ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ ПОЛІТЕХНІЧНИЙ УНІВЕРСИТЕТ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Кафедра комп’ютерних інтелектуальних систем та мереж

Шевчук Данило Олегович

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

ПРОГРАМНА СИСТЕМА ТРАСУВАННЯ МАРШРУТІВ НА ОСНОВІ МЕТОДІВ ФРОНТУ ХВИЛІ

Напрям 6.050102 – Комп'ютерна інженерія

Спеціальності 05010201 – Комп'ютерні системи та мережі

Керівник: Защолкін Костянтин В’ячеславович,

канд. техн. наук.

Одеса – 2020

**АНОТАЦІЯ**

Шевчук Данило Олегович АМДР.АМ162.1447. Тема роботи «Програмна система трасування маршрутів на основі методів фронту хвилі». Пояснювальна записка, розміром у N сторінок, містить: N зображень, N таблиць та N джерел посилань.

Мета дипломної роботи – розробка програмного забезпечення для знаходження найкоротшого шляху при трасування маршрутів за допомогою хвильового алгоритму.

У дипломній роботі розглянуті і проаналізовані алгоритми пошуку маршрутів, обґрунтовані переваги хвильового методу та виконана реалізація ПЗ згідно алгоритму. Розроблено методику ініціалізації, поширення хвилі і відновлення шляху. Програма для методики знаходження найкоротшого шляху розроблена на мові програмування JavaScript.

Програмний продукт допоможе покращити роботу комп’ютерних інженерів усіх профілів. За допомогою даного програмного продукту можна видати найкоротший шлях з великою ефективністю за рахунок принципу зворотного трасування. А також є можливість задати різний рівень пріоритету проходження конкретних ділянок шляху.

Ключові слова: трасування, точка старту, точка цілі, напрямок обходу, перешкоди, точки пріоритету, JavaScript.

**SUMMARY**

Shevchuk Danylo Olegovich AMDR.AM162.1447. Subject of the diploma is «Routing software system based on wavefront methods». This work, size of N pages contains: N images, N tables та N source links.

The purpose of the thesis is to develop software for finding the shortest path when tracing routes using a wave algorithm.

The thesis considers and analyzes the algorithms of search in depth (English Depth-first search, DFS) and search in width (alternative name - the method of the wave front). A method of initialization, wave propagation and path recovery has been developed. The program for the method of finding the shortest path is developed in the JavaScript programming language.

The software product will help computer engineers of different profiles. With this software product, you can give the shortest path with high efficiency due to the principle of reverse tracing. Also you have a possibility of marking certain area with heir or lower priority.

Keywords: trace, start point, target point, bypass direction, obstacles, priority area, JavaScript.

# ЗМІСТ

|  |  |  |
| --- | --- | --- |
|  | Вступ | 7 |
| 1 | Аналітичний огляд за темою дипломної роботи | 10 |
| 1.1 | Постановка задачі трасування маршрутів | 10 |
| 1.2 | Область використання трасування маршрутів | 10 |
| 1.2.1 | Використання трасування маршрутів у системах проектування печатних платформ | 11 |
| 1.2.2 | Використання трасування маршрутів у комп'ютерних іграх. | 12 |
| 1.2.3 | Використання трасування маршрутів у робототехніці | 12 |
| 1.2.4 | Використання трасування маршрутів в задачах навігації | 12 |
| 1.3 | Алгоритми пошуку маршрутів | 14 |
| 1.3.1 | Алгоритм Дейкстри | 14 |
| 1.3.2 | Алгоритм Беллмана-Форда | 15 |
| 1.3.3 | Алгоритм пошуку A \* | 17 |
| 1.3.4 | Алгоритм Флойда-Уоршелла | 18 |
| 1.4 | Висновки | 18 |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ**

* 1. MOODLE (англ. *Modular Object-Oriented Dynamic Learning Environment*) – Модульна Об’єктно-Орієнтована Система Динамічного Навчання.
  2. CLR (англ. *Common Language Runtime*) – загальномовне виконуюча середа
  3. LINQ (англ. *Language-Integrated Query*) – язык добавления запросов
  4. XML (англ. *eXtensible Markup Language*) – расширяемый язык разметки
  5. ПЗ – Програмне Забезпечення

# ВСТУП

Пошук шляху (англ. Pathfinding) — це побудова найкоротшого шляху між двома точками за допомогою комп'ютерної програми. Це практичніший варіант розв'язування лабіринтів. Ця галузь досліджень ґрунтується на алгоритмі Дейкстри для пошуку найкоротшого шляху на зваженому графі.

