КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра теоретичної кібернетики

Кваліфікаційна робота
на здобуття ступеня бакалавра
за спеціальністю 122 Комп'ютерні науки
на тему:

Пошук циклічних маршрутів на картах з туристичними даними

Виконала студентка 4-го курсу
Назаренко Поліна Василівна _____
(підпис)

Науковий керівник:
доцент, кандидат фіз.-мат. наук
Ставровський Андрій Борисович _____
(підпис)

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

Студент _____
(підпис)

Робота заслухана на засіданні кафедри теоретичної кібернетики та рекомендована до захисту в ЕК,
протокол № ……
від …….2021р.
Завідувач кафедри
Крак Ю.В.

Київ – 2021

РЕФЕРАТ

Структура й обсяг дипломної роботи:

Кваліфікаційна робота на здобуття ступеня бакалавра, 40 сторінок, 27 малюнків, 2 таблиці, 24 джерел.

Ключові слова: Maps APIs, TSP, geolocation, Python, Leaflet, graph algorythms, JavaScript, Flask, OpenStreetMap.

Актуальність теми полягає в аналізі алгоритмів для знаходження циклічних маршрутів, і побудові додатку за допомогою сучасних бібліотек і фреймворків, таким чином отримавши кращу продуктивність, і більш зручний користувацький інтерфейс з розширеним функціоналом.

Мета дослідження є побудова додатка який знаходить класифіковані точки у певній дальності від користувача, та будує зручний циклічний маршрут не вимагаючи ручного введення кожної точки.

Для досягнення мети були поставлені такі завдання як дослідження існуючих картографічних сервісів, огляд варіантів розв'язань задачі побудови оптимального маршруту, обґрунтування NP-складності.

Зміст

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

TSP (Travelling salesman problem) - завдання побудови оптимального маршруту

GPS (Global Positioning System) - система глобального позиціонування

HTML (HyperText Markup Language) - мова розмітки гіпертекстових документів

CSS (Cascading Style Sheets) - каскадні таблиці стилів

API (Application Programming Interface) - інтерфейс програмування додатків

SQL (Structured query language) - мова структурованих запитів

JSON (JavaScript Object Notation) - об'єктній запис JavaScript

NP (Nondeterministic Polynomial time) - Недетермінований поліноміальний час

ВСТУП

Зараз здебільшого здається дивною можливість заблукати в місті, навіть в незнайомому, адже зараз майже у кожного є смартфон і доступ до інтернету. Якщо щось трапляється з смартфоном, наприклад розрядився, або закінчився мобільний інтернет, в прешу чергу ми поспішаємо відновити стан при якому у нас є смартфон і зв’язок. В такій ситуації ми дуже можливо пригадаємо всі кафе з написом вай-фай, всі магазини техніки, банкомати, чи сервіс центри повз які ми проходили у цьому районі. Навіть побудуємо маршрут через усі найближчі точки що нам підходять.

Але ніхто не гарантує що це буде саме та частина міста яку ви хоч трохи знаєте, може навіть інша країна. Можливо ситуація не така екстремальна, щоб мозок перебрав усі найдрібніші шматочки інформації про навколишне середовище. Мозок не здається надійним гідом по місцях де ти ніколи раніше не буваав, а от карта яка базується на постійно оновлюваних даних, які завантажують та підтримують в актуальному стані люди які знайомі з певною місцевістю вже довгий час викликає довіру, і дозволяє ознайомитись і орієнтуватись без особливих проблем.

Під час пандемії досліджувати нові країни стало значно важче, але навіть без польотів і туризму завжди можна знайти багато цікавого просто поряд з домом. Або знайти оптимальний маршрут відвідування усіх навколишніх кав’ярень, або пробігтись усіма книжковими магазинами, так щоб було більше часу на вибір книги, а не на біг між книгарнями. Тож і коли ми майже нікуди не подорожуємо, ми все одно не позбавляємось повсякденних задач прокладання маршрутів.

Задача пошуку маршрутів зараз дуже актуальна через появу величезної кількості даних у вільному доступі, яких навіть так постійно замало, і вони постійно змінюються, адже реальний світ не залишається статичним. Заклади відкриваються і закриваються швидше ніж їх додають на карти. Будівлі

перебудовують, щось зносять, щось будують. У нас стає більше можливостей для розрахунку, і так само більше варіантів їх застсування.

Хоч сама задача в математичному і навіть програмному сенсі вже досить давно відома, постійна зміна умов і потреб, а також можливостей техніки змушує продовжувати шукати нові рішення, або оптимізувати і адаптувати ті що вже існують.

Скоріш за все, колись задача пошуку шляху перестане існувати в явному вигляді, можливо нам усім вживлять чіп який буде забезпечувати нас достовірною картою з GPS варто лише подумати, і для ресурсів людського мозку складність обчислення не грає такої ролі як для обмежених можливостей комп’ютерів. Можливо у нас будуть телепорти, і більше не буде потреби шукати найкоротший шлях, бо він завжди нульовий.

Робота ставить на меті побудову додатка який знаходить класифіковані точки у певній дальності від користувача, та будує зручний циклічний маршрут не вимагаючи ручного введення кожної точки. Додаток може виступати як допоміжний інструмент для туристів, що не знають міста, до якого приїхали, та для вирішення побутових задач пошуку.

Функціонал програми буде полягає в побудові оптимального маршруту за заданими даними, виведення його на інтерактивній карті, виведення інформації щодо пересування по маршруту, візуалізація будь–якого елемента маршруту.

РОЗДІЛ 1. ЗАДАЧА ПРО ЦИКЛІЧНИЙ МАРШРУТ ТА СУЧАСНІ КАРТОГРАФІЧНІ ЗАСОБИ

1.1 Типова задача

Завдання побудови оптимального маршруту – класична задача комбінаторики. Щоб наочніше показати її важливість, нижче наведена коротка історія вирішення поставленого завдання.

У 1857 Гамільтон створив гру Ікосіан, в правилах якої сказано, що учасники повинні з'єднати 20 точок додекаедру так, щоб кожна точка використовувалася не більше одного разу, також кінцева точка шляху повинна збігатися з вихідною. Ця гра лягла в основу появи гамільтонова графа.[2]

Задача пошуку оптимально маршруту була вперше розглянута з математичної точки зору в 1930-их роках математиком і економістом Карлом Менджером у Відні. У 1940-их статистики Мехаланобіс, Джессен, Гош і Маркс намагалися знайти застосування цій задачі в сільськогосподарському секторі, що призвело до її популяризації – починаючи з середини 1950х років методи розв'язання задачі стали публікуватися в наукових журналах.

Задача побудови циклічного маршруту має просте формулювання, проте не має ефективного способу розв'язання [3]. У 1972 Р. Карп довів, що задача Гамільтонового циклу належить до класу NP-повних задач (Nondeterministic Polynomial time), звідки випливає, що задача комівояжера (TSP) є NP-важкою [4]. Цей результат дав пояснення обчислювальної складності знаходження оптимальних маршрутів.

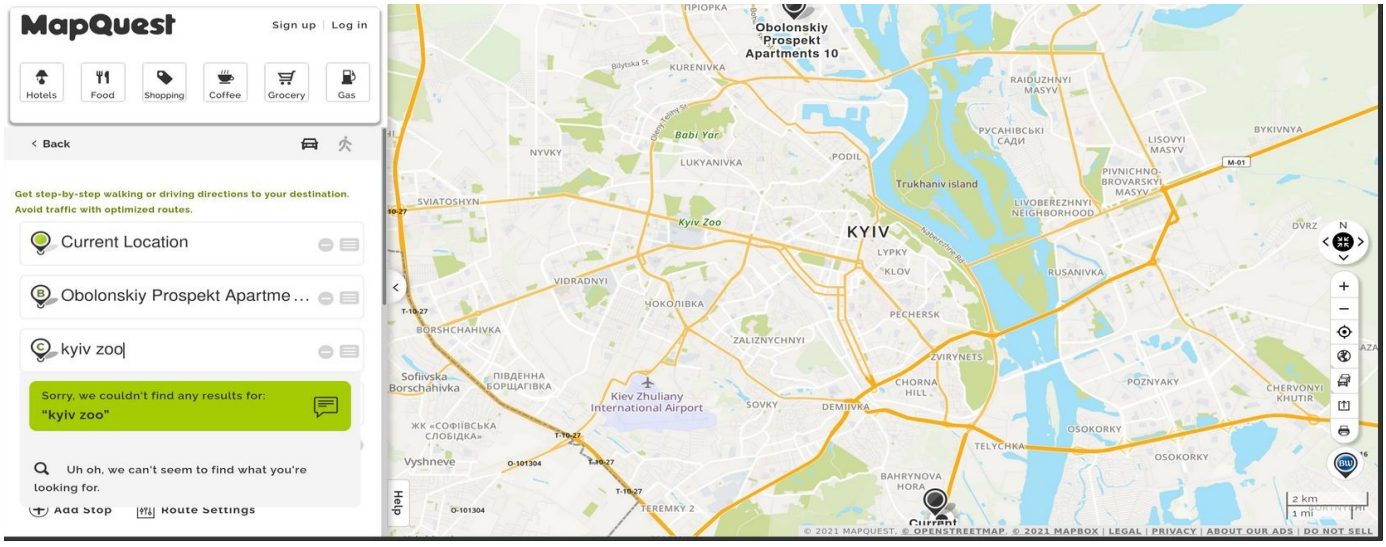
Проте TSP є дуже важливою практичною задачею, тому для її ефективного розв'язання було винайдено різноманітні методи, і дослідження продовжуються.

1.2 Аналіз існуючих сервісів

Популярні картографічні сервіси як Google Maps, не зважаючи на весь їх сучасний функціонал, не пропонують користувачам можливість пошуку оптимального шляху. При введенні деякої кількості координат сервіс вибудовує маршрут в тому порядку, в якому дані були введені. Користувач може обирати засоби пересування (на машині, пішки, велосипедом, на маршрутному транспорті), але всі ці зміни впливають виключно на варіанти побудови маршруту між точками які він задає. Послідовність точок у маршруті змінюється лише власноруч користувачем, що не є зручним способом для побудови довгого маршрута з великою кількістю зупинок, особливо на новій місцевості.

Аналізуючи різні картографічні сервіси можна помітити, що дуже мало з них мають реалізований багатозупиночний маршрут. Також ті карти в яких ця функція присутня, нажаль, не дуже зручні у використанні.

1.2.1 MapQuest



мал.1

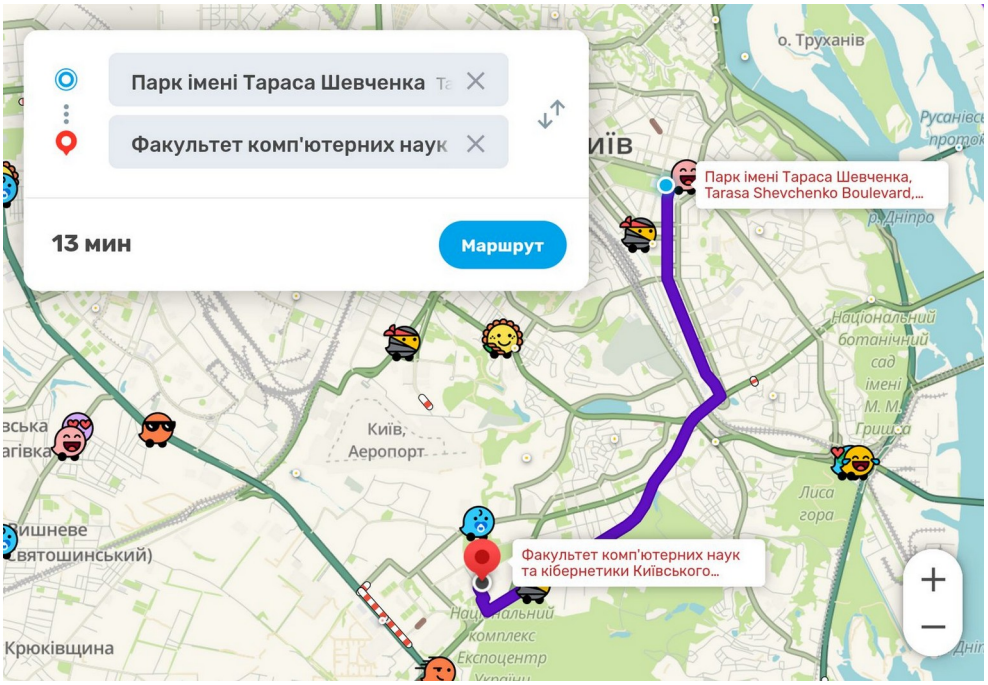
Наприклад MapQuest[5]. Має функцію побудови по декільком точкам, та навіть перемикач яким можна включити сортування маршрута для зручності. Але

додавати точки до маршруту дуже не зручно, принаймні в Україні, бо підказки при наборі адресси чи назви здебільшого з США, і воно не знаходить за назвою закладу, або не повною адресою (повною саме в тому форматі в якому записано в карті, або координатами). Також там наявна реклама, що займає забагато уваги, особливо відео яке вискакує на панелі інструментів.

На малюнку 1 видно, що сервіс не знаходить точку за назвою яка зазначена на карті. Тобто не зважаючи на наявність додаткового і корисного функціоналу, ця карта не є зручною ані для спеціальних потреб, ані для звичайного користувача.

1.2.2 Waze

Якщо казати про знаходження найшвидшого маршруту до точки в реальному часі, то waze[6](мал.2) є одним з найзручніших варіантів. Простий і зрозумілий у використанні, та дуже потужний навігатор, нажаль максимально ефективним є лише для автомобіля. Також можливості побудови маршрута обмежені лише однією зупинкою на поїздку, а інформація про будинки і будь які споруди поза дорогою мінімізована наскільки це можливо.

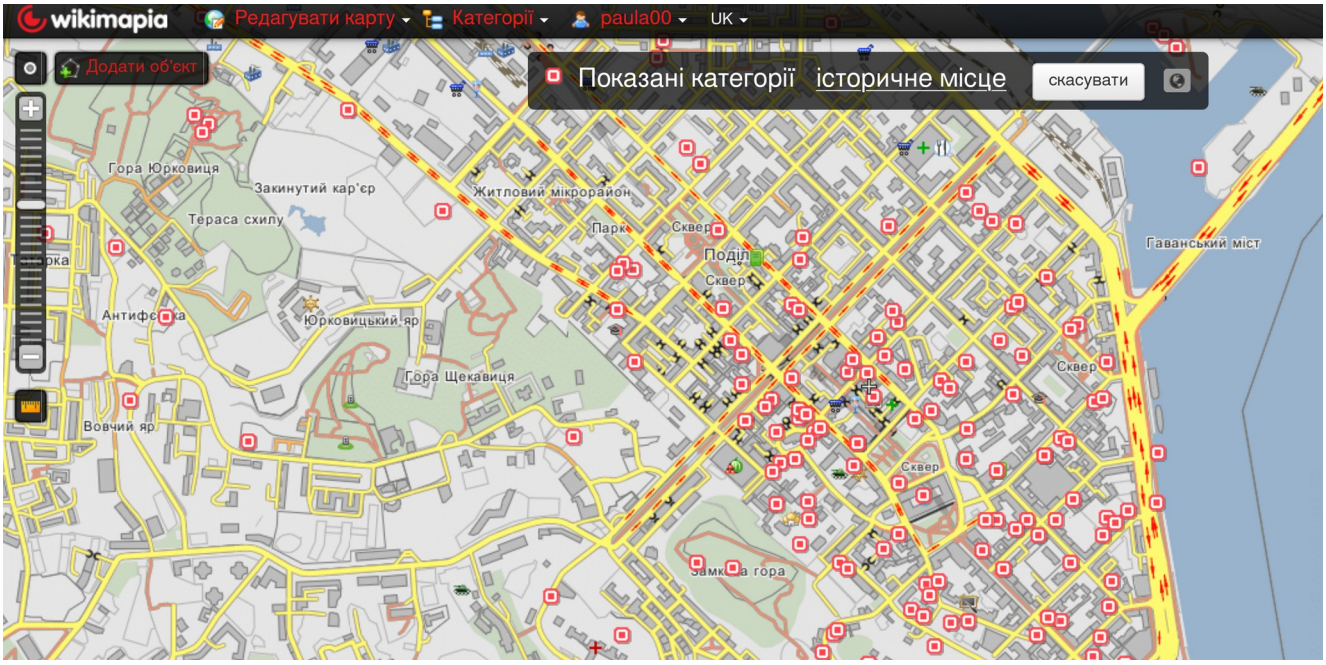


мал.2

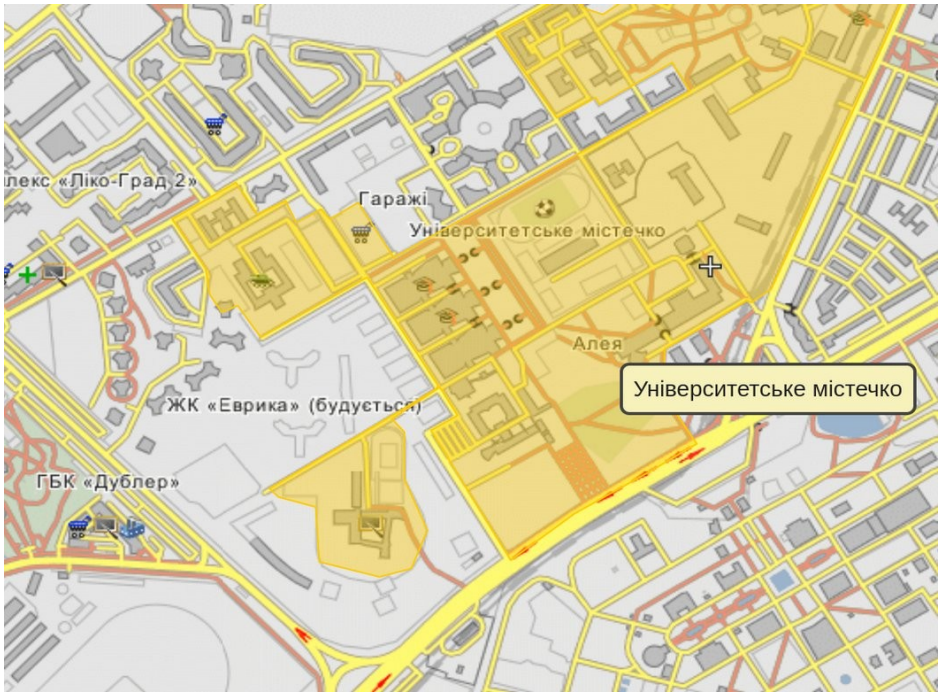
Окрім якості навігаторської частини розробники вклали багато в інтерфейс, і дали користувачам змогу бачити на карті один одного. Це не було необхідним для додатку такого формату, але точно виділило його серед інших.

1.2.3 Wikimapia

Багато сервісів не мають побудови маршрутів, але вони також мають цікаві можливості. Наприклад Wikimapia[7], показує кожну будівлю, дорогу, площу, район чи область, і до кожного об’єкта показує коротку інформацію: до яких категорій відноситься, довжина і широта, посилання на офіційні сторінки, список організацій що знаходяться в будівлі, тощо.



Мал.3 – відмічені всі об’єкти з категорії



Мал.4

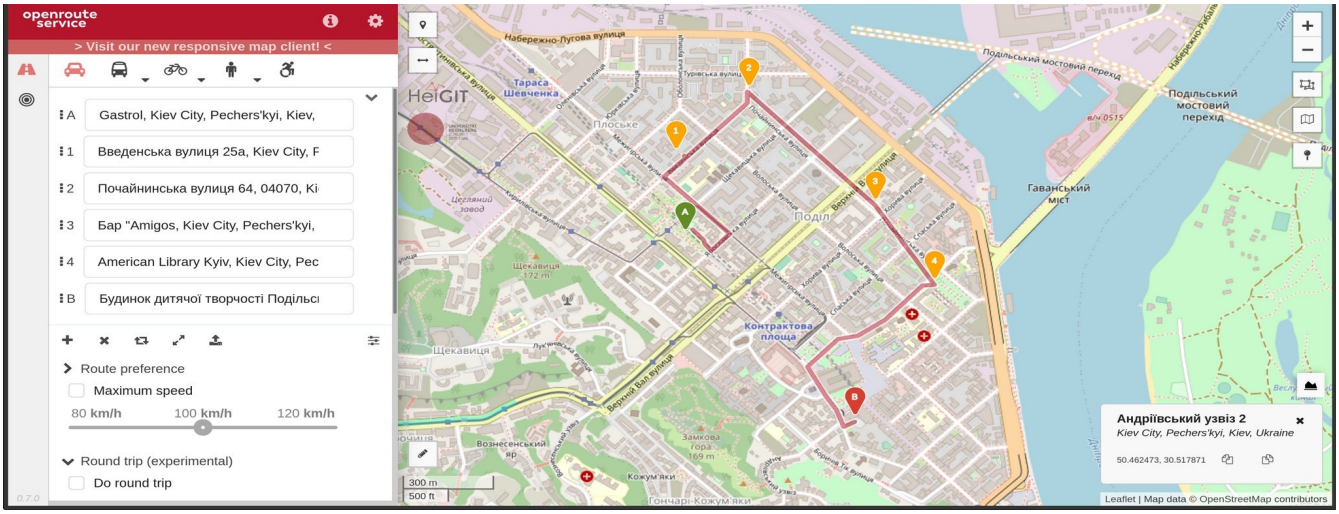
На малюнку 4 продемонстровано виділення об’єкта з категорії невидимий, яка використовується для позначення незначних споруд як підземний перехід, або сукупності будівель і певних територій.

Wikimapia Api[8] - це система, яка дозволяє отримувати дані з карт та легко інтегрувати геодані у зовнішній додаток або веб-сайт. Безкоштовний доступ до БД Wikimapia надається з метою допомогти географічному співтовариству використовувати ці дані не лише на власному веб-сайті, але через будь-який інший веб-сайт або додаток, включаючи мобільні. API на момент написання даної роботи ще перебуває на стадії розробки (бета-версія), незабаром планується додати більше можливостей, таких як дороги, використання графічних плиток, а потім додавання інформації з мобільних пристроїв до БД Wikimapia.

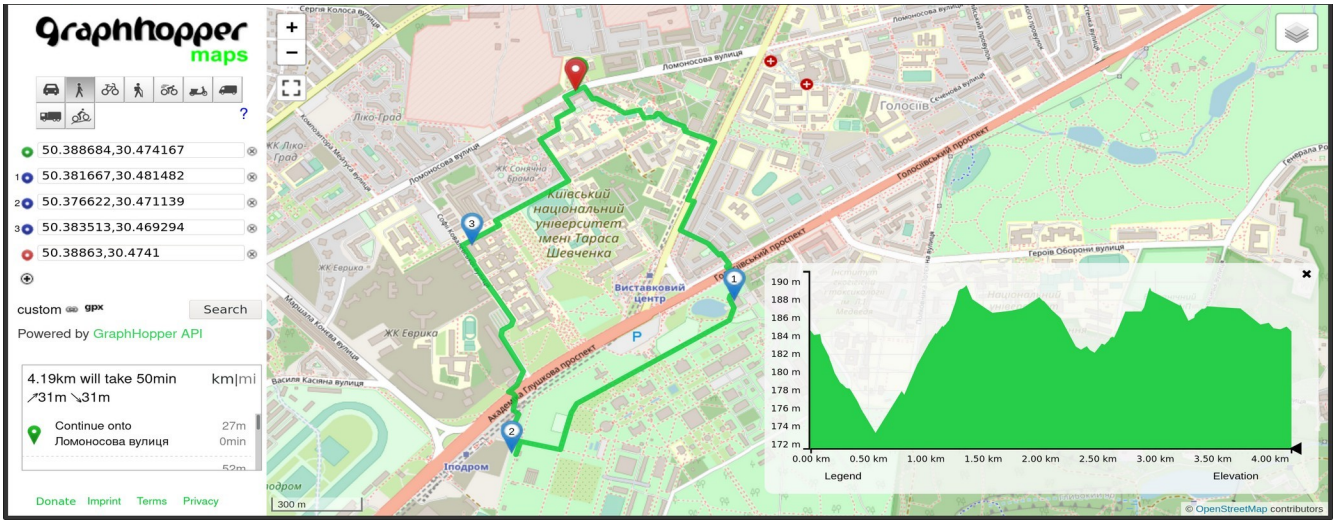
1.2.4 OpenStreetMap

– мапа світу сворена людьми і відкрита до використання за відкритою ліцензією. І це просто грааль в призначеній для користувача картографії.

На базі OpenStreetMap створено багато персональних карт, в тому числі 1й приклад - MapQuest, а також більш функціональна Openrouteservice[9], і доволі популярний GraphHoper[10].



мал.5



мал.6

Як видно Openrouteservice(мал.5) і GraphHoper(мал.6) доволі схожі за будовою. В обох можна обирати яким чином ти будеш пересуватись, зручний механізм додавання нових точок без потреби знати адресу, на правий клік вивалюється меню з вибором чи ти хочеш зробити цю точку стартовою, фінішною, або прохідною. Також в обох варіантах нам показують графік висоти, в Openroute у

правому нижньому іконка з горою розгортає схожий графік, і це впливає на вибір маршрута для різних способів пересування.

1.2.5 Leaflet

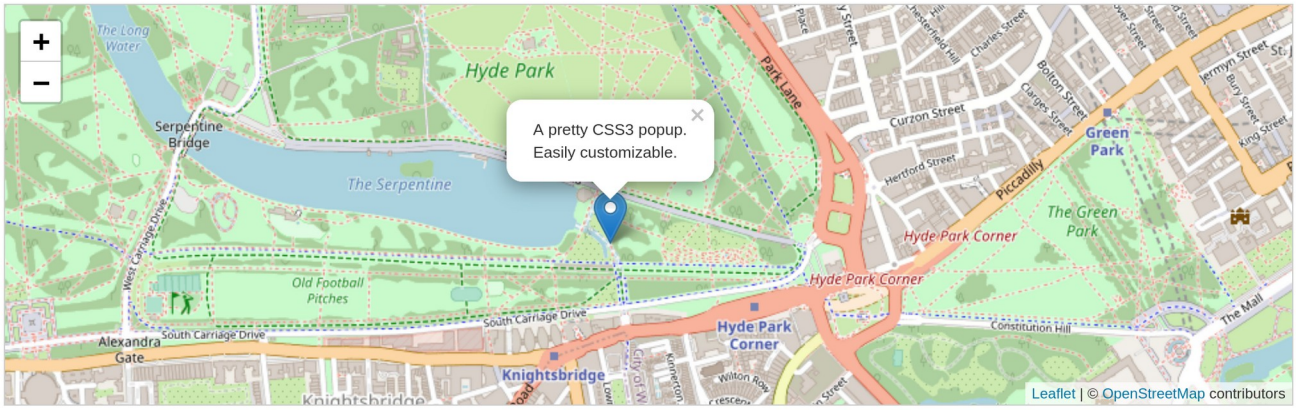
Провідна бібліотека JavaScript з відкритим кодом для побудови зручних мобільних та інтерактивних карт. Вона має всі функції для роботи з картами, які потрібні більшій частині розробників.

Leaflet розроблена з урахуванням простоти, продуктивності та зручності використання. Вона ефективно працює на всіх основних платформах, може бути доповнена різноманітними плагінами, і має простий у використанні API зі зрозумілою і докладною документацією.

Поєднання Leaflet та OSM дозволяє будувати круті надбудови над картами, і додавати нові утиліти, а також робити власні картографічні додатки.

Openrouteservice з прикладу вище побудований саме таким чином.

Мал.7 - приклад Leaflet.



мал.7

1.3 Постановка задачі

Аналіз існуючих додатків і сервісів, представлений в попередньому розділі, допомагає сформулювати вимоги, які можна висунути для розробки додатку по розрахунку оптимального маршруту, адже представлені картографічні засоби не розв'язують поставлену задачу.

1. Функціонал програми полягає в пошуку точок на карті за запитом клієнта, побудові оптимального маршруту за знайденими даними, виведення його на інтерактивній карті, візуалізація елементів маршруту
2. Додаток буде реалізовано англійською для універсальності
3. Зручніше з точки зору реалізації, та для доступності для нових клієнтів буде реалізація програми як веб-сайту, потім можна буде адаптувати для мобільних пристроїв
4. У зв'язку зі стрімкістю розвитку технологій, сервіс повинен бути спроектований таким чином, щоб обсяг робіт по заміні оптимізаційного алгоритму, або будь-якої іншої частини був мінімальний. Повинен мати гнучку і здатну до змін архітектуру.

1.4 Висновки по Розділу 1

В цьому розділі було представлено короткий огляд деяких з доступних технологій для побудови маршрутів на картах, а також коротка історія формування завдання пошуку оптимального маршрута.

Представлені картографічні засоби не розв'язують поставлену задачу.

РОЗДІЛ 2. ОГЛЯД ІСНУЮЧИХ МЕТОДІВ РОЗВ'ЯЗВННЯ ЗАДАЧІ

2.1 Обгрунтування NP-складності

Доведення NP-складності задача комівояжера є в [12]. Розглянемо його:

Спершу ми показуємо, що що задача комівояжера є NP. Враховуючи приклад проблеми ми використовуємо послідовність n вершин у турі. Алгоритм перевірки перевіряє, чи ця послідовність містить кожну вершину рівно один раз, підсумовує граничні витрати і перевіряє, чи не більше суми k. Цей процес, безумовно, може бути здійснений за поліноміальний час.

Для того щоб довести NP-складність задачі комівояжера, ми показуємо, що Гамільтонів цикл поліноміально зводиться до неї. Нехай $G = (V, E)$ — Гамільтонів цикл. Будуємо приклад задачі комівояжера наступним чином. Ми формуємо граф $G` = (V, E`)$, *де* $E` = \{ (i, j): i, j \in V, i \neq j \}$, і визначаємо функцію витрат c:

$$c(i, j) = \begin{cases} 0 & \text{if } (i, j) \in E \ , \\ 1 & \text{if } (i, j) \notin E \ . \end{cases}$$

(Оскільки G не спрямований, він не має автоциклів, і тому $c(v, v) = 1$ для всіх вершин $v \in V$.) Тоді екземпляром задача комівояжера є $<G`, c, 0>$, який ми можемо легко створити за поліноміальний час.

Тепер ми показуємо, що граф G має гамільтонів цикл тоді і тільки тоді, коли граф G` має тур витрат не більше 0. Припустимо, що графік G має гамільтонів цикл *h*. Кожне ребро в *h* належить *E* і, отже, коштує 0 у G`. Отже, *h* - тур в G` із вартістю 0. І навпаки, припустимо, що граф G` має тур *h`* вартості не більше 0. Оскільки витрати ребер в E` дорівнюють 0 і 1, вартість туру h` дорівнює рівно 0, і кожне ребро в турі повинно коштувати 0. Отже, h` містить лише ребра в E. Ми робимо висновок, що h` є гамільтоновим циклом у графі G.

2.2 Алгоритмічні інструменти для розв'язання NP-складних здач.

Багато реальних задач є "NP-складними" і видаються нерозв'язними за допомогою завжди коректних і завжди швидких алгоритмів. Коли у проекті з’являється NP-складна задача, потрібно іти на компроміс щодо правильності або швидкості. Розглянемо старі методи (наприклад, повний перебір, динамічне програмування) та нові (як розв'язувачі MIP та SAT) для розробки точних алгоритмів, які різко покращуються при вичерпному пошуку. Ми також розглянемо старі методи (наприклад, жадібні алгоритми) і нові (як локальний пошук) для розробки швидких евристичних алгоритмів, які є «приблизно правильними» або неточними, із застосуванням програм для планування.[11]

2.3 Точні алгоритми

Існує дві группи методів для розв’язання точних алгоритмів - одна з них це методи релаксації лінійного програмування TSP : алгоритм Гомори, метод внутрішньої точки, метод гілок і меж. Друга це методи динамічного програмування.

Особливість методів для реалізації точних алгоритмів - гарантія знаходження оптимальних рішень при загальній трудомісткості процесу.[11]

Повний перебір (Brute Force)

Метод повного перебору або грубої сили - один з найбільш очевидних методів розв’язання задачі обчислення оптимального маршруту. Його суть полягає в грубому переборі всіх варіантів шляхів, алгоритм розв’язку можна записати як:

1. Визначити число можливих гамільтонових контурів
2. Визначити вагу кожного контуру, склавши ваги всіх його ребер

3. Обрати контур з мінімальною вагою

Метод повного перебору він гарантує знаходження рішення задачі TSP, при цьому він прямолінійний і простий в виконанні, що доволі непогано. Однак, яким би легким не здавався такий спосіб для людини, алгоритм вважається неефективним при роботі з великим обсягом даних, бо для знаходження оптимального маршруту вимагає знайти ваги (n - 1)! гамільтонових контурів.

Метод гілок і меж (Branch and Bound)

Метод гілок і меж часто використовується для знаходження оптимального рішення задач комбінаторної оптимізації.

Його суть полягає в розбитті множини на підзадачі і виключення неоптимальних рішень.[13]

2.4 Неточні алгоритми

Алгоритми даної групи роблять акцкент на швидкості знаходження рішення а не на його точності. У свою чергу так алгоритми можна розділити на
дві категорії: наближені (Approximation Algorithms) і евристичні (Heuristic Algorithms).

Алгоритм найближчого сусіда (NearestNeighbour)

Один з найпростіших евристичних методів розв'язання TSP. Головне правило алгоритму - завжди вибирати сусіднє місто (сусіда). Алгоритм складається з наступних кроків:

1. Вибрати будь-яке місто;
2. Визначити сусіднє місто, не включене в маршрут, і перейти в нього;
3. Перевірити чи залишилися міста, не включені в маршрут, якщо так - повторити другий крок;

5. Додати ребро між останнім включеним містом і початковим.

У загальному випадку трудомісткість рішення задачі дорівнює O (n^2).

Жадібний алгоритм (Greedy)

Жадібний алгоритмі формує рішення по частинах, завжди вибираючи наступний фрагмент, який пропонує найбільш очевидну та негайну користь. Тож проблеми, або такі конкретні дані, де вибір місцево оптимальних рішень також веде до глобального рішення, найкраще підходять для таких алгоритмів.

Щоб розв’язати TSP з використанням жадібного алгоритму, ми передивляємось всі ребра, виходять з точки-вузла, і вибираємо *n* найкоротших дуг. Якщо ті *n* коротких дуг формують гамільтонов цикл, тоді ми знайшли оптимальне рішення.

Трудомісткість рішення задачі жадібним алгоритмом дорівнює O (n^2).

Нижня межа вартості оптимального маршруту вище нижньої межі Хелда-Карпа на 15-20%.

Алгоритм Ліна-Кернігана (Lin-Kernighan)

Алгоритм Ліна-Кернігана вважається одним із найбільш ефективних методів пошуку оптимальних або майже оптимальних маршрутів для задачі побудови маршрутів.

В алгоритмі вказані заміщення, які можуть перетворити один розв'язок-кандидат у інший. Враховуючи здійсненний тур TSP, алгоритм неодноразово виконує обміни, що зменшують тривалість поточного туру, доки не буде досягнуто тур, для якого жоден обмін не дасть покращення. Цей процес може повторюватися багато разів з початкових турів, сформованих випадковим чином. Алгоритм Ліна-Кернігана (LK) виконує так звані "k-opt" ходи в турах. Виконання k-opt змінює тур,

замінюючи k-шляхи з туру на k-шляхи таким чином, що досягається коротший тур. Нехай буде поточний тур. На кожному кроці ітерації алгоритм намагається знайти два набори ребер, X = {x1, ..., xk} і Y = {y1, ..., yk}, такі, що якщо ребра X видаляються з T та, замінюються на ребра Y, результат - кращий тур. Ребра X називаються позаребрами. Ребра Y називаються внутрішніми. Два набори X і Y будуються елемент за елементом.

Якщо коротко, це передбачає обмін парами під-турів, щоб зробити новий тур. Це узагальнення (наприклад 2-opt та 3-opt) працюють, перемикаючи два-три шляхи, щоб зробити тур коротшим. Алгоритм Ліна-Кернігана адаптивний і на кожному кроці вирішує, скільки шляхів між точками потрібно змінити, щоб знайти коротший шлях. Розробка і реалізація алгоритму лише проста, так як алгоритм складається з безлічі кроків, більшість з яких сильно впливає на роботу алгоритму. [14]

Алгоритм пошуку з заборонами (TabuSearch)

Одна з неприємних проблем алгоритму найближчого сусіда полягає в тому, що він часто застрягає в точці локального оптимуму. Цього можна уникнути, застосувавши алгоритм пошуку із заборонами. Даний метод дозволяє переходити від одного локального оптимуму до іншого в пошуках глобального. Після переходу ребро потрапляє в список заборон і повторно не використовується, крім випадків, коли воно може поліпшити побудований оптимальний шлях. На практиці заборонений набір зберігається як комбінація пройдених кроків, яка ддозволяє побудувати подальший шлях, що продовжує поточний розв'язок. [15]

Недоліком цього методу можна вважати час його виконання – трудомісткість алгоритму оцінюється як O (n^3). [16]

Мурашиний алгоритм (Ant Colony Optimization)

Мурашиний алгоритм – ефективний поліноміальних алгоритм, розроблений на базі поведінки справжніх мурах. Вперше його принципи були описані в 1991 Марко Дориго. Мурахи співпрацюють в пошуках харчового ресурсу, тому вони залишають слід специфічних феромонів на їх шляху від колонії до місця де вони знайшли їжу.[17] Цей тип невербальної комунікації називають стігмергія – стимуляція, заснована на досвіді попередніх мурах і спрямована на підвищення продуктивності. [18]

Алгоритм полягає в тому, щоб випустити деяку кількість “мурах”, які будуть рухатись випадковим чином, і відмічати пройдені шляхи, але не будуть заходити в точки які вони відвідали, окрім випадку завершення маршруту. Таким чином на найкоротшому шляху буде найбільше відміток проходу, і наступна мураха з великою ймовірністю обере саме цей шлях.

2.5 Генетичний алгоритм

Генетичні алгоритми належать до методів оптимізації, в основу яких лягли біологічні процеси. Еволюційна теорія дає визначення природного відбору, згідно з яким особи, більш пристосовані до умов навколишнього середовища, мають більше шансів на виживання, а отже і на продовження роду, і навпаки – непристосовані особини мають мінімальні шанси на залишення потомства. Основою відбору є мутації генів і їх комбінації, що формуються при розмноженні і передаються потомству. В ході природного добору виживають особини, що краще пристосувались, тобто ті що мають найбільшу пристосованість. Ця чисельна характеристика, може змінюватися в залежності від умов конкретного завдання та середовища. Пристосовані особини схрещуються і дають потомство. Випадкові мутації також можуть впливати на розвиток популяції.

Кожен крок алгоритму можна розбити на три етапи:

1. Пропорційний відбір особин поточного покоління з найбільшою пристосованістю, які можуть виробляти потомство, і формування з них проміжної популяції;

2. Проміжну популяцію ділять на пари і з якоюсь імовірністю схрещують, в результаті чого в нове покоління потрапляє сама пара або її нащадки при їх присутності. Нащадки формуються з частин батьківських послідовностей, розділеної деякою точкою.

3. Відбувається мутація отриманого покоління, що не допускає передчасної збіжності. [20]

Критерієм отримання оптимального результату може бути задане число зміни поколінь, або виродження популяції (коли всі рядки є майже ідентичними та знаходяться в області екстремуму). Результатом роботи алгоритму буде особина з найбільшим значенням функції пристосованості.

При використанні генетичного алгоритму для розв’язання завдання пошуку оптимального шляху виконуються всі перераховані вище кроки, а в якості пристосованості особини виступає довжина маршруту.

Існують кілька різних реалізацій класичного генетичного алгоритму в залежності від підходу до етапів схрещування і мутації. Розглянемо приклад: спочатку є n пунктів, з'єднаних дорогами, тобто з будь-якого пункту можна потрапити в будь-який інший. В якості набору хромосом особини буде одновимірний масив. Він представляє собою послідовий список всіх міст, такий що в результаті повинні повернутися початкову точку. Наприклад – 5 міст, у особини масив 2-4-3-1-5-2 або 5-1-4-3-2-5 і так далі, отже маємо покоління яке складається з k особин, у кожної з яких є власний шлях обходу (можуть повторюватися). Початкові значення цього обходу обираються випадковим чином. Далі буде організовано зміну поколінь, тобто особини можуть розмножуватися і

іноді мутувати. Мутувати мають не часто, аби не відходити задалеко від вигідних мутацій які вони вже мають.

Схрещування особин в генетичному алгоритмі зазвичай реалізується просто - набір хромосом двох особин розривається на 2 частини і однією частиною обмінюються один з одним. Однак до задачі комівояжера потрібно додати деякі додаткові модифікації, бо ми не можемо роздирати два маршрути і робити з них один, адже кожне місто обходиться тільки один раз. Тобто номер міста може бути в масиві 2-3-1-4-5-2 тільки один раз (виключаючи крайні) тому треба застосовувати один з можливих методів:

1. Частково відображувальне схрещення
2. Впорядковане схрещування
3. Циклічне схрещення

Згідно з [20] деякі з цих методів показали хороші показники при тестуванні. Але, як і у випадку з алгоритмом пошуку із заборонами, трудомісткість процесу створює проблему при достатньо великій кількості вузлів.

РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ

Третя глава описує використані для розробки програмні інструменти, також наводиться алгоритм, який будує оптимальний маршрут за заданими на мапі точками. В кінці розділу аналіз роботи програми.

3.1. Огляд використовуваних в розробці програми інструментів

Framework - **Flask**

Flask — mikroframework, де „Micro” означає, що Flask прагне зробити ядро, простим, але розширюваним. Flask не прийме багато рішень замість розробника, наприклад, яку базу даних використовувати. Ті рішення, які він приймає, легко змінити. Все інше залежить від розобника, тож Flask може бути всім, що потрібно, і нічого зайвого. Зручний для використання як у великих проектах, так і для програм з кількох рядків коду.

За замовчуванням Flask не включає рівень абстракції бази даних, перевірку форми чи щось інше, де вже існують різні бібліотеки, які можуть це реалізувати. Натомість Flask підтримує розширення для додавання такої функціональності до програми. Численні розширення забезпечують інтеграцію баз даних, перевірку форми, обробку завантажень, різні технології відкритої автентифікації тощо. За рахунок такої гнучкості може бути налаштованим під різні проекти з різними потребами без додаткового налаштування відносин між частинами програми.

Flask має багато значень конфігурації, з розумними значеннями за замовчуванням, і кілька умов для початку. За домовленістю, шаблони та статичні файли зберігаються у підкаталогах у дереві джерела Python програми, із іменами **templates** та **static** відповідно.

Flask надає дуже простий шар для приєдняння найкращого, що може запропонувати Python, а також містить безліч можливостей для налаштування поведінки. [21]

OpenStreetMap - це проект зі створення безкоштовної та редагованої карти світу, що буде постійно оновлюватись. Геодані, що лежать в основі карти, вважаються основним результатом проекту. Створення та зростання OSM було зумовлене обмеженою доступністю картографічних ресурсів та появою недорогих портативних пристроїв для супутникової навігації.

Дані карти збираються волонтерами, які проводять систематичні наземні обстеження за допомогою таких інструментів, як портативний GPS, блокнот, або цифрова камера, можливо вже використовують дрони. Далі дані вводяться в базу даних OpenStreetMap за допомогою ряду програмних засобів, включаючи JOSM та Mercator. Змагальні заходи Mapathon також проводяться командою OpenStreetMap та неприбутковими організаціями спільно з органами місцевого самоврядування, щоб скласти карту певної області.

Наявність аерофотознімків та інших даних з комерційних та державних джерел додало важливі джерела даних для ручного редагування та автоматизованого імпорту.

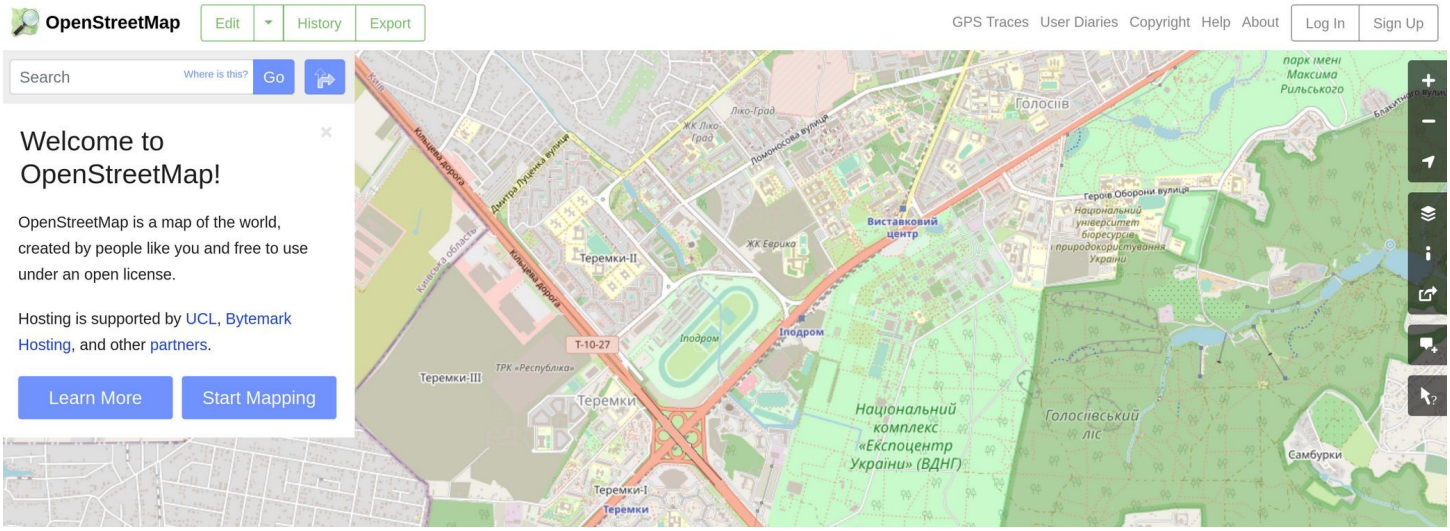
OpenStreetMap використовує топологічну структуру даних з чотирма основними елементами (також відомими як примітиви даних):

Вузли - це точки з географічним положенням, що зберігаються як координати (пари широти та довготи) відповідно до WGS 84.

Шляхи є впорядкованими списками вузлів, що представляють полілінію або, можливо, багатокутник, якщо вони утворюють замкнутий цикл. Вони використовуються як для представлення лінійних об’єктів, таких як вулиці та річки, так і районів, таких як ліси, парки, паркувальні ділянки та озера.

Відношення - це впорядковані списки вузлів, шляхів і відносин (разом звані "члени"), де кожен член може за бажанням мати "роль" (рядок). Відносини використовуються для представлення зв'язку вузлів і шляхів. Приклади включають обмеження поворотів на дороги, маршрути, що охоплюють декілька існуючих шляхів (наприклад, автомагістраль на великі відстані), та ділянки з дірками.

Теги - це пари ключ-значення (обидва довільні рядки). Вони використовуються для зберігання метаданих про об’єкти на карті (наприклад, про їх тип, назву та їх фізичні властивості). Теги не стоять окремо, а завжди прикріплюються до об’єкта: до вузла, способу чи відношення. [22][23]



Leaflet - це бібліотека JavaScript з відкритим кодом, яка використовується для створення додатків веб-відображення. Вперше випущений в 2011 році, він підтримує більшість мобільних і настільних платформ, підтримуючи HTML5 та CSS3. Серед його користувачів - FourSquare, Pinterest та Flickr.

Leaflet дозволяє розробникам без фону GIS дуже легко відображати веб-карти, розміщені на загальнодоступному сервері, з необов’язковими плиточними накладками. Він може завантажувати дані про функції з файлів GeoJSON, стилювати їх та створювати інтерактивні шари, такі як маркери зі спливаючими вікнами при натисканні. [24]

3.2 Логіка роботи додатку

Додаток являє собою веб-сайт, тобто backend та frontend.

На клієнтській частині реалізовано відображення інтерактивної карти, та панелі інструментів де можна задавати час бажаної прогулянки, метод пересування, та обрати тип місць які користувач бажає відвідати, наприклад історичні будівлі, або музеї.

При вході на сайт відображається мапа, але для використання функцій побудови маршруту необхідна авторизація. Якщо користувач не авторизован у системі, то на панелі інструментів всі слайдери та інструменти будуть відображені як не активні (тобто сірі, і не реагують на курсор), але кнопка входу і посилання на реєстрацію будуть активні, і при натисканні на неактивні буде відображатись пропозиція зареєструватись або увійти у свій акаунт.

3.2.1 Реалізація авторизації

Розширення Flask-WTF використовує класи Python для подання веб-форм. Клас форми лише визначає поля форми як змінні класу.

```
from flask_wtf import FlaskForm
from wtforms import StringField, PasswordField, BooleanField, SubmitField
from wtforms.validators import DataRequired

class LoginForm(FlaskForm):
    username = StringField('Username', validators=[DataRequired()])
    password = PasswordField('Password', validators=[DataRequired()])
    remember_me = BooleanField('Remember Me')
    submit = SubmitField('Sign In')
```

Чотири класи, які представляють типи полів, використані для цієї форми, імпортуються безпосередньо з пакету WTForms. Для кожного поля об'єкт створюється як змінна класу в класі LoginForm. Кожному полю присвоюється опис або мітка як перший аргумент.

Додатковий аргумент validators, використовується для прив'язки поведінки перевірки до полів. Валідатор DataRequired лише перевіряє, що поле не відправлено порожнім.

Шаблон форми входу буде наслідувати базовий шаблон сайту.

Функція перегляду яка відображає шаблон:

```
@app.route('/login')
def login():
    form = LoginForm()
    return render_template('login.html', title='Sign In', form=form)
```

Тут імпортувано клас LoginForm з forms.py, та створено екземпляр об'єкта з нього і відправлено його в шаблон. Синтаксис form = form може виглядати дивно, але просто передає об'єкт форми, створений в рядку вище, в шаблон з формою імені. Це все, що потрібно для відображення полів форми.

Функція перегляду, що приймає і перевіряє дані введені користувачем

```
from flask import render_template, flash, redirect

@app.route('/login', methods=['GET', 'POST'])
def login():
    form = LoginForm()
    if form.validate_on_submit():
        flash('Login requested for user {}, remember_me={}'.format(
            form.username.data, form.remember_me.data))
        return redirect('/index')
    return render_template('login.html', title='Sign In', form=form)
```

Шаблон форми входу

```
{% extends "base.html" %}

{% block content %}
    <h1>Sign In</h1>
    <form action="" method="post"  novalidate>
        {{ form.hidden_tag() }}
        <p>
            {{ form.username.label }}<br>
            {{ form.username(size=32) }}<br>
            {% for error in form.username.errors %}
            <span style="color: red;">[{{ error }}]</span>
            {% endfor %}
        </p>
        <p>
            {{ form.password.label }}<br>
            {{ form.password(size=32) }}<br>
            {% for error in form.password.errors %}
            <span style="color: red;">[{{ error }}]</span>
            {% endfor %}
        </p>
        <p>{{ form.remember_me() }} {{ form.remember_me.label }}</p>
        <p>{{ form.submit() }}</p>
    </form>
 {% endblock %}
```

Цикли після поляів username і password відображають повідомлення про помилки, додані валідаторами в червоному кольорі. Як правило, всі поля, що мають прикріплені перевіряючі елементи, матимуть повідомлення про помилки, що додаються в form. <Field_name> .errors. Це буде список, тому що поля можуть мати кілька валідаторів і повідомлень про помилки може бути більше одного для відображення користувачеві.

Якщо ви спробуєте відправити форму з порожнім ім'ям користувача або паролем, ви отримаєте червоне повідомлення про помилку.

Хешування паролів (Password hashing) - це складна тема, але є кілька простих у використанні бібліотек, які реалізують всю цю логіку таким чином, щоб її можна було викликати з програми.

Одним з пакетів, що реалізують хешування паролів, є Werkzeug, який був встановленим Flask.

Демонстрація хешування:

```
>>> from werkzeug.security import generate_password_hash
>>> hash = generate_password_hash('foobar')
>>> hash
'pbkdf2:sha256:50000$vT9fkZM8$04dfa35c6476acf7e788a1b5b3c35e217c78dc04539d29
5f011f01f18cd2175f'
```

Для реалізації логіки хешування паролів достатньо лише двох нових методів в користувацькій моделі:

```
class User(db.Model):
    # ...

    def set_password(self, password):
        self.password_hash = generate_password_hash(password)

    def check_password(self, password):
        return check_password_hash(self.password_hash, password)
```

Після авторизації всі поля панелі інструментів активні, а замість кнопки входу відображається аватар і ім’я користувача.

Для користувачів без аватару, він автоматично виставляється за допомогою md5 з hashlib.

```
from hashlib import md5
# ...

class User(UserMixin, db.Model):
    # ...
    def avatar(self, size):
        digest = md5(self.email.lower().encode('utf-8')).hexdigest()
        return 'https://www.gravatar.com/avatar/{}?d=identicon&s={}'.format(
            digest, size)
```

Функція повертає геометричне зображення приблизно такого вигляду:

3.2.2 Панель інструментів

На панелі послідовні поля для обрання параметрів маршрута, а саме:

Поле для задання часу прогулянки має список запропонованих варіантів (20хв, 30хв, 40хв, 2год, тощо), і можливість вводу довільного часу з вибором одиниць виміру хв/год, який за замовчуванням стоїть на хвилинах.

Поле для вибору способу пересування, на якій намальовані декілька різних варіантів, і кнопка розширення варіантів.

Тобто постійно показані декілька найпопулярніших, наприклад машина, повільна хода, велосипед, а у повному списку всі інші можливі варіанти, такі як скутери та електронні види транспорту, інвалідний візок, тощо. За замовчуванням стоїть звичайна хода, або спосіб обраний користувачем в профілі як основний.

Ім’я	**Опис**	**Обмеження**	**Позначка**
car	Режим поїздки звичайним авто	Автомобільний доступ	
small_truck	Мала вантажівка як Mercedes Sprinter або Ford Transit	висота=2.7m, ширина=2+0.4m, довжина=5.5m, вага=2080+1400 kg	
truck	Вантажівка як MAN або Mercedes-Benz Actros	висота=3.7m, ширина=2.6+0.5m, довжина=12m, вага=13000 + 13000 kg, hgv=yes, 3 Axes	
scooter	Режим мопеда + Пішохід або ходьба без небезпечних навантажень	Швидкий по місту, здатен ігнорувати певні перешкоди, максимальна швидкість приблизно 50 км / год.	
foot	Пішохід або ходьба без небезпечних навантажень	Пішохідний доступ	
hike	Піші прогулянки з пріоритетом для більш мальовничих маршрутів і, можливо, трохи довші. На тривалість ходьби впливають перепади висот.	Пішохідний доступ	

Ім’я	Опис	Обмеження	Позначка
bike	Трекінговий велосипед, уникаючи пагорбів	Велосипедний доступ	
mtb	Гірський велосипед	Велосипедний доступ	
racingbike	Міський або шосейний велосипед	Велосипедний доступ	

Табл.1 — позначки і відмінності режимів пересування

Потім йде поле вводу типу цілей прогулянки, з списком запропонованих варіантів, наприклад історичні будівлі, музеї, пам’ятки архітектури, тощо.
Під цим кнопка для запуску розрахунку маршрута, яка не активується без заповнення поля з цілями прогулянки. Після заповнення і відправлення запита користувачеві показується анімація для очікування результата.

3.2.3 Пошук

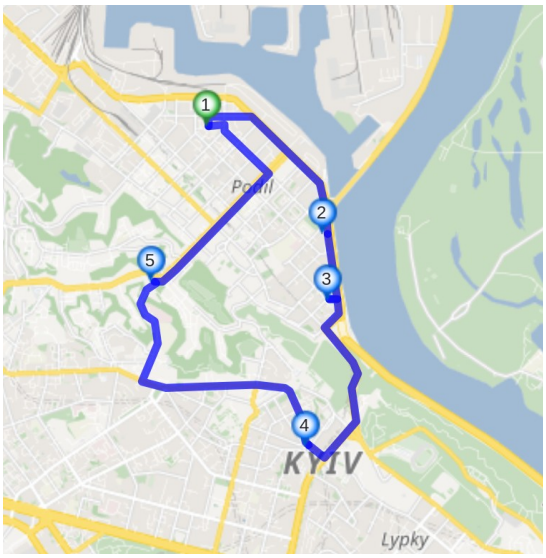
Якщо користувач дозволив додатку доступ до його локації, то вона буде введена як точка відправлення, в іншому випадку потрібно буде задати її власноруч.

За часом і способом пересування визначається зона пошуку. Тобто якщо користуачеві потрібно повернутись на стартову позицію, то найдальша можлива точка буде на дистанції в половину шляху який можна пройти за заданий час.

Отримавши область пошуку, програма шукає всі об’єкти що підходять за описом або типом, та додає їх координати до списку у форматі JSON:

```
{
     "id": "location_666",
     "name": "visit_666",
     "address": {
       "location_id": "location_666",
       "lon": 0.000,
       "lat": 0.000
       }
 }
```

Список та інші модифікації передаються API для побудови маршрута



Distance: 8km, transport-time: 17min

stop	arrival	departure	distance
0	-	00:00:00	0
1	00:03:17	00:03:17	1.54
2	00:04:16	00:04:16	2.11
3	00:07:24	00:07:24	3.8
4	00:13:32	00:13:32	6.28
5	00:17:56	-	8.16

Також є можливість для прибирання і додавання точок до маршрута.

Також буде додана можливість зберігати маршрути в профілі користувача.

3.3 Тестування додатку та аналіз отриманих даних

Розглянемо як відрізняється маршрут побудований для різних способів пересування при заданому деякому маршруті. Наприклад через декілька вікових дубів Голосієва:

Дуб Вітовта - ботанічної пам’ятки природи місцевого значення. Вік близько 300 років. Обсяг 4 м. Висота 30 м. Координати: 50°23'3"N 30°29'49"E

Дуб Слави – приблизно 300 років, Координати: 50°22'32"N 30°29'20"E

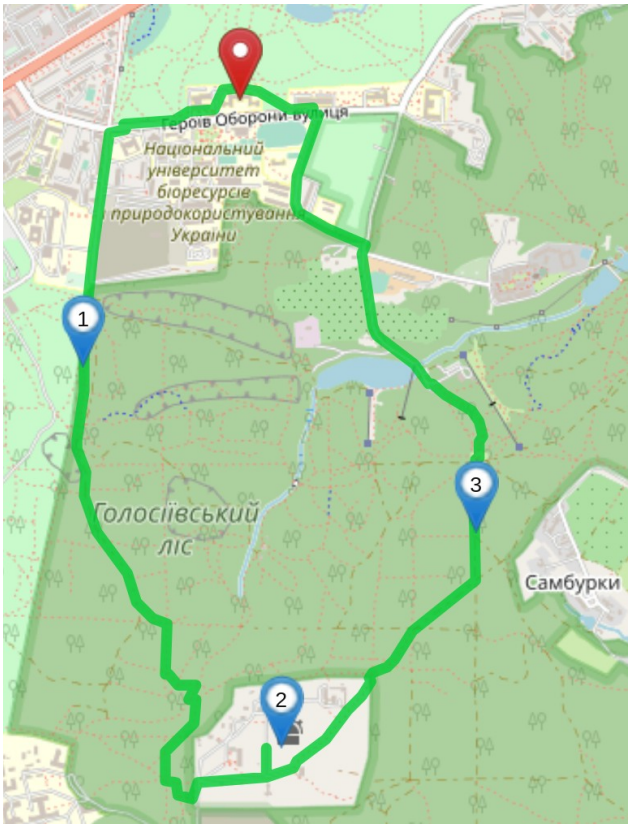
Віковий дуб біля обсерваторії, теж ~300, Координати: 50°21'52"N 30°30'6"E

Самбурський дуб імені Михалка - Один з півтисячолітніх дубів біля хутора Самбурки, названий на честь Михайла Михалка - діяча у сфері охорони природи, ініціатора створення Національного природного парку «Голосіївський», учасника дисидентського руху. Координати: 50°22'14"N 30°30'28"E

пішки

6.87km will take 1h 22min

↗118m ↘120m

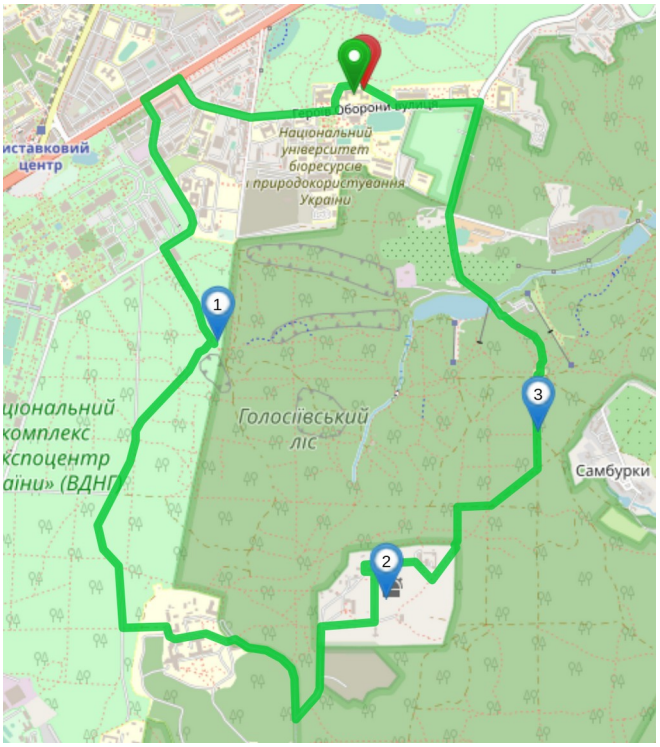


Велосипед

9.21km will take 52min

↗122m ↘124m

Для звичайного велосипеда обираються маршрути без некомфортних перепадів висоти

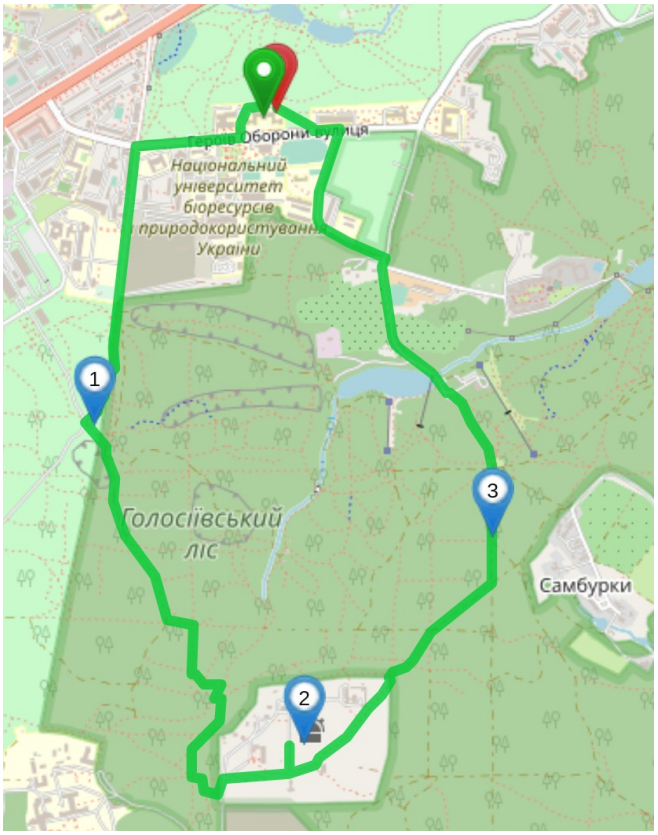


Гірський велосипед

6.88km will take 25min

↗119m ↘120m

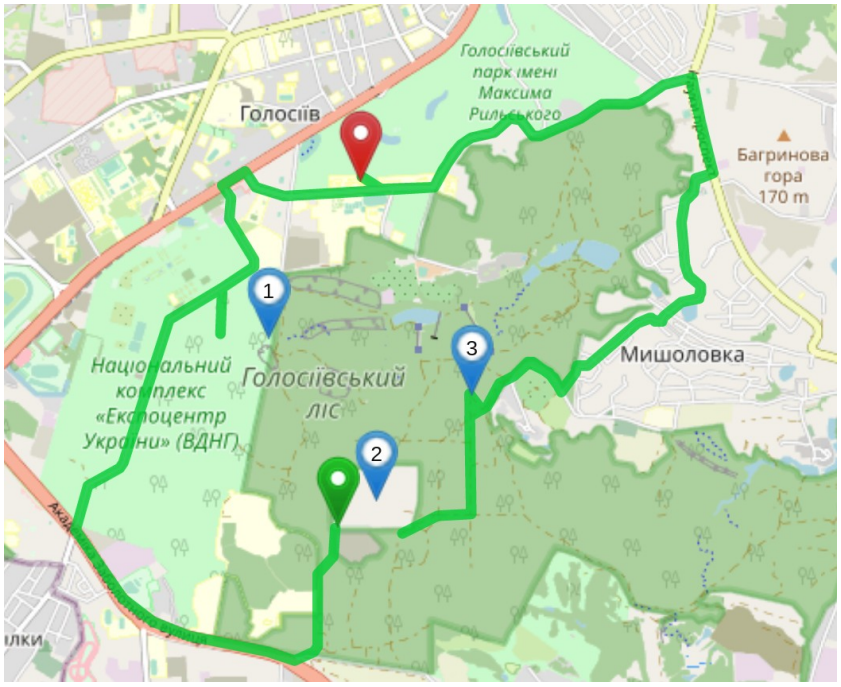
Як бачимо, при уточненні можливостей велосипеда, маршрут перебудовується з розрахунком на доріжки які можна проїхати на гірському велосипеді, і стає подібнішим до пішохідного. Але треба пам’ятати, що можливості транспорту не гарантують безпеку за відсутності відповідних навичків велосипедиста.



Мопед

23.12km will take 1h 11min

↗292m ↘292m

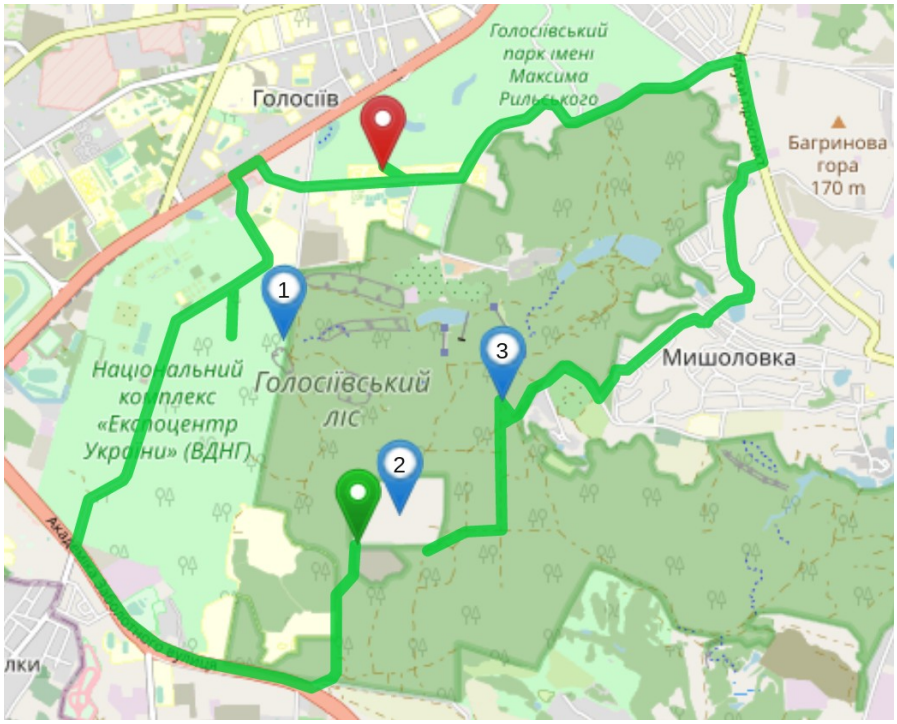


Така карта є максимально правильною для мопеда за всіма стандартами, але здебільшого грунтова дорога без особливих пошкоджень не є перешкодою для виїзду на пікнік на скутері.

Легковий автомобіль

23.12km will take 1h 13min

↗292m ↘292m



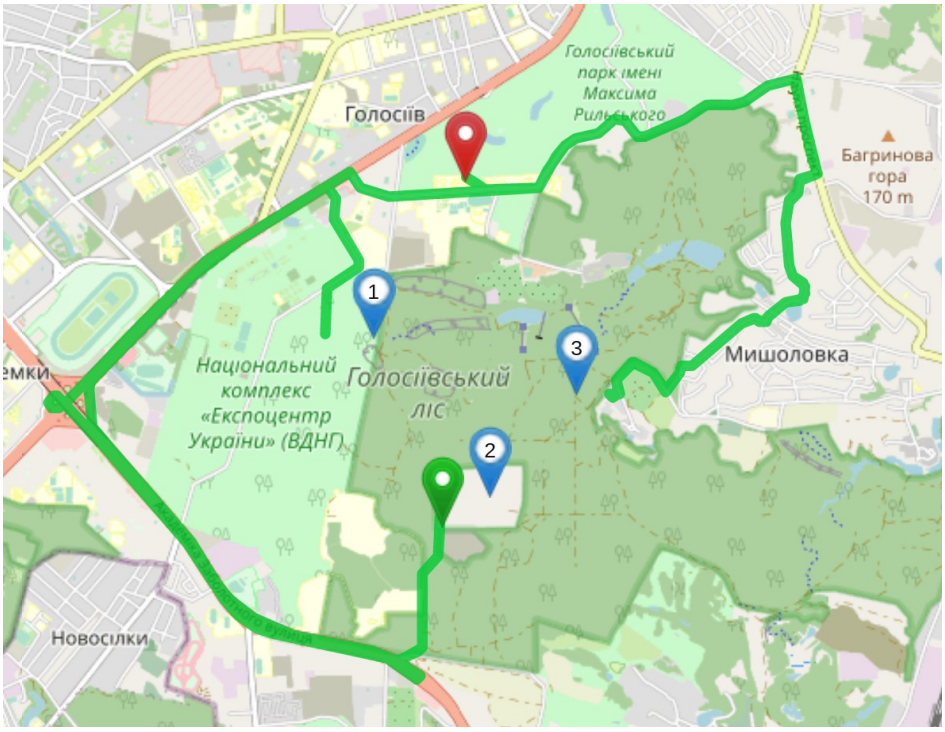
Майже не відрізняється від мопеду. За стандартами звичайний транспорт не поїде грунтовими або пішохідними дорогами. Хоча деякі дороги які в наших картах відмічені як нормальні дороги, насправді будуть важчими для проїзду транспорту ніж не відмічені грунтові дороги.

Мала вантажівка

35.96km will take 1h 50min

↗381m ↘381m

Деякі дороги заборонені для проїзду великогабаритного транспорту.

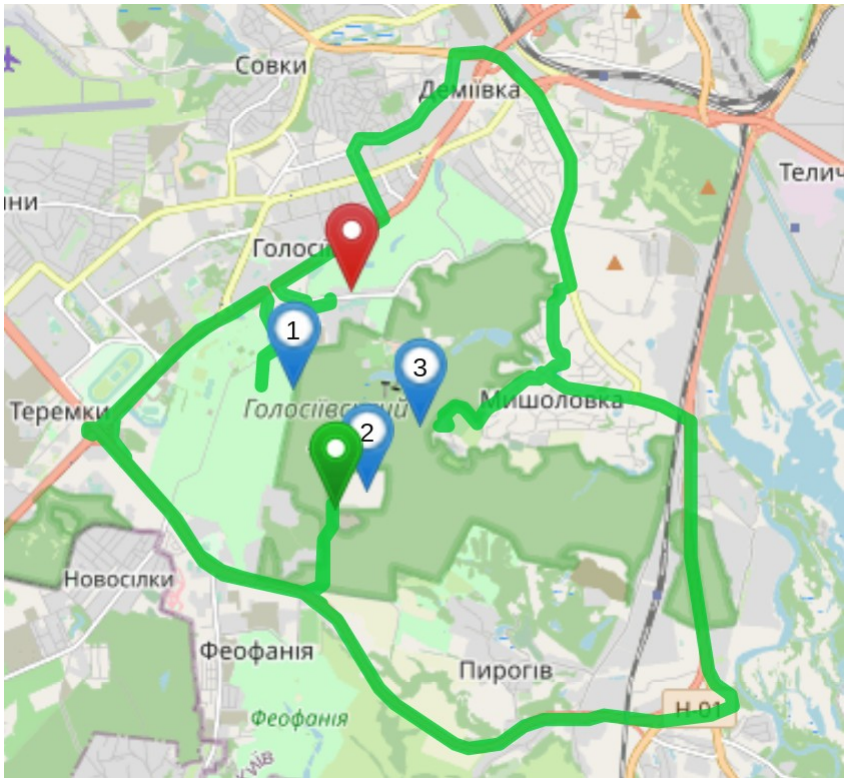


Вантажівка

42.83km will take 2h 54min

↗482m ↘481m

Нажаль програма не розпізнає доцільність використання певних способів пересування за специфічних обставин, а лише прокладає маршрут.



3.4 Висновки по розділу 3

Загалом побудовані маршрути оптимальні та час наближений до реального, розбіжності можуть бути спричиненими неточністю локальних даних на картах, наприклад відсутності деякого шляху на карті, або неправильне внесення на карту інформації щодо будівель та інших об’єктів.

Також маркування доріг не завжди буде відповідати стандартним можливостям певного транспорту, через те що наприклад пішки можна пройти скрізь де немає будівель, і навіть так не завжди. Коли ми йдемо пішки, або їдемо на гірському велосипеді, ми насправді можемо рухатись як завгодно, адже на відміну від машин ми не обмежені дорогами з певними характеристиками. Деякі дороги через ліси або луки спершу були витоптані, а потім їх наносили на карти як стежки.

ВИСНОВОК

Не зважаючи на величезну кількость доступних рішень, завжди можна знайти деякі недоліки як сервісах, так і в інструментах що вони можуть представити. Щось не відображає релевантні дані для вашого регіону, або і зовсім ваш район. В деяких за доступ до оперування або редагування додатковим функціоналом стоять немалі ціни, або інші незручності. Наприклад арі гугла має менше функцій ніж у них є в додатку.

Задача пошуку маршрутів зараз дуже актуальна через появу величезної кількості даних, яких навіть так постійно замало, і вони постійно змінюються, адже реальний світ не залишається статичним. У нас стає більше можливостей для розрахунку, і так само більше варіантів їх застсування. Можливо за кілька років стануть популярними персональні види летючого транспорту, і задача стане ще складнішою, потрібно буде додати ще величезну кількість даних, наприклад постійно враховувати швидкість і напрямок вітру.

Що більше можливих рішень задачі пошуку шляхів ми будуємо, то більше нам стає потрібно, адже неможливо побудувати програму яка буде повністю покривати всі специфічні задачі і потреби, і це чудово. Постійно виикає ще більше рішень як у програмному, так і просто візуальному форматі, і інструменти які є у нас на сьогоднішній день значно перевищують за продуктивністю ті самі декілька років тому. А ті що будуть в майбутньому нам ще довго будувати і покращувати.

СПИСОК ЛІТЕРАТУРИ

1 Google Maps https://maps.google.com/

2 Applegate D.L., Bixby R.E., Chvátal V., Cook W.J. The Traveling Salesman Problem. Princeton University Press, 2007.

3 Левітін А. Алгоритми. Введення в розробку і аналіз. Вільямс, 2006.

4 Гері М., Джонсон Д. Обчислювальні машини і складнообчислювальні завдання. Світ, 1982.

5 MapQuest https://www.mapquest.com/ukraine/kyiv/kyiv/

6 Waze https://www.waze.com/

7 Wikimapia http://wikimapia.org/

8 Wikimapia Api http://wikimapia.org/api/

9 Openrouteservice https://classic-maps.openrouteservice.org/

10 GraphHoper https://graphhopper.com/maps/

11 Algorithms Part 4: Algorithms for NP-Hard Problems Tim Roughgarden

12 Introduction to Algorithms, 3rd Edition (2009), Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, Theorem 34.14

13 Applegate D., Bixby R., Chvatal V., Cook W. TSP cuts outside the template paradigm. Donet, 2000

14 Generalk-opt submoves for the Lin–Kernighan TSPheuristicKeld Helsgaun 2009

15 Alsalibi B.A., Jelodar M.B., Venkat I. A Comparative Study between the Nearest Neighbor and Genetic Algorithms: A revisit to the Traveling Salesman Problem // International Journal of Computer Science and Electronics Engineering (IJCSEE), 2013.

16 Basu S. Tabu Search Implementation on Traveling Salesman Problem and Its Variations: A Literature Survey. Indian Institute of Management Calcutta, 2012. P. 1-8.

17 Gupta R., Chauhan C., Pathak K. Survey of Methods of Solving TSP along

with its Implementation using Dynamic Programming Approach. International Journal of Computer Applications, 2012.

18 Dorigo M., Caro G. D. Ant Algorithms for Discrete Optimization // Artificial Life, 1999

19 Nilsson C. Heuristics for the Traveling Salesman Problem. Linkoping University, 2011. P. 1-6.

20 Захарова О.М., Мінашіна І.К. Огляд методів багатовимірної оптимізації // Інформаційні процеси, 2014.

21 https://flask.palletsprojects.com/en/2.0.x/

22 https://en.wikipedia.org/wiki/OpenStreetMap

23 https://www.openstreetmap.org/about

24 https://leafletjs.com/index.html