У цій статті вперше пропонується модель прийняття рішень для планування малоефективних туристичних маршрутів в межах об'єкта Європейського Союзу, що має суспільне значення. Пропонована методологія спрямована на планування оптимізованих шляхів руху на основі віртуального ландшафту, побудованого як біологічних, так і на логістичних критеріях [43]. Запропонована модель складається з чотирьох етапів:

а) вибір біологічних і логістичних обмежень;

б) вибір біологічних і логістичних чинників;

в) використання нечіткого скорингу для стандартизації та вагових критеріїв;

г) застосування моделювання з найменшими витратами до віртуального ландшафту, отриманому в результаті попередніх етапів.

Було визначено три оптимізованих маршрути для нових туристичних маршрутів, які чинять найменший можливий вплив на існуюче біорізноманіття (місця проживання, види рослин і тварин), а також відповідають логістичним критеріям. Пропонований підхід також дозволяє кількісно оцінити вплив пропонованих шляхів, що дозволяє проводити порівняння з існуючими шляхами. Пропоновані шляхи, базуються на оптимізації, працюють значно краще; крім того, було запропоновано виведення з експлуатації восьми існуючих шляхів, які впливають на біорізноманіття.

Ця робота не тільки пропонує основу для планування малоефективних туристичних маршрутів у межах територій, що охороняються, але і припускає ідею про те, що завдяки використанню належних моделей оптимізації можна досягти задовільного балансу між природою і діяльністю людини.

1.1.4 Комбінована об'єктивна оптимізація маршруту руху засобів пересування з використанням генетичного алгоритму

Проблема маршрутизації засобів пересування (VRP) є імперативним сегментом логістичної обробки. Це допомагає транспортним і розподільчим компаніям в маршрутизації,

допомагаючи встановити правильний баланс між кількістю транспортних засобів і загальною відстанню. У даній роботі розглядається комбіноване об'єктивне завдання маршрутизації ємнісних транспортних засобів (CVRP). CVRP-це комбінаторна задача оптимізації, в якій парк транспортних засобів з обмеженнями по потужності доступний для обслуговування набору клієнтів з центрального депо з їх індивідуальними запитами, відомими заздалегідь [44]. Для вирішення поставленого завдання прийнята методика генетичного алгоритму.

Продуктивність генетичного алгоритму багато в чому залежить від типу використовуваних генетичних операторів. У даній роботі були обрані два різних кросовера і зроблена спроба виявити вплив цих обраних кросоверів на якість створюваних рішень. Оскільки основна увага приділяється двом цілям VRP, тобто кількістю транспортних засобів і відстані, використовується процедура призначення придатності за допомогою генетичного алгоритму, відома як Fitness Aggregated Genetic Algorithm (FAGA). Модель перевіряється з використанням інформації, отриманої від дистриб'юторської фірми.

Отриманий результат дозволяє зробити висновок про те, що запропонований алгоритм є висококонкурентним і надзвичайно ефективним для об'єктивної комплексної оптимізації маршруту руху засобів пересування.

1.1.5 Рішення задачі маршрутизації фідерного транспортного засобу з використанням оптимізації мурашиної колонії

У даній статті досліджується проблема маршрутизації фідерних транспортних засобів (FVRP), новий варіант задачі маршрутизації транспортних засобів (VRP), в якому кожен клієнт обслуговується або великим (вантажівка), або малим транспортним засобом (мотоцикл) [45]. В цьому конкретному виді доставки вантажівки і мотоцикли повинні виїхати з депо, відвідати клієнтів і в кінцевому підсумку повернутися в депо. Під час процесу доставки мотоцикли переміщуються в місця розташування вантажівок для перевантаження. Для вирішення поставленої задачі використовується алгоритм оптимізації мурашиної колонії (ACO) з метою визначення кількості диспетчерських субпарків і оптимальних маршрутів руху для мінімізації загальних витрат (фіксованих маршрутів і шляхових витрат). Для аналізу продуктивності FVPR створюються три базових набору даних. Для цілей порівняння всі екземпляри виконуються шляхом відправки тільки вантажівок, як в традиційному VRP і чотириступінчастої ієрархічної евристики. Крім того, ACO порівнюється з оптимальними рішеннями для невеликих екземплярів. Результати показують, що запропонований алгоритм ACO дає перспективні рішення, особливо для великих екземплярів, в розумні строки і ефективним чином.

1.2 Мурашиний алгоритм

Одним з існуючих методів вирішення задач пошуку оптимальних маршрутів на графах є мурашиний алгоритм (ACO). Марко Доріго виступив автором цієї ідеї[8]. Підхід полягає у тому, щоб використовувати моделі поведінки мурашок, яка є мета-евристичною оптимізацією та шукає шлях від колонії до джерела їжі. Колонія мурашок може розглядатися як багаточасткова система, в якій кожна частина (мураха) функціонує автономно за простими правилами. На відміну від примітивної поведінки частин, поведінка всієї системи є цілісною. На сьогодні відомо результати антиоптимізації таких складних комбінаторних задач, як: завдання квадратичного призначення, оптимізація графіків мережі, розфарбовування графіків, завдання мандрівного продавця для оптимізації маршрутів вантажних автомобілів, завдання планування тощо [9 - 14]. Незважаючи на дуже швидкий успіх мурашиних алгоритмів, все ще переважна більшість експертів з дослідницьких операцій є незнайомі з цією технологією оптимізації.

У реальному світі мурахи спочатку йдуть випадковим чином і повертаються до своєї колонії за їжею. Мурахи мають два способи передачі інформації: прямий - обмін їжею, нижньощелепні, зорові та хімічні контакти та непрямий - стігмержі. Стігмержі - це тимчасовий проміжок взаємодії, коли один суб'єкт змінює частину навколишнього середовища, а решта використовує цей статус пізніше, коли знаходиться біля нього. Біологічно стігмержі виробляються через феромон - особливий секрет, який належним чином осідає при переміщенні мурах. Феромон - досить стабільна речовина. Мурашки можуть сприймати його протягом декількох днів. Чим вище концентрація феромону на сліді, тим більше мурашок буде рухатися по ньому. Якщо інші мурахи знайдуть такі стежки, вони, ймовірно, йдуть за ними. Замість того, щоб відстежувати ланцюг, вони підсилюють її після повернення, якщо вони врешті-решт знайдуть джерело з їжею. З часом феромон випаровується, дозволяючи мурахам адаптуватися до змін навколишнього середовища. Чим довше потрібно пройти шлях до цілі та назад, тим більше випаровується феромоновий слід. За короткий термін, для порівняння, проходження буде швидшим і в результаті цього щільність феромонів буде залишатися високою. Випаровування феромонів має в собі властивість знаходити способи пошуку оптимального локально рішення. Якщо б феромони не випаровувалися, то шлях, який було обрано першим, був би найбільш привабливим. У цьому випадку вивчення просторових рішень було б обмеженим. Отже, коли одна мурашка знайде короткий шлях від колонії до джерела їжі, тоді інші мурахи скоріше пройдуть цим шляхом, а позитивні відгуки, у вигляді феромонів, зрештою приведуть усіх мурашок до єдиного, найкоротшого шляху.

Мурашині алгоритми використовують за основу імітацію природніх механізмів мурашиної самоорганізації. Самоорганізація являється основою поведінки колонії мурах і є множиною динамічних механізмів, які забезпечують виконання системою глобальної цілі у результаті низькорівневої взаємодії елементів цієї системи. Головною особливістю такої взаємодії являється використання елементами системи тільки локальної інформації. Причому виключається будь-яке звернення до глобального образу і централізоване управління, що репрезентує собою систему в зовнішньому світі.

Самоорганізація являється результатом взаємодії чотирьох компонентів:

а) випадковість;

б) багаторазовість;

в) зворотний зв'язок, який є позитивним;

г) зворотний зв'язок, що є негативним.

1.3 Дослідження алгоритмів прокладання туристичного маршруту

Для прокладання туристичного маршруту використання жорстко формалізованих математичних моделей надає оптимальне рішення з точки зору суворо закладеного в програму алгоритму, але при такому підході майже неможливо врахувати сформовані в обраному регіоні традиції та звички населення, екологічне становище, переміщення, і інші вимоги, котрі не піддаються формальному опису. Через це найбільш ефективним вважається підхід, під час якого експерт проводить аналіз і дослідження отриманих результатів та приймає кінцеве рішення.

Основою всієї конструкції є визначення величини пасажиропотоку за напрямками транспортної кореспонденції. У свою чергу, вони знаходяться на основі транспортного розрахунку районів. Чим правильніше туристська територія ділиться на транспортні зони, тим більш правильно (точніше) оцінюються значення пасажиропотоку і, таким чином, найбільшою мірою маршрутна мережа пасажирського транспорту задовольняє потреби туриста.

Щоб вирішити завдання проектування маршрутних мереж транспортного туризму необхідно подати її як математичну модель.

Потрібно описати транспортну мережу туризму у вигляді спрямованого графа G(V,E), де V - множина вершин (точок туристичних зупинок), E - множина дуг мережі (реальна ділянка дороги, що з'єднує туристичні зупинки). Напрямок дуги визначає курс проходження

транспортних засобів. Магістралі, котрі мають двосторонній рух, відповідно мають парні дуги, які є протилежно орієнтованими.

Під час дослідження потокоутворюючих факторів у множині вершин V можна виділити дві підмножини: перша $S \subseteq V$, котра містить пункти, які створюють потоки, які являються елементами множини S. Їх будемо називати джерелами. Друга підмножина $D \subseteq V$, що містить пункти, котрі поглинають потоки, буде називатися підмножиною стоків. Відносно задачі моделювання потоків для зимніх сезонів масового відпочинку у горах, міста та решта населених пунктів становляться джерелами, а туристичні райони Карпат, наприклад, стоками. Представимо множину всіх потокоутворюючих пар як декартовий добуток $W = \{w = (i, j) : i \in S, j \in D\}$.

Для кожної пари "джерело-стік" $w = (i, j) \in W$ існує свій попит на перевезення, ρW – загальна кількість користувачів, котрі з пункту «i» повинні добратися до пункту «j». А матрицею кореспонденцій називається набір $\{\rho W : w \in W\}$.

Маршрутом в мережі G, який з'єднує вершини «i» та «j», будемо називати таку послідовність дуг:

$$f_1 = (i \to k_1), f_2 = (k_1 \to k_2), \ ..., f_l = (k_{l-1} \to k_l), \ f_{l+1} = (k_l \to j),$$

де $f_t \in F$ при всіх $t = 1, ..., l+1$.

Тут передбачається відсутність в маршрутах циклів і петель. Множину альтернативних маршрутів позначимо як PW, за якими для кожної пари $w = (i, j) \in W$, що виходить із джерела «i» потік може досягати стоку «j». Усю сукупність шляхів у мережі G позначимо через

$$P = \prod_{w \in W} P_W .$$

Нехай x_p буде величиною потоку, який йде за шляхом $p \in P$. Для транспортних задач, традиційно, потокові змінні мають бути невід'ємними а також задовольняти балансовим рамкам. Через це для кожної пари w потоки $p\ x$, де $p \in P$, мабть належати множині

$$X_W = \left\{ x_p \geq 0 : p \in P_W , \ \sum_{p \in P_W} x_p = \rho_W \right\}.$$

Спершу об'єднаємо x_p у вектор $x = (x_p : = p \in P)$. Далі допустимою областю для вектора x буде множина, що була утворена як декартовий добуток всіх X_w :

$$X = \prod_{w \in W} X_W = \left\{ x \geq 0 : \ \sum_{p \in P_W} x_p = \rho_w, w \in W \right\}.$$

На подолання всіх шляхів $p \in P$ потрібні деякі витрати (паливо, гроші, час, зношеність дороги, амортизація автомобіля тощо). Кількість таких витрат може залежати від

інтенсивності та щільності потоку (руху) в мережі. Зазвичай, в моделях розглядаються фінансові або часові витрати. Позначимо G_p як питомі витрати користувачів щоб мати змогу проїхати по шляху p. Так як на витрати на одному маршруті можуть впливати, наприклад, завантаження інших маршрутів, то в суцільному випадку G_p є функціями від перевантаження усієї мережі, тобто $G_p = G_p(x)$.

Часто рішення подібних транспортних задач зводиться до вирішення варіаційних нерівностей, або оптимізаційної задачі в окремому випадку, що дає змогу адаптувати для їх рішення чисельні методи.

На даний час вже відомо багато різних методів знаходження рішення для задачі маршрутизації транспорту. Завдання маршрутизації транспорту являється узагальненням широко відомої задачі комівояжера, в якій необхідно побудувати одразу декілька замкнутих маршрутів, котрі проходять через якусь загальну вершину. Ці завдання є класом задач комбінаторної оптимізації та являються NP-складними. На даний момент не існує методів, які могли б знайти їх точне рішення і перевірити наближені на оптимальність.

Існує досить точний алгоритм, яким можна вирішити завдання маршрутизації транспорту, використовуючи за основу метод гілок і границь, але через швидке зростання часу, необхідного на обчислення, його практично неможливо застосовувати для задач, які мають більше ніж 30 вершин.

В останні часи до наближених алгоритмів проявляється дуже багато інтересу. Ще під час 60-х років XX століття евристичні методи отримали активний розвиток, а в сучасні дні вони отримали назву класичні. За останні двадцять п’ять років більшість зусиль було спрямовано на розвиток метаевристичних методів [6, 7]. Вони не є закінченими евристиками, які були б готові для практичного використання. Ці методи представляють собою деякий метод, що використовується для побудови закінченої евристики для певного завдання.

Значна частина цих методів були засновані на спостереженнях за живим і неживим навколишнім середовищем, тобто природою. Їх відмінність полягає в тому, що вони здатні подолати точки локального оптимуму щоб продовжити пошук. Через це потенційно якщо порівнювати класичні евристики та метаевристичні методи, останні спроможні знаходити більш точні рішення.

1.4 Сутність мурашиного алгоритму

Мураха обирає шлях пошуку їжі залежно від інтенсивності феромону, котрий розпростирається на цьому шляху. Потім після знаходження їжі мураха повертається у колонію, залишаючи за собою слід феромону. Чим коротшим є шлях, тим більш інтенсивним

на ньому буде феромон, бо він не буде встигати випаровуватися, чого не можна сказати про більш довгі шляхи. Таким чином, в результаті залишається тільки приблизно найкоротший шлях до їжі. Детально алгоритм буде розглянуто у розділі 2.

1.5 Застосування мурашиних алгоритмів для туристичних задач

Завдання можна сформулювати як задачу пошуку маршруту, який був би мінімальним за вартістю замкнутого маршруту, що проходить через всі вершини без повторень на повному графі з n кількістю вершин. Вершини графа являються містами, через які турист повинен пройти, а вага кожного ребра відображає відстань або вартість проїзду. Дана задача є NP-складною, і точне рішення методом повного перебору має факторіальну складність.

На основі розподілу феромону на стежці (ребрі графа) моделюється поведінка мурашок. Ймовірність включення стежки в маршрут окремої мурашки прямо пропорційна загальної кількості феромону на цій стежці, а кількість залишеного феромону вираховується пропорційно довжині усього маршруту. Тобто, чим коротшим є маршрут, тим більшою буде кількість феромону який відкладено на його стежці та більша кількість мурашок буде включати його в розрахунок власних маршрутів. Під час моделювання цього підходу використовується тільки позитивний зворотній зв'язок, що призводить до завчасної збіжності – переважна кількість мурашок пересувається за локально-оптимальним маршрутом. Щоб уникнути цього, потрібно змоделювати негативний зворотний зв'язок, наприклад, у вигляді ефекту випаровування феромону.

Маршрутна система пасажирського транспорту є найважливішою частиною транспортної інфраструктури для проведення туристичних поїздок, котрі, як правило, визначають динаміку для розвитку сучасних регіонів,. Під час розвитку туристичного регіону, маршрутна система цього регіону потребує періодичного удосконалення. Це пов'язується з численними змінами у забудові санаторіїв та туристичних баз, модернізацією вуличної мережі туристичних територій, зміною місцеположення місць прокладання маршрутів туристів.

Проектування нової маршрутної мережі транспорту водним, повітряним чи наземним методом до популярних туристичних осередків є дуже складним процесом, який включає в собі декілька етапів роботи. Найкращим підходом до вирішення даного завдання є його автоматизація за допомогою експерта, котрий проведе аналіз отриманих результатів та прийме остаточне рішення. Але автоматизація завдань у даній галузі вимагає також проведення наукових досліджень, які матимуть за мету формалізацію та розробку алгоритмів, що будуть придатні для роботи на практиці.

1.6 Версії Android SDK

За кожним випуском з кодовою назвою слідують інкрементні випуски. Наприклад, платформа Ice Cream Sandwich (ICS) була спочатку випущена як Android 4.0 (API рівня 14) і майже негайно замінена інкрементними випусками, які в кінцевому результаті привели до появи Android 4.0.3 і 4.0.4 (API рівня 15).

Пристрої Android зі старими версіями не піддаються негайному оновленню або заміні при появі нової версії. Станом на червень 2015 року більше 10% пристроїв все ще працювали під управлінням ICS або Gingerbread. Версія Android 4.0.4 (останнє оновлення ICS) була випущена в травні 2012 року.

На багатьох пристроях продовжують працювати старі версії Android в основному через гостру конкуренцію між виробниками пристроїв Android і операторами стільникового зв'язку. Оператори прагнуть мати можливості і телефони, яких немає у інших мереж. Виробники пристроїв теж відчувають тиск — всі їхні телефони базуються на одній ОС, але їм потрібно виділятися на тлі конкурентів. Поєднання цього тиску з боку ринку і операторів стільникового зв'язку призвело до появи численних пристроїв зі спеціалізованими модифікаціями Android.

Пристрій зі спеціалізованою версією Android не зможе перейти на нову версію, випущену Google. Замість цього йому доведеться чекати сумісного «фірмового» оновлення, яке може вийти через кілька місяців після випуску версії Google.

Через затримки оновлення в поєднанні з регулярним випуском нових версій сумісність стає важливою проблемою в програмуванні на Android. Щоб залучити більш широку аудиторію, розробники Android повинні створювати додатки, які добре працюють на пристроях з різними версіями Android: Jelly Bean, KitKat, Lollipop і більш сучасними версіями, а також на пристроях різних форм-факторів.

Підтримка різних розмірів не настільки складна, як може здатися. Смартфони випускаються з екранами різних розмірів, але система макетів Android добре пристосовується до них. З планшетами справа йде складніше, але в цьому випадку на допомогу приходять кваліфікатори конфігурацій. Втім, для пристроїв Android TV і Android Wear (також працюють під управлінням Android) відмінності в інтерфейсі настільки серйозні, що розробнику доводиться переосмислювати організацію взаємодії з користувачем і дизайн програми.

Частка пристроїв з версіями Froyo, Gingerbread і ICS падає від місяця до місяця, і обсяг роботи, необхідної для підтримки старих версій, переважує користь від них. Інкрементні випуски не створюють особливих проблем зі зворотною сумісністю. Зміна основної версії — зовсім інша справа. Робота, необхідна для підтримки тільки пристроїв 4.x, не так вже й велика, але якщо необхідно також підтримувати пристрої 2.x, доведеться витратити час на

аналіз відмінностей між версіями. Підтримка Android 5.0 (Lollipop) поряд з версіями 4.x вимагатиме певних зусиль, але компанія Google надала спеціальні бібліотеки, які спростять цю задачу. Підтримка пристроїв 2.x створює багато проблем. Випуск Honeycomb, Android 3.0, став поворотним моментом в Android, ознаменувавши собою появу нового інтерфейсу і нових архітектурних компонентів. Версія Honeycomb призначалася тільки для планшетів, так що нові розробки набули масового поширення тільки з виходом Ice Cream Sandwich. Версії, що виходили після цього, були менш революційними. Система Android підтримує забезпечення зворотної сумісності. Також існують сторонні бібліотеки, які допомагають у цьому.

Крім мінімальної підтримуваної версії, також можна задати цільову версію (target version) і версію для побудови (build version). Всі ці властивості задаються у файлі build.gradle в модулі app. Версія для побудови задається виключно в цьому файлі. Мінімальна і цільова версії SDK задаються у файлі build.gradle, але використовуються для заміни або ініціалізації значень з файлу AndroidManifest.xml.

Нижче наведено приклад файлу build.gradle, що знаходиться в модулі app.

```
...
compileSdkVersion 22
buildToolsVersion "23.0.0"
defaultConfig {
      applicationId "com.bignerdranch.android.geoquiz"
      minSdkVersion 16
      targetSdkVersion 22
      ...
}
...
```

Значення minSdkVersion визначає нижню межу, за якою ОС відмовляється встановлювати додаток. Вибираючи API рівня 16 (Jelly Bean), ви дозволяєте Android встановлювати GeoQuiz на пристроях з версією Jelly Bean і вище. Android відмовиться встановлювати GeoQuiz на пристрої, припустимо, з системою Froyo.

API рівня 16 є хорошим вибором для мінімальної версії SDK: в цьому випадку додаток можна буде встановлювати на 88 % використовуваних пристроїв.

Значення targetSdkVersion повідомляє Android, для якого рівня API проектувалося ваш додаток. Найчастіше в цьому полі вказується новітня версія Android.

Зниження цільової версії SDK гарантує, що додаток буде працювати з зовнішнім виглядом і поведінкою цільової версії, в якій воно добре працює. Цей параметр існує для забезпечення сумісності з новими версіями Android, так як всі зміни наступних версій ігноруються до збільшення targetSdkVersion.

Останній параметр SDK позначений в прикладі ім'ям compileSdkVersion. У файлі AndroidManifest.xml цей параметр відсутній. Якщо мінімальна і цільова версії SDK поміщаються в маніфест і повідомляються ОС, версія SDK для побудови відноситься до

закритої інформації, відомої тільки вам і компілятору. Функціональність Android використовується через класи і методи SDK. Версія SDK, що використовується для побудови, також називається метою побудови (build target), вказує, яка версія повинна використовуватися при побудові вашого кода. Коли Android Studio шукає класи та методи, на які ви посилаєтеся в директивах імпорту, версія SDK для побудови визначає, в якій версії SDK буде здійснюватися пошук.

Кращим варіантом мети побудови є новітній рівень API, наприклад 27. Проте при необхідності мету побудови існуючого додатку можна змінити, наприклад, при випуску чергової версії Android, щоб ви могли використовувати нові методи і класи, що з'явилися в цій версії. Всі ці параметри — мінімальну версію SDK, цільову версію SDK, версію SDK для побудови можна змінити у файлі build.gradle, проте слід пам'ятати, що зміни в цьому файлі проявляться тільки після синхронізації проекту зі зміненими файлами.

Відмінності між мінімальною версією SDK і версією SDK для побудови створюють проблеми сумісності, якими необхідно керувати. Наприклад, що відбудеться в додатку при виконанні коду, розрахованого на версію SDK після мінімальної версії Jelly Bean (API рівня 16)? Якщо встановити такий додаток і запустити його на пристрої з Jelly Bean, відбудеться збій.

Раніше тестування в подібних ситуаціях було справжнісіньким жахом. Проте завдяки вдосконаленню Android Lint проблеми, породжені викликом нового коду на старих пристроях, тепер успішно виявляються. Якщо ви використовуєте код версії, що перевищує мінімальну версію SDK, Android Lint повідомить про помилку побудови.

1.7 Розробка додатку з використанням навігаційних мап на прикладі Maps API

1.7.1 Імпорт картографічної бібліотеки Play Services Maps

Перш ніж розпочати розробку, необхідно імпортувати картографічну бібліотеку. Вона також входить до складу бібліотек Play Services. Необхідно відкрити вікно структури проекту і додати в модуль app наступну залежність: “com.google. android.gms:play-services-maps:7.0.0”. Номер версії залежності play-services змінюється з часом, тому необхідно завжди використовувати останню версію бібліотеки.

1.7.2 Робота з мапами в операційній системі Android

Дані про поточне місцезнаходження телефону майже не мають сенсу для користувача без візуального інтерфейсу, який можна б було використовувати.

Ймовірно, картографічні програми стали першим бестселером серед додатків для смартфонів; ось чому робота з мапами підтримується Android з найперших днів. Дані мап великі, складні і вимагають підтримки цілої системи серверів, що надають базові картографічні дані. Велика частина системи Android може існувати самостійно як частина Android Open Source Project. Проте про мапи цього сказати не можна. Отже, хоча мапи завжди існували в Android, вони також завжди були відокремлені від інших API системи Android. Поточна версія Maps API, версія 3 існує в Google Play Services поряд з Fused Location Provider. Їх використання вимагає пристрій з встановленим додатком Play Store, або емулятор з Google APIs.

Щоб розпочати роботу з Maps API потрібно виконати наступні кроки:

а) переконайтеся в тому, що пристрій підтримує Play Services;

б) імпортуйте потрібну бібліотеку Play Services;

в) за допомогою Google Play Services Util переконайтеся в тому, що у вас встановлена новітня версія програми Play Store.

1.7.3 Отримання ключа Maps API

Для використання Maps API необхідно оголошення ключа API в маніфесті, а для цього необхідно отримати власний ключ API. Цей ключ дозволить додатку використовувати картографічний сервіс Google.

Щоб отримати ключ API, необхідно отримати хеш ключа підписання і використовувати його для реєстрації Google Maps v2 API в Google Developer Console.

Для отримання ключа API необхідно ідентифікувати себе за допомогою ключа підписання — математично захищеного набору чисел, який належить вам і тільки вам. Кожен додаток, що встановлюється на пристрої Android, підписується унікальним ключем, за яким Android визначає, хто створив додаток.

Android Studio автоматично створює ключ підписання за замовчуванням, який називається налагоджувальним ключем. Кожен раз при побудові програми APK підписується випробувальні ключем перед його розгортанням.

Знаючи хеш SHA1 налагоджувального ключа, можна переходити до отримання ключа API. Інструкції наведені в документації Google: https://developers.google.com/maps /documentation/android/start. Після виконання цих інструкцій можна отримати ключ API свого проекту, відповідний налагоджувальному ключу підписання. Додайте його в маніфест.

Приклад маніфесту з доданим ключем наведено нижче.

```
<application
       android:allowBackup="true"
       android:icon="@mipmap/ic_launcher"
       android:label="@string/app_name"
       android:theme="@style/AppTheme" >
       <meta-data
            android:name="com.google.android.maps.v2.API_KEY"
            android:value="XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"/>
       ...
</application>
```

1.7.4 Створення мапи у додатку

Попереднє налаштування Maps API виконано - тепер потрібно створити мапу.

Мапи відображаються в віджеті MapView. Цей віджет схожий на інші віджети уявлень, не враховуючи одної відмінності: для його правильної роботи необхідно вручну передавати всі події життєвого циклу:

```
public void onCreate( Bundle instanceState ){
      super.onCreate( instanceState );

      mMapView.onCreate(instanceState);
}
```

Набагато простіше доручити цю роботу SDK: для цього використовується клас MapFragment або, при використанні фрагментів з бібліотеки підтримки, — SupportMapFragment. MapFragment створює MapView і виконує функції хоста, включаючи відповідні передачі зворотних викликів життєвого циклу.

Для початку потрібно замінити використання Fragment на SupportMapFragment. SupportMapFragment використовує власне перевизначення onCreateView(…), і в результаті цих дій все буде готове. Після запуску, в додатку повинна з'явитися карта.

1.8 Аналіз існуючих додатків

На даний момент існує багато мобільних додатків на операційній системі Android, котрі забезпечують використання мап, побудову та оптимізацію маршруту. Це Google Maps, HERE WeGO, Maps.me та Waze. Проаналізуємо, що представляє собою кожен з них, та пізніше порівняємо їх.

1.8.1 Google Maps

Google Maps являється картографічним сервісом, котрий розроблюється компанією Google з 2005 року.

Плюси:

а) простота використання;

б) дуже гарна фільтрація результатів пошуку;

в) дані про погоду;

г) знаходження найкращого маршруту;

ґ) управління голосом;

д) вуличний вигляд;

е) приблизна оцінка дорожнього трафіку.

Мінуси:

а) немає інформації про напрямок руху в вуличному вигляді;

б) обмежена кількість шляхів;

в) високий рівень споживання батареї;

г) в залежності від країни, надає неточну інформацію про дорожній трафік.

1.8.2 Waze

Waze це навігаційний і соціальний застосунок, котрий дозволяє прокладати оптимальний маршрут, відстежувати ситуацію дорожнього трафіку в реальному часі, дізнаватися про місцезнаходження радарів швидкості, тощо. Waze був створений в 2008 році, компанією Linqmap. На даний час він був викуплений компанією Google, і продовжує розроблятися нею ж. Особливістю Waze є те, що він створюється самими користувачами, а саме спільнотою редакторів-волонтерів.

Плюси:

а) Waze збирає інформацію і швидко гарантує, що ви отримаєте найшвидший маршрут;

б) оновлення дорожнього трафіку в реальному часі;

в) швидкий і оптимізований додаток;

г) знаходження найкращого та найшвидшого маршруту завдяки вбудованій ігровій формі, яка заохочує користувачів знаходити нові маршрути.

Мінуси:

а) неефективний інтерфейс, переповненість різними іконками в великих містах, або пусті екрани в маленьких;

б) багато відволікаючих факторів під час користування додатком;

в) поліцейські скаржаться, що повідомлення водія про наявність поліцейських пасток дуже сильно заважає їх роботі;

г) високий рівень споживання заряду батареї та інтернет трафіку.

1.8.3 HERE WeGo

Here WeGo це веб картографічна та навігаційна служба, яку розробляє компанія Here Technologies. Раніше була розроблена компанією Nokia і вперше побачила світ у 2012 році.

Плюси:

а) надає інформацію про дорожній трафік та публічний транспорт;

б) є можливість завантажити на смартфон мапу цілою країни і користуватися нею офлайн;

в) повідомляє про два наступні повороти, що надає полегшення водію під час їзди.

Мінуси:

а) недостатньо інформації про дорожній рух, тільки на базовому рівні;

б) активна зміна маршруту погано працює.

1.8.4 Maps.me

Maps.me це додаток для мобільних пристроїв на основі вільної географічної мапи OpenStreetMap. Вперше додаток побачив світ в 2010, а зараз він знаходиться під керівництвом Mail.ru Group з 2014 року. Основним напрямом розвитку є робота в офлайн режимі.

Плюси:

а) отримує дані з суспільної мапи OpenStreetMap, котра може редагуватися будь-якою людиною;

б) швидкий рендерінг карти на мобільних девайсах;

в) офлайн маршрутизація та покрокова навігація;

г) дуже різноманітний вибір зон для збереження;

ґ) є можливість завантажити дані про дорожній трафік за бажанням;

д) сучасний дизайн мапи;

е) офлайн векторні мапи;

є) інтеграція з booking.com;

ж) дуже легко додавати або редагувати закладки.

Мінуси:

а) немає опції для переключення між різними виглядами мапи, такими як супутниковий, топографічний, тощо;

б) неможливо відрізнити найшвидший, найефективніший або найкоротший шлях;

в) немає відображення обмеження скорості в зоні;

г) повільний пошук та обчислення маршруту;

ґ) інформація про дорожній трафік наявна не у всіх країнах та містах;

д) немає деталізованих мап для туристів;

е) в офлайн режимі не відображаються зображення.

1.8.5 Порівняння додатків

Усі чотири додатки мають свої плюси та мінуси, але деякі з них мають вагомі переваги понад іншими додатками в залежності від цілей користування. Якщо цілью є подорож по місцям без інтернет покриття, то фаворитом буде Maps.me, бо він надає найкращу підтримку офлайн карт. Для користування навігатором у повсякденному житті під час їзди по місту, найкращим вибором буде Waze, завдяки дуже великій кількості корисних функцій. Але якщо користувачу необхідний надійний, всесторонній картографічний сервіс та навігатор, який би вміщав у собі найнеобхідніші функції з якісним виконанням, то потрібно покласти свій погляд на Google maps.

1.9 Формулювання завдань та методів їх вирішення

В магістерській роботі повинні бути вирішені наступні завдання:

а) проаналізувати сучасні роботи з тематики дослідження;

б) дослідити алгоритми прокладання туристичного маршруту;

в) ґрунтовно проаналізувати предмет дослідження;

г) проаналізувати та дослідити методи вирішення задачі комівояжера;

д) сформулювати сутність мурашиного алгоритму та дослідити його застосування для туристичних задач;

е) практично реалізувати алгоритми у вигляді Android-додатку.

Необхідно розробити Android-додаток, який використовує мурашиний алгоритм для оптимізації маршруту. Він має містити в собі можливості:

а) прокладати шлях між двома або більше точками на мапі;

б) виконувати пошук локації;

в) змінювати локалізації;

г) оптимізувати маршрут, використовуючи мурашиний алгоритм.

1.10 Висновки до першого розділу

У цьому розділі було розглянуто та проаналізовано методику розв'язання задач пошуку оптимальних туристичних маршрутів алгоритмами мурашиних колоній. Оглянуто сучасні роботи на тему оптимізації маршрутів. Було розглянуто найпопулярніші Android-додатки, які реалізують прокладання та оптимізацію маршрутів, та проведено їх порівняння. Сформульовані основні завдання магістерської роботи та методи їх вирішення з використанням узагальненого мурашиного алгоритму.

2 АНАЛІЗ ТА УДОСКОНАЛЕННЯ МУРАШИНОГО АЛГОРИТМУ ДЛЯ ОПТИМІЗАЦІЇ ТУРИСТИЧНОГО МАРШРУТУ

2.1 Аналіз вимог

2.1.1 Проблема складності рішень задач вибору

При розв'язанні практичних обчислювальних задач часто виникає проблема, що пов’язана зі складністю обчислень і обчислювальних процесів.

Складністю обчислювальних процесів являється поняття в теорії складності обчислень, тобто оцінка ресурсів (часу, зазвичай) необхідних для виконання алгоритму.

Існує багато критеріїв для оцінювання алгоритмів. Найбільше уваги приділяється порядку росту часу та розміру пам'яті, необхідних для розв'язання задачі, при збільшенні розміру вхідних даних [15].

Клас складності - це набір задач розпізнавання, для яких існують подібні за обчислювальною складністю алгоритми. Завдання можна розділити на класи за складністю їх розв’язання. Усі класи складності є ієрархічними: деякі містять інші. Клас P, який є найнижчим, містить усі задачі, які можна вирішити за поліноміальний час. Клас NP включає всі задачі, які можливо вирішити за поліноміальний час лише використовуючи недетерміновану машину Тьюрінга [16].

Поняття поліноміального алгоритму є базовим поняттям теорії складності. Поліноміальний алгоритм — алгоритм, у якого час роботи (необхідна кількість елементарних двійкових операцій щоб виконати його на детермінованій машині Тьюрінга) на вхідному рядку з довжиною l є обмеженим згори певним поліномом $P(l)$ [17].

Класом P називають множину задач, для яких є «швидкі» алгоритми знаходження рішення (час роботи в них поліноміально залежить вхідних даних, а саме їх розміру) [16].

Класом NP називають множину задач розпізнавання, які можливо розв'язати за наявності певних додаткових відомостей, тобто існує можливість «швидко» перевірити розв'язок використовуючи машину Тьюрінга. Тобто, клас NP можна визначити як сукупність завдань, для яких можна «швидко» знайти рішення на недетермінованій машині Тьюрінга [16].

У теорії складності та теорії алгоритмів NP-повною задачею є задача, котра належить до класу NP, а також всі задачі з класу NP можливо звести до неї за деякий поліноміальний час [18]. Ця задача неформально належить до класу NP-повних, якщо вона належить до класу NP та є такою ж «складною», як і будь-яка інша задача, які відноситься до класу NP. NP-повні

задачі зіставляють собою підмножину NP-задач та відрізняються тим, що всі NP-задачі тим або іншим способом можуть бути зведені до них. Тобто, якщо для NP-повної задачі знайти P-розв'язок, то також буде знайдено P-розв'язок для всіх задач класу NP [19].

Як відзначено у вступі, в основі будь-якого методу розв'язання задач дискретної математики лежить повний перебір. Складність методу повного перебору пов'язана з кількістю всіх можливих варіантів вирішення задачі. З великими просторами рішень метод повного перебору рідко буває ефективним, бо у цьому випадку він може шукати результат на протязі років або століть.

2.1.2 Постановка задачі комівояжера

Задача комівояжера – задача, яка належить до комбінаторної оптимізації, та полягає у пошуку найвигіднішого шляху, котрий проходить через кожне місто хоча б по одному разу з наступним поверненням у початкове місто.

Оптимізаційна постановка задачі належить до класу NP-складних задач, як і більшість її приватних випадків.

Існує кілька окремих випадків загальної постановки завдання для задачі комівояжера:

а) геометрична(матриця відстаней показує відстані між усіма точками на площині);

б) трикутна;

в) симетрична й асиметрична.

Ще існує узагальнення задачі, що називається узагальнена задача комівояжера. Вихідними даними для неї є матриця вартостей переходу від однієї вершини до іншої, множина вершин, розбиття цієї множини вершин на так звані кластери [20, 21].

Сутність задачі комівояжера полягає у пошуку оптимального маршруту руху при необхідності відвідати всі заплановані об'єкти з найменшими фінансовими і тимчасовими витратами. Зазвичай, мова йде про просте переміщення по заданим точкам, або про перевезення вантажу невеликого формату на транспортному засобі [21].

Дуже наочно задачу комівояжера можна зобразити у вигляді графа $G=(X, U)$, вершини якого відповідають містам, а дуги — дорогам, що з'єднують міста між собою. Для кожної дуги, що з'єднує міста i, k, приписується число d_{ik}, зване довжиною дуги і рівне відстані між містами i та k. Якщо з кожного міста є безпосередній шлях, який веде до іншого міста без входження в проміжні міста, то кожні дві вершини графа будуть з'єднані дугами в обох напрямках. Граф, який отримується, виявляється повним.

Елементарний шлях в графі, який проходить через всі вершини, називається гамільтоновим шляхом. Замкнутий елементарний шлях, тобто елементарний контур, який проходить через кожну вершину графа, має назву гамільтоновий контур [22].

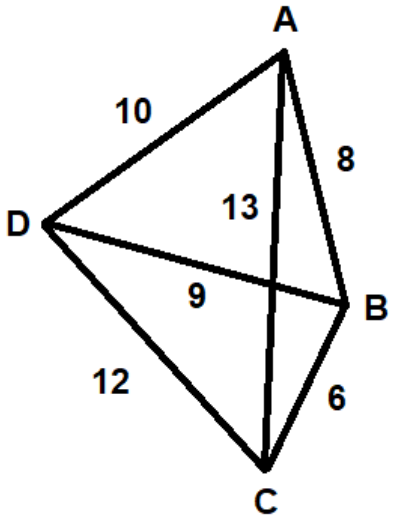


Рисунок 2.1 – Гамільтонів граф

На рисунку 2.1 зображено чотирикутник з 6 трикутними гранями і 4 вершинами. Потрібно, рухаючись по ребрах чотирикутника, обійти всі вершини, заходячи в кожну з них тільки по одному разу та повернутися у вихідну вершину. Кожен такий шлях, якщо він існує, має назву гамільтоновий цикл. Приведемо даний граф до матричного вигляду(табл 2.1).

Таблиця 2.1 – Матричний вигляд графу, зображеного на рис. 1

	A	B	C	D
A	0	8	13	10
B	8	0	6	9
C	13	6	0	12
D	10	9	12	0

Довжинам граней d_{ij}, де $i=j$, присвоюємо значення 0. Отримуємо задачу комівояжера, і далі можна вирішувати її. Якщо взяти за початкову вершину А, то обираючи найкоротші довжини, можна отримати шлях ABCDA.

2.1.3 Методи розв’язання задачі комівояжера. Метод прямого перебору

Для розв'язання задачі комівояжера використовують різні групи методів [20]:

а) точні, що містять повний перебір та метод гілок і меж;

б) наближені методи: методи випадкового перебору, жадібний і дерев'яний алгоритми; метод імітації відпалу;

в) «біологічні методи»: генетичні алгоритмів, а також алгоритм мурашиних колоній.

Прямий перебір полягає у вичерпному розгляду усіх можливих варіантів всіляких варіантів. Метод легко програмується, але він обмежений невеликою розмірністю задачі. У випадку дуже великого простору рішень, прямий перебір має можливість не надати результатів на протязі дуже великого проміжку часу [23].

Таблиця 2.2 – Залежність кількості варіантів рішень від (N-1) вершин

Розмірність	Кількість варіантів
5	24
10	362880
15	87178291200
20	1,21645E+17
25	6,20448E+23
30	8,84176E+30
35	2,95233E+38
40	2,03979E+46
45	2,65827E+54
50	6,08282E+62

У таблиці 2.2 можна спостерігати дуже швидке зростання кількості варіантів рішень задачі комівояжера та зробити висновок, що прямий перебір може бути ефективним тільки для малої розмірності задачі.

Будь-яке завдання, яке належить до класу NP, може бути вирішене прямим перебором. Більш того, навіть якщо можна обчислити цільову функцію від кожного конкретного можливого вирішення завдання за поліноміальний час, залежачи від кількості усіх можливих рішень, прямий перебір може зажадати експонентного часу роботи [23].

2.1.4 Метод гілок і меж

У результаті роботи алгоритму знаходиться максимум функції на допустимій множині. Де множина може бути як раціональною, так і дискретною. Під час роботи алгоритму виконується всього дві операції: поділення вихідної множини на підмножини(гілки), та установлення оцінок(меж). Існує дві оцінки множини: згори та знизу. Оцінкою згори є точка, яка гарантовано не є меншою аніж максимум на раніше заданій підмножині. Оцінкою знизу називається точка, яка гарантовано не є більшою аніж мінімум на підмножині, що була задана. Множина, котра має найбільшу оцінку зверху, називається рекордною. Спочатку прийнято вважати всю множину рекордною.

Алгоритм методу:

а) вихідна множина розбивається на дві або більше підмножини;

б) для нових підмножин знаходяться оцінки знизу та згори;

в) серед усіх підмножин визначається максимальна оцінка знизу;

г) видаляються ті множини, котрі мають максимальну оцінку знизу більшу аніж оцінку зверху;

ґ) серед усіх підмножин знаходиться максимальна оцінка згори, та вважається рекордною;

д) якщо не було досягнуто необхідної точності, тоді перейти до пункту а).

У результаті роботи одержується значення між оцінкою знизу та згори для рекордної множини. Різниця між нижньою та верхньою оцінками називається точністю, тобто коли ці оцінки збігаються, для дискретних множин алгоритм вважається завершеним [24].

Цей метод використовується для знаходження рішень певних NP-повних задач. Від способу визначення оцінок та вигляду функції залежить швидкість алгоритму, але вона гарантовано є не більше за прямий перебір. На практиці метод гілок і меж дозволяє суттєво (у інколи – на 2 порядки) скоротити обсяг обчислень у порівнянні з простим перебором. Але для задач високої розмірності цього буває недостатньо.

2.1.5 Метод Монте-Карло

Знаходження рішення для задачі комівояжера методом Монте-Карло будується на випадковому виборі кожного наступного міста, через який буде проходити шлях. При використанні програми відповідь носить імовірнісний характер і може сильно відрізнятися від правильного розв'язання задачі комівояжера. Теоретично при збільшенні числа випробувань похибка відповіді зменшується. Але практично для задач високої розмірності кількість

випадкових переборів становиться дуже малою у порівнянні з загальною кількістю варіантів, тому ймовірність знайти точне оптимальне рішення є малою. На практиці метод Монте-Карло може бути дуже корисним для підбору приємного для дослідника варіанту, тобто, наближеного рішення [25].

2.1.6 Біологічний підхід у сучасній кібернетиці

В останні роки у кібернетиці та прикладної математиці широко використовується «біологічний» підхід, оснований на використанні методів біоніки.

Біонікою називається прикладна наука про застосування в технічних пристроях і системах функцій, структур і властивостей живої природи, принципів організації, тобто сутності живого в природі та їх промислові аналоги [26].

Розрізняють такі напрями біоніки [26]:

а) біологічна біоніка, яка вивчає процеси, котрі відбуваються в біологічних системах;

б) теоретична біоніка, яка будує математичні моделі цих процесів;

в) технічна біоніка, що для вирішення інженерних завдань застосовує моделі теоретичної біоніки.

Біоніка дуже тісно пов'язана з фізикою, біологією, кібернетикою, хімією та інженерними науками.

Основний принцип біоніки – «підглянути у Природі», тобто, знайти, як Природа вирішує певні задачі, використовувати виявлені принципи для вирішення практичних завдань у науці й техніці.

На основі біоніки та біологічного підходу у кібернетиці розроблені ефективні методи наближеного пошуку оптимуму в задачах дискретної математики, зокрема, генетичні й мурашині алгоритми.

2.1.7 Генетичні алгоритми

Генетичний алгоритм - це евристика пошуку, натхненна теорією природної еволюції Чарльза Дарвіна. Цей алгоритм відображає процес природного відбору, коли найбільш пристосовані особини відбираються для розмноження з метою отримання потомства наступного покоління. Відмінністю генетичного алгоритму є використання оператора «схрещування», який виконує операцію рекомбінації рішень-кандидатів, яка має роль, аналогічну ролі схрещування в живій природі [27].

Завдання програмується таким чином, аби його рішення можна було б представити у вигляді вектору хромосома. Потрібно створити випадковим чином деяку кількість початкових векторів.

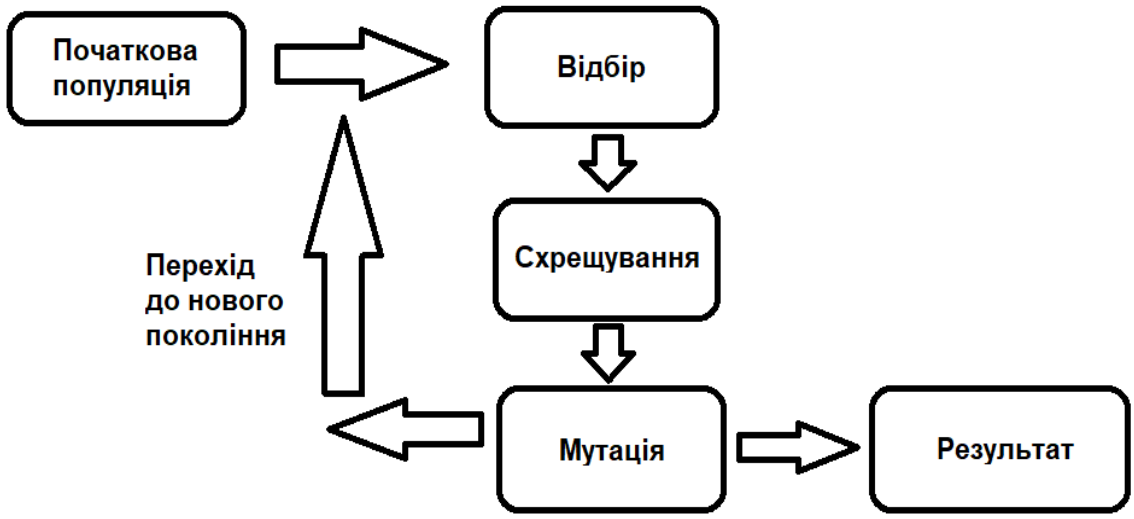


Рисунок 2.2 – Схема генетичного алгоритму

На схемі (рис. 2.2) зображено класичний алгоритм роботи генетичного алгоритму. Можна виділити наступні етапи роботи алгоритму [27]:

а) створення першої популяції;

б) визначення функції пристосованості для особин популяції;

в) (початок циклу):

– вибір індивідів з поточної популяції (селекція);

– схрещування та\або мутація;

– обчислення функцій пристосованості для всіх особин;

– утворення нового покоління;

– якщо виконуються умови зупинки, то (кінець циклу), інакше (початок циклу).

Критерієм зупинки може бути [27]:

– знаходження глобального, або субоптимального рішення;

– вичерпання числа поколінь, відпущених на еволюцію;

– вичерпання часу, який було відпущено на еволюцію.

Функція пристосованості — використовується для порівняння хромосом одного покоління, присвоює кожній хромосомі значення пристосованості. Потім обираються найкращі хромосоми, котрі схрещуються і мутують, створюючи нове покоління.

Схрещування — процес обміну частинами хромосом між двома або більше хромосомами.

Мутація — перетворення хромосоми, під час якого випадково змінюється один або декілька генів.

Генетичні алгоритми насамперед використовуються для пошуку рішень в дуже великих, складних просторах пошуку.

Переваги генетичного алгоритму [28]:

– концептуальна простота;
– широка застосовність;
– паралелізм;
– стійкість до динамічних змін;
– рішення проблем, для яких відсутній досвід рішень.

Недоліки генетичного алгоритму [29]:

– не гарантує отримання оптимального рішення;
– ефективно сформулювати задачу може тільки спеціаліст;
– висока обчислювальна ресурсоємкість;
– невисока ефективність на кінцевих фазах моделювання еволюції;
– проблема самоадаптації.

2.1.8 Мурашині алгоритми

Мурашині алгоритми являються імовірнісною жадібною евристикою, в якій ймовірності визначаються, виходячи з якості рішення, що була отримана з попередніх рішень. Вони знаходять найкоротше рішення, але не можуть гарантувати, що це рішення буде найкращим. Мурашині алгоритми можуть використовуватися як для динамічних , так і статичних комбінаторних оптимізаційних задач. У будь-якому випадку отримується оптимальне рішення, тобто збіжність гарантована, але швидкість збіжності невідома [30].

Ідеєю мурашиного алгоритму є імітація поведінки мурах, котра пов'язана з навичками мурах швидко знаходити найкращий (найкоротший) маршрут від мурашника до джерела з їжею, та під час пошуку адаптуватися до динамічних умов, знаходячи новий маршрут, який є найкоротшим [30].

Під час руху мураха залишає на шляху феромономи, які використовуються іншими мурашками задля вибору маршруту. Завдяки цьому елементарному правилу поведінки і визначається здатність мурашок знаходити новий маршрут, навіть в тому випадку, коли старий маршрут виявляється недоступним.

Феромони - це хімічні речовини, здатні діяти як гормони поза тіла виділяючого індивідуума, щоб впливати на поведінку сприймаючих індивідуумів. Існують феромони тривоги, феромони харчового сліду, статеві феромони і багато інших, які впливають на поведінку або фізіологію.

Мурашиний алгоритм — це імовірнісний метод вирішення обчислювальних завдань, який може бути зведено до знаходження хороших шляхів через графи. Суттю алгоритму є мета евристична оптимізація та аналіз і використання моделі поведінки мурах, котрі шукають маршрут від колонії до джерела з їжею.

Метаевристика - метод оптимізації, багаторазово використовуючий прості правила евристики для досягнення субоптимального або оптимального рішення.

В 1992 році була запропонована перша версія алгоритму італійським вченим Марко Доріго, яка була спрямована на знаходження оптимального маршруту в графі.

Для кращого розуміння розглянемо особливості поведінки мурахів у природі.

2.1.9 Основи біоніки мурах

Мурахи — найбільш еволюційне просунуте сімейство комах з точки зору екології, етології та фізіології. Їх сім'ї являються складними соціальними групами з розподілом праці та розвинутими системами самоорганізації і комунікації, що дозволяють мурахам координувати свої дії між собою при виконанні завдань, котрі непідвладні одному індивіду. Є такі види мурашок, шо володіють розвиненою «мовою» і здатні ділитися складною інформацією. Більш того, більшість видів мурах мають високорозвинені симбіотичні стосунки з іншими комахами, бактеріями, грибами і рослинами [31].

Мурахи спілкуються між собою використовуючи хімічні комунікації шляхом виділення спеціальних хімічних речовин – феромонів, в також – шляхом фізичних комунікацій – за допомогою звуків або дотиків.

Збереження всієї складної структури мурашиної колонії, зв'язків всіх особин і їх здібностей розпізнавати інших членів колонії обумовлено трофаллаксисом (обміном проковтнутої рідкою їжею) та хімічної комунікацією.

Для спілкування мурахи використовують феромони. Вони сприймають запахи своїми тонкими і довгими вусиками, котрі надають їм інформацію про інтенсивність та напрямок запаху. Так як мурахи живуть на землі, то поверхня ґрунту вважається гарним місцем для того, щоб залишати слід феромону, який можуть відчувати інші мурашки. Для видів, які виробляють їжу в групі, фуражир (той хто шукає їжу, розвідник), знайшовши їжу, відзначає свій шлях назад до мурашника, і по цій стежці йдуть інші мурахи, які також відзначають

феромонами свій шлях назад до гнізда в разі знаходження їжу за вказаним маршрутом. Коли джерело їжі виснажується, мурахи більше не відзначають цей шлях, і запах поступово розсіюється. Така поведінка мурах допомагає впоратися із змінами навколишнього середовища. Наприклад, якщо встановлений шлях до їжі перешкоджає перешкода, фуражири починають знаходити новий шлях до їжі. Якщо пошук був успішним, на шляху назад мураха позначає найкоротший шлях його повернення до гнізда. Інші мурахи йдуть цим успішним маршрутом, покращуючи оптимальний маршрут і поступово знаходячи найкращий шлях до їжі.

Мурахи використовують феромони не тільки для маршрутизації. Поранена мураха залишаэ тривожний феромон, який здалеку викликає мурах і змушує всіх особин, що знаходяться поблизу, атакувати ворога. Деякі мурахи навіть використовують "пропагандистський феромон", щоб залякати своїх ворогів і змусити їх битися між собою. У сім'ях видів, що мають цариць, робітники починають вирощувати нову царицю для сім'ї, якщо чинна цариця не виробляє необхідних феромонів [31].

Фізичні комунікаційні сигнали можуть використовуватися мурахами в поєднанні з феромонами. Наприклад, мурахи можуть спілкуватися за допомогою тактильних стимулів і звуків. Зокрема, деякі мурахи видають стрекочучі звуки, використовуючи сегменти черевця або нижньої щелепи. Звуки використовуються для спілкування між членами сім'ї або з іншими видами тварин. Мурашки також дуже чутливі до вібрацій твердих тіл.

Багато тварин можуть навчатися за допомогою наслідування, але мурахи, можливо, єдина група, крім ссавців, у яких спостерігається інтерактивне навчання. Інформований фуражир виду Temnothorax albipennis приводить товариша до нещодавно відкритого джерела живлення за допомогою бігу в тандемі. «Учень» отримує інформацію від «лідера». При цьому «лідер» і «учень» завжди знаходяться в контакті і слідкують за просуванням один одного. Отримавши урок, «учні» часто самі стають «вчителями», таким чином інформація про положення їжі поширюється по всьому гнізді.

Мурахи впізнають членів сім'ї по запаху, що надходить від вуглеводневих виділень, які знаходяться на їхньому екзоскелеті. Якщо мураха відділена від материнської сім'ї, вона з часом втрачає запах сім'ї. Будь-яка мураха, котра входить у гніздо і не має відповідного запаху, буде атакована.

У реальному світі мурашки (спочатку) переміщаються хаотично і у випадку знаходження їжі повертаються в свою колонію, залишаючи сліди феромонів. Якщо інші мурахи знайдуть такі сліди, вони, швидше за все, підуть по ним. Замість того щоб відслідковувати ланцюжок, вони підсилюють його після повернення, якщо в кінці кінців знайдуть джерело їжі. З плином часу стежка феромонів починає випаровуватися, таким чином

зменшуючи привабливу міцність. Чим більше часу потрібно, щоб пройти шлях до цілі і назад, тим більше феромонний шлях випарується. На короткому шляху проходження буде швидше, і в результаті густота феромонів залишається високою. Випаровування феромону має властивість уникати прагнення до локально оптимального рішення. Якщо б феромони не випарувалися, то шлях, обраний першим, був би найбільш привабливим. В такому випадку дослідження просторових рішень були б обмежені. Отже, коли один мураха знаходить короткий шлях від колонії до джерела їжі, інші мурахи з більшою вірогідністю підуть по цьому шляху, і позитивний зворотній зв'язок в кінцевому підсумку призводить всіх мурах до одного, найкоротшого шляху [31].

2.1.10 Формулювання узагальненого мурашиного алгоритму

Незалежно від модифікацій, Будь-який мурашиний алгоритм можна сформулювати у наступному вигляді (рис. 2.3):

Поки умови виходу не виконані:

– утворюємо мурах;

– знаходимо рішення;

– оновлюємо феромони;

– допоміжні дії {опціонально}.

Рисунок 2.3 – Схема узагальненого мурашиного алгоритму

2.2 Аналіз програмних та інструментальних засобів

2.2.1 Мова програмування Java

Java - це комп'ютерна мова програмування. Вона дозволяє програмістам писати комп'ютерні інструкції за допомогою команд англійською мовою замість того, щоб писати в числових кодах. Java відома як мова високого рівня, тому що її код легко читати і писати.

Як і англійська мова, Java має набір правил, які визначають, як пишуться інструкції. Ці правила відомі як її синтаксис. Як тільки програма написана, високорівневі інструкції переводяться в числові коди, які комп'ютери можуть зрозуміти і виконати.

На початку 90-х років Java, яка спочатку називалася Oak, а потім Green, була створена командою на чолі з Джеймсом Гослінгом для Sun Microsystems, компанії, яка нині належить Oracle.

Java спочатку була розроблена для використання на цифрових мобільних пристроях, таких як мобільні телефони. Однак, коли Java 1.0 була випущена для широкої публіки в 1996 році, її основну увагу було перенесено на використання в інтернеті, забезпечуючи інтерактивність користувачів, надаючи розробникам можливість створювати анімовані веб-сторінки.

З часу версії 1.0 було багато оновлень, таких як J2SE 1.3 в 2000 році, J2SE 5.0 в 2004 році, Java SE 8 в 2014 році і Java SE 10 в 2018 році.

Протягом багатьох років Java розвивалася як успішна мова для використання як в Інтернеті, так і поза ним.

Java була розроблена з урахуванням декількох ключових принципів:

а) простота використання: основи Java прийшли з мови програмування під назвою C++. Хоча C++ є потужною мовою, він складний за своїм синтаксисом і не відповідає деяким вимогам Java. Java була побудована на ідеях C++ і поліпшила їх, щоб забезпечити потужну і просту у використанні мову програмування;

б) надійність: Java необхідна для зниження ймовірності фатальних помилок від помилок програміста. З урахуванням цього було введено об'єктно-орієнтоване програмування. Коли дані та їх маніпуляції були упаковані разом в одному місці, Java була надійною;

в) безпека: оскільки Java спочатку призначалася для мобільних пристроїв, які будуть обмінюватися даними мереж, вона була побудована з урахуванням високого рівня безпеки. Java, ймовірно, є найбезпечнішою мовою програмування на сьогоднішній день;

г) незалежність від платформи: програми повинні працювати незалежно від машин, на яких вони виконуються. Java була написана, щоб бути портативною і крос-платформенною

мовою, яка не піклується про операційну систему, обладнання або пристрої, на яких вона працює.

Команда Sun Microsystems успішно об'єднала ці ключові принципи, і популярність Java можна пояснити тим, що вона є надійною, безпечною, простою у використанні і портативною мовою програмування.

2.2.2 Розширювана мова розмітки XML

Розширювана мова розмітки (XML) використовується для опису даних. Стандарт XML - це гнучкий спосіб електронного обміну структурованими даними через Інтернет або через корпоративні мережі і створення інформаційних форматів.

XML-код, формальна рекомендація Консорціуму World Wide Web Consortium (W3C), схожий на мову гіпертекстової розмітки (HTML). І XML, і HTML містять символи розмітки для опису вмісту сторінки або файлу. HTML-код описує вміст веб-сторінки (в основному текст і графічні зображення) тільки з точки зору того, як воно має відображатися і взаємодіяти.

XML-дані відомі як самовизначні, що означає, що структура даних вбудована в дані, тому, коли дані надходять, немає необхідності попередньо будувати структуру для зберігання даних; вона динамічно розуміється в XML. Формат XML може бути використаний будь-якою особою або групою осіб або компаній, які хочуть обмінюватися інформацією узгодженим чином. XML насправді є більш простим і зручним у використанні підмножиною стандартної узагальненої мови розмітки (SGML), який є стандартом для створення структури документа.

Основним будівельним блоком XML-документа є елемент, який визначається тегами. Елемент має початковий і кінцевий тег. Всі елементи в XML-документі містяться в самому зовнішньому елементі, відомому як кореневий елемент. XML також може підтримувати вкладені елементи або елементи всередині елементів. Ця здатність дозволяє XML підтримувати ієрархічні структури. Імена елементів описують зміст елемента, а структура описує відносини між елементами.

XML-документ вважається "добре сформованим" (тобто здатним бути прочитаним і зрозумілим синтаксичним аналізатором XML), якщо його формат відповідає специфікації XML, якщо він правильно розмічений і якщо елементи правильно вкладені. XML також підтримує можливість визначення атрибутів елементів і опису характеристик елементів елемента в початковому тегу.

Додатки для XML нескінченні. Наприклад, виробники комп'ютерів можуть домовитися про стандартний або поширений спосіб опису інформації про продукт комп'ютера (швидкість

процесора, обсяг пам'яті і т. д.), а потім описати формат інформації про продукт за допомогою коду XML. Такий стандартний спосіб опису даних дозволив би користувачеві відправити інтелектуальний агент (програму) на веб-сайт кожного виробника комп'ютерів, зібрати дані і потім провести коректне порівняння.

Переваги XML іноді виявлялися революційними за своїми масштабами незабаром після його появи. Однак, як концепція, вона не була революційною. Він також не був панацеєю від усіх бід. Надмірне застосування XML в багатьох областях техніки знижує його реальну цінність і призводить до великої кількості непотрібної плутанини. Можливо, найбільш руйнівним є передбачувана поведінка багатьох постачальників, які прагнуть переробити XML, використовуючи свій власний набір пропрієтарних розширень. Хоча деякі хочуть додати цінність XML, інші прагнуть лише прив'язати користувачів до своїх продуктів.

Сила XML полягає в його простоті. Він може взяти великі шматки інформації і об'єднати їх в XML - документ-значущі фрагменти, які забезпечують структуру і організацію інформації.

2.2.3 Система автоматичної збірки Gradle

Gradle-це вдосконалена система управління збіркою загального призначення, заснована на Groovy і Kotlin. Gradle підтримує автоматичне завантаження та налаштування залежностей або інших бібліотек. Він підтримує репозиторії Maven і Ivy для вилучення цих залежностей. Це дозволяє повторно використовувати артефакти існуючих систем складання. Gradle підтримує мультипроектні і мульти-артефактні збірки. У Gradle є поняття проектів і завдань.

Збірка Gradle складається з одного або декількох проектів. Проекти можуть бути чимось, що повинно бути побудовано або щось, що повинно бути зроблено. Кожен проект складається із завдань. Завдання являє собою частину роботи, яку виконує збірка, наприклад, компілює вихідний код або генерує Javadoc.

Проект з використанням Gradle описує його збірку за допомогою build.gradle файлу. Цей файл знаходиться в кореневій папці проекту. Файл збірки для Gradle збірки заснований на доменній мові (DSL). У цьому файлі можна використовувати комбінацію декларативних і імперативних тверджень. Ви також можете написати код Groovy або Kotlin, коли вам це потрібно. Завдання також можуть бути створені і розширені динамічно під час виконання. Цей файл збірки визначає проект і його завдання. Gradle - це система збірки загального призначення, тому цей файл збірки може виконувати будь-яке завдання.

Система збірки Gradle використовує плагіни для розширення своєї основної функціональності. Плагін - це розширення для Gradle, яке зазвичай додає деякі попередньо

налаштовані завдання. Gradle поставляється з декількома плагінами, і ви можете розробляти власні плагіни. Одним із прикладів є плагін Java. Цей плагін додає в ваш проект завдання, які дозволяють компілювати вихідний код Java, запускати модульні тести і створювати JAR-файл.

2.2.4 Особливості ОС Android

Одна з головних переваг платформи Android - її відкритість [40]. Операційна система Android побудована на основі відкритого вихідного коду і знаходиться у вільному поширенні. Це дозволяє розробникам отримати доступ до вихідного коду Android і зрозуміти, яким чином реалізовані властивості і функції додатків. Будь-який користувач може взяти участь у вдосконаленні операційної системи Android. В Інтернеті доступні різні додатки Android з відкритим вихідним кодом, пропоновані компанією Google і рядом інших виробників.

Відкритість платформи сприяє швидкому розвитку. На відміну від закритої системи iOS компанії Apple, доступною тільки на пристроях Apple, система Android доступна на пристроях десятків виробників устаткування (OEM, Original Equipment Manufacturer) і телекомунікаційних компаній по всьому світу. Всі вони конкурують між собою, що йде на користь кінцевому споживачеві.

2.2.5 Робота навігації в Android

Сервіси на основі визначення місця розташування надають мобільним користувачам безліч переваг для отримання інформації про своє поточне місцезнаходження та обробки цих даних, щоб отримати корисну інформацію поруч з їх місцем розташування. За допомогою A-GPS в телефонах і через веб-сервіси з використанням GPRS, засновані на місцезнаходження послуги можуть бути реалізовані на смартфонах на базі Android, щоб забезпечити ці додаткові послуги: консультування клієнтів про поточні умови руху, надання інформації про маршрут, допомагаючи їм знайти довколишні готелі.

Глобальна система позиціонування (GPS) використовує сузір'я з 24 супутників, що обертаються навколо Землі. GPS знаходить положення користувача, обчислюючи різницю в часі, протягом якого сигнали з різних супутників досягають приймача. Сигнали GPS декодуються, тому смартфон повинен мати вбудований GPS-приймач.

Assisted-GPS (A-GPS) являє собою нову технологію для смартфонів, яка інтегрує мобільну мережу з GPS, щоб забезпечити кращу точність від 5 до 10 метрів. Це фіксує положення протягом декількох секунд, має краще охоплення і може, в деяких випадках, використовуватися всередині будівель, споживає менше енергії акумулятора і вимагає менше

супутників. Деталізація інформації про місцезнаходження є найбільш точною (широти і довготи).

У стандартній комплектації Android надає базовий варіант Location API, що дозволяє отримувати дані про розташування від різних джерел. На більшості телефонів такими джерелами є точні дані від приймача GPS і наближені дані від веж стільникового зв'язку або мереж WiFi. Такі API існують з перших версій Android. Їх можна знайти в пакеті android.location. Якщо сигнал GPS недоступний, то рішення з даними від веж стільникового зв'язку може бути оптимальним. Якщо жоден з цих сигналів не доступний, навіть позиціонування з акселерометром і гіроскопом все ж краще, ніж повна відсутність даних. У минулому якісним додатками доводилося спеціально підписуватися на всі ці різнорідні джерела даних і перемикатися між ними в міру потреби. Таке рішення не назвеш ні простим, ні зручним.

В даний час на мобільному пристрої для визначення його місця розташування використовуються кілька систем глобального позиціонування. Найбільш поширеним провайдером є GPS (Global Positioning System) — американська система глобального позиціонування, яка дозволяє в будь-якому місці Землі визначити місце розташування об'єкта. Крім GPS, існують і інші системи, поки володіють меншими можливостями, ніж GPS: ГЛОНАСС-російська і Galileo-європейська система глобального позиціонування.

Принцип роботи всіх систем глобального позиціонування полягає у визначенні місця розташування шляхом вимірювання відстаней від мобільного пристрою до супутників. Відстань обчислюється за часом затримки прийому сигналу від супутників до мобільного пристрою.

Крім GPS, мобільний пристрій на платформі Android також підтримує визначення місця розташування за допомогою мережевого провайдера, використовуючи сигнал від станцій стільникового зв'язку. Ще один варіант визначення місця розташування - через точки доступу Wi-Fi-з'єднань. Однак слід мати на увазі, що мережеві провайдери розташування менш надійні, ніж супутникові системи глобального позиціонування. Задля цих типів провайдерів в класі LocationManager визначені константи: GPS_PROVIDER і NETWORK_PROVIDER.

2.2.6 Вибір середовища розробки

Середовищем для розробки мобільного додатку стало інтегроване середовище розробки (Integrated Development Environment - IDE) Android Studio. Причина цього вибору – Android Studio найпопулярніше IDE для розробки мобільних додатків, які працюють під

управлінням ОС Android. Воно має зручний інтерфейс та стабільно оновлюється розробниками.

2.2.7 Android Studio

Android Studio - це офіційне інтегроване середовище розробки (IDE) для розробки додатків на Android. Він заснований на ідеї IntelliJ, інтегрованому середовищі розробки Java для програмного забезпечення, і включає в себе інструменти редагування коду і розробника.

Для підтримки розробки додатків в операційній системі Android Android Studio використовує систему збірки на основі Gradle, емулятор, шаблони коду та інтеграцію з Github. Кожен проект в Android Studio має одну або кілька модальностей з вихідним кодом і файлами ресурсів. До таких модальностей відносяться модулі додатків для Android, бібліотечні модулі і модулі движка додатків Google.

Android Studio використовує функцію Instant Push для передачі коду і змін ресурсів в занедбаний додаток. Редактор коду допомагає розробнику писати код і пропонує його завершення, рефакторинг і аналіз. Програми, вбудовані в Android Studio, потім компілюються у форматі APK для відправки в Google Play Store.

Програмне забезпечення було вперше анонсовано на Google I/O в травні 2013 року, а перша стабільна збірка була випущена в грудні 2014 року. Android Studio доступна для настільних платформ Mac, Windows і Linux. Він замінив Eclipse Android Development Tools (ADT) в якості основного середовища розробки додатків для Android.

2.2.8 Емулятор для розробки додатків Android

Комплект для розробки програмного забезпечення (SDK) для Android поставляється з Android Studio, плагіном, який називається набором інструментальних засобів для розробки на Android (ADT). Це інструмент розробки для створення, налагодження та тестування додатків на Java. Android SDK може використовуватися без ADT; замість інструментів, можна використовувати інструменти командного рядка. Емулятор підтримує використання обох підходів, і з його допомогою можна запустити, виправити і перевірити додатки. 90% розробки додатків можуть бути завершені навіть без використання реального пристрою. Повністю функціональний емулятор для Android відтворює найбільш вивчені характеристики пристрою. Серед тих функцій, які не можуть бути імітовані в емуляторі, є USB-підключення, робота камери та відео, імітація роботи навушників, батареї і технології Bluetooth.

Android Emulator базується на технології з відкритим вихідним кодом "імітація" процесора під назвою QEMU, який був розроблений Фабрісом Белларом. Та ж сама технологія може емулювати одну операційну систему на іншій, незалежно від того, яка використовується процесором. QEMU забезпечує емуляцію рівня процесора. При використанні емулятор Android імітує процесор, що функціонує на базі ARM (Advanced RISC Machine). ARM - 32-розрядна архітектура мікропроцесорів на базі RISC (Reduced Instruction Set Computer, комп'ютер з скороченим набором команд команд), яка за рахунок скорочення кількості команд досягає простоти конструкції і підвищення продуктивності. Емулятор використовує процесор в такій модельованій версії Linux, яка використовується в Android. ARM широко використовується в мобільних пристроях і вбудованих електронних пристроях, де важливо розподілити невелику кількість енергії. Багато з наявних в продажу мобільних пристроїв мають процесори з цією архітектурою. Наприклад, Apple Newton на базі процесора ARM6. Ігрові автомати, Nintendo DS і Game Boy Advance працюють на архітектурі ARM версії 4, котра використовує близько 30 000 транзисторів. Класичний Pentium містить 3,2 мільйона транзисторів.

2.2.9 Інтерфейс користувача Android

Користувацький інтерфейс використовуємий в рамках ОС Android, можна порівняти з іншими повнофункціональними UI-структурами, застосовуваними на локальних комп'ютерах. Це більш сучасний і асинхронний характер. Насправді, UI-фреймворк Android належить до четвертого покоління, якщо рахувати перше покоління традиційного прикладного програмування інтерфейсу Microsoft Windows, засноване на C і MFC (класів Microsoft Foundation, Microsoft бібліотеки базових класів, заснованих на C++) - другий. У цьому випадку, Swing UI - фреймворк, заснований на Java, буде третім поколінням, і,як було запропоновано в його конструктивних можливостях, набагато перевершує MFC по гнучкості. Android UI, JavaFX, Microsoft Silverlight і мова Mozilla XML (XUL) інтерфейси включають новий тип UI-фреймворк четвертого покоління, який є декларативним UI і підтримує незалежну тематизацію.

При програмуванні в інтерфейсі Android використовується оголошення інтерфейсу в файлах XML. Тоді визначення представлення XML завантажуються в додаток з призначеним для користувача інтерфейсом, як в Windows. Навіть меню програми завантажуються з файлу XML. Екрани (вікна) Android часто називають активністями, які можуть бути різних типів. Види (представлення) є основними елементами, які знаходяться в інтерфейсі Android. Форми можуть бути об'єднані в групи (групи Видів). Для внутрішньої організації цього виду, вони

використовують протягом тривалого часу відому в програмуванні концепцію полотна і взаємодію користувача з системою. Такі композитні уявлення, які включають в себе види і групи видів, працюють на основі спеціальної логіки компонента інтерфейсу користувача Android. Одним з ключових понять бази є управління Android-Power життєвим циклом вікон подій (вікна діяльності). Під час роботи використовуються протоколи системи, так що Android може керувати ситуацією поки користувач виконує дії щоб приховати, відновити, зупинити або закрити вікно події.

2.2.10 Використання API ключів

Хмарні ендпоінти обробляють як API ключі, так і схеми аутентифікації, такі як Firebase або Auth0. Головна відмінність між цими двома поняттями полягає в наступному:

а) API ключі визначають проект, що викликається, додаток або сайт, виконуючи виклик до API;

б) токени аутентифікації ідентифікують користувача, людину, яка використовує додаток або сайт.

Щоб вирішити, яка схема є найбільш підходящою, важливо зрозуміти, що API ключі та аутентифікація можуть надати.

API ключі забезпечують:

а) ідентифікація проекту – ідентифікація додатку або проекту, що викликає цей API;

б) авторизація проекту перевірка, чи отримав викликаючий додаток доступ до API і включив API в своєму проекті.

Ключі API не такі безпечні, як токени аутентифікації, але вони ідентифікують додаток або проект, що викликає API. Вони генеруються в проекті, що виконує виклик, і ви можете обмежити їх використання таким середовищем, як діапазон IP-адрес або додаток для Android або iOS.

Ідентифікуючи викликаючий проект, ви можете використовувати API-ключі для асоціації інформації про використання з цим проектом. API ключі дозволяють Cloud Endpoints Frameworks фреймворкам хмарних ендпоінтів відхиляти виклики від проектів, які не мають доступу або включені в API.

API ключі, як правило, не вважаються безпечними; вони зазвичай доступні для клієнтів, що дозволяє легко вкрасти ключ API. Після того, як ключ було вкрадено, він не має терміну дії, тому його можна використовувати нескінченно, якщо тільки власник проекту не

скасує або не оновить ключ. Хоча обмеження, які ви можете встановити на ключ API, пом'якшують це, існують кращі підходи для авторизації.

API може обмежити деякі або всі свої методи вимогою API ключів. Це має сенс зробити, якщо:

а) потрібно заблокувати анонімний трафік;

б) потрібно контролювати кількість викликів, що виконуються до API;

в) потрібно визначити режим використання трафіку вашого API;

г) необхідно фільтрувати логи по ключу API.

API ключі не можуть бути використані для:

а) ідентифікації окремих користувачів - API ключі не ідентифікують користувачів, вони ідентифікують проекти;

б) безпечної авторизації;

в) визначення творців проекту.

2.2.11 Об’єкти Intent

Intent - це повідомлення, які не є синхронними і дозволяють різним компонентам програми запитувати функціональність у інших компонентів Android. Вони є об'єктами android.content.Intent і використовуються для запуску дії, як метод startActivity(), що дозволяє Intent почати дію. Intent також може містити дані через пакет, який може бути використаний приймаючим компонентом.

Активності розпочаті іншими активностями, називаються субактивностями, вони також можуть бути запущені Intent через startService(Intent) метод. Intent зазвичай перераховує дію, яку потрібно виконати, і надає дані, за якими така дія проти події відбувається.

Об'єкт Intent - це група інформації для компонента, який отримує інформацію intent plus, що представляє інтерес для операційної системи Android.

Операційна система Android визначає компоненти, які можуть реагувати на певний Intent за допомогою фільтра Intent, який визначає типи Intent, на які може реагувати активність, послуга або широкомовний приймач. Вони реєструються або шляхом перерахування їх в AndroidManifest.xml або динамічно за кодом програми.

Існує два типи Intent:

а) явні Intent, які явно визначають компонент, який повинен бути викликаний системою Android, використовуючи клас Java в якості ідентифікатора;

б) неявні Intent, які визначають дію, яка повинна бути виконана, і необов'язково дані,

які надають інформацію для дії.

У Intent є три частини інформації, які використовуються для дозволу: дія, тип і категорія. З допомогою цієї інформації виконується запит в PackageManager для компонента, який може обробляти Intent.

2.3 Принцип розробки задачі комівояжера з відомим рішенням

2.3.1 Сутність задачі

Необхідно розробити задачу комівояжера з відомим рішенням та будь-якої розмірності. Для розробки задачі з відомим рішенням потрібно створити матрицю з великими відстанями. Потім сгенерувати шлях рішення з короткими відстанями, котрі в декілька разів менші за будь-які інші відстані.

2.3.2 Принцип розробки

Спочатку потрібно згенерувати матрицю з великими значеннями відстаней, які мають бути, як мінімум, в декілька разів більше аніж значення, котрі ви збираєтесь використовувати в очікуваному рішенні задачі. Елементам матриці, котрі посилаються на самих себе присвоюємо значення 0.

Далі створюємо масив з використаними містами в матриці. Потім для першого рядка в матриці випадково обираємо комірку, котра не повинна містити в собі нуль, місто, на яке посилається ця комірка, не повинно бути використаним, а також обрана комірка не повинна бути першою в рядку матриці. Генеруємо число від 1 до 10 та присвоюємо його обраній комірці, і додаємо місто, на котре вона посилається до масиву використаних міст. Далі переходимо до рядку(міста), на котре посилалася попередньо обрана комірка, і повторюємо виконані дії доти, поки умови не будуть виконуватися. Потім беремо останнє обране місто, і з нього йдемо в першу комірку(місто), таким чином завершуючи шлях і повертаючись в початкове місто. На кожній ітерації додаємо сгенеровані відстані, і в кінці отримаємо найкоротшу відстань, яку будемо використовувати як очікуване рішення при тестуванні алгоритму.

2.4 Висновки до другого розділу

У цьому розділі було розглянуто та проаналізовано методи та засоби удосконалення мурашиного алгоритму для оптимізації туристичного маршруту. Проаналізовано проблему складності рішень задач вибору та розглянуто принципи розробки задачі комівояжера з відомим рішенням. Були розглянуті засоби для розробки додатку на Android.

3 РОЗРОБКА ПРОГРАМНОГО ДОДАТКУ

3.1 Постановка задачі

Потрібно розробити Android-додаток, котрий надаватиме змогу користувачу знаходити оптимальний шлях між декількома точками на мапі. Додаток повинен використовувати мурашиний алгоритм для знаходження оптимального шляху. Користувач також повинен мати змогу обирати не менше 10 точок на маршруті.

3.2 Архітектура Android-додатка

Мобільний додаток на ОС Android складається з декількох екранів, котрі мають назву активностей (Activity). Кожен екран користувацького інтерфейсу представляється класом. Android-додаток може складатися з декількох активностей і може переключатися між ними під час роботи додатку. Проект має в собі декілька класів з розширенням .java, які являються реалізаціями функціоналу цих активностей. Візуальний опис кожної активності лежить всередині xml-файлів. Кожна активність вміє адаптувати розміри своїх внутрішніх об’єктів до розміру екрану мобільного пристрою. Також всередині проекту лежить .xml файл, у якому описана активність, що буде запускатися першою при старті мобільного додатку.

Робота зі сторонніми API відбувається за допомогою HTTP-запитів з JSON-вмістом всередині. Завдяки стороннім сервісам можливо вести зручну розробку навігаційного додатку, не турбуючись про створення візуальної складової мапи.

Функціонально, додаток складається з приведених нижче модулів (активностей):

- основна активність містить в собі функціональну панель та інтерактивну мапу;
- активність меню налаштувань, де можна змінити мову, одиниці виміру або транспортний засіб;
- активність допомога, де надана коротка інформація по користуванню додатком;
- активність пошук на мапі, котрий необхідний для швидкого пошуку якогось місця.

3.2.1 Життєвий цикл активності

Коли користувач працює з мобільним телефоном, він постійно відкриває, закриває активності або переміщує їх на задній план. Активність має три стани:

а) активний (active або running) — коли активність знаходиться на передньому плані екрану смартфону і є центром для інтерактивної взаємодії з користувачем;

б) призупинений (paused) — коли активність втратила фокус, але користувач все ще може її бачити. Тобто інша активність знаходиться частково перекриває дану активність і знаходиться зверху;

в) зупинений (stopped) — якщо дана активність повністю закрита іншою активністю. Користувач більше не може її побачити та система може її знищити для звільнення пам’яті.

Якщо активність, яка була знищена системою, знову потрібно відобразити на екрані, вона повинна бути повністю перезапущена і відновлена в своєму попередньому стані. Активність при переході від одного стану до іншого отримує повідомлення через захищені методи:

а) onCreate() — викликається при створенні активності. Усередині цього методу налаштовують статичний інтерфейс активності — створюють уявлення, пов'язують дані зі списками і т.д. Цей метод приймає один параметр — об'єкт Bundle, що містить попередній стан активності (якщо цей стан було збережено);

б) onRestart() — викликається після того, як активність була зупинена і знову була запущена користувачем. Завжди супроводжується викликом методу onStart();

в) onStart() — викликається безпосередньо перед тим, як активність стає видимою. Супроводжується викликом методу onResume(), якщо активність отримує передній план, або викликом методу onStop(), якщо стає прихованою;

г) onResume() - викликається безпосередньо перед тим, як активність починає взаємодію з користувачем. Завжди супроводжується викликом методу onPause();

ґ) onPause() — викликається під час того, як система збирається запустити іншу активність;

д) onResume(), якщо активність повертається знову на передній план, або метод onStop(), якщо вікно активності стає невидимим для користувача;

е) onStop() — викликається, коли активність стає невидимою. Це може статися при її знищенні, або якщо була запущена інша активність(існуюча або нова), перекрив вікно поточної активності. Завжди супроводжує будь-який виклик методу onRestart(), якщо активність повертається на передній план для взаємодії з користувачем, або методу onDestroy(), якщо цяактивність знищується;

є) onDestroy() — викликається перед знищенням активності. Це останній запит, який отримує активність від системи. Даний метод викликається після закінчення роботи активності, при виклику методу finish() або в разі, коли система руйнує цей екземпляр активності для звільнення ресурсів. Ці два сценарії знищення можна визначити викликом методу isFinishing().

3.2.2 Запуск активності з використанням об'єктів Intent

Компоненти Android-додатків, в тому числі і активність, запускаються через об'єкти Intent. Таким чином виконується пізнє зв’язування між компонентами одного чи більше додатків під час виконання роботи додатку.

У кожному випадку система Android знаходить відповідну активність, щоб відповісти на Intent, ініціалізуючи його в разі необхідності.

Явний Intent зазвичай використовують для повідомлень всередині програми, наприклад, коли одна активність запускає іншу активність з цього додатка.

Неявний Intent використовують для запуску компонентів інших додатків. У файлі маніфесту програми зазвичай декларується фільтр Intent. За відсутності визначається адресата система Android переглядає фільтри Intent всіх додатків і знаходить компонент (Activity, Service або Broadcast Receiver), фільтр Intent якого є найбільш підходящим для виконання даного неявного Intent.

3.3 Використані API

Під час розробки були використані API GraphHopper та API Mapilion котрі є безкоштовними та знаходяться у відкритому доступі. Вони надають доступ до навігаційних мап, роботи з координатами, побудови шляху на мапі та візуального стилю мапи.

За допомогою API GraphHopper можна інтегрувати планування маршруту від А до Б, покрокову навігацію, оптимізацію маршруту та багато іншого в своєму додатку. Щоб дістатися з пункту А в пункт Б, можна використовувати маршрутизацію GraphHopper. Цей API надає час у дорозі, відстані, інструкції по повороту і геометрія маршруту на основі профілю подорожі, який користувач може відобразити на будь-якій карті, яку він забажає.

Mapilion надає користувачу змогу використовувати векторну мапу з можливістю її масштабування.

Перевагами Mapilion є:

а) масштабність: він призначений для обробки мільйонів запитів в хвилину. Запити обслуговуються за допомогою високо масштабованої хмарної інфраструктури;

б) висока доступність: використовується кілька рівнів кешування і кілька реплік для забезпечення високої доступності;

в) растрові плитки: пропонується два растрових стилю: OSM Bright і Kurviger Liberty;

г) плитка для затінення пагорбів: Mapilion надає два різних набору плиток для затінення пагорбів. V1 базується на даних SRTM, V2-на даних SRTM, ASTER GDEM і Copernicus.

Використання мапи Mapilion наведено в методі нижче , який запускається на старті додатку.

```
      @Override
          protected void onCreate (@Nullable Bundle savedInstanceState) {
              super.onCreate(savedInstanceState);
              setContentView(R.layout.activity_navigation_launcher);
              Mapbox.getInstance(this.getApplicationContext(),
getString(R.string.mapbox_access_token));
              Telemetry.disableOnUserRequest();
              ButterKnife.bind(this);
              mapView.setStyleUrl(getString(R.string.map_view_styleUrl));
              mapView.onCreate(savedInstanceState);
              mapView.getMapAsync(this);
              localeUtils = new LocaleUtils();
              if (getSupportActionBar() != null) {
                  getSupportActionBar().setTitle("");
              }
              showFirstStartIfNecessary();
          }
```

Файл налаштувань сторонніх API, в якому описані змінні котрі використовуються для роботи з GraphHopper API та Mapilion API.

```
      <?xml version="1.0" encoding="utf-8"?>
      <resources>
          <string name="mapbox_access_token" translatable = "false" >pk.test<
/string>
          <string name="gh_key" translatable="false">[GH_API_KEY]</string>
          <string name="base_url"
translatable="false">https://graphhopper.com/api/1/navigate/</string>

          <!-- Це URL-адреси стилю карти -->
          <!-- Стиль, який використовується для регулярного перегляду карти і
планування маршруту -->
          <string name="map_view_styleUrl" translatable =
"false">https://tiles.mapilion.com/assets/osm-
bright/style.json?key=[MAP_API_KEY]</string>
          <!-- Стиль "день", який використовується для навігації -->
          <string name="navigation_guidance_day" translatable =
"false">https://tiles.mapilion.com/assets/osm-
bright/style.json?key=[MAP_API__KEY]</string>
          <!-- Стиль "ніч", який використовується для навігації -->
          <string name="navigation_guidance_night" translatable =
"false">https://tiles.mapilion.com/assets/osm-
bright/style.json?key=[MAP_API_KEY]</string>
      </resources>
```

3.4 Розробка інтерфейсу додатку

До проектування дизайну потрібно віднестись якомога серйозніше, бо досвід користувача напряму залежить від візуальної складової додатку. В багатьох випадках саме дизайн вирішує, буде додаток успішним серед користувачів чи ні. Погано спроектований, не інтуітивно зрозумілий інтерфейс відлякує людей, та заважає швидкому та ефективному користуванню додатком.

Орієнтирами в дизайні були обрані максимальна простота, інтуітивність та приємні кольори. Користувач повинен розуміти за що відповідає кожна іконка тільки поглянувши на неї. Екран має бути вільним від великих кнопок, функціональні кнопки повинні займати мінімальний простір для кращого досвіду користування.

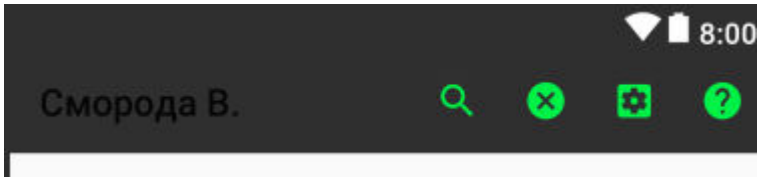


Рисунок 3.1 – Вигляд функціональної панелі

Нижче приведений код інтерфейсу функціональної панелі в файлі “navigation_view_activity_menu.xml”.

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">

    <item
        android:id="@+id/geocoding_search_btn"
        android:icon="@drawable/ic_search_black_24dp"
        android:title="@string/geocoding_search"
        app:showAsAction="always" />

    <item
        android:id="@+id/reset_route_btn"
        android:icon="@drawable/ic_cancel_black_24dp"
        android:title="@string/reset_route"
        app:showAsAction="always" />

    <item
        android:id="@+id/settings"
        android:icon="@drawable/ic_settings_applications_black_24dp"
        android:title="@string/settings"
        app:showAsAction="always" />

    <item
        android:id="@+id/help"
        android:icon="@drawable/ic_help_black_24dp"
        android:title="@string/help"
```

```
            app:showAsAction="always" />

    </menu>
```

Основним кольором кнопок було обрано зелений. Це було обгрунтовано тим, що людина на природньому рівні краще сприймає цей кольор. Спектральний зелений, а також колір листя і трави діє на нервову систему позитивно, кожна людина може відчути це, перебуваючи в лісі, в саду, на лузі. Нижче наведений код відповідний за кольори в функціональному рядку.

```
    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">

        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>

        <item
name="navigationViewMapStyle">@string/navigation_guidance_day</item>
    </style>
```

Меню налаштувань має простий вигляд:

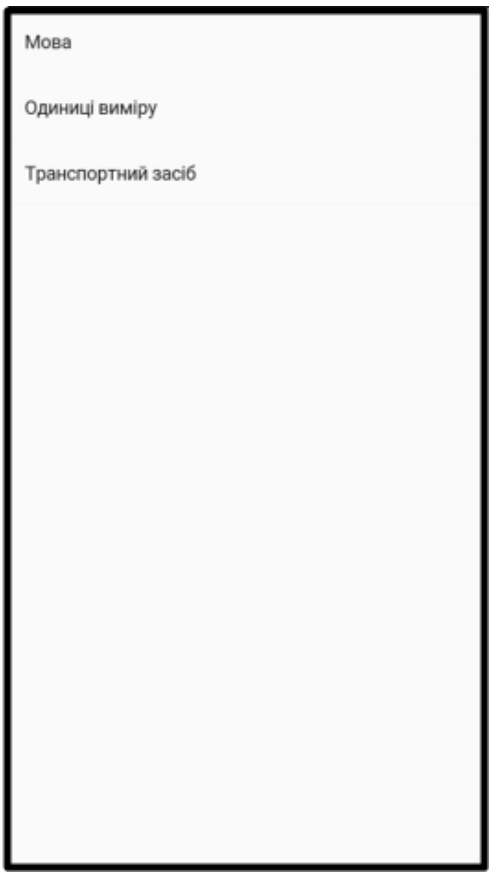


Рисунок 3.2 – Вигляд меню налаштувань

Воно містить можливість змінити мову, одиниці виміру та транспортний засіб для якого буде будуватися шлях між точками на мапі.

Код інтерфейсу меню налаштувань в xml:

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <ListPreference
        android:defaultValue="@string/default_locale"
        android:entries="@array/language_array"
        android:entryValues="@array/language_values_array"
        android:key="@string/language_key"
        android:title="@string/language" />
    <ListPreference
        android:defaultValue="@string/default_unit_type"
        android:entries="@array/unit_type_array"
        android:entryValues="@array/unit_type_values_array"
        android:key="@string/unit_type_key"
        android:title="@string/unit_type" />
    <ListPreference
        android:entries="@array/route_profile_array"
        android:entryValues="@array/route_profile_values_array"
        android:key="@string/route_profile_key"
        android:title="@string/route_profile" />
</PreferenceScreen>
```

3.5 Опис основних методів

Створення основного екрану з мапою:

```
protected void onCreate(@Nullable Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_navigation_launcher);
    Mapbox.getInstance(this.getApplicationContext(),
getString(R.string.mapbox_access_token));
    Telemetry.disableOnUserRequest();
    ButterKnife.bind(this);
    mapView.setStyleUrl(getString(R.string.map_view_styleUrl));
    mapView.onCreate(savedInstanceState);
    mapView.getMapAsync(this);
    localeUtils = new LocaleUtils();
    if (getSupportActionBar() != null) {
        getSupportActionBar().setTitle("");
    }
}
```

При виборі одного з пунктів налаштувань викликається метод наведений нижче:

```
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.settings:
            showSettings();
            return true;
```

```
            case R.id.reset_route_btn:
                clearRoute();
                return true;
            case R.id.geocoding_search_btn:
                showGeocodingInputDialog();
                return true;
            case R.id.help:
                showHelpDialog();
                return true;
            default:
                return super.onOptionsItemSelected(item);
        }
    }
```

Метод, котрий спрацьовує при натисканні на кнопку «Допомога»:

```
    private void showHelpDialog() {
            AlertDialog.Builder builder = new AlertDialog.Builder(this);
            builder.setTitle(R.string.help_message_title);
            builder.setMessage(Html.fromHtml("Щоб додати точку на мапі,
натисніть з утриманням. Ви повинні додати 2 точки для знаходження маршруту. Для
пошуку місця на мапі натисніть на лупу. Для того, щоб видалити всі обрані точки
маршруту натисніть на кнопку з хрестиком. Щоб відкрити меню налаштувань,
натисніть на шестерню."));
            SharedPreferences sharedPreferences =
PreferenceManager.getDefaultSharedPreferences(this);
            builder.setPositiveButton(R.string.agree, new
DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int id) {
                    SharedPreferences.Editor editor = sharedPreferences.edit();
                    editor.putInt(getString(R.string.first_start_dialog_key),
FIRST_START_DIALOG_VERSION);
                    editor.apply();
                }
            });

            AlertDialog dialog = builder.create();
            dialog.show();
        }
```

При довгому натисканні на мапу викликається метод додавання нової точки до маршруту:

```
    @Override
        public void onMapLongClick(@NonNull LatLng point) {
            addPointToRoute(point.getLatitude(), point.getLongitude());
            updateRouteAfterWaypointChange();
        }
```

Видалення всіх точок маршруту відбувається в методі clearRoute:

```
        private void clearRoute() {
            waypoints.clear();
            mapRoute.removeRoute();
            route = null;
            if (currentMarker != null) {
                mapboxMap.removeMarker(currentMarker);
                currentMarker = null;
            }
        }
```

Метод, котрий спрацьовує при пошуку локації та виводе на мапу знайдені точки:

```
public void onPostExecuteGeocodingSearch(List<GeocodingLocation> locations) {
        clearGeocodingResults();
        markers = new ArrayList<>(locations.size());

        if (locations.isEmpty()) {
            onError(R.string.error_geocoding_no_location);
            return;
        }

        List<LatLng> bounds = new ArrayList<>();
        Location lastKnownLocation = getLastKnownLocation();
        if (lastKnownLocation != null)
            bounds.add(new LatLng(lastKnownLocation.getLatitude(),
lastKnownLocation.getLongitude()));

        for (GeocodingLocation location : locations) {
            GeocodingPoint point = location.getPoint();
            MarkerOptions markerOptions = new MarkerOptions();
            LatLng latLng = new LatLng(point.getLat(), point.getLng());
            markerOptions.position(latLng);
            bounds.add(latLng);
            markerOptions.title(location.getName());
            String snippet = "";
            if (location.getStreet() != null) {
                snippet += location.getStreet();
                if (location.getHousenumber() != null)
                    snippet += " " + location.getHousenumber();
                snippet += "\n";
            }
            if (location.getCity() != null) {
                if (location.getPostcode() != null)
                    snippet += location.getPostcode() + " ";
                snippet += location.getCity() + "\n";
            }
            if (location.getCountry() != null)
                snippet += location.getCountry() + "\n";

            snippet += "\n\n Натисніть на це вікно\n щоб додати точку на
маршрут";
            if (!snippet.isEmpty())
                markerOptions.snippet(snippet);

markerOptions.icon(IconFactory.getInstance(this.getApplicationContext()).fromReso
urce(R.drawable.ic_map_marker));
            markers.add(mapboxMap.addMarker(markerOptions));
        }

        if (bounds.size() >= 2) {
            LatLngBounds.Builder boundsBuilder = new LatLngBounds.Builder();
            boundsBuilder.includes(bounds);
            animateCameraBbox(boundsBuilder.build(), CAMERA_ANIMATION_DURATION,
padding);
        } else if (bounds.size() == 1) {
            animateCamera(bounds.get(0));
        }
        hideLoading();
    }
```

Метод updateRouteAfterWaypointChange оновлює маршрут після додавання нової точки до маршруту:

```
private void updateRouteAfterWaypointChange() {
        if (this.waypoints.isEmpty()) {
            hideLoading();
        } else {
            Point lastPoint = this.waypoints.get(this.waypoints.size() -
1);
            LatLng latLng = new LatLng(lastPoint.latitude(),
lastPoint.longitude());
            setCurrentMarkerPosition(latLng);
            if (this.waypoints.size() > 0) {
                fetchRoute();
            } else {
                hideLoading();
            }
        }}
```

3.6 Опис роботи методів мурашиного алгоритму

Метод, приведений нижче запускає алгоритм по маршруту та оновлення феромонів.

```
private void runAlgorithm(int startingCity) {
        initializeVisitedCities();
        int selectedCity = startingCity;
        travel = new ArrayList<>();
        travel.add(startingCity);

        while (hasMoreCitiesToGo()) {
            updateSumProbability(selectedCity);
            getProbabilityForCity(selectedCity);
            selectedCity = chooseCity(selectedCity);
            if (selectedCity > 2000) {
                break;
            }
            travel.add(selectedCity);
        }
        setCompletedLength(travel.get(0), travel.get(travel.size() - 1));
        updatePheromones();
    }
```

Вибір наступного міста(точки) відбувається в методі нижче:

```
private int chooseCity(int currentCity) {
        ArrayList<Double> probabsForChoose = new ArrayList<>();
        for (int i = 0; i < CITIES; i++) {
            probabsForChoose.add(probabilities[currentCity][i]);
        }
        Collections.sort(probabsForChoose);

        Random random = new Random();
        int randomValue = random.nextInt(100);
        double city = Collections.max(probabsForChoose);
        double ender = probabsForChoose.get(1);
        for (int i = 0; i < CITIES; i++) {
            //Если i = последнему, выбираем последний город
```

```
                if (i == CITIES - 1) {
                    city = probabsForChoose.get(probabsForChoose.size() - 1);
                    break;
                    //Если случайное число попадает в отрезок
                } else if (randomValue > probabsForChoose.get(i) && randomValue
< ender) {
                    city = probabsForChoose.get(i + 1);
                    break;
                }
                //Правый край отрезка для сравнения
                if (i == CITIES - 2) {
                    ender                =                ender                +
probabsForChoose.get(probabsForChoose.size() - 1);
                } else {
                    ender = ender + probabsForChoose.get(i + 2);
                }
            }//end for
            for (int i = 0; i < CITIES; i++) {
                //Ищем город по вероятности
                //Если вероятность совпала, выбираем этот город
                if (probabilities[currentCity][i] == city) {
                    visitedCities[i] = true;
                    lengthOfWay += roadLength[currentCity][i];
                    return i;
                }
            }//end for
            return 2200;
        }
```

Оновлення феромонів викликається в методі updatePheromones:

```
    private void updatePheromones() {
            double pieceOfPheromone = Q / lengthOfWay;
            for (int i = 0; i < CITIES; i++) {
                for (int j = 0; j < CITIES; j++) {
                    roadPheromone[i][j] = roadPheromone[i][j] * (1 - P);
                }
            }
            int first = 0;
            int last = 1;
            for (int i = 0; i < CITIES; i++) {
                if (i == CITIES - 1) {
                    roadPheromone[travel.get(travel.size() - 1)][travel.get(0)]
+= pieceOfPheromone;
                } else {
                    roadPheromone[travel.get(first)][travel.get(last)]       +=
pieceOfPheromone;
                    first++;
                    last++;
                }
            }
        }
```

3.7 Оптимізація алгоритму

Для тонкого налаштування мурашиного алгоритму маємо п’ять змінних:

- ALPHA, відповідає за жадібність алгоритму, від 0 до 1, де 0 обирає найближчі точки;

- BETA, відповідає на вплив відстані на вибір наступної точки, від 0 до 1, де 0 ігнорує відстань;

- P, коефіцієнт випаровування феромонів, від 0 до 1;

- Q, коефіцієнт для знаходження нових феромонів;

- TIME, кількість циклів запуску алгоритму, що впливає на точність.

Проведеними дослідженнями було виявлено оптимальне налаштування змінних мурашиного алгоритму для отримання задовільного результату. Змінну ALPHA, котра відповідає за жадібність потрібно виставляти як 1. Змінну BETA, для знаходження оптимальних маршрутів рекомендується обирати як 1. Інакше можна втратити неочевидні маршрути для алгоритму, і при відносно невеликій кількості циклів оптимальний результат може не бути досягнений. Коефіцієнт випаровування P було вирішено вибрати 0.2, що дозволяє уникнути зациклень на одному маршруті та виявити нові, більш оптимальні маршрути. Оптимальне значення Q було знайдено як 300. TIME, що напряму впливає на точність алгоритму, виставлено на 200 ітерацій. Ця цифра є оптимальною з точки зору часу і продуктивності роботи алгоритму. Різниця між 200 і 500 ітерацій в більшості випадків незначна, і алгоритм знаходить оптимальний шлях у межах 200 ітерацій.

3.8 Приклад роботи додатку

При запуску додатку запускається головна активність, на якій користувачу відображається мапа світу і функціональна панель. Знайдемо на мапі столицю України, місто Київ, та довгими натисками додамо декілька точок на маршрут.

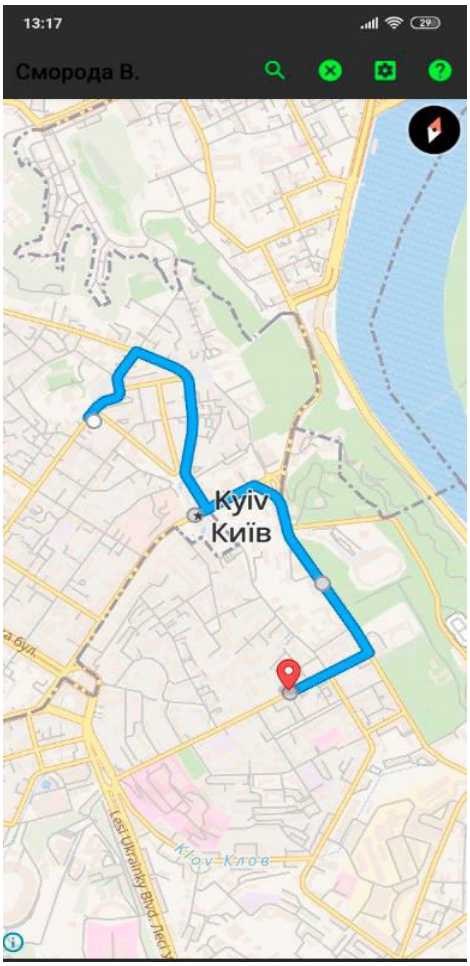


Рисунок 3.3 – Вигляд побудованого маршруту

Відразу ж було отримано оптимізований маршрут, котрий проходить послідовно через чотири вказаних точки на мапі.

Натиснемо на лупу і спробуємо знайти наступну точку для нашого маршруту.

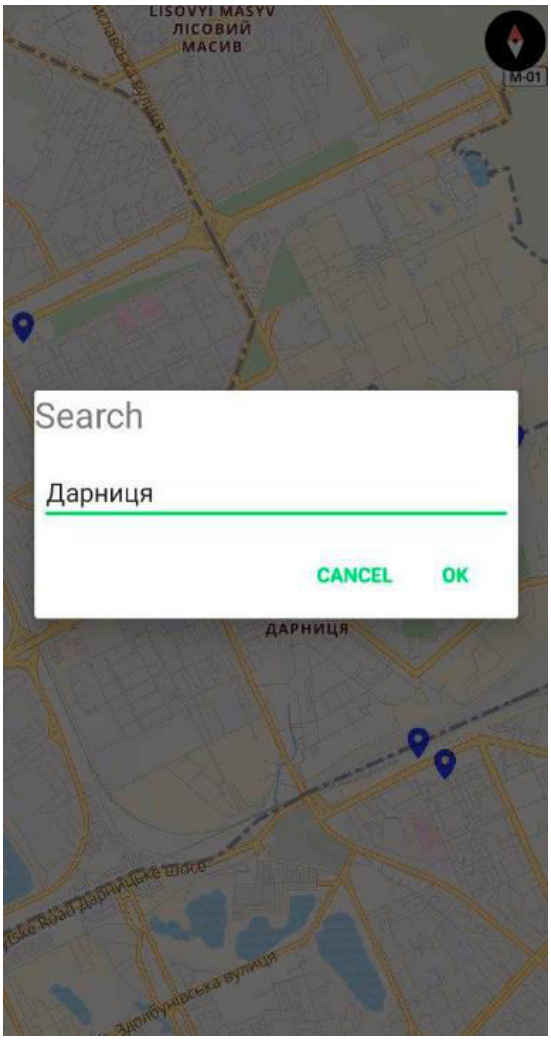


Рисунок 3.4 – Вікно пошуку локації

Введемо у поле вводу назву шуканого місця, наприклад Дарниця, та натиснемо на кнопку ОК.

Рисунок 3.5 – Результат пошуку локації

На мапі знайшовся район Дарниця і синіми позначками відмічено місця з назвою Дарниця. Довгим натиском на позначку додамо її на маршрут.

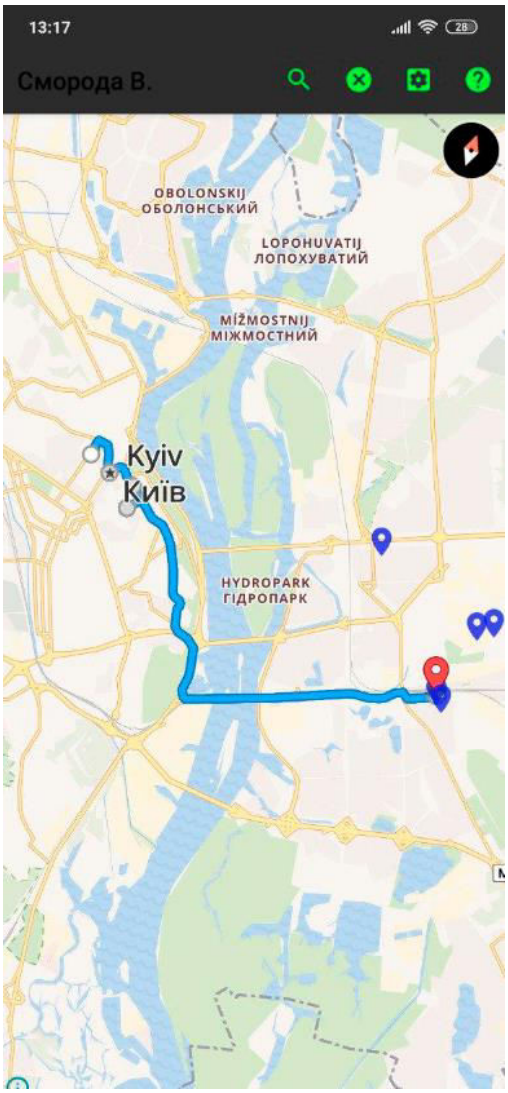


Рисунок 3.6 – Результат додавання ще одної точки на маршруті

Було отримано маршрут, котрий прокладено через п’ять вказаних точок на мапі.

3.9 Висновки до третього розділу

Був розроблений Android-додаток для прокладання туристичних маршрутів з використанням мурашиного алгоритму для оптимізації їх протяжності. Була спроектована архітектура додатка, до складу якого входять дві активності. Був розроблений дизайн кожної активності цього додатка, кінцевою метою якого є зручне, інтуїтивне використання всіх можливостей додатка. Був розроблений і налагоджений програмний код, який забезпечує динамічне функціонування додатка. В роботі додатка використовуються сторонні API, такі як GraphHopper та Mapilion. Наведено приклад роботи додатку при розробці реального туристичного маршруту.

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

В даному розділі проведено аналіз потенційних небезпечних та шкідливих виробничих факторів, причин пожеж. Розглянуті заходи, які дозволяють забезпечити гігієну праці і виробничу санітарію. На підставі аналізу розроблені заходи з техніки безпеки та рекомендації з пожежної профілактики.

Завданням даної роботи бакалавра було дослідження та удосконалення мурашиного алгоритму для розв'язання задачі комівояжера, і як результат було створено програмне забезпечення, котре використовує удосконалений алгоритм. За цим програмний забезпеченням в подальшому розроблятиметься реальна система, яка значно полегшить процес розв’язання задачі комівояжера. Так як в процесі проектування використовувалося спеціалізоване приміщення, то аналіз потенційно небезпечних і шкідливих виробничих чинників виконується для персонального комп'ютера на якому буде використовуватися розроблене програмне забезпечення.

4.1 Загальні питання з охорони праці

Умови праці на робочому місці, безпека технологічних процесів, машин, механізмів, устаткування та інших засобів виробництва, стан засобів колективного та індивідуального захисту, що використовуються працівником, а також санітарно-побутові умови повинні відповідати вимогам нормативних актів про охорону праці. В законі України «Про охорону праці» [32] визначається, що охорона праці - це система правових, соціально-економічних, організаційно-технічних, санітарно-гігієнічних і лікувально-профілактичних заходів та засобів, спрямованих на збереження життя, здоров'я і працездатності людини у процесі трудової діяльності.

При роботі з обчислювальною технікою змінюються фізичні і хімічні фактори навколишнього середовища: виникає статична електрика, електромагнітне випромінювання, змінюється температура і вологість, рівень вміст кисню і озону в повітрі.

4.2 Аналіз стану умов праці

Робота над створенням програмного забезпечення проходить в побутовому приміщенні. Для даної роботи достатньо однієї людини, для якої надано робоче місце зі стаціонарним комп’ютером.

4.2.1 Вимоги до приміщення

Для зручності спільної роботи з іншими працівниками (обговорення ідей, з’ясування проблем і т.д.) в кімнаті є диван і журнальний стіл. Задля дотримання визначеного рівня мікроклімату в будівлі встановлено систему опалення та кондиціонування.

Згідно до санітарних норм мікроклімату виробничих приміщень [33] розмір площі для одного робочого місця оператора персонального комп’ютера має бути не менше 6 кв. м, а об’єм – неменше 20 куб. м.

4.2.2 Вимоги до організації робочого місця

При порівнянні відповідності характеристик робочого місця нормативним основні вимоги до організації робочого місця [35] і відповідними фактичними значеннями для робочого місця, констатуємо повну відповідність.

За ступенем пожежної безпеки приміщення належить до категорії В. Кабінет оснащений переносним вуглекислотним вогнегасником ВВК-5 .

Наявна аптечка для надання долікарської допомоги, а також у кабінеті роблять вологе прибирання та щоденно провітрюють приміщення.

4.2.3 Навантаження та напруженість процесу праці

За фізичним навантаженням робота відноситься до категорії легкі роботи (Іа), її виконують сидячи з періодичним ходінням. Щодо характеру організування виконання дипломної роботи, то він підпадає під нав’язаний режим, оскільки певні розділи роботи необхідно виконати у встановлені конкретні терміни. За ступенем нервово-психічної напруги виконання роботи можна віднести до ІІ – ІІІ ступеня і кваліфікувати як помірно напружений – напружений за умови успішного виконання поставлених завдань.

Рекомендовано застосування екранних фільтрів, локальних світлофільтрів (засобів

індивідуального захисту очей) та інших засобів захисту, а також інші профілактичні заходи [35][38].

4.3 Виробнича санітарія

Роботу, пов'язану з ЕОП з ВДТ, у тому числі на тих, які мають робочі місця, обладнані ЕОМ з ВДТ і ПП, виконують із забезпеченням виконання НПАОП 0.00.-7.15-18 «Вимоги щодо безпеки та захисту здоров`я працівників під час роботи з екранними пристроями» [34].

4.3.1 Аналіз небезпечних та шкідливих факторів при розробці виробу

Роботу, пов'язану з ЕОП з ВДТ, у тому числі на тих, які мають робочі місця, обладнані ЕОМ з ВДТ і ПП, виконують із забезпеченням виконання , які встановлюють вимоги безпеки до обладнання робочих місць, до роботи із застосуванням ЕОМ з ВДТ і ПП. Переважно роботи за проектами виконують у кабінетах чи інших приміщеннях, де використовують різноманітне електрообладнання, зокрема персональні комп’ютери (ПК) та периферійні пристрої.

Робочі місця мають відповідати вимогам ДСанПіН 3.3.2.007-98 «Правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» [39].

4.3.2 Пожежна безпека

Згідно НАПБ А. 01.001-2014 «Правила пожежної безпеки в Україні» [36], небезпека розвитку пожежі на обчислювальному центрі обумовлюється застосуванням розгалужених систем електроживлення ЕОМ, вентиляції і кондиціювання. Небезпека загоряння пов'язана з особливістю комп'ютерів – іззначною кількістю щільно розташованих на монтажній платі і блоках електронних вузлів і схем, електричних і комутаційних кабелів, резисторів, конденсаторів, напівпровідникових діодів і транзисторів. Надійна робота окремих елементів і мікросхем в цілому забезпечується тільки в певних інтервалах температури, вологості і при заданих електричних параметрах. При відхиленні реальних умов експлуатації від розрахункових можуть виникнути пожежонебезпечні ситуації.

4.3.3 Електробезпека

На робочому місці виконуються наступні вимоги електробезпеки: ПК, периферійні пристрої та устаткування для обслуговування, електропроводи і кабелі за виконанням та ступенем захисту відповідають класу зони за ПУЕ (правила улаштування електроустановок), мають апаратуру захисту від струму короткого замикання та інших аварійних режимів. Лінія електромережі для живлення ПК, периферійних пристроїв і устаткування для обслуговування, виконана як окрема групова три- провідна мережа, шляхом прокладання фазового, нульового робочого та нульового захисного провідників. Нульовий захисний провідник використовується для заземлення (занулення) електроприймачів.

4.4 Гігієнічні вимоги до параметрів виробничого середовища

4.4.1 Мікроклімат

Дане приміщення обладнане системами опалення, кондиціонування повітря або припливно-витяжною вентиляцією. У приміщенні на робочому місці забезпечуються оптимальні значення параметрів мікроклімату: температури, відносної вологості й рухливості повітря у відповідності до [33]. Рівні позитивних і негативних іонів у повітрі мають відповідати [33]. Контроль параметрів мікроклімату в холодний і теплий період року здійснюється не менше 3-х разів на зміну (на початку, середині, в кінці).

4.4.2 Освітлення

Штучне освітлення в робочому приміщенні передбачається здійснювати з використанням люмінесцентних джерел світла в світильниках загального освітлення, рівень якого відповідає ДБН В.2.5-28:2018 «Природне і штучне освітлення» [37], оскільки люмінесцентні лампи мають високу потужність (80 Вт), тривалий термін служби (до 10000 годин), спектральний складом випромінюваного світла, близький до сонячного. При експлуатації ЕОМ виконується зорова робота IVв розряду точності (середня точність). При цьому нормована освітленість на робочому місці (Ен) рівна 200 лк. Джерелом природного освітлення є сонячне світло.

Розрахунок освітлення

Для виробничих та адміністративних приміщень світловий коефіцієнт приймається не менше -1/8, в побутових – 1/10:

$$S_b = \left(\frac{1}{5} / \frac{1}{10}\right) * S_n \quad (4.1)$$

$S_n = a \cdot b = 6 \cdot 4 = 24$ м^2,

$S = 1/10 \cdot 24 = 2{,}4$ м^2.

Приймаємо 1 вікно площею S=2,4 м^2.

Світильники загального освітлення розташовуються над робочими поверхнями в рівномірно-прямокутному порядку. Для організації освітлення в темний час доби передбачається обладнати приміщення, довжина якого складає 5 м, ширина 5 м, світильниками ЛПО2П, оснащеними лампами типа ЛБ (дві по 80 Вт) з світловим потоком 5400 лм кожна.

Розрахунок кількості світильників n виробляється по формулі (4.2):

$$n = \frac{E \times S \times Z \times K}{F \times U \times M}, \quad (4.2)$$

Підставивши числові значення у формулу (4.2), отримуємо:

$$n = \frac{300 \times 24 \times 1.1 \times 1.5}{5400 \times 0.575 \times 2} = 1.9$$

Приймаємо освітлювальну установку, яка складається з 2-х світильників, які складаються з 2-х люмінесцентних ламп загальною потужністю 80 Вт, напругою – 220 В.

4.4.3 Вентилювання

У приміщенні, де знаходяться ЕОМ, повітрообмін реалізується за допомогою природної організованої вентиляції. Цей метод забезпечує приток потрібної кількості свіжого повітря, що визначається в [33].

Також має здійснюватися провітрювання приміщення, в залежності від погодних умов, тривалість повинна бути не менше 10 хв. Найкращий обмін повітря здійснюється при наскрізному провітрюванні.

4.5 Заходи з організації виробничого середовища та попередження виникнення надзвичайних ситуацій

Відповідно до санітарно-гігієнічних нормативів та правил експлуатації обладнання наводимо приклади деяких заходів безпеки.

Заходи безпеки під час експлуатації персонального комп'ютера та периферійних пристроїв передбачають:

а) правильне організування місця праці та дотримання оптимальних режимів праці та відпочинку під час роботи з ПК;

б) експлуатацію сертифікованого обладнання;

в) дотримання заходів електробезпеки;

г) забезпечення оптимальних параметрів мікроклімату;

ґ) забезпечення раціонального освітлення місця праці (освітленість робочого місця не перевищувала 2/3 нормальної освітленості приміщення);

Заходи безпеки під час експлуатації інших електричних приладів передбачають дотримання таких правил:

а) постійно стежити за справним станом електромережі, розподільних щитків, вимикачів, штепсельних розеток, лампових патронів, а також мережевих кабелів живлення, за допомогою яких електроприлади під'єднують до електромережі;

б) постійно стежити за справністю ізоляції електромережі та мережевих кабелів, не допускаючи їхньої експлуатації з пошкодженою ізоляцією;

в) не тягнути за мережевий кабель, щоб витягти вилку з розетки;

г) не закривати меблями, різноманітним інвентарем вимикачі, штепсельні розетки;

ґ) не залишати включені електроприлади без нагляду;

д) не ставити на електроприлади матеріали, які можуть під дією теплоти, що виділяється, загорітися (канцелярські товари, сувенірну продукцію тощо).

4.5.1 Розрахунок захисного заземлення

Згідно з класифікацією приміщень за ступенем небезпеки ураження електричним струмом [35][38], приміщення в якому проводяться всі роботи відноситься до першого класу (без підвищеної небезпеки). Під час роботи використовуються електроустановки з напругою живлення 36 В, 220 В, та 360 В. Опір контуру заземлення повинен мати не більше 4 Ом.

Послідовність розрахунку:

а) Визначається необхідний опір штучних заземлювачів $R_{шт.з.}$:

$$R_{\text{шт.з.}} = \frac{R_{\text{д}} \cdot R_{\text{пр.з.}}}{R_{\text{пр.з.}} - R_{\text{д}}}, \quad (4.3)$$

де $R_{\text{пр.з.}}$–опір природних заземлювачів; $R_{\text{д}}$–допустимий опір заземлення. Якщо природні заземлювачі відсутні, то $R_{\text{шт.з.}} = R_{\text{д}}$.

Підставивши числові значення у формулу (4.3), отримуємо:

$$R_{\text{шт.з.}} = \frac{4 \cdot 40}{40 - 4} \approx 4\ \text{Ом}$$

б) Опір заземлення в значній мірі залежить від питомого опору ґрунту ρ,Ом·м. Приблизне значення питомого опору глини приймаємо $\rho = 40$ *Ом·м (табличне значення).*

в) Розрахунковий питомий опір ґрунту, $\rho_{розр}$, Ом·м, визначається відповідно для вертикальних заземлювачів $\rho_{розр.в}$, і горизонтальних $\rho_{розр.г}$, Ом·м за формулою:

$$\rho_{\text{розр.}} = \Psi \cdot \rho \quad (4.4)$$

$$\rho_{\text{розр.в}} = 1{,}7 \cdot 40 = 68\text{Ом} \cdot \text{м}$$

$$\rho_{\text{розр.в}} = 5{,}5 \cdot 40 = 220\text{Ом} \cdot \text{м}$$

г) Розраховується опір розтікання струму вертикального заземлювача $R_{\text{в}}$, Ом, за (4.5).

$$R_{з} = \frac{\rho}{2\pi \cdot l} \cdot \left(\ln \frac{2 \cdot l}{d} + \frac{1}{2} \ln \frac{4 \cdot t + l}{4 \cdot t - l} \right), \quad (4.5)$$

$$t = h_E + \frac{1_E}{2}, \quad (4.6)$$

$$t = 0{,}8 + \frac{3}{2} = 2{,}3\ \text{м}$$

$$R_{\text{в}} = \frac{68}{2 \cdot \pi \cdot 3} \cdot \left(\ln \frac{2 \cdot 3}{0{,}05} + \frac{1}{2} \cdot \ln \frac{4 \cdot 2{,}3 + 3}{4 \cdot 2{,}3 - 3} \right) = 18{,}5$$

Визначається теоретична кількість вертикальних заземлювачів n штук, без урахування коефіцієнта використання $\eta_в$:

$$n = \frac{2R_E}{R_Д} = \frac{2\times 18,5}{4} = 9,25 \quad (4.7)$$

І визначається коефіцієнт використання вертикальних електродів групового заземлювача без врахування впливу з'єднувальної стрічки$\eta_в$ = 0,57 (табличне значення).

Визначається необхідна кількість вертикальних заземлювачів з урахуванням коефіцієнта використання$\eta_в$,шт:

$$\mathrm{n} = \frac{2\cdot \mathrm{R_E}}{\mathrm{R_Д}\cdot {}^{\eta}\mathrm{E}} = \frac{2\cdot 18,5}{4\cdot 0,57} \approx 16 \quad (4.8)$$

Визначається довжина з'єднувальної стрічки горизонтального заземлювача l_c, м:

$$1_C = 1,05 \cdot L_B \cdot (n_B - 1), \quad (4.9)$$

де $L_в$– відстань між вертикальними заземлювачами, (прийняти за $L_в$ = 3м);

$n_в$– необхідна кількість вертикальних заземлювачів.

$$1_c = 1,05 \cdot 3 \cdot (16 - 1) \approx 48 \text{ м}.$$

Визначається опір розтіканню струму горизонтального заземлювача (з'єднувальної стрічки)$R_г$, Ом:

$$R_т = \frac{220}{2 \cdot \pi \cdot 48} \cdot \ln \frac{2 \cdot 48^2}{0,95 \cdot 0,15 \cdot 0,5} = 8,1 \text{ Ом} \quad (4.10)$$

Визначається коефіцієнт використання горизонтального заземлювача $\eta_{с.}$ відповідно до необхідної кількості вертикальних заземлювачів $n_в$.

Коефіцієнт використання з'єднувальної смуги $\eta_с$ = 0,3.

Розраховується результуючий опір заземлювального електроду з урахуванням з'єднувальної смуги:

$$R_{\text{заг.}} = \frac{R_E \cdot R_г}{R_E \cdot {}^{\eta}c + R_г \cdot n_E \cdot {}^{\eta}E} \leq R_д, \quad (4.11)$$

Висновок: дане захисне заземлення буде забезпечувати електробезпеку будівлі, так як виконується умова: $R_{заг} < 4$ Ом, а саме:

$$R_{заг} = \frac{18,5 \cdot 8,1}{18,5 \cdot 0,3 + 8,1 \cdot 16 \cdot 0,57} = 1,9 \leq R_д$$

4.5.2 Охорона навколишнього природного середовища

Діяльність за темою магістерської роботи, а саме робота за комп'ютером в процесі її виконання впливає на навколишнє природнє середовище і регламентується нормами діючого законодавства.

Основним екологічним аспектом в процесі діяльності за даними спеціальностями є процеси впливу на атмосферне повітря та процеси поводження з відходами, які утворюються, збираються, розміщуються, передаються на відалення (знешкодження), утилізацію, тощо в ІТ галузі.

Вплив на атмосферне повітря при нормальних умовах праці не оказує, бо не має в приміщенні сканерів, принтерів та інших джерел викиду забруднюючих речовин в повітря робочої зони.

В процесі діяльності комп'ютера виникають процеси поводження з відходами ІТ галузі. Нижче надано перелік відходів, що утворюються в процесі роботи:

а) Відпрацьовані люмінесцентні лампи - І клас небезпеки

б) Батарейки та акумулятори (малі) -ІІІ клас небезпеки

в) Змінні носії інформації - ІV клас небезпеки

г) Відходи друкуючих пристроїв - ІV клас небезпеки

ґ) Відпрацьований ізолюючий матеріал, дроти та кабелі - ІV клас небезпеки

д) Макулатура - ІV клас небезпеки

е) Побутові відходи - ІV клас небезпеки

Відходи в міру їх накопичення збирають у тару, відповідну класу небезпеки, з дотриманням правил безпеки, після чого доставляють до місця тимчасового зберігання відходів відповідно до затвердженої схеми їх розміщення. Зазначені для зберігання відходів місця чи об'єкти повинні використовуватися лише для заявлених відходів.

Не допускається зберігання відходів у невстановлених схемою місцях, а також перевищення норм тимчасового зберігання відходів.

Способи тимчасового зберігання відходів визначаються видом, агрегатним станом і

класом небезпеки відходів:

- Відходи I класу небезпеки зберігаються в герметичній тарі (сталеві бочки, контейнери). У міру наповнення тару з відходами закривають герметично сталевий кришкою;

- Відходи III класу небезпеки зберігаються в тарі, яка забезпечує локалізацію зберігання, дозволяє виконувати вантажно-розвантажувальні і транспортні роботи і виключає поширення в ОС шкідливих речовин;

- Відходи IV класу небезпеки можуть зберігатися відкрито на промисловому майданчику у вигляді конусоподібної купи, звідки їх автонавантажувачем перевантажують у самоскид і доставляють на місце утилізації або захоронення.

З метою визначення та прогнозування впливу відходів на навколишнє середовище, своєчасного виявлення негативних наслідків, їх запобігання відповідно до Закону України «Про відходи» повинен здійснюватися моніторинг місць утворення, зберігання, і видалення відходів.

4.6 Висновки до четвертого розділу

В результаті проведеної роботи було зроблено аналіз умов праці, шкідливих та небезпечних чинників, з якими стикається робітник. Було визначено параметри і певні характеристики приміщення для роботи над запропонованим проектом написаному в кваліфікаційній роботі, описано, які заходи потрібно зробити для того, щоб дане приміщення відповідало необхідним нормам і було комфортним і безпечним для робітника.

Приведені рекомендації щодо організації робочого місця, а також важливу інформацію щодо пожежної та електробезпеки. Були наведені розміри приміщення та наведено значення температури, вологості й рухливості повітря, необхідна кількість і потужність ламп та інші параметри, значення яких впливає на умови праці робітника, а також – наведені інструкції з охорони праці, техніки безпеки при роботі на комп’ютері.

Були зазначені заходи щодо поводження з екологічними аспектами впливу на навколишнє природне середовище.

ВИСНОВКИ

1. Проведено аналіз і дослідження існуючих методів оптимізації маршруту.
2. У результаті досліджень було знайдено параметри для налаштування мурашиного алгоритму, за якими він демонструє найкращі результати для оптимізації маршруту.
3. Здійснено удосконалення мурашиного алгоритму, щоб досягти більш точних результатів під час знаходження рішення.
4. Проведено розрахунки, встановлено, що при великій різниці відстаней за оптимальним маршрутом, у порівнянні з іншими, мурашиний алгоритм дозволяє знайти вірне рішення з високою ймовірністю навіть при розмірності матриці відстаней 100×100.
5. Мовою програмування Java розроблено Android-додаток, котрий використовує мурашиний алгоритм для оптимізації побудови маршруту на мапі.
6. Додаток розроблено за допомогою середовища програмування Android Studio. Для розробки графічного інтерфейсу програми було використано вбудований в Android Studio графічний редактор XML-коду.
7. Android-додаток, отриманий в результаті роботи, може бути використаний для прокладання найбільш коротких маршрутів на мапі.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Жеронимус Б.Л. Математико-статистический метод выборочного обследования пасажиропотоков / Б.Л. Жеронимус, Д.Д. Джумаев. – Автомобильний транспорт. – 1966. – № 4. – С. 43-44
2. Блатнов М.Д. Пасажирские автомобильние перевозки // М.Д. Блатнов – М.: Транспорт, 1981. – 222 с.
3. Дли М.И. Алгоритмы поддержки принятия решений по управлению инфраструктурными проектами на основе моделей муравьиных колоний / М.И. Дли, В.В. Гимаров, С.И. Глушко // Весник СГТУ. – 2012. – № 1 (64). – Вып. 2. – С. 423-427
4. Глушко С.И. Многоколониальные алгоритмы муравьиных колоний для решения двухкритериальной задачи выбора маршрута / С.И. Глушко // Информационные технологии, энергетика и экономика: Сб. тр. X Междунар. Науч.-техн. конф. – Смоленск: Универсум,. – 2013. – Т. 2. – С. 25-28
5. Ольховский С.Ю. Моделирование функционирования и розвития машрутизированих систем городского пасажирского транспорта: монография / С.Ю. Ольховский, В.В. Яворский. – Омск: Изд-во СибАДИ, 2001. – 138 с.
6. Мартинова Ю.А. Анализ опыта проектирования рациональних маршрутних сетей городского пасажирского транспорта / Ю.А. Мартинова // Интернет-журнал "Науковедения". – 2014. – № 2. – С. 121-131
7. Goss S. Self-organized shortcuts in the Argentine ant / S. Goss, S. Aron, J.L. Deneuborg // Naturwissenshaffen. – 1989. – Vol. 76.– P. 579-581
8. Dorigo M. Optimization, learning and natural algorithm / M. Dorigo – Ph.D. thesis, Politecnico di Milano, Italy, 1992
9. Dorigo M. Th. Stutzle, Ant Colony Optimization, / M. Dorigo. – 2004. – Massachusetts Institute of Technology. – 306 p.
10. Bonavear E. Swarm Intelligence: from Natural to Artificial Systems / E. Bonavear, M. Dorig. – Oxford University Press, 1999. – 307 p.
11. Corne D. New Ideas in Optimization / D. Corne, M. Dorigo, F. Glover. – McGravvHill, 1999. – 450 p.
12. Dorigo M. Swarm Intelligence, Ant Algorithms and Ant Colony Optimization / M. Dorigo // Reader for CEU Summer University Course "Complex System". – Budapest, Central European University, 2001. – P. 1-38
13. Штовба С.Д. Муравьиные алгоритмы / С.Д. Штовба // Exponenta Pro. Математика в приложениях. – 2003. – № 4. – С. 70-75

14. МакКоннелл Дж. Основы современных алгоритмов / Дж.МакКоннелл. – М.: Техносфера, 2004. – 368 с.
15. Обчислювальна складність [Електронний ресурс] / Режим доступу: https://www.wikiwand.com/uk/Обчислювальна_складність
16. Теорія складності обчислень [Електронний ресурс] / Режим доступу: https://www.wikiwand.com/uk/Теорія_складності_обчислень
17. Клас складності P [Електронний ресурс] / Режим доступу: https://www.wikiwand.com/uk/Клас_складності_P
18. NP-повна задача [Електронний ресурс] / Режим доступу: https://www.wikiwand.com/uk/NP-повна_задача
19. Теория алгоритмов и математическая логика, Тема 14 Складність алгоритмів [Електронний ресурс] / Режим доступу: https://elearning.sumdu.edu.ua/free_content/lectured:5de5178bb62ca7a97fe35cba8b92d1b337ee8101/latest/8125/index.html
20. Задача комівояжера [Електронний ресурс] / Режим доступу: https://www.wikiwand.com/ru/Задача_комівояжера
21. Реферат – Задача коммивояжёра 2 - Информатика [Електронний ресурс] / Режим доступу: https://www.ronl.ru/referaty/informatika/403425
22. Коршунов Ю. М. К 70 Математические основы кибернетики: Учеб. пособие для вузов. — 2-е изд., перераб. и доп. — М.: Энергия, 1980. — 424 с, ил.
23. Полный перебор – это… Что такое полный перебор? [Електронний ресурс] / Режим доступу: https://dikc.academic.ru/dic.nsf/ruwiki/201612
24. Метод гілок і меж [Електронний ресурс] / Режим доступу: https://www.wikiwand.com/uk/Метод_гілок_і_меж
25. Зенг В.А., Нифонтова Л.С. МЕТОД МОНТЕ-КАРЛО ДЛЯ РЕШЕНИЯ ЗАДАЧИ КОММИВОЯЖЁРА // Научное сообщество студентов XXI столетия. ТЕХНИЧЕСКИЕ НАУКИ: сб. ст. по мат. XIV междунар. студ. науч.-практ. конф. № 14. URL: http://sibac.info/archive/technic/8(11).pdf
26. Бионика [Електронний ресурс] / Режим доступу: https://www.wikiwand.com/ru/Бионика
27. Применение генетических алгоритмов для решения задачи коммивояжера | Наука | FANDOM Powered by Wikia [Електронний ресурс] / Режим доступу: ru.science.wikia.com/wiki/Применение_генетических_алгоритмов_для_решения_задачи_коммивояжера

28. НОУ ИНТУИТ | Лекция | Введение.Основы генетических алгоритмов [Електронний ресурс] / Режим доступу: https://www.intuit.ru/studies/courses/14227/1284/lecture/24168?page=12
29. Достоинства и недостатки генетических алгоритмов [Електронний ресурс] / Режим доступу: https://studopedia.su/14_162922_dostoinstva-i-nedostatki-geneticheskih-algoritmov.html
30. Муравьиные алгоритмы [Електронний ресурс] / Режим доступу: http://uran.donntu.org/~masters/2008/kita/khaustova/library/3.htm
31. Муравьи [Електронний ресурс] / Режим доступу: https://www.wikiwand.com/ru/Муравьи
32. Закон України «Про охорону праці». Вводиться в дію Постановою ВР № 2695-XII від 14.10.92, ВВР, 1992, № 49, ст.669. – Режим доступу: www. URL: https://zakon.rada.gov.ua/laws/show/2694-12
33. ДСН 3.3.6.042-99 «Санітарні норми мікроклімату виробничих приміщень». Вводиться в дію Постановою ВР № 42 від 01.12.1999. – Режим доступу: www. URL: https://zakon.rada.gov.ua/rada/show/va042282-99
34. НПАОП 0.00-7.15-18 «Вимоги щодо безпеки та захисту здоров`я працівників під час роботи з екранними пристроями». Зареєстровано в Міністерстві юстиції України 25 квітня 2018 р. за № 508/31960. – Режим доступу: www. URL: https://zakon.rada.gov.ua/laws/show/z0508-18
35. НПАОП 40.1-1.01-97 «Правила безпечної експлуатації електроустановок». Затверджено наказом Державного комітету України по нагляду за охороною праці № 257 від 6 жовтня 1997 р. – Режим доступу: www. URL: https://zakon.rada.gov.ua/laws/show/z0011-98
36. НАПБ А. 01.001-2014 «Правила пожежної безпеки в Україні». Затверджено Наказом Міністерства внутрішніх справ України № 1417 від 30.12.2014. – Режим доступу: www. URL: https://zakon4.rada.gov.ua/laws/show/z0252-15
37. ДБН В.2.5-28:2018 «Природне і штучне освітлення». – Режим доступу: www. URL: http://www.minregion.gov.ua/wp-content/uploads/2018/12/V2528-1.pdf
38. НПАОП 40.1-1.21-98 «Правила безпечної експлуатації електроустановок споживачів». Затверджено Наказом Держнаглядохоронпраці України № 4 від 09.01.98 – Режим доступу: www. URL: https://zakon.rada.gov.ua/laws/show/z0093-98
39. ДСанПіН 3.3.2.007-98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин». Вводиться в дію Постановою ВР № 7 від 10.12.1998. – Режим доступу: www. URL: https://zakon.rada.gov.ua/rada/show/v0007282-98

40. Android для разработчиков. 3-е изд. — СПб.: Питер, 2016. — 512 с.: ил. — (Серия «Библиотека программиста»)
41. M Anranur Uwaisy. Recommendation of Scheduling Tourism Routes using Tabu Search Method (Case Study Bandung) [Електронний ресурс] / M Anranur Uwaisy, Z.K.A. Baizal, M Yusza Reditya // Elsevier. – 2019. – Режим доступу до ресурсу: https://www.sciencedirect.com/science/article/pii/S1877050919310701
42. An optimized routing algorithm for vehicle ad-hoc networks [Електронний ресурс] / H. Bello-Salau, A.M. Aibinu, Z. Wang та ін.] // Elsevier. – 2019. – Режим доступу до ресурсу: https://www.sciencedirect.com/science/article/pii/S2215098618301162
43. Planning low-impact tourist paths within a Site of Community Importance through the optimisation of biological and logistic criteria [Електронний ресурс] / Alessandro Ferrarini, Graziano Rossi, Gilberto Parolo, Maria Ferloni // Elsevier. – 2018. – Режим доступу до ресурсу: https://www.sciencedirect.com/science/article/abs/pii/S0006320708000530
44. Arjun T Mulloorakam. Combined Objective Optimization for Vehicle Routing Using Genetic Algorithm [Електронний ресурс] / Arjun T Mulloorakam, Nidhish Mathew Nidhiry // Elsevier. – 2019. – Режим доступу до ресурсу: https://www.sciencedirect.com/science/article/pii/S2214785318329316
45. Solving the Feeder Vehicle Routing Problem using ant colony optimization [Електронний ресурс] / Ying-Hua Huang, Carola A. Blazquez, Shan-Huen Huang та ін.] // Elsevier. – 2019. – Режим доступу до ресурсу: https://www.sciencedirect.com/science/article/abs/pii/S0360835218305199.

ДОДАТОК А

Лістинг програми

```
1) import android.annotation.SuppressLint;
2) import android.app.AlertDialog;
3) import android.app.DialogFragment;
4) import android.content.DialogInterface;
5) import android.content.Intent;
6) import android.content.SharedPreferences;
7) import android.location.Location;
8) import android.net.Uri;
9) import android.os.Bundle;
10) import android.preference.PreferenceManager;
11) import android.support.annotation.NonNull;
12) import android.support.annotation.Nullable;
13) import android.support.design.widget.Snackbar;
14) import android.support.v7.app.AppCompatActivity;
15) import android.text.Html;
16) import android.view.Menu;
17) import android.view.MenuItem;
18) import android.view.View;
19) import android.widget.EditText;
20) import android.widget.ProgressBar;
21) import android.widget.Toast;
22) import com.graphhopper.directions.api.client.model.GeocodingLocation;
23) import com.graphhopper.directions.api.client.model.GeocodingPoint;
24) import com.mapbox.android.core.permissions.PermissionsListener;
25) import com.mapbox.android.core.permissions.PermissionsManager;
26) import com.mapbox.api.directions.v5.DirectionsCriteria;
27) import com.mapbox.api.directions.v5.models.DirectionsResponse;
28) import com.mapbox.api.directions.v5.models.DirectionsRoute;
29) import com.mapbox.core.constants.Constants;
30) import com.mapbox.geojson.LineString;
31) import com.mapbox.geojson.Point;
32) import com.mapbox.mapboxsdk.Mapbox;
33) import com.mapbox.mapboxsdk.annotations.IconFactory;
34) import com.mapbox.mapboxsdk.annotations.Marker;
35) import com.mapbox.mapboxsdk.annotations.MarkerOptions;
36) import com.mapbox.mapboxsdk.camera.CameraPosition;
37) import com.mapbox.mapboxsdk.camera.CameraUpdateFactory;
38) import com.mapbox.mapboxsdk.exceptions.InvalidLatLngBoundsException;
39) import com.mapbox.mapboxsdk.geometry.LatLng;
40) import com.mapbox.mapboxsdk.geometry.LatLngBounds;
41) import com.mapbox.mapboxsdk.maps.MapView;
42) import com.mapbox.mapboxsdk.maps.MapboxMap;
43) import com.mapbox.mapboxsdk.maps.OnMapReadyCallback;
44) import com.mapbox.mapboxsdk.maps.Telemetry;
45) import com.mapbox.mapboxsdk.plugins.locationlayer.LocationLayerPlugin;
46) import com.mapbox.mapboxsdk.plugins.locationlayer.modes.RenderMode;
47) import com.mapbox.services.android.navigation.ui.v5.NavigationLauncher;
48) import
com.mapbox.services.android.navigation.ui.v5.NavigationLauncherOptions;
49) import com.mapbox.services.android.navigation.ui.v5.route.NavigationMapRoute;
50) import
com.mapbox.services.android.navigation.ui.v5.route.OnRouteSelectionChangeListener
;
51) import com.mapbox.services.android.navigation.v5.navigation.NavigationRoute;
52) import com.mapbox.services.android.navigation.v5.utils.LocaleUtils;
53) import java.util.ArrayList;
54) import java.util.List;
55) import java.util.Locale;
```

```
56) import butterknife.BindView;
57) import butterknife.ButterKnife;
58) import retrofit2.Call;
59) import retrofit2.Response;
60) public class NavigationLauncherActivity extends AppCompatActivity implements
OnMapReadyCallback,
61) MapboxMap.OnMapLongClickListener, OnRouteSelectionChangeListener,
62) SolutionInputDialog.NoticeDialogListener, FetchSolutionTaskCallbackInterface,
63) FetchGeocodingTaskCallbackInterface,
GeocodingInputDialog.NoticeDialogListener,
64) PermissionsListener {
65) private static final int CAMERA_ANIMATION_DURATION = 1000;
66) private static final int DEFAULT_CAMERA_ZOOM = 16;
67) private static final int CHANGE_SETTING_REQUEST_CODE = 1;
68) private static final int FIRST_START_DIALOG_VERSION = 1;
69) private static final int FIRST_NAVIGATION_DIALOG_VERSION = 1;
70) private LocationLayerPlugin locationLayer;
71) private NavigationMapRoute mapRoute;
72) private MapboxMap mapboxMap;
73) private PermissionsManager permissionsManager;
74) @BindView(R.id.mapView)
75) MapView mapView;
76) @BindView(R.id.loading)
77) ProgressBar loading;
78) private Marker currentMarker;
79) private List<Marker> markers;
80) private List<Point> waypoints = new ArrayList<>();
81) private DirectionsRoute route;
82) private LocaleUtils localeUtils;
83) private final int[] padding = new int[]{50, 50, 50, 50};
84) private String currentJobId = "";
85) private String currentVehicleId = "";
86) private String currentGeocodingInput = "";
87) @Override
88) protected void onCreate(@Nullable Bundle savedInstanceState) {
89) super.onCreate(savedInstanceState);
90) setContentView(R.layout.activity_navigation_launcher);
91) Mapbox.getInstance(this.getApplicationContext(),
getString(R.string.mapbox_access_token));
92) Telemetry.disableOnUserRequest();
93) ButterKnife.bind(this);
94) mapView.setStyleUrl(getString(R.string.map_view_styleUrl));
95) mapView.onCreate(savedInstanceState);
96) mapView.getMapAsync(this);
97) localeUtils = new LocaleUtils();
98) if (getSupportActionBar() != null) {
99) getSupportActionBar().setTitle("");
100) }
101) showFirstStartIfNecessary();
102) }
103) @Override
104) public boolean onCreateOptionsMenu(Menu menu) {
105) getMenuInflater().inflate(R.menu.navigation_view_activity_menu, menu);
106) return true;
107) }
108) @Override
109) public boolean onOptionsItemSelected(MenuItem item) {
110) switch (item.getItemId()) {
111) case R.id.settings:
112) showSettings();
113) return true;
114) case R.id.navigate_btn:
115) launchNavigationWithRoute();
```

```
116) return true;
117) case R.id.reset_route_btn:
118) clearRoute();
119) return true;
120) case R.id.fetch_solution_btn:
121) showSolutionInputDialog();
122) return true;
123) case R.id.geocoding_search_btn:
124) showGeocodingInputDialog();
125) return true;
126) case R.id.help:
127) showHelpDialog();
128) return true;
129) default:
130) return super.onOptionsItemSelected(item);
131) }
132) }
133) private void showSettings() {
134) startActivityForResult(new Intent(this,
NavigationViewSettingsActivity.class), CHANGE_SETTING_REQUEST_CODE);
135) }
136) @Override
137) protected void onActivityResult(int requestCode, int resultCode, Intent
data) {
138) super.onActivityResult(requestCode, resultCode, data);
139) if (requestCode == CHANGE_SETTING_REQUEST_CODE && resultCode == RESULT_OK) {
140) boolean shouldRefetch =
data.getBooleanExtra(NavigationViewSettingsActivity.UNIT_TYPE_CHANGED, false)
141) || data.getBooleanExtra(NavigationViewSettingsActivity.LANGUAGE_CHANGED,
false)
142) || data.getBooleanExtra(NavigationViewSettingsActivity.PROFILE_CHANGED,
false);
143) if (waypoints.size() > 0 && shouldRefetch) {
144) fetchRoute();
145) }
146) }
147) }
148) @SuppressWarnings({"MissingPermission"})
149) @Override
150) protected void onStart() {
151) super.onStart();
152) mapView.onStart();
153) if (locationLayer != null) {
154) locationLayer.onStart();
155) }
156) }
157) @SuppressWarnings({"MissingPermission"})
158) @Override
159) public void onResume() {
160) super.onResume();
161) mapView.onResume();
162) handleIntent(getIntent());
163) }
164) @Override
165) protected void onNewIntent(Intent intent) {
166) super.onNewIntent(intent);
167) setIntent(intent);
168) }
169) private void handleIntent(Intent intent) {
170) if (intent != null) {
171) Uri data = intent.getData();
172) if (data != null && "graphhopper.com".equals(data.getHost())) {
173) if (data.getPath() != null) {
```

```
174) if (this.mapboxMap == null) {
175) return;
176) }
177) if (data.getPath().contains("maps")) {
178) clearRoute();
179) setRouteProfileToSharedPreferences(data.getQueryParameter("vehicle"));
180) List<String> points = data.getQueryParameters("point");
181) for (String point : points) {
182) String[] pointArr = point.split(",");
183) addPointToRoute(Double.parseDouble(pointArr[0]),
Double.parseDouble(pointArr[1]));
184) }
185) setStartFromLocationToSharedPreferences(false);
186) updateRouteAfterWaypointChange();
187) }
188) if (data.getPath().contains("api/1/vrp/solution")) {
189) clearRoute();
190) List<String> pathSegments = data.getPathSegments();
191) fetchVrpSolution(pathSegments.get(pathSegments.size() - 1),
data.getQueryParameter("vehicle_id"));
192) }
193) }
194) }
195) }
196) }
197) private void showFirstStartIfNecessary() {
198) SharedPreferences sharedPreferences =
PreferenceManager.getDefaultSharedPreferences(this);
199) if (sharedPreferences.getInt(getString(R.string.first_start_dialog_key), -1)
< FIRST_START_DIALOG_VERSION) {
200) showHelpDialog();
201) }
202) }
203) private void showHelpDialog() {
204) AlertDialog.Builder builder = new AlertDialog.Builder(this);
205) builder.setTitle(R.string.help_message_title);
206) builder.setMessage(Html.fromHtml("Щоб додати точку на мапі, натисніть з
утриманням. Ви повинні додати 2 точки для знаходження маршруту. Для пошуку місця
на мапі натисніть на лупу. Для того, щоб видалити всі обрані точки маршруту
натисніть на кнопку з хрестиком. Щоб відкрити меню налаштувань, натисніть на
шестерню."));
207) SharedPreferences sharedPreferences =
PreferenceManager.getDefaultSharedPreferences(this);
208) builder.setPositiveButton(R.string.agree, new
DialogInterface.OnClickListener() {
209) public void onClick(DialogInterface dialog, int id) {
210) SharedPreferences.Editor editor = sharedPreferences.edit();
211) editor.putInt(getString(R.string.first_start_dialog_key),
FIRST_START_DIALOG_VERSION);
212) editor.apply();
213) }
214) });
215) builder.setNeutralButton(R.string.github, new
DialogInterface.OnClickListener() {
216) public void onClick(DialogInterface dialog, int id) {
217) startActivity(new Intent(Intent.ACTION_VIEW,
Uri.parse("https://github.com/graphhopper/graphhopper-navigation-example")));
218) }
219) });
220) AlertDialog dialog = builder.create();
221) dialog.show();
222) }
223) private void showNavigationDialog() {
```

```
224) AlertDialog.Builder builder = new AlertDialog.Builder(this);
225) builder.setTitle(R.string.legal_title);
226) SharedPreferences sharedPreferences =
PreferenceManager.getDefaultSharedPreferences(this);
227) builder.setPositiveButton(R.string.agree, new
DialogInterface.OnClickListener() {
228) public void onClick(DialogInterface dialog, int id) {
229) SharedPreferences.Editor editor = sharedPreferences.edit();
230) editor.putInt(getString(R.string.first_navigation_dialog_key),
FIRST_NAVIGATION_DIALOG_VERSION);
231) editor.apply();
232) launchNavigationWithRoute();
233) }
234) });
235) AlertDialog dialog = builder.create();
236) dialog.show();
237) }
238) private void fetchVrpSolution(String jobId, String vehicleId) {
239) currentJobId = jobId;
240) currentVehicleId = vehicleId;
241) showLoading();
242) new FetchSolutionTask(this, getString(R.string.gh_key)).execute(new
FetchSolutionConfig(currentJobId, currentVehicleId));
243) }
244) @Override
245) public void onPause() {
246) super.onPause();
247) mapView.onPause();
248) }
249) @Override
250) public void onLowMemory() {
251) super.onLowMemory();
252) mapView.onLowMemory();
253) }
254) @Override
255) protected void onStop() {
256) super.onStop();
257) mapView.onStop();
258) if (locationLayer != null) {
259) locationLayer.onStop();
260) }
261) }
262) @Override
263) protected void onDestroy() {
264) super.onDestroy();
265) mapView.onDestroy();
266) }
267) @Override
268) protected void onSaveInstanceState(Bundle outState) {
269) super.onSaveInstanceState(outState);
270) mapView.onSaveInstanceState(outState);
271) }
272) private void clearRoute() {
273) waypoints.clear();
274) mapRoute.removeRoute();
275) route = null;
276) if (currentMarker != null) {
277) mapboxMap.removeMarker(currentMarker);
278) currentMarker = null;
279) }
280) }
281) private void clearGeocodingResults() {
282) if (markers != null) {
```

```
283) for (Marker marker : markers) {
284) this.mapboxMap.removeMarker(marker);
285) }
286) markers.clear();
287) }
288) }
289) @Override
290) public void onMapReady(MapboxMap mapboxMap) {
291) this.mapboxMap = mapboxMap;
292) this.mapboxMap.getUiSettings().setAttributionDialogManager(new
GHAttributionDialogManager(this.mapView.getContext(), this.mapboxMap));
293) this.mapboxMap.addOnMapLongClickListener(this);
294) initMapRoute();
295) this.mapboxMap.setOnInfoWindowClickListener(new
MapboxMap.OnInfoWindowClickListener() {
296) @Override
297) public boolean onInfoWindowClick(@NonNull Marker marker) {
298) for (Marker geocodingMarker : markers) {
299) if (geocodingMarker.getId() == marker.getId()) {
300) LatLng position = geocodingMarker.getPosition();
301) addPointToRoute(position.getLatitude(), position.getLongitude());
302) updateRouteAfterWaypointChange();
303) marker.hideInfoWindow();
304) return true;
305) }
306) }
307) return true;
308) }
309) });
310) // Check for location permission
311) permissionsManager = new PermissionsManager(this);
312) if (!PermissionsManager.areLocationPermissionsGranted(this)) {
313) permissionsManager.requestLocationPermissions(this);
314) } else {
315) initLocationLayer();
316) }
317) handleIntent(getIntent());
318) }
319) @Override
320) public void onMapLongClick(@NonNull LatLng point) {
321) addPointToRoute(point.getLatitude(), point.getLongitude());
322) updateRouteAfterWaypointChange();
323) }
324) private void addPointToRoute(double lat, double lng) {
325) waypoints.add(Point.fromLngLat(lng, lat));
326) }
327) @Override
328) public void onNewPrimaryRouteSelected(DirectionsRoute directionsRoute) {
329) route = directionsRoute;
330) }
331) @SuppressWarnings({"MissingPermission"})
332) private void initLocationLayer() {
333) locationLayer = new LocationLayerPlugin(mapView, mapboxMap);
334) locationLayer.setRenderMode(RenderMode.COMPASS);
335) Location lastKnownLocation = getLastKnownLocation();
336) if (lastKnownLocation != null) {
337) // TODO we could think about zoom to the user location later on as well
338) animateCamera(new LatLng(lastKnownLocation.getLatitude(),
lastKnownLocation.getLongitude()));
339) }
340) }
341) private void initMapRoute() {
342) mapRoute = new NavigationMapRoute(mapView, mapboxMap);
```

```
343) mapRoute.setOnRouteSelectionChangeListener(this);
344) }
345) private void fetchRoute() {
346) NavigationRoute.Builder builder = NavigationRoute.builder(this)
347) .accessToken("pk." + getString(R.string.gh_key))
348) .baseUrl(getString(R.string.base_url))
349) .user("gh")
350) .alternatives(true);
351) boolean startFromLocation = getStartFromLocationFromSharedPreferences();
352) if (!startFromLocation && waypoints.size() < 2 || startFromLocation &&
waypoints.size() < 1) {
353) onError(R.string.error_not_enough_waypoints);
354) return;
355) }
356) if (startFromLocation) {
357) Location lastKnownLocation = getLastKnownLocation();
358) if (lastKnownLocation == null) {
359) onError(R.string.error_location_not_found);
360) return;
361) } else {
362) Point location = Point.fromLngLat(lastKnownLocation.getLongitude(),
lastKnownLocation.getLatitude());
363) if (lastKnownLocation.hasBearing())
364) builder.origin(location, (double) lastKnownLocation.getBearing(), 90.0);
365) else
366) builder.origin(location);
367) }
368) }
369) for (int i = 0; i < waypoints.size(); i++) {
370) Point p = waypoints.get(i);
371) if (i == 0 && !startFromLocation) {
372) builder.origin(p);
373) } else if (i < waypoints.size() - 1) {
374) builder.addWaypoint(p);
375) } else {
376) builder.destination(p);
377) }
378) }
379) showLoading();
380) setFieldsFromSharedPreferences(builder);
381) builder.build().getRoute(new SimplifiedCallback() {
382) @Override
383) public void onResponse(Call<DirectionsResponse> call,
Response<DirectionsResponse> response) {
384) if (validRouteResponse(response)) {
385) route = response.body().routes().get(0);
386) mapRoute.addRoutes(response.body().routes());
387) boundCameraToRoute();
388) } else {
389) Snackbar.make(mapView, R.string.error_calculating_route,
Snackbar.LENGTH_LONG).show();
390) }
391) hideLoading();
392) }
393) @Override
394) public void onFailure(Call<DirectionsResponse> call, Throwable throwable) {
395) super.onFailure(call, throwable);
396) Snackbar.make(mapView, R.string.error_calculating_route,
Snackbar.LENGTH_LONG).show();
397) hideLoading();
398) }
399) });
400) }
```

```
401) @SuppressLint("MissingPermission")
402) private Location getLastKnownLocation() {
403) if (locationLayer != null) {
404) return locationLayer.getLastKnownLocation();
405) }
406) return null;
407) }
408) private void updateRouteAfterWaypointChange() {
409) if (this.waypoints.isEmpty()) {
410) hideLoading();
411) } else {
412) Point lastPoint = this.waypoints.get(this.waypoints.size() - 1);
413) LatLng latLng = new LatLng(lastPoint.latitude(), lastPoint.longitude());
414) setCurrentMarkerPosition(latLng);
415) if (this.waypoints.size() > 0) {
416) fetchRoute();
417) } else {
418) hideLoading();
419) }
420) }
421) }
422) private void setFieldsFromSharedPreferences(NavigationRoute.Builder builder)
{
423) builder
424) .language(getLanguageFromSharedPreferences())
425) .voiceUnits(getUnitTypeFromSharedPreferences())
426) .profile(getRouteProfileFromSharedPreferences());
427) }
428) private String getUnitTypeFromSharedPreferences() {
429) SharedPreferences sharedPreferences =
PreferenceManager.getDefaultSharedPreferences(this);
430) String defaultUnitType = getString(R.string.default_unit_type);
431) String unitType =
sharedPreferences.getString(getString(R.string.unit_type_key), defaultUnitType);
432) if (unitType.equals(defaultUnitType)) {
433) unitType = localeUtils.getUnitTypeForDeviceLocale(this);
434) }
435) return unitType;
436) }
437) private Locale getLanguageFromSharedPreferences() {
438) SharedPreferences sharedPreferences =
PreferenceManager.getDefaultSharedPreferences(this);
439) String defaultLanguage = getString(R.string.default_locale);
440) String language =
sharedPreferences.getString(getString(R.string.language_key), defaultLanguage);
441) if (language.equals(defaultLanguage)) {
442) return localeUtils.inferDeviceLocale(this);
443) } else {
444) return new Locale(language);
445) }
446) }
447) private boolean getShouldSimulateRouteFromSharedPreferences() {
448) SharedPreferences sharedPreferences =
PreferenceManager.getDefaultSharedPreferences(this);
449) return sharedPreferences.getBoolean(getString(R.string.simulate_route_key),
false);
450) }
451) private boolean getStartFromLocationFromSharedPreferences() {
452) SharedPreferences sharedPreferences =
PreferenceManager.getDefaultSharedPreferences(this);
453) return
sharedPreferences.getBoolean(getString(R.string.start_from_location_key), true);
454) }
```

```
455) private void setStartFromLocationToSharedPreferences(boolean
setStartFromLocation) {
456) SharedPreferences sharedPreferences =
PreferenceManager.getDefaultSharedPreferences(this);
457) SharedPreferences.Editor editor = sharedPreferences.edit();
458) editor.putBoolean(getString(R.string.start_from_location_key),
setStartFromLocation);
459) editor.apply();
460) }
461) private void setRouteProfileToSharedPreferences(String ghVehicle) {
462) if (ghVehicle == null)
463) return;
464) SharedPreferences sharedPreferences =
PreferenceManager.getDefaultSharedPreferences(this);
465) SharedPreferences.Editor editor = sharedPreferences.edit();
466) String routeProfile;
467) switch (ghVehicle) {
468) case "foot":
469) case "hike":
470) routeProfile = "walking";
471) break;
472) case "bike":
473) case "mtb":
474) case "racingbike":
475) routeProfile = "cycling";
476) break;
477) default:
478) routeProfile = "driving";
479) }
480) editor.putString(getString(R.string.route_profile_key), routeProfile);
481) editor.apply();
482) }
483) private String getRouteProfileFromSharedPreferences() {
484) SharedPreferences sharedPreferences =
PreferenceManager.getDefaultSharedPreferences(this);
485) return sharedPreferences.getString(
486) getString(R.string.route_profile_key), DirectionsCriteria.PROFILE_DRIVING
487) );
488) }
489) private void launchNavigationWithRoute() {
490) if (route == null) {
491) Snackbar.make(mapView, R.string.error_route_not_available,
Snackbar.LENGTH_SHORT).show();
492) return;
493) }
494) SharedPreferences sharedPreferences =
PreferenceManager.getDefaultSharedPreferences(this);
495) if
(sharedPreferences.getInt(getString(R.string.first_navigation_dialog_key), -1) <
FIRST_NAVIGATION_DIALOG_VERSION) {
496) showNavigationDialog();
497) return;
498) }
499) Location lastKnownLocation = getLastKnownLocation();
500) if (lastKnownLocation != null && waypoints.size() > 1) {
501) float[] distance = new float[1];
502) Location.distanceBetween(lastKnownLocation.getLatitude(),
lastKnownLocation.getLongitude(), waypoints.get(0).latitude(),
waypoints.get(0).longitude(), distance);
503) if (distance[0] > 100) {
504) AlertDialog.Builder builder = new AlertDialog.Builder(this);
505) builder.setTitle(R.string.error_too_far_from_start_title);
506) builder.setMessage(R.string.error_too_far_from_start_message);
```

```
507) builder.setPositiveButton(R.string.ok, new DialogInterface.OnClickListener()
{
508) public void onClick(DialogInterface dialog, int id) {
509) waypoints.set(0, Point.fromLngLat(lastKnownLocation.getLongitude(),
lastKnownLocation.getLatitude()));
510) fetchRoute();
511) }
512) });
513) builder.setNegativeButton(R.string.cancel, new
DialogInterface.OnClickListener() {
514) public void onClick(DialogInterface dialog, int id) {
515) _launchNavigationWithRoute();
516) }
517) });
518) AlertDialog dialog = builder.create();
519) dialog.show();
520) } else {
521) _launchNavigationWithRoute();
522) }
523) } else {
524) _launchNavigationWithRoute();
525) }
526) }
527) private void _launchNavigationWithRoute() {
528) NavigationLauncherOptions.Builder optionsBuilder =
NavigationLauncherOptions.builder()
529) .shouldSimulateRoute(getShouldSimulateRouteFromSharedPreferences())
530) .directionsProfile(getRouteProfileFromSharedPreferences())
531) .waynameChipEnabled(false);
532) optionsBuilder.directionsRoute(route);
533) NavigationLauncher.startNavigation(this, optionsBuilder.build());
534) }
535) private boolean validRouteResponse(Response<DirectionsResponse> response) {
536) return response.body() != null && !response.body().routes().isEmpty();
537) }
538) private void hideLoading() {
539) if (loading.getVisibility() == View.VISIBLE) {
540) loading.setVisibility(View.INVISIBLE);
541) }
542) }
543) private void showLoading() {
544) if (loading.getVisibility() == View.INVISIBLE) {
545) loading.setVisibility(View.VISIBLE);
546) }
547) }
548) private void boundCameraToRoute() {
549) if (route != null) {
550) List<Point> routeCoords = LineString.fromPolyline(route.geometry(),
551) Constants.PRECISION_6).coordinates();
552) List<LatLng> bboxPoints = new ArrayList<>();
553) for (Point point : routeCoords) {
554) bboxPoints.add(new LatLng(point.latitude(), point.longitude()));
555) }
556) if (bboxPoints.size() > 1) {
557) try {
558) LatLngBounds bounds = new
LatLngBounds.Builder().includes(bboxPoints).build();
559) animateCameraBbox(bounds, CAMERA_ANIMATION_DURATION, padding);
560) } catch (InvalidLatLngBoundsException exception) {
561) Toast.makeText(this, R.string.error_valid_route_not_found,
Toast.LENGTH_SHORT).show();
562) }
563) }
```

```
564) }
565) }
566) private void animateCameraBbox(LatLngBounds bounds, int animationTime, int[]
padding) {
567) CameraPosition position = mapboxMap.getCameraForLatLngBounds(bounds,
padding);
568) mapboxMap.animateCamera(CameraUpdateFactory.newCameraPosition(position),
animationTime);
569) }
570) private void animateCamera(LatLng point) {
571) mapboxMap.animateCamera(CameraUpdateFactory.newLatLngZoom(point,
DEFAULT_CAMERA_ZOOM), CAMERA_ANIMATION_DURATION);
572) }
573) private void setCurrentMarkerPosition(LatLng position) {
574) if (position != null) {
575) if (currentMarker == null) {
576) MarkerOptions markerOptions = new MarkerOptions()
577) .position(position);
578) currentMarker = mapboxMap.addMarker(markerOptions);
579) } else {
580) currentMarker.setPosition(position);
581) }
582) }
583) }
584) private void updateWaypoints(List<Point> points) {
585) if (points.size() > 24) {
586) onError(R.string.error_too_many_waypoints);
587) return;
588) }
589) clearRoute();
590) this.waypoints = points;
591) updateRouteAfterWaypointChange();
592) }
593) private void showSolutionInputDialog() {
594) SolutionInputDialog dialog = new SolutionInputDialog();
595) dialog.setJobId(currentJobId);
596) dialog.setVehicleId(currentVehicleId);
597) dialog.show(getFragmentManager(), "gh-example");
598) }
599) private void showGeocodingInputDialog() {
600) GeocodingInputDialog dialog = new GeocodingInputDialog();
601) dialog.setGeocodingInput(currentGeocodingInput);
602) dialog.show(getFragmentManager(), "gh-example");
603) }
604) @Override
605) public void onDialogPositiveClick(DialogFragment dialog) {
606) EditText jobId = dialog.getDialog().findViewById(R.id.job_id);
607) if (jobId != null) {
608) EditText vehicleId = dialog.getDialog().findViewById(R.id.vehicle_id);
609) fetchVrpSolution(jobId.getText().toString(),
vehicleId.getText().toString());
610) }
611) EditText search = dialog.getDialog().findViewById(R.id.geocoding_input_id);
612) if (search != null) {
613) currentGeocodingInput = search.getText().toString();
614) showLoading();
615) String point = null;
616) LatLng pointLatLng = this.mapboxMap.getCameraPosition().target;
617) if (pointLatLng != null)
618) point = pointLatLng.getLatitude() + "," + pointLatLng.getLongitude();
619) new FetchGeocodingTask(this, getString(R.string.gh_key)).execute(new
FetchGeocodingConfig(currentGeocodingInput,
getLanguageFromSharedPreferences().getLanguage(), 5, false, point, "default"));
```

```
620) }
621) }
622) @Override
623) public void onError(int message) {
624) Snackbar.make(mapView, message, Snackbar.LENGTH_LONG).show();
625) }
626) @Override
627) public void onPostExecuteGeocodingSearch(List<GeocodingLocation> locations)
{
628) clearGeocodingResults();
629) markers = new ArrayList<>(locations.size());
630) if (locations.isEmpty()) {
631) onError(R.string.error_geocoding_no_location);
632) return;
633) }
634) List<LatLng> bounds = new ArrayList<>();
635) Location lastKnownLocation = getLastKnownLocation();
636) if (lastKnownLocation != null)
637) bounds.add(new LatLng(lastKnownLocation.getLatitude(),
lastKnownLocation.getLongitude()));
638) for (GeocodingLocation location : locations) {
639) GeocodingPoint point = location.getPoint();
640) MarkerOptions markerOptions = new MarkerOptions();
641) LatLng latLng = new LatLng(point.getLat(), point.getLng());
642) markerOptions.position(latLng);
643) bounds.add(latLng);
644) markerOptions.title(location.getName());
645) String snippet = "";
646) if (location.getStreet() != null) {
647) snippet += location.getStreet();
648) if (location.getHousenumber() != null)
649) snippet += " " + location.getHousenumber();
650) snippet += "\n";
651) }
652) if (location.getCity() != null) {
653) if (location.getPostcode() != null)
654) snippet += location.getPostcode() + " ";
655) snippet += location.getCity() + "\n";
656) }
657) if (location.getCountry() != null)
658) snippet += location.getCountry() + "\n";
659) snippet += "\n\n Натисніть на це вікно\n щоб додати точку на маршрут";
660) if (!snippet.isEmpty())
661) markerOptions.snippet(snippet);
662)
markerOptions.icon(IconFactory.getInstance(this.getApplicationContext()).fromReso
urce(R.drawable.ic_map_marker));
663) markers.add(mapboxMap.addMarker(markerOptions));
664) }
665) if (bounds.size() >= 2) {
666) LatLngBounds.Builder boundsBuilder = new LatLngBounds.Builder();
667) boundsBuilder.includes(bounds);
668) animateCameraBbox(boundsBuilder.build(), CAMERA_ANIMATION_DURATION,
padding);
669) } else if (bounds.size() == 1) {
670) animateCamera(bounds.get(0));
671) }
672) hideLoading();
673) }
674) @Override
675) public void onPostExecute(List<Point> points) {
676) if (getStartFromLocationFromSharedPreferences() && !points.isEmpty()) {
677) points.remove(0);
```

```
678) }
679) updateWaypoints(points);
680) }
681) @Override
682) public void onRequestPermissionsResult(int requestCode, @NonNull String[]
permissions,
683) @NonNull int[] grantResults) {
684) permissionsManager.onRequestPermissionsResult(requestCode, permissions,
grantResults);
685) }
686) @Override
687) public void onExplanationNeeded(List<String> permissionsToExplain) {
688) Toast.makeText(this, "This app needs location permissions to work
properly.",
689) Toast.LENGTH_LONG).show();
690) }
691) @Override
692) public void onPermissionResult(boolean granted) {
693) if (granted) {
694) initLocationLayer();
695) } else {
696) Toast.makeText(this, "You didn't grant location permissions.",
697) Toast.LENGTH_LONG).show();
698) }
699) }
700) }
701) package com.graphhopper.navigation.example;
702) import java.util.ArrayList;
703) import java.util.Collections;
704) import java.util.Random;
705) /**
706) * Класс реализует муравьиный алгоритм, а также вспомогательные методы
707) *
708) * @author Влад Смородa
709) */
710) public class AlgorithmService {
711) private static final double ALPHA = 1;//жадность, где 0 - выбор где поближе
712) private static final double BETA = 1;//если = 0, то игнорируем расстояние
713) private static final double P = 0.2;//коэфициент испарения феромонов
714) private static final double Q = 300;//коэфициент для нахождения феромона
нового
715) private static final int TIME = 200;//количество запусков алгоритма
716) private static int CITIES;
717) private int[][] roadLength;
718) private double[][] roadPheromone;
719) private double[][] probabilities;
720) private double[] sumProbability;
721) private ArrayList<Integer> travel;
722) private int lengthOfWay;
723) private static ArrayList<Integer> listTravel;
724) boolean[] visitedCities;
725) public AlgorithmService(int cities) {
726) CITIES = cities;
727) sumProbability = new double[CITIES];
728) visitedCities = new boolean[CITIES];
729) probabilities = new double[CITIES][CITIES];
730) }
731) /**
732) * Находит кратчайший путь
733) *
734) * @param input введенная матрица
735) * @param size размерность матрицы
736) * @return кратчайший путь
```

```
737) */
738) static StringBuilder getShortestWay(int[][] input, int size) {
739) AlgorithmService algo;
740) StringBuilder resultTravel;
741) StringBuilder minWay = new StringBuilder();
742) int minWayLength = Integer.MAX_VALUE;
743) for (int i = 0; i < 10; i++) {
744) algo = new AlgorithmService(size);
745) algo.inputMatrix(input, size);
746) for (int j = 0; j < TIME; j++) {
747) algo.runAlgorithm(0);
748) }
749) resultTravel = new StringBuilder();
750) resultTravel.append("Длина пути: ").append(algo.lengthOfWay).append(" |
Путь:");
751) for (Integer city : algo.travel) {
752) resultTravel.append(" -> ").append(city + 1);
753) }
754) resultTravel.append(" -> ").append(algo.travel.get(0) + 1);
755) if (algo.lengthOfWay < minWayLength) {
756) minWayLength = algo.lengthOfWay;
757) listTravel = algo.travel;
758) minWay = resultTravel;
759) }
760) }
761) return minWay;
762) }
763) /**
764) * Запускает полный цикл алгоритма по поиску города и обновлению феромонов
765) *
766) * @param startingCity начальный город
767) */
768) private void runAlgorithm(int startingCity) {
769) initializeVisitedCities();
770) int selectedCity = startingCity;
771) travel = new ArrayList<>();
772) travel.add(startingCity);
773) while (hasMoreCitiesToGo()) {
774) updateSumProbability(selectedCity);
775) getProbabilityForCity(selectedCity);
776) selectedCity = chooseCity(selectedCity);
777) if (selectedCity > 2000) {
778) break;
779) }
780) travel.add(selectedCity);
781) }
782) setCompletedLength(travel.get(0), travel.get(travel.size() - 1));
783) updatePheromones();
784) }
785) /**
786) * Инициализирует массив посещенных городов 'false' значениями, а также длину
787) * пути Initializing array of visited cities with false value, length of
788) * way.
789) */
790) public void initializeVisitedCities() {
791) for (int i = 0; i < visitedCities.length; i++) {
792) visitedCities[i] = false;
793) }
794) lengthOfWay = 0;
795) }
796) /**
797) * Проверяем, остались ли еще непосещенные города
798) *
```

```
799) * @return true или false
800) */
801) public boolean hasMoreCitiesToGo() {
802) for (int i = 0; i < visitedCities.length; i++) {
803) if (visitedCities[i] == false) {
804) return true;
805) }
806) }
807) return false;
808) }
809) /**
810) * Обновляет массив с сумарными вероятностями пойти в другой непосещенный
811) * город
812) *
813) * @param currentCity текущий город
814) */
815) private void updateSumProbability(int currentCity) {
816) visitedCities[currentCity] = true;
817) for (int k = 0; k < CITIES; k++) {
818) sumProbability[k] = 0;
819) }
820) int i = currentCity;
821) for (int j = 0; j < CITIES; j++) {
822) if (visitedCities[j] != true) {
823) sumProbability[i] = sumProbability[i] + (Math.pow(roadPheromone[i][j],
ALPHA) / Math.pow(roadLength[i][j], BETA));
824) }
825) }
826) }
827) /**
828) * Метод расчитывает вероятность попасть в каждый из городов
829) *
830) * @param currentCity текущий город
831) */
832) private void getProbabilityForCity(int currentCity) {
833) for (int j = 0; j < CITIES; j++) {
834) if (visitedCities[j] == false) {
835) probabilities[currentCity][j] = 100 *
((Math.pow(roadPheromone[currentCity][j], ALPHA) /
(Math.pow(roadLength[currentCity][j], BETA)))
836) / (sumProbability[currentCity]));
837) } else {
838) probabilities[currentCity][j] = 0;
839) }
840) }
841) }
842) /**
843) * Выбирается следующий город для посещения
844) *
845) * @param currentCity текущий город, отсчет с нуля
846) * @return возвращает номер города, отсчет с нуля
847) */
848) private int chooseCity(int currentCity) {
849) ArrayList<Double> probabsForChoose = new ArrayList<>();
850) for (int i = 0; i < CITIES; i++) {
851) probabsForChoose.add(probabilities[currentCity][i]);
852) }
853) Collections.sort(probabsForChoose);
854) Random random = new Random();
855) int randomValue = random.nextInt(100);
856) double city = Collections.max(probabsForChoose);
857) double ender = probabsForChoose.get(1);
858) for (int i = 0; i < CITIES; i++) {
```

```
859) //Если i = последнему, выбираем последний город
860) if (i == CITIES - 1) {
861) city = probabsForChoose.get(probabsForChoose.size() - 1);
862) break;
863) //Если случайное число попадает в отрезок
864) } else if (randomValue > probabsForChoose.get(i) && randomValue < ender) {
865) city = probabsForChoose.get(i + 1);
866) break;
867) }
868) //Правый край отрезка для сравнения
869) if (i == CITIES - 2) {
870) ender = ender + probabsForChoose.get(probabsForChoose.size() - 1);
871) } else {
872) ender = ender + probabsForChoose.get(i + 2);
873) }
874) }//end for
875) for (int i = 0; i < CITIES; i++) {
876) //Ищем город по вероятности
877) //Если вероятность совпала, выбираем этот город
878) if (probabilities[currentCity][i] == city) {
879) visitedCities[i] = true;
880) lengthOfWay += roadLength[currentCity][i];
881) return i;
882) }
883) }//end for
884) return 2200;
885) }
886) /**
887) * Назначаем длинну полного пути муравья.
888) *
889) * @param firstCity первый город
890) * @param lastCity последний город
891) */
892) private void setCompletedLength(int firstCity, int lastCity) {
893) lengthOfWay += roadLength[lastCity][firstCity];
894) }
895) /**
896) * Обновляет значения феромонов
897) */
898) private void updatePheromones() {
899) double pieceOfPheromone = Q / lengthOfWay;
900) for (int i = 0; i < CITIES; i++) {
901) for (int j = 0; j < CITIES; j++) {
902) roadPheromone[i][j] = roadPheromone[i][j] * (1 - P);
903) }
904) }
905) int first = 0;
906) int last = 1;
907) for (int i = 0; i < CITIES; i++) {
908) if (i == CITIES - 1) {
909) roadPheromone[travel.get(travel.size() - 1)][travel.get(0)] +=
pieceOfPheromone;
910) } else {
911) roadPheromone[travel.get(first)][travel.get(last)] += pieceOfPheromone;
912) first++;
913) last++;
914) }
915) }
916) }
917) /**
918) * Создаются массивы длин и феромонов на основании введенной матрицы Массив
919) * феромонов заполняется случайными значениями от 1 до 3
920) *
```

```
921) * @param matrix заданная матрица
922) * @param size размерность матрицы
923) */
924) private void inputMatrix(int[][] matrix, int size) {
925) CITIES = size;
926) Random rnd = new Random();
927) roadLength = new int[size][size];
928) roadPheromone = new double[size][size];
929) for (int r = 0; r < size; r++) {
930) for (int c = 0; c < size; c++) {
931) roadLength[r][c] = matrix[r][c];
932) }
933) }
934) for (int i = 0; i < size; i++) {
935) for (int j = 0; j < size; j++) {
936) roadPheromone[i][j] = rnd.nextInt(2) + 1;
937) }
938) }
939) }
940) static ArrayList getListTravel() {
941) return listTravel;
942) }
943) }
944) public class FetchGeocodingConfig {
945) final String query;
946) final String locale;
947) final int limit;
948) final boolean reverse;
949) final String point;
950) final String provider;
951) FetchGeocodingConfig(String query, String locale, int limit, boolean
reverse, String point, String provider) {
952) this.query = query;
953) this.locale = locale;
954) this.limit = limit;
955) this.reverse = reverse;
956) this.point = point;
957) this.provider = provider;
958) }
959) }
960) import android.os.AsyncTask;
961) import com.graphhopper.directions.api.client.ApiException;
962) import com.graphhopper.directions.api.client.api.GeocodingApi;
963) import com.graphhopper.directions.api.client.model.GeocodingLocation;
964) import com.graphhopper.directions.api.client.model.GeocodingResponse;
965) import java.util.ArrayList;
966) import java.util.List;
967) import timber.log.Timber;
968) public class FetchGeocodingTask extends AsyncTask<FetchGeocodingConfig,
Void, List<GeocodingLocation>> {
969) private final String ghKey;
970) private final FetchGeocodingTaskCallbackInterface callbackInterface;
971) FetchGeocodingTask(FetchGeocodingTaskCallbackInterface callbackInterface,
String ghKey) {
972) this.callbackInterface = callbackInterface;
973) this.ghKey = ghKey;
974) }
975) @Override
976) protected List<GeocodingLocation> doInBackground(FetchGeocodingConfig...
geocodingConfigs) {
977) if (geocodingConfigs.length != 1)
978) throw new IllegalArgumentException("It's only possible to fetch one
geocoding at a time");
```

```
979) List<GeocodingLocation> locations = new ArrayList<>();
980) GeocodingApi api = new GeocodingApi();
981) try {
982) FetchGeocodingConfig geocodingConfig = geocodingConfigs[0];
983) GeocodingResponse res = api.geocodeGet(ghKey, geocodingConfig.query,
geocodingConfig.locale, geocodingConfig.limit, geocodingConfig.reverse,
geocodingConfig.point, geocodingConfig.provider);
984) locations = res.getHits();
985) if (locations.isEmpty())
986) callbackInterface.onError(R.string.error_location_not_found);
987) } catch (ApiException e) {
988) callbackInterface.onError(R.string.error_fetching_geocoding);
989) Timber.e(e, "An exception occured when fetching geocoding results for %s",
geocodingConfigs[0].query);
990) }
991) return locations;
992) }
993) @Override
994) protected void onPostExecute(List<GeocodingLocation> locations) {
995) callbackInterface.onPostExecuteGeocodingSearch(locations);
996) }
997) }
998) import com.graphhopper.directions.api.client.model.GeocodingLocation;
999) import java.util.List;
1000) public interface FetchGeocodingTaskCallbackInterface {
1001) void onError(int message);
1002) void onPostExecuteGeocodingSearch(List<GeocodingLocation> points);
1003) }
1004) public class FetchSolutionConfig {
1005) final String jobId;
1006) final String vehicleId;
1007) FetchSolutionConfig(String jobId, String vehicleId) {
1008) this.jobId = jobId;
1009) this.vehicleId = vehicleId;
1010) }
1011) }
1012) public class FetchSolutionConfig {
1013) final String jobId;
1014) final String vehicleId;
1015) FetchSolutionConfig(String jobId, String vehicleId) {
1016) this.jobId = jobId;
1017) this.vehicleId = vehicleId;
1018) }
1019) }
1020) import android.os.AsyncTask;
1021) import com.graphhopper.directions.api.client.ApiException;
1022) import com.graphhopper.directions.api.client.api.SolutionApi;
1023) import com.graphhopper.directions.api.client.model.Activity;
1024) import com.graphhopper.directions.api.client.model.Address;
1025) import com.graphhopper.directions.api.client.model.Route;
1026) import com.mapbox.geojson.Point;
1027) import java.util.ArrayList;
1028) import java.util.List;
1029) import timber.log.Timber;
1030) public class FetchSolutionTask extends AsyncTask<FetchSolutionConfig, Void,
List<Point>> {
1031) private final String ghKey;
1032) private final FetchSolutionTaskCallbackInterface callbackInterface;
1033) FetchSolutionTask(FetchSolutionTaskCallbackInterface callbackInterface,
String ghKey) {
1034) this.callbackInterface = callbackInterface;
1035) this.ghKey = ghKey;
1036) }
```

```
1037) @Override
1038) protected List<Point> doInBackground(FetchSolutionConfig... solutions) {
1039) if (solutions.length != 1)
1040) throw new IllegalArgumentException("It's only possible to fetch one
solution at a time");
1041) List<Point> points = new ArrayList<>();
1042) SolutionApi api = new SolutionApi();
1043) try {
1044) com.graphhopper.directions.api.client.model.Response res =
api.getSolution(ghKey, solutions[0].jobId);
1045) List<Route> routes = res.getSolution().getRoutes();
1046) for (Route route : routes) {
1047) if (route.getVehicleId().equals(solutions[0].vehicleId) ||
solutions[0].vehicleId == null) {
1048) // Found the right vehicle
1049) List<Activity> activities = route.getActivities();
1050) for (int i = 0; i < activities.size(); i++) {
1051) Activity activity = activities.get(i);
1052) Address address = activity.getAddress();
1053) points.add(Point.fromLngLat(address.getLon(), address.getLat()));
1054) }
1055) break;
1056) }
1057) }
1058) if (points.isEmpty())
1059) callbackInterface.onError(R.string.error_vehicle_not_found);
1060) } catch (ApiException e) {
1061) callbackInterface.onError(R.string.error_fetching_solution);
1062) Timber.e(e, "An exception occured when fetching a solution with jobId %s",
solutions[0].jobId);
1063) }
1064) return points;
1065) }
1066) @Override
1067) protected void onPostExecute(List<Point> points) {
1068) callbackInterface.onPostExecute(points);
1069) }
1070) }
```

ДОДАТОК Б

Комп’ютерна презентація:

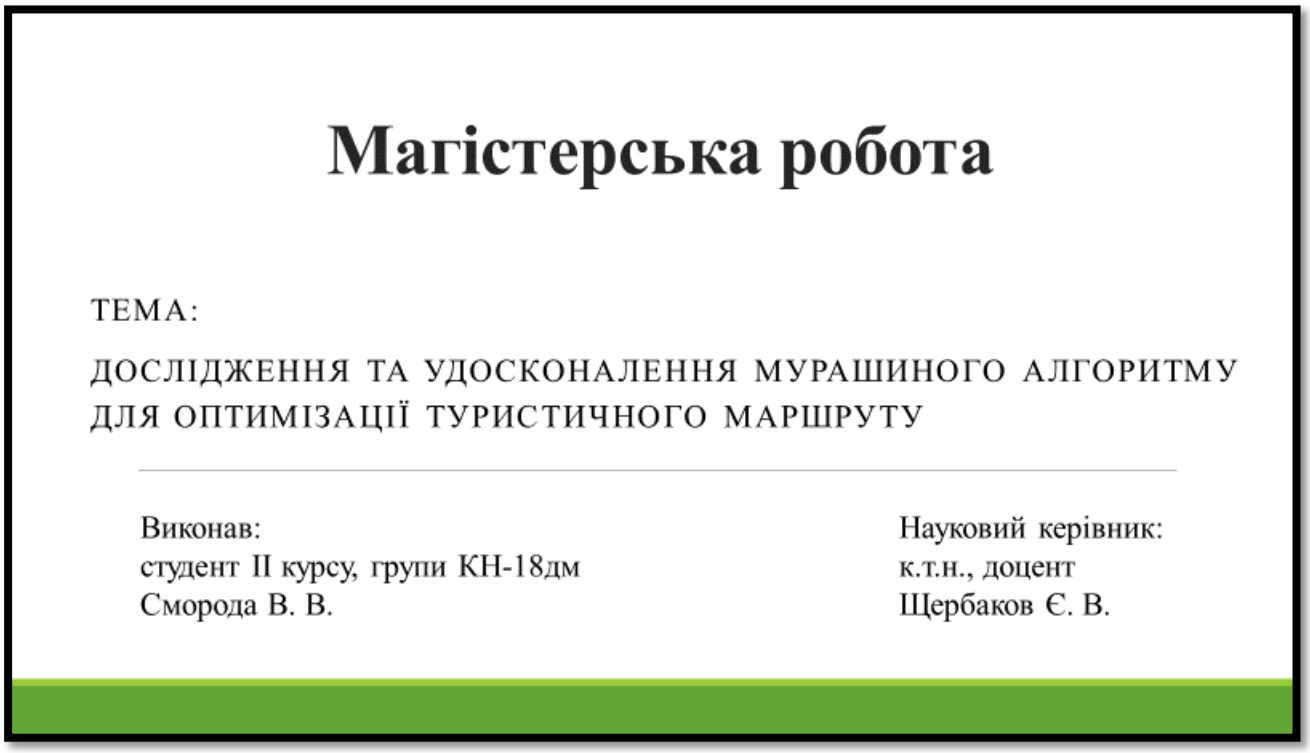


Рисунок Б.1 – Титульний лист

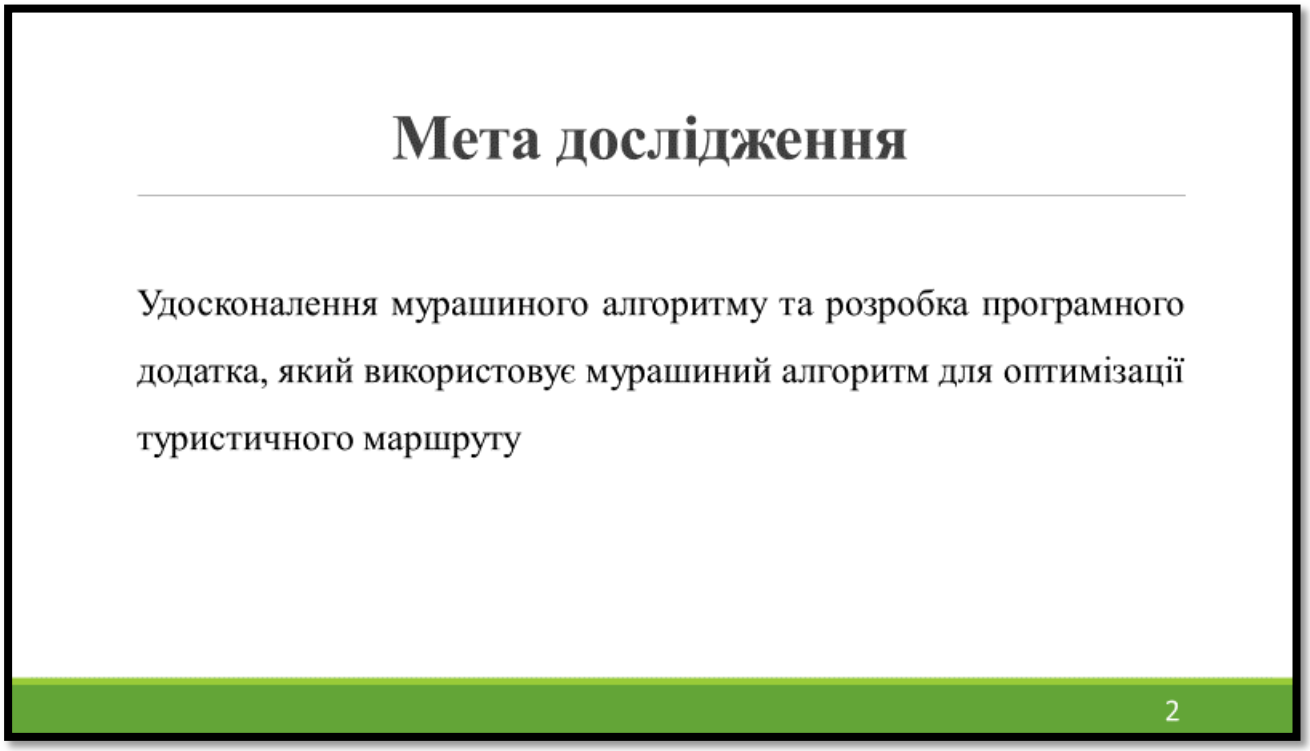


Рисунок Б.2 – Мета дослідження

Рисунок Б.3 – Розвиток туристичних маршрутів

Рисунок Б.4 – Задача комівояжера

Задача комівояжера

Групи методів для розв'язання задачі комівояжера:

а) точні, що містять повний перебір й метод гілок і меж;

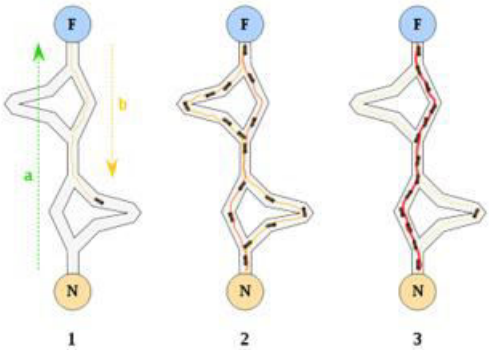
б) наближені методи: методи випадкового перебору, жадібний і дерев'яний алгоритми; метод імітації відпалу;

в) «біологічні методи»: генетичні алгоритми, а також алгоритми мурашиних колоній.

5

Рисунок Б.5 – Групи методів

Мурашині алгоритми



Перша версія алгоритму, запропонована італійським вченим Марко Доріго в 1992 році, була спрямована на пошук оптимального шляху в графі.

6

Рисунок Б.6 – Мурашині алгоритми

Рисунок Б.7 – Ідея мурашиного алгоритму

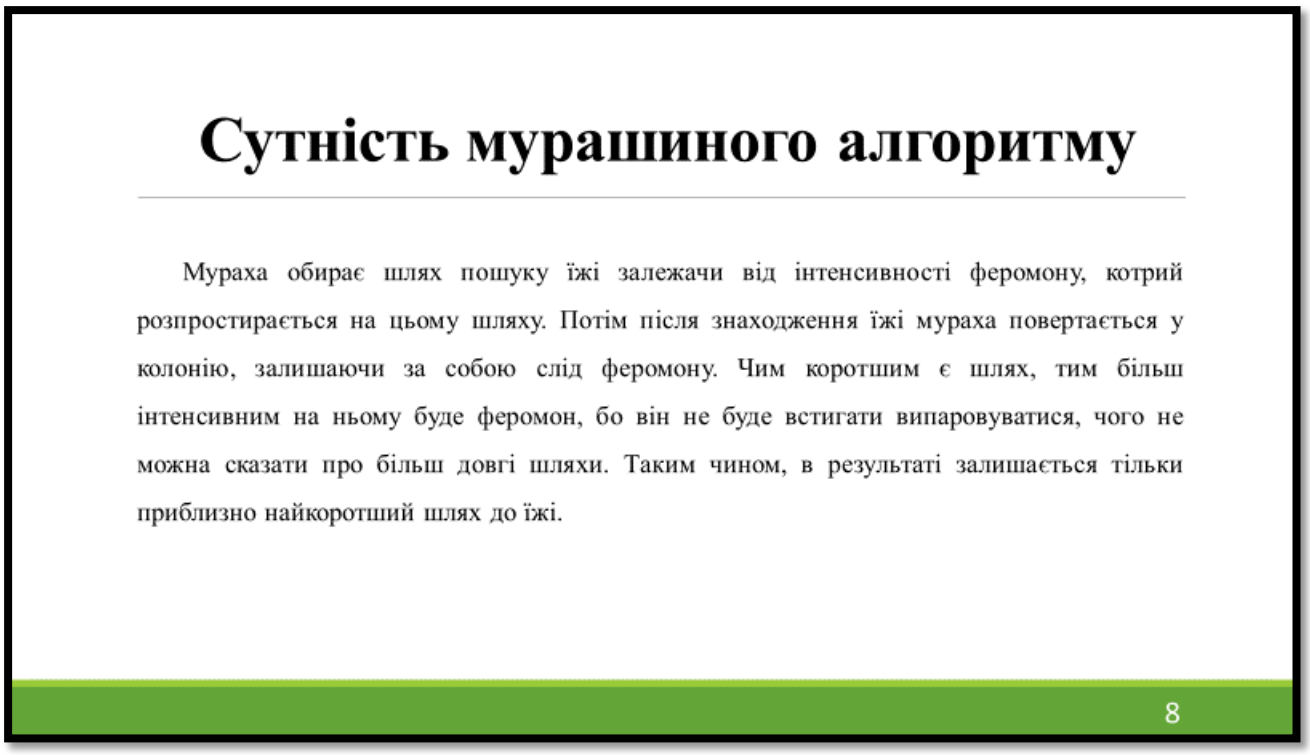


Рисунок Б.8 – Сутність мурашиного алгоритму

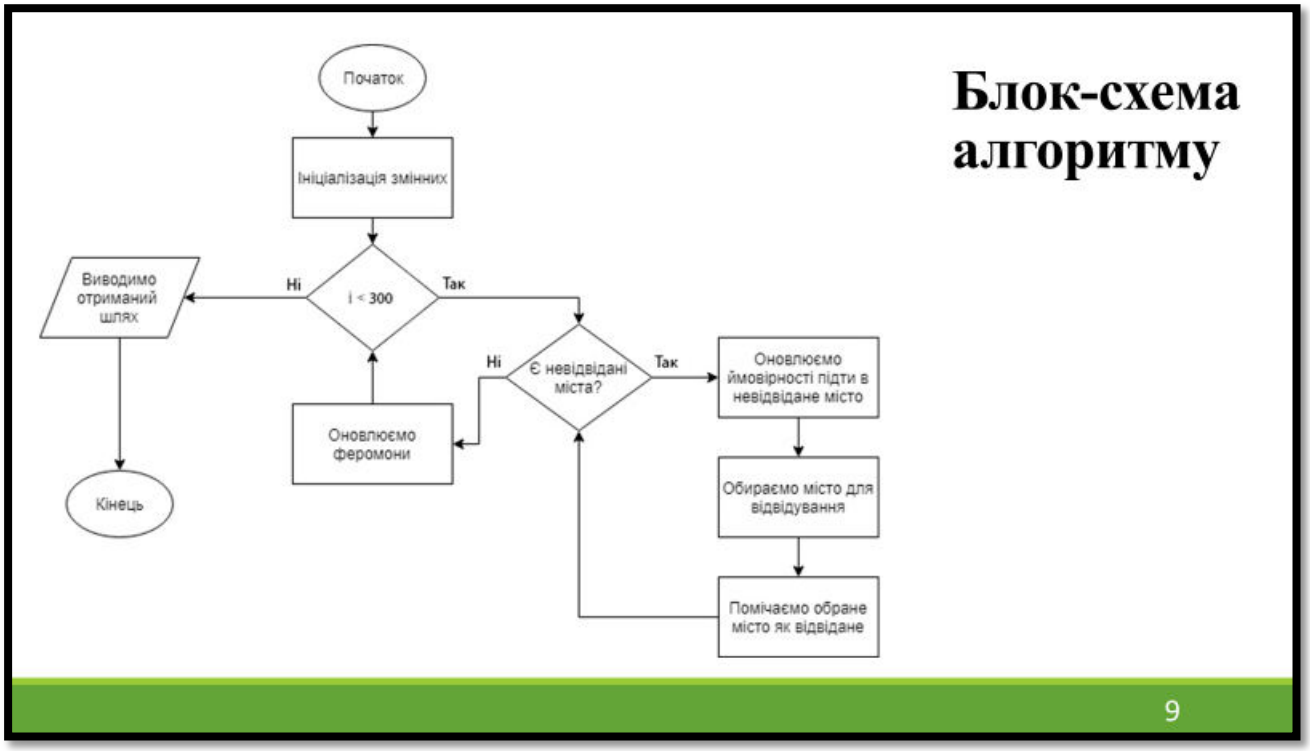


Рисунок Б.9 – Блок-схема алгоритму

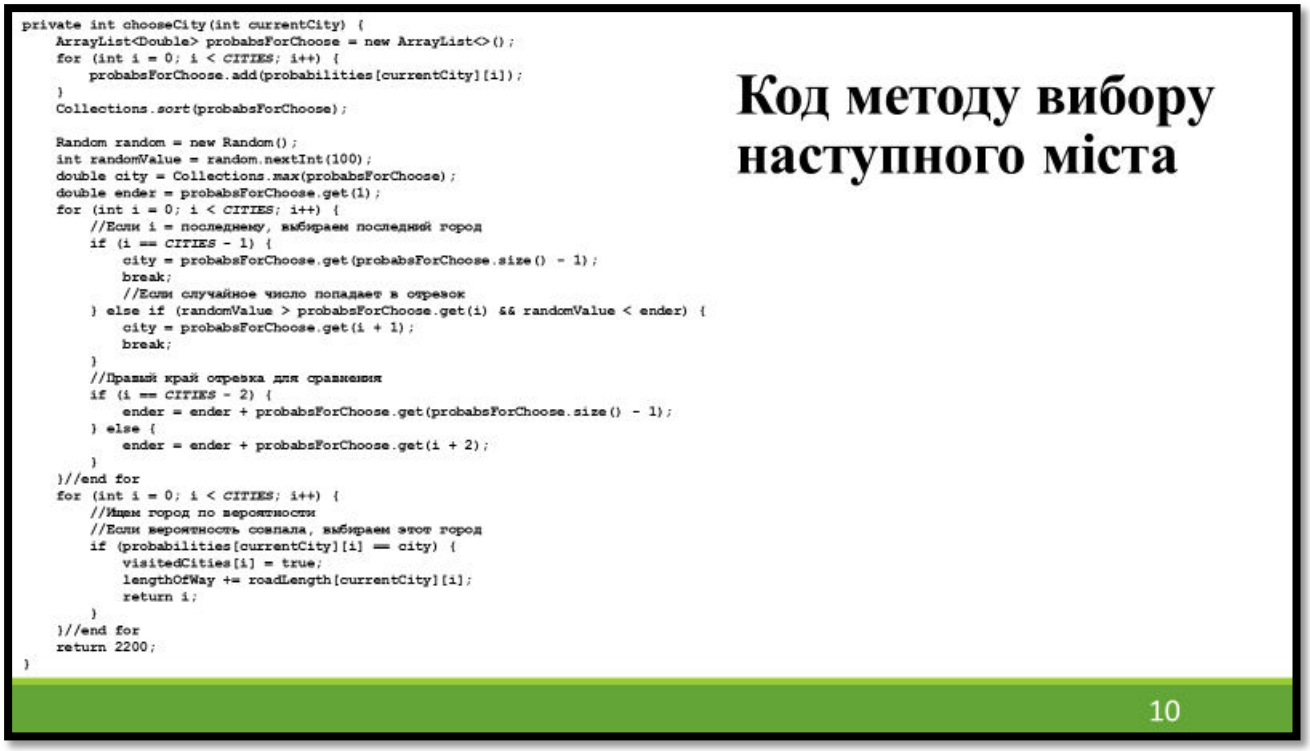


Рисунок Б.10 – Код

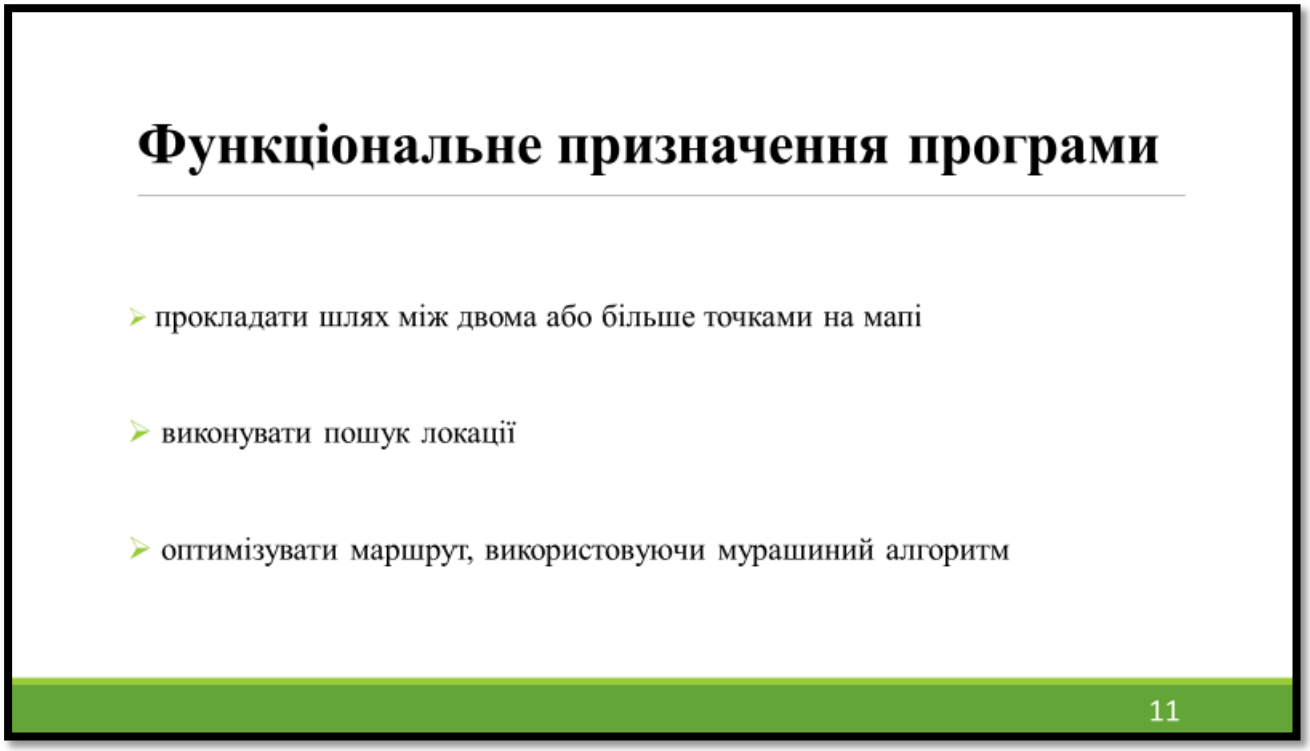


Рисунок Б.11 – Функціональне призначення програми

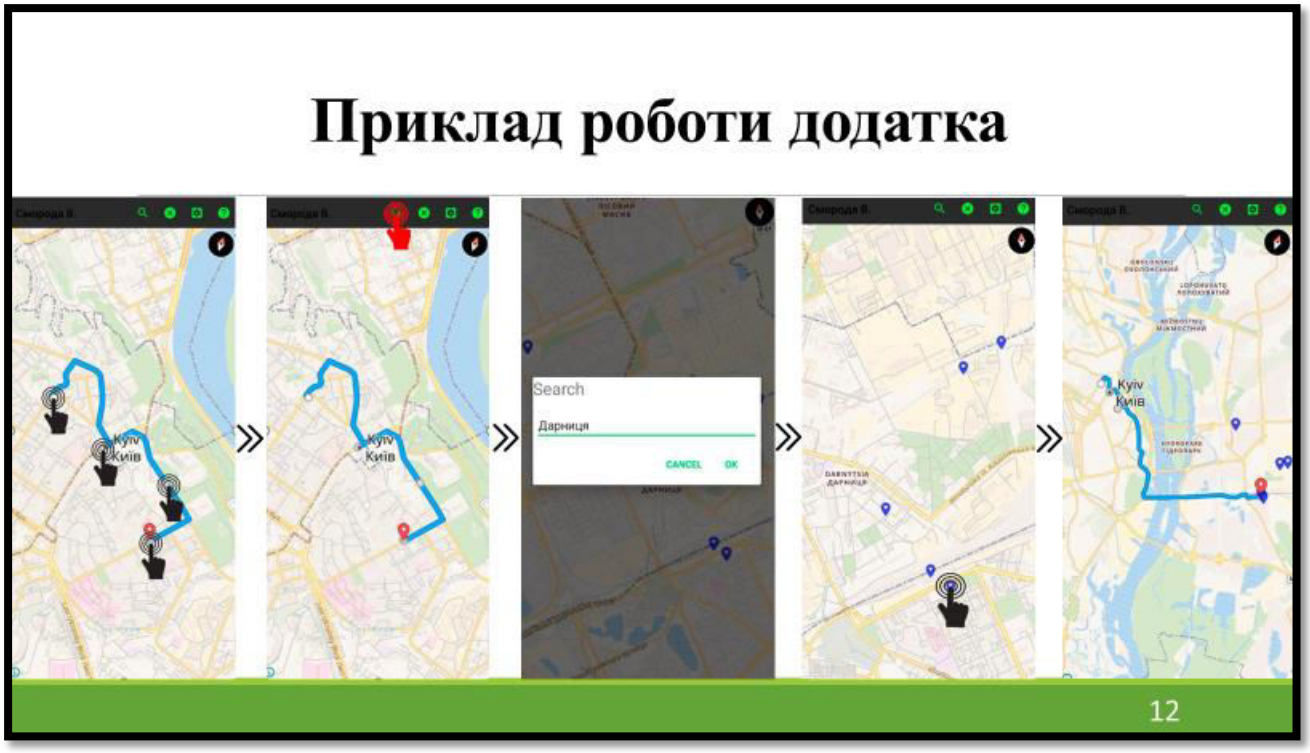


Рисунок Б.12 – Приклад роботи додатка

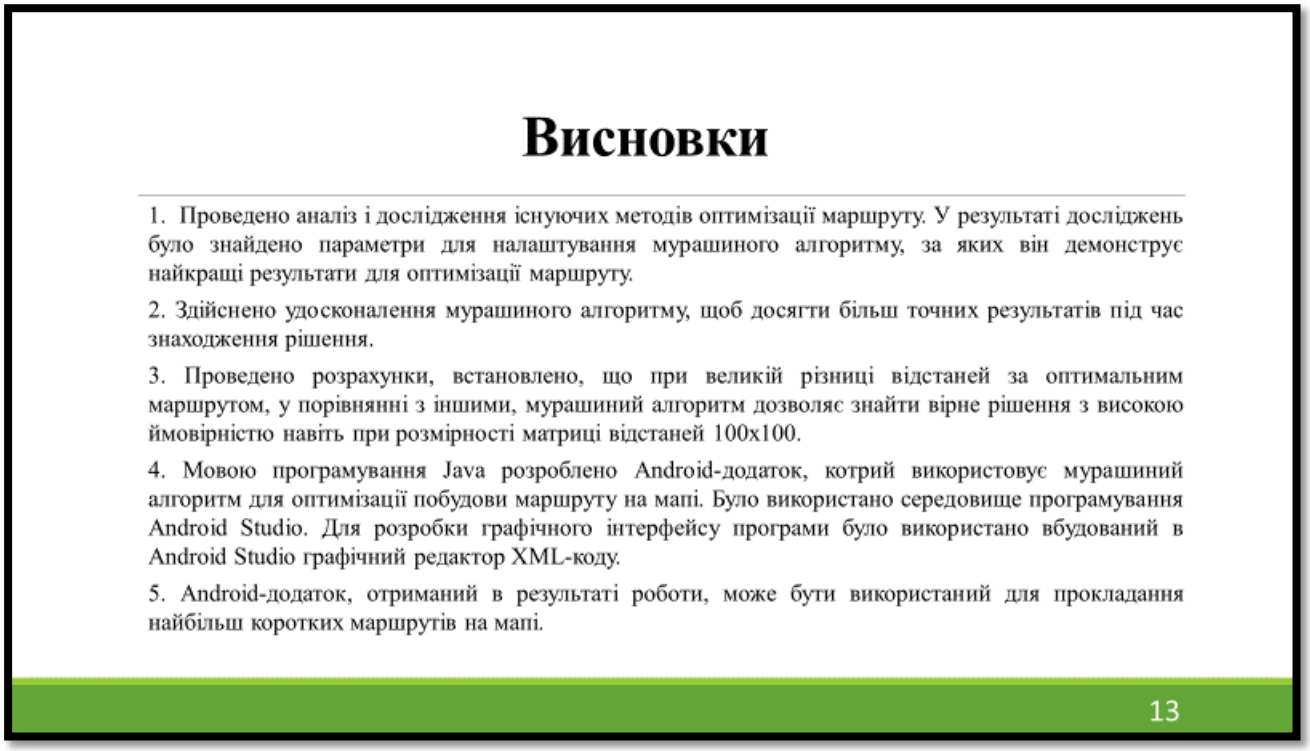


Рисунок Б.13 – Висновки