Задача пошуку шляху тісно пов'язана з задачею про найкоротший шлях у рамках теорії графів, яка розглядає визначення шляху, що найкраще відповідає деяким критеріям (найкоротший, найдешевший, найшвидший і так далі) між двома точками у великій мережі.

Хвильовий алгоритм - категорія алгоритмів, які застосовуються у випадках де потрібно знайти мінімальний шлях у аланарному графі, тобто двомірній матриці, із однаковою довжиною ребр. Ця категорія алгоритмів була заснована завдяки алгоритму пошуку в ширину.  
Алгоритм пошуку в ширину було створено для пошуку вершини графа. Якщо задано початкову вершину графа s, алгоритм пошуку в ширину систематично обходить всі досяжні із s вершини. Перший крок: вершина s позначається як пройдена, до масиву додаються всі сусідні вершини, які досяжні з вершини s без проходження проміжних вершин. Кожен наступний крок всі поточні вершини масиву відмічаються, як пройдені, а новий масив формується із сусідніх вершин, які ще не були відмічені пройденими сусідами поточних вершин графа. Елементи масиву оперуються завдяки черзі та принципу FIFO. Доки не досягнена вершина s, чи коли в масив не було включено кожну вершину, виконання алгоритму продовжується. Якщо алгоритм було запинено через другий варіант, це означає, що всі вершини, було пройдено, а шлях до цільової вершини не знайдений, тобто неможливо дійти до цільової вершини.

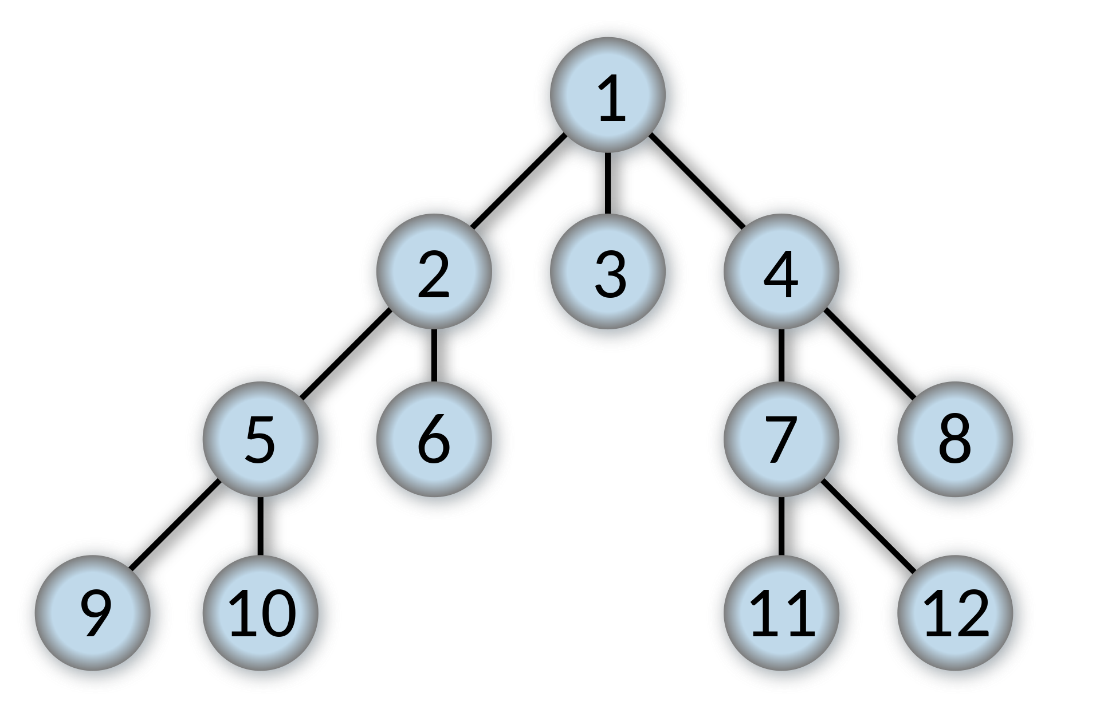


Рис. 1 - Алгоритм пошуку в ширину

Хвильовий алгоритм оперує елементи графу подібно алгоритму пошуку в ширину, але при цьому допустимо що елементи графу можуть бути перешкодами, тобто шлях не може проходити через цей елемент. У цьому випадку шлях повинен поширюватися в усіх напрямках, окрім тих, де наступний елемент є перешкодою.

Хвильові алгоритми прийнято поділяти за двома типами поширювання: ортоганальний та діагонально-ортогональний. Кожен крок у першому із цих двох повинен поширювати хвилю тільки у елементи які мають повну сусідню межу, тобто різниця координат цих елементів дорівнює одиницею тільки у одній осі координат, наприклад сусідній елемент до елементу із координатами (1, 1) (х та у відповідно) у цьому випадку є (0, 1), (1, 0), (2,1) та (1, 2). Приклад на рисунку 2.

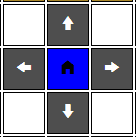


Рис. 2 – Ортогональні сусіди  
Діагонально-ортогональний тип поширювання, у свою чергу не зобов’язує повного контакту елементів, та може поширюватися до елементів модуль різниць координат яких у сумі дає 2, тобто до елементу (1, 1) (х та у відповідно) сусідніми елементами у такому випадку є елементи із координатами (0, 1), (0, 0), (1, 0), (0, 2), (2, 0), (2, 2), (2,1) та (1, 2). Приклад на рисунку 3.

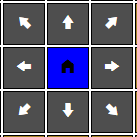


Рис. 3 – Діагонально-ортогональні сусіди  
У більшості випадків шлях, який було створено завдяки діагонально- ортогональному типу поширення є коротший, за той, який би можна було отримати при застосуванні ортогонального методу. У деяких випадках, коли неможливо знайти шлях завдяки ортогональному типу, застосуванні діагонально-ортогональному типу поширення хвилі, шлях може бути знайдено. Наприклад випадок який зображено на рисунку НОМЕР.  
 У цьому випадку усі ортогональні сусіди цільового елементу, або елементу “finish”, э перешкодами, або “wall”, але із сусідніх діагонально-ортогональних елементів є порожні елементи, тобто шлях може пролягати через ці елементи, що дозволяє хвилі поширюватися через них до цільового елементу.

1. АНАЛІТИЧНИЙ ОГЛЯД ЗА ТЕМОЮ ДИПЛОМНОЇ РОБОТИ

У даному розділі розглянуто математичний опис завдання пошуку маршрутів.

* 1. Постановка задачі трасування маршрутів

Назва алгоритму – хвильовий – пов’язана з тим, що визначення індексів k вершин графу G відбувається, як розповсюдження з початкової вершини v певної хвилі, яка спрямовується за напрямком дуг. Коли хвиля дійде до кінцевої вершини u, це буде означати, що алгоритм закінчив свою роботу. Значення індексу, „принесеного хвилею”, у вершині u буде відповідати довжині знайденого маршруту. А для того, щоб визначити цей маршрут (послідовність вершин), потрібно з кінцевої вершини u повертатися в зворотному до розповсюдження хвилі напрямку і відзначати послідовно одну довільну вершину зі значеннями індексу k–1, k–2, …, 1, 0. Зрозуміло, що вершина з індексом 0, – це початкова вершина v. Вершини u1, u2, …, uk–1, взагалі, можуть бути визначені неоднозначно. Ця неоднозначність відповідає випадкам, коли існує кілька різних мінімальних шляхів з v до u в орграфі G. Для процесу аналізу тестування впроваджуються різні методи.

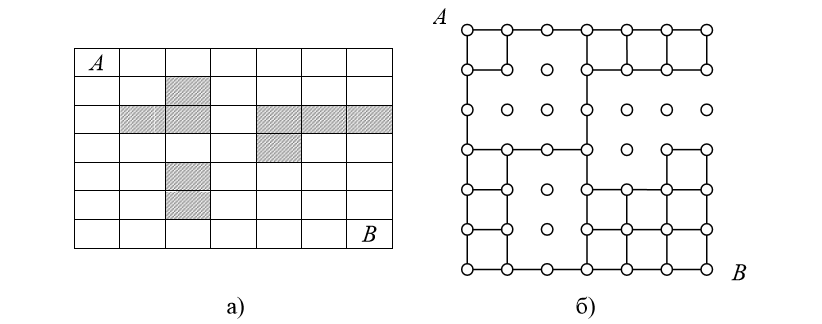
Наприклад, визначимо мінімальний шлях з v1 до v6 в орієнтованому графі G, заданому матрицею суміжності (відповідний граф представлено на рис. 2)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | *v*1 | *v*2 | *v*3 | *v*4 | *v*5 | *v*6 |
| *v*1 | 0 | 0 | 0 | 1 | 1 | 0 |
| *v*2 | 1 | 0 | 0 | 0 | 0 | 1 |
| *v*3 | 0 | 1 | 0 | 0 | 0 | 1 |
| *v*4 | 0 | 1 | 0 | 0 | 1 | 0 |
| *v*5 | 1 | 0 | 1 | 0 | 0 | 0 |
| *v*6 | 0 | 0 | 1 | 0 | 1 | 0 |

## 

## Рис. 2 - Приклад роботи хвильового алгоритму.

Діючи згідно з хвильовим алгоритмом, послідовно знаходимо F1(v1) = {v4, v5}; F2(v1) = D(F1(v1)) \ {v1, v4, v5} = {v2, v3}; F3(v1) = D(F2(v1)) \ {v1, v2, v3, v4, v5} = {v6}. Таким чином, v6 ∈ F3(v1), а, отже, за пунктом 3 існує шлях з v1 до v6 завдовжки 3, і цей шлях є мінімальним. Знайдемо тепер мінімальний шлях із v1 до v6. Визначимо множину F2(v1) ∩ D -1(v6) = {v2, v3} ∩ {v2, v3} = {v2, v3}. Виберемо будь-яку вершину зі знайденої множини, наприклад, v3. Визначимо далі множину F1(v1) ∩ D -1(v3) = {v4, v5} ∩ {v4, v5, v6} = {v4, v5}. Виберемо будь-яку вершину зі знайденої множини, наприклад, v5. Тоді (v1, v5, v3, v6) – шуканий мінімальний шлях з v1 до v6 в орієнтованому графі G, а відстань між v1 та v6 дорівнює 3. Очевидно, хвильовий алгоритм може застосовуватися не тільки для орієнтованих, а й для неорієнтованих графів. В останньому випадку, пересування з однієї вершини до іншої можливі в обидві сторони. Хвильовий алгоритм широко застосовується у розробці комп’ютерних ігор – коли необхідно визначити оптимальний маршрут пересування гравця або певного „юніта” з однієї точки віртуальної місцевості (карти) до іншої. Наведемо приклад такої задачі. Нехай потрібно знайти найкоротший маршрут з точки А до точки В на карті, яка зображена на рис. 3, а. На ній заштриховані комірки відповідають певним перепонам на шляху, тобто в цих частинах місцевості гравець не зможе пройти. Також будемо вважати, що гравець може пересуватись тільки по вертикалі та горизонталі. Відповідний цієї карті граф представлено на рис. 3, б.



## Рис. 3 - Приклад карти території та відповідного графу.

## Після роботи хвильового алгоритму отримаємо наступні індекси вершин – комірок карти (рис. 4, а). На рис. 4, б зображено два зі знайдених маршрутів з вершини А у вершину В. Як можна побачити, довжина знайденого маршруту дорівнює 12.

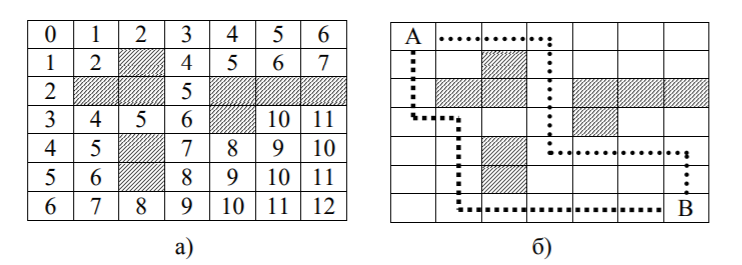
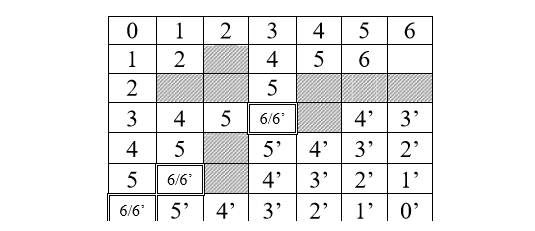


Рис. 4 - Робота хвильового алгоритму для графу на рис. 3

Можна зробити процес пошуку найкоротшого маршруту за допомогою хвильового алгоритму більш економнішим, а відтак, й більш швидким. Для цього будемо розповсюджувати хвилю не тільки з початкової вершини А (перша хвиля), а й з кінцевої вершини В (друга хвиля). Для того, щоб відрізняти індекси першої хвилі від другої, індекси останньої будемо позначати зі штрихом. Робота модифікованого алгоритму закінчується коли обидві хвилі зустрінуться (рис. 5).

Рис. 5 - Двонаправлений хвильовий алгоритм

## Комірки, де дві хвилі зустрічаються, позначені подвійними лініями. З порівняння рис. 30.4 та 30.5 видно, що знайдені найкоротші маршрути співпадають. І хоча в цьому прикладі економія склала всього лиш одну комірку (яка не була відмічена), можна зрозуміти, що на більш складних, тобто насичених „перепонами”, картах робота модифікованого хвильового алгоритму буде більш ефективнішою за простий хвильовий алгоритм.

* 1. Область використання трасування маршрутів

У даному підрозділі розглянуті області використання трасування маршрутів.

* + 1. Використання трасування маршрутів у системах проектування печатних платформ

Трасування друкованих плат — один з етапів проектування радіоелектронної апаратури, що полягає в покроковому проектуванні структури провідників друкованої плати вручну або з використанням однієї з САПР друкованих плат.

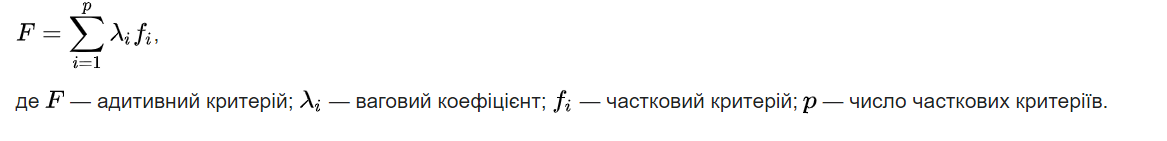
Трасування з'єднань є, як правило, заключним етапом конструкторського проєктування РЕА і полягає у визначенні топології та геометрії провідників, що з'єднують контакти елементів проєктованого пристрою.

Задача трасування є складною комбінаторною задачею — однією з найбільш трудомістких в загальній проблемі автоматизації проєктування РЕА. Це пов'язано з кількома факторами, зокрема з різноманіттям способів конструктивно-технологічної реалізації з'єднань, для кожного з яких при алгоритмічному вирішенні задачі застосовуються специфічні критерії оптимізації і обмеження. З математичної точки зору задача трасування — оптимізаційна задача, що полягає у виборі раціонального рішення з величезної кількості можливих варіантів рішень.

Кількість можливих варіантів рішень в задачах трасування така, що навіть для відносно нескладних систем вирішення задачі трасування шляхом перебору всіх можливих варіантів з'єднань неможлива. Саме тому представляють інтерес методи трасування, які відшукують не найкращий варіант трасування (що потребувало б гарантованого перебору та оцінки всіх варіантів), а раціональний варіант, який незначно поступається оптимальному, але може бути знайдений за відносно невеликий час.

Основна задача трасування формулюється в такий спосіб: за заданою схемою з'єднань прокласти необхідні провідники на площині (платі, кристалі і т.п.) таким чином, щоб досягти зазначеного критерію якості трасування з урахуванням заздалегідь заданих обмежень. Основними є обмеження на ширину провідників і мінімальні відстані між ними.

Вихідною інформацією для рішення задачі трасування є список ланцюгів, параметри конструкції елементів і комутаційного поля, а також дані про розміщення елементів. Представлення цільової функції часто виконують в адитивній або мультиплікативній форму функції оцінки, наприклад такого вигляду:



Хвильові алгоритми, засновані на ідеях Лі та розроблені Ю. Л. Зіманом і Г. Г. Рябовим. Дані алгоритми набули широкого поширення в існуючих САПР, оскільки вони дозволяють легко враховувати технологічну специфіку друкованого монтажу зі своєю сукупністю конструктивних обмежень. Ці алгоритми завжди гарантують побудова траси, якщо шлях для неї існує.

* + 1. Використання трасування маршрутів у комп'ютерних іграх.

Пошук шляху в комп'ютерних іграх це одна з проблем ігрового штучного інтелекту. Для її вирішення досліджуються та використовуються різні алгоритми пошуку найкоротшого шляху.

Невід'ємною частиною сучасних комп'ютерних ігор є наявність ігрового штучного інтелекту (ШІ). Хоча підхід до ігрового ШІ серйозно відрізняється від підходу до традиційного ШІ без нього обійтися вже важко. Одним з яскравих прикладів ігрового штучного інтелекту є створення ігрового персонажу – програми-робота, що імітує партнерів в грі, в мережевих поєдинках, командних боях і т. п. Програма створення ігрового персонажу заснована на модулі штучного інтелекту, який адаптований до особливостей гри: мапі, правил, а також до типу гри. Незалежно від типу штучного персонажу та його призначення первинною задачею є його переміщення. Для руху по відкритій території, наприклад, з точки А в точку Б можна дістатися по прямій лінії. Але якщо рух відбувається в приміщенні з кімнатами, або виникають перешкоди, то потрібно вже більш складний підхід. У найпростішому випадку приміщення має прямокутну форму і розбито на квадратні клітини (комірки) одного розміру, сторони яких паралельні координатним осям. Всі клітини поділяються на прохідні – білі і непрохідні (стіни, перешкоди) – чорні. З кожної клітини є можливість переміститися тільки в сусідні клітини лише по горизонталі або по вертикалі. Зазвичай приміщення складається з залів (кімнат), які з'єднуються коридорами. У залах можуть перебувати перешкоди, які необхідно обходити. Припустимо, що в кожному прохідному залі побудовані найкоротші внутрішні траси, що з'єднують різні входи. Побудовані траси – сегменти геометричного графа: його вершини відповідають точкам входу, а ребра - побудованим трасам. Даний граф, по суті, є дорожньою картою (roadmap). Далі вирішується задача пошуку найкоротших маршрутів по дорожній карті.

До найбільш популярних алгоритмів пошуку маршруту в графі можна віднести: − Пошук в ширину (англ. Breadth-First Search, BFS) дозволяє обчислити найкоротші відстані (в термінах кількості ребер) від виділеної вершини орієнтованого графа до всіх інших вершин, і / або побудувати кореневе спрямоване дерево, відстані в якому збігаються з відстанями в вихідному графі . Крім того, пошук в ширину дозволяє вирішувати задачу перевірки досяжності (чи існують шляхи між вершиною джерелом і іншими вершинами графа).

* + 1. Використання трасування маршрутів у робототехніці

Одна з основних функцій робота полягає у виконанні руху в задану точку в залежності від геометричних форм перешкод. Однак не завжди трасування ставить перед собою мету знайти найкоротший шлях; найчастіше це просто неможливо. Програмування даної функції впирається в проблему, яка зводиться до алгоритму обходу перешкод . В основу програм трасування можуть бути покладені алгоритми:

- з дискретним поданням зони руху;

- засновані на теорії графів;

- засновані на методах потенційних полів;

- засновані на евристичних моделях;

- з поданням інформації у вигляді рівнянь.

Розглянемо найпростішу задачу трасування: обхід двовимірних лабіринтів по заданій цифровий (растрової) карті місцевості у вигляді двовимірної площині, розділеної на клітини (квадратні, трикутні або гексагональних), з зазначеними точкою старту (точки, з якої ми повинні провести маршрут), точкою цілі (точка, в яку ми повинні провести маршрут) і перешкодами різної геометричної форми. Спочатку необхідно вибрати стратегію обходу перешкоди. Ми можемо піти двома шляхами: перший - прокладати шлях «на ходу», ігноруючи перешкоди до зіткнення з ними; другий - заздалегідь спланувати шлях до початку переміщення. Найбільш прості алгоритми засновані на припущенні, що перешкоди можна «торкатися рукою» і слідувати його контуру, поки вона не буде обійдена. Спочатку вибираємо напрям для руху до мети. рухаємося до зіткнення з перешкодою. При зіткненні вибираємо інший напрямок в Відповідно до стратегії обходу, наприклад: - переміщення у випадковому напрямку. Робимо невеликий зсув в випадковому напрямку при зустрічі перешкоди;

- трасування навколо перешкоди. При зустрічі перешкоди йдемо по його контуру до певного моменту, що визначається евристикою (наприклад кількість переміщень в одному напрямку; зупинити обхід перешкоди при можливості пересування в напрямку, який був бажаним на початку трасування).

При реалізації багатьох алгоритмів на графах виникає необхідність організувати систематичний перебір вершин графа, при якому кожна вершина проглядається точно один раз. Такий перебір можна організувати двома способами: пошуком в глибину або пошуком в ширину.

* + 1. Використання трасування маршрутів в задачах навігації

Алгоритми знаходження найкоротшого шляху на графі застосовуються для знаходження шляхів між фізичними об'єктами на таких картографічних сервісах, як карти Google або OpenStreetMap.

Оптимізація маршруту по мережі доріг є вирішеним завданням та широко використовується в сучасних навігаційних системах. Проте, це завдання істотно відрізняється від завдання прокладення оптимального маршруту в умовах бездоріжжя, оскільки припускає ряд обмежень. При проектуванні мереж з динамічно змінюваною топологією це завдання набуває особливої актуальності при відсутності навігаційних (векторних) карт. Використання методів варіаційного числення при рішенні поставленої проблеми пов’язано зі значними математичними труднощами, що обумовлює вибір дискретних методів теорії графів і дослідження операцій як альтернативу континуальним методам. Один з найефективніших рішень задачі трасування маршрутів руху мобільних вузлів мережі є пошук оптимального шляху на основі алгоритму методів фронту хвилі.

* 1. Алгоритми пошуку маршрутів

У даному підрозділі детально описано 4 найбільш використовуваних методів трасування маршрутів

* + 1. Алгоритм Дейкстри

Алгоритм Дейкстри знаходить найкоротший шлях від однієї з вершин графа до всіх інших. Алгоритм працює тільки для графів без ребер з негативною вагою;

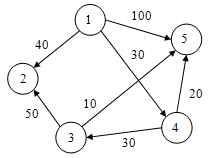
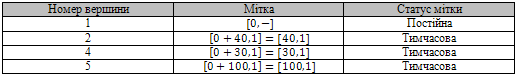
Розглянемо деякий орієнтований граф, для якого потрібно знайти найкоротший маршрут від вершини 1 до всіх інших вершин. Для розв'язку задач такого типу доцільно використовувати алгоритм Дейкстри.

Рис. 6.1 - Приклад роботи алгоритму Дейкстри

Слідуючи алгоритму, початковій вершині (вершина під номером 1) присвоюємо постійну мітку Алгоритм Дейкстри після чого переходимо до першого етапу.

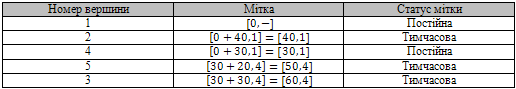
Етап 1: з вершини 1 існує орієнтоване ребро до вершин 2, 4 і 5. Обчислимо для даних вершин відповідні мітки. В результаті отримаємо наступну таблицю.



Таблиця. 1.1 - Приклад роботи алгоритму Дейкстри

Серед вершин з тимчасовими мітками, вибиремо ту, значення відстані для якої є найменшим. В нашому випадку такою є вершина 4, з відстанню algoritm\_dejkstru\_pruklad4. Міняємо статус даної вершини на «постійна» і переходимо до другого етапу.

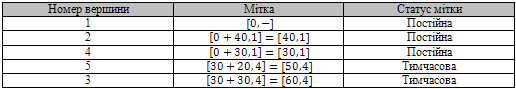
Етап 2: з четвертої вершини існує орієнтоване ребро до вершин 3 і 5. Обчислюємо для них мітки, після чого таблиця набуде наступного вигляду.



Таблиця. 1.2 - Приклад роботи алгоритму Дейкстри

В даній таблиці, тимчасову мітку [100, 1] для вершини 5, отриману на попередньому етапі, замінено на нову [50, 4], також тимчасову мітку. Це говорить про те, що до вершини 5 знайдено більш короткий маршрут, який проходить через вершину 4. Пісял чого, аналогічно першому етапу, вибираємо вершину значення відстані для якої є найменшим (вершина під номером 2) і міняємо її статус на постійну.

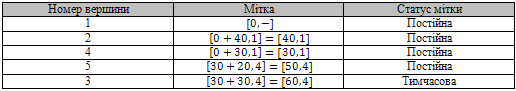
Етап 3: від вершини 2 не можливо потрапити до інших вершин графа, тому на даному етапі ми отримаємо аналогічну другому етапу таблицю, тільки статус вершини 2 змінено на постійна.



Таблиця. 1.3 - Приклад роботи алгоритму Дейкстри

Серед вершин, статус мітки яких не дорінює постійна, вибираємо ту, для якої значення відстані Алгоритм Дейкстри є найменшим. Тобто вершину 5, для якої Алгоритм Дейкстри. Міняємо її статус на постійну і переходимо до четвертого етапу.

Етап 4: з вершини 5 не існує орієнтованих ребер до інших вершин графа, тому таблиця міток залишається незмінною, тільки статус вершини 5 замінено на постійну.



Таблиця. 1.4 - Приклад роботи алгоритму Дейкстри

На четвертому етапі, ми отримали таблицю, в якій з тимчасовою міткою залишилась тільки вершина 3. І виходячи з того, що з даної вершини можна потрапити у вершину 2 або 5, статус яких становить — постійна, то на цьому процес обчислень за алгоритмом Дейкстри закінчується.

Таким чином ми отримали маршрут найкоротшої довжини, який починається у вершині 1 і обходить всі вершини заданого графа.

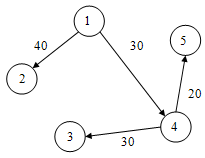


Рис. 6.2 - Приклад роботи алгоритму Дейкстри

* + 1. Алгоритм Беллмана-Форда

Алгоритм Беллмана-Форда знаходить найкоротші шляхи від однієї вершини графа до всіх інших у зваженому графі. Вага ребер може бути негативною;

Запишемо алгоритм Беллмана-Форда більш детально. Для цього, розглянемо деякий орієнтований граф *G* із зваженими ребрами, який не містить циклів від'ємної довжини. І припустимо, що потрібно знайти найкоротші шляхи від виділеної вершини  *a* до всіх інших вершин даного графа:

Перед початком виконання алгорітму всі вершини графа вважаються непройденими, а ребра — не переглянутими. Кожній вершині в ході виконання алгоритму присвоюється число *dx* , рівне довжині найкоротшого шляху з вершини *a* у вершину *x*, що включає тільки пройдені вершини. На першому кроці покладаємо http://www.mathros.net.ua/wp-content/uploads/2019/02/algorytm_bellmana-forda12.gif і http://www.mathros.net.ua/wp-content/uploads/2019/02/algorytm_bellmana-forda13.gif для сіх *x*, відмінних від *a*. Також, на даному кроці, вершині *a* присвоюється мітка «пройдена» і покладається http://www.mathros.net.ua/wp-content/uploads/2019/02/algorytm_bellmana-forda14.gif ( *y* — остання з пройдених вершин).

Для кожної з вершин графа *G* наступним чином перерахувати величину http://www.mathros.net.ua/wp-content/uploads/2019/02/algorytm_bellmana-forda16.gif : (де http://www.mathros.net.ua/wp-content/uploads/2019/02/algorytm_bellmana-forda17.gif - вага ребра http://www.mathros.net.ua/wp-content/uploads/2019/02/algorytm_bellmana-forda18.gif). Якщо http://www.mathros.net.ua/wp-content/uploads/2019/02/algorytm_bellmana-forda13.gif для всіх непройдених вершин *x*, процедуру алгоритму Беллмана-Форда необхідно звершити (у вихідному графі відсутні шляхи з вершини у непройдені вершини *a*). В іншому випадку, мітку «пройдена» необхідно присвоїти тій з вершин *x*, для якої величина *dx* є найменшою. Крім того, ребро, що веде в обрану на даному етапі вершину *x* вважається переглянутим (для цієї дуги досягався мінімум відповідно до виразу (1)). Після цього, поклавши http://www.mathros.net.ua/wp-content/uploads/2019/02/algorytm_bellmana-forda19.gif , ітераційний процес продовжується далі. Відмітимо, що якщо для деякої пройденої вершини *x* відбувається зменшення величини *dx*, то з цієї вершини і инцидентного їй переглянутого ребра відповідні мітки знімаються.

Процедура алгоритму Беллмана-Форда завершується тільки тоді, коли всі вершини графа *G* пройдені і коли після чергового виконання кроку номер два жодне з чисел *dx* не змінило свого значення.

# 

* + 1. Алгоритм пошуку A \*

Алгоритм пошуку A \* знаходить маршрут з найменшою вартістю від однієї вершини (початкової) до іншої (цільової, кінцевої), використовуючи алгоритм пошуку по першому найкращому збігу на графі;

Алгоритм використовує допоміжну функцію (евристику), аби скеровувати напрям пошуку та скорочувати його тривалість. Алгоритм повний в тому сенсі, що завжди знаходить оптимальний розв'язок, якщо він існує.

Алгоритм А\* спершу відвідує ті вершини, які ймовірно ведуть до найкоротшого шляху до мети. Аби розпізнати такі вершини, кожній відомій вершині *x* співставляється значення *f(x)*, яке дорівнює довжині найкоротшого шляху від початкової вершини до кінцевої, який пролягає через обрану вершину. Вершини з найменшим значенням *f* обираються в першу чергу.

Функція *f(x)* для вершини *x* визначається так:

*f(x)=g(x)+h(x)*

де:

*g(x)* функція, значення якої дорівнюють вартості шляху від початкової вершини до *x*,

*h(x)* евристична функція, оцінює вартість шляху від вершини *x* до кінцевої.

Використана евристика не повинна давати завищену оцінку вартості шляху. Прикладом оцінки може служити пряма лінія: загальний шлях не може бути коротшим за пряму лінію.

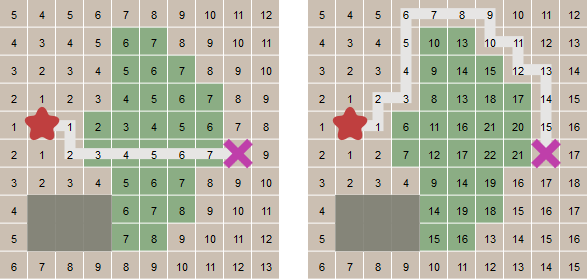


Рис. 7 - Приклад роботи алгоритму пошуку A \*

Алгоритм ділить вершини на три класи:

*невідомі вершини*: ці вершини ще не були знайдені. Ще не відомий шлях до них. На початку роботи алгоритму всі вершини, окрім початкової, належать до класу невідомих.

*відомі вершини* (OpenList): вже відомий (можливо не оптимальний) шлях до цих вершин. Всі відомі вершини разом зі значеннями *f* зберігаються в списку. З цього списку вибираються, в першу чергу, найперспективніші вершини. Конкретна реалізація цього списку має суттєвий вплив на швидкодію алгоритму, і зазвичай має вигляд черги з пріоритетом (наприклад, бінарна купа). На початку роботи алгоритму до відомих вