**УДК 681.3**

Саламаха О.І.

Національний університет "Львівська політехніка",

кафедра електронних обчислювальних машин

**Сервіс для пошуку вільних паркомісць**

*© Саламаха О.І., 2020*

**Розглянуто сервіс для пошуку вільних паркомісць. Метою є створення системи, що використовуючи доступ до бази даних і засобів геолокацій дозволить користувачеві знайти найкоротший та найзручніший шлях для паркування свого транспортного засобу.**

**Ключові слова: розумне паркування автомобіля.**

**In this work the service for searching available parking lots has been developed.**

**The objective of the current work is to create the IOS app that will enable the users to find the shortest and the most convenient way to park their vehicles by accessing the data base and geolocation facilities**

**Keywords: smart parking**

**Вступ.** Система для пошуку вільних паркомісць є частиною проекту створення розумного міста. Ідея даної системи виникла через бажання сприяння збільшенню комфорту власникам автомобілів. Адже наразі однією з проблем кожного міста-мільйонника є паркування. Зазвичай водіям дуже важко швидко знайти для себе оптимальне місце, це тягне за собою великі витрати часу, коштів, палива. Це також сприяє викидам газів, а відповідно забрудненнню навколишнього середовища. Існує статистика про те, що середня кількість витраченого палива на пошук вільних місць сягає 170 тисяч літрів. Тому, впровадження системи для пошуку вільних паркомісць зекономить витрати ресурсів.

Після інтенсивного впровадження “розумних” телефонів та будинків, багато компаній впроваджують інновації за допомогою смарт-рішень для паркування автомобілів. У зв’язку із значним збільшенням автомобілей, проблема стає все більш актуальною. На сьогоднішній момент було придумано систему для пошуку вільних паркомісць як варіант вирішення даної задачі. Рішення набуває популярності та все більше і більше компаній намагаються реалізувати дану систему

**Стан проблеми.** Саме визначення системи Smart Parking означає вид типової розумної програми, смарт парковка є гарним прикладом того, як зробити інтернет речі, частиною повсякденного життя, щоб був моментальний доступ до інформації в режимі онлайн.Інтернет може бути використаний для надання водієві доступу до Smart Parking через мобільний додаток, щоб знайти та забронювати місце стоянки в будь якій зоні для парковки. Це надає можливість зробити оплату чи передоплату за місце для паркування за допомогою кредитної картки. Система Smart Parking дозволяє знайти вільні парковки, повідомити про імовірність того, що місце стоянки ще є доступним та приймає рішення про резервування та передоплату за таке місце паркування [9]. Одним із варіантів реалізації система Smart Parking є моніторинг стоянки, де кожне паркувальне місце обладнане датчиком руху,камерами для виявлення наявності або відсутності транспортних засобів з метою створення карти наявності, яка може бути використана для керування паркуванням, резервуванням та іншими послугами.

Важливим аспектом при розгортанні поняття Smart Parking є побудова оптимальної топології сенсорної бездротової мережі на парковці. Економія коштів на обладнання для сенсорних бездротових мереж дозволить привернути увагу інвесторів, власників торгових центрів, власників автостоянок до системи Smart Parking. Також важливим аспектом є зменшення об’єму роботи для мережевих інженерів, які зможуть автоматизувати процес створення топології бездротової мережі на парковці. Тобто програма , що буде розраховувати оптимальну топологію бездротової мережі буде економити не лише кошти на обладнання для мережі, а й звільнить від цієї задачі мережевих інженерів, що є безумовним плюсом. Також програма дозволятиме змінювати конфігурацію приладів для бездротової мережі а також розробляти та завантажувати до неї будь які схеми парковок, зо описані в правильному форматі (JSON чи XML). Це дозволить використовувати програму для будь яких мережевих приладів та різн их конфігурацій парковки [8].

**Постановка задачі.** Метою є створення системи, що використовуючи доступ до бази даних і засобів геолокацій, дозволить користувачеві знайти найкоротший та найзручніший шлях для паркування свого транспортного засобу.

.

**Розв’язання задачі.** Із всіх алгоритмів які існують на даний момент, було вибрано: «Алгоритм Дейкстри».  
 Алгоритм працює покроково, на кожному кроці він відвідує вершину і намагається зменшити мітки. Робота алгоритму завершається тоді, коли всі вершини були відвідані. Мітка вершини а =0, а мітки всіх інших вершин дорівнюють безкінечності, це відображає те, що відстань від а до інших вершин поки не відомі. Якщо всі вершини відвідані, то алгоритм завершається. В іншому випадку, із відвіданих вершин вибирається вершина u, яка має мінімальну мітку. Ми розглядаємо всі можливі маршрути, в яких u являє собою останній пункт.  
 Вершини, в які ведуть ребра з u, назвемо сусідами цієї вершини. Для кожного сусіда вершини u, крім позначених як відвідані, розглянемо нову довжину шляху, що дорівнює сумі значень поточної мітки u і довжини ребра, що з'єднує u з цим сусідом. Якщо отримане значення довжини менше значення мітки сусіда, замінимо значення мітки отриманим значенням довжини. Розглянувши всіх сусідів, позначимо вершину u як відвідану і повторимо крок алгоритму [12].

Застосування алгоритму Дейкстри

Стосовно нашого завдання пошуку найкоротшого шляху від точки а до точки б в якості заданої вершини a вказуємо вершину Client і шукаємо найкоротший шлях до вершини Parkinglot.Тобто робота алгоритму, закінчується тоді, коли всі вершини Parkinglot були відвідані.При запуску системи,значення Client =0, а Parkinglot = безкінечності, так як відстань від клієнта до місця паркування поки ще не відома.Якщо програма відвідає всі паркомісця, то робота алгоритму завершується, а в іншому випадку вибирається Parkinglot з найменшою міткою. В результаті застосування даного алгоритму ми отримаємо послідовність вершин графа, що позначають місця, які найкращим чином   
 підходять для найращого паркування.

Тепер продемонструємо роботу алгоритму в мові Swift.

Клас Node буде не чим іншим, як властивістю (використовується алгоритмом), щоб побачити, чи ми його вже відвідали, і масив з'єднань з іншими вузлами.

|  |
| --- |
| class Node { |
|  | var visited = false |
|  | var connections: [Connection] = [] |
|  | } |

Як видно з визначення вузла, кожне підключення присвоюється певному вузлу, тому все, що нам потрібно визначити в самому з'єднанні, це його вага (також відомий як вартість) і вузол, до якого він підключається.

Очевидно, використовувати краще мононаправлені з'єднання, таким чином легше керувати двонаправленими та однонаправленими графіками.

|  |
| --- |
| class Connection { |
|  | public let to: Node |
|  | public let weight: Int |
|  |  |
|  | public init(to node: Node, weight: Int) { |
|  | assert(weight >= 0, "weight has to be equal or greater than zero") |
|  | self.to = node |
|  | self.weight = weight |
|  | } |
|  | } |

Потрібно визначити шлях: шлях не є нічим іншим, як послідовністю вузлів.Це допоможе нам відстежити, які шляхи нашого графіка ми вже відвідали, і як ми туди потрапили. Більш важливо, що наш алгоритм поверне нам цей об'єкт для опису найкоротшого шляху між нашим вихідним вузлом і вузлом призначення.

Для цього визначення ми будемо використовувати рекурсивний спосіб:

|  |
| --- |
| class Path { |
|  | public let cumulativeWeight: Int |
|  | public let node: Node |
|  | public let previousPath: Path? |
|  |  |
|  | init(to node: Node, via connection: Connection? = nil, previousPath path: Path? = nil) { |
|  | if |
|  | let previousPath = path, |
|  | let viaConnection = connection { |
|  | self.cumulativeWeight = viaConnection.weight + previousPath.cumulativeWeight |
|  | } else { |
|  | self.cumulativeWeight = 0 |
|  | } |
|  |  |
|  | self.node = node |
|  | self.previousPath = path |
|  | } |
|  | } |

Для зручності я також додаю властивість cumulativeWeight, щоб відстежувати вартість для досягнення вузла шляху: ця вартість - це сума всіх ваг підключень, які ми пройшли від вихідного вузла до цього вузла.

По-перше, ми визначимо кордон: кордон - це набір шляхів до вузлів, які можуть досягати з вузлів, які ми відвідували до цих пір.

Спочатку вона пуста, але як тільки ми запустимо скрипт, ми додамо шлях до нашого початкового вузла

Тепер ми можемо розпочати наступні кроки алгоритму Дейкстри:

Знаходження найдешевшого вузла

Для цього ми витягуємо найдешевший шлях з нашого кордону (, перевіряємо, чи вузол ще не був відвіданий, а якщо ні, то переходимо до наступного кроку).

Позначаємо його як відвіданий і відстежуємо, на яких вузлах можна відвідати його. Як тільки ми дійшли до цього кроку, обов'язково позначимо наш вузол як відвіданий , а потім додаємо всі нові (невідомі) досяжні вузли з цього вузла, досліджуючи його з'єднання .Повторюємо.Теперішній цикл завершено, тому ми дійсно повторюємо два вищевказані кроки!

|  |
| --- |
| func shortestPath(source: Node, destination: Node) -> Path? { |
|  | var frontier: [Path] = [] { |
|  | didSet { frontier.sort { return $0.cumulativeWeight < $1.cumulativeWeight } } |
|  | } |
|  |  |
|  | frontier.append(Path(to: source)) |
|  |  |
|  | while !frontier.isEmpty { |
|  | let cheapestPathInFrontier = frontier.removeFirst() |
|  | guard !cheapestPathInFrontier.node.visited else { continue } |
|  |  |
|  | if cheapestPathInFrontier.node === destination { |
|  | return cheapestPathInFrontier |
|  | } |
|  |  |
|  | cheapestPathInFrontier.node.visited = true |
|  |  |
|  | for connection in cheapestPathInFrontier.node.connections where !connection.to.visited { |
|  | frontier.append(Path(to: connection.to, via: connection, previousPath: cheapestPathInFrontier)) |
|  | } |
|  | } |
|  | return nil |

**Висновки**. В результаті виконання роботи для пошуку вільних паркомісць було ознайомлено із загальною проблемою Smart Parking в Україні, проаналізовано елементу базу сенсорів, розроблено алгоритм для вирішення задачі знаходження оптимального паркомісця та на його основі створено програму.

Було проаналізовано всі типи датчиків руху,відеокамер, та типів систем для пошуку вільного паркомісця

**Література**

1. Amaldi, E.; Capone, A.; Malucelli, F.; Signori, F. UMTS radio planning: Optimizing base station configuration. In Proceedings of the 2002 IEEE Vehicular Technology Conference, Vancouver, BC, Canada, 24–28 September 2002; Volume 2, pp. 768– 772.
2. Bi, Y.; Sun, L.; Zhu, H.; Yan, T.; Luo, Z. A parking management system based on wireless sensor network. Acta Autom. Sin. 2006, 32, 968–977
3. Dhillon, S.S.; Chakrabarty, K. Sensor placement for effective coverage and surveillance in distributed sensor networks. In Proceedings of the 2003 IEEE Wireless Communications and Networking Conference, New Orleans, LA, USA, 16–20 March 2003; Volume 3, pp. 1609–1614.
4. Jourdan, D.B.; de Weck, O.L. Multi-Objective Genetic Algorithm for the Automated Planning of a Wireless Sensor Network to Monitor a Critical Facility. In Proceedings of the SPIE, Sensors, and Command, Control, Communications, and Intelligence (C3I) Technologies for Homeland Security and Homeland Defense III, Orlando, FL, USA, 12–16 April 2004; Volume 5403, pp. 565–575.  
   Howard, A.; Mataric,´ M.J.; Sukhatme, G.S. Mobile sensor network deployment using potential fields: A distributed, scalable solution to the area coverage problem. Distrib. Auton. Robot. Syst. 2002, 5, 299–308
5. Li, X.; Ranga, U.K. Design and implementation of a digital parking lot management system. Technol. Interface J. 2009, 10
6. Mehzer, A.; Stulman, A. The maximal covering location problem with facility placement on the entire plane. J. Reg. Sci. 1982, 22, 361–365
7. Pala, Z.; Inanç, N. Smart parking applications using RFID technology. In Proceedings of the 1 st RFID Eurasia Conference, Istanbul, Turkey, 5–6 September 2007; pp. 121– 123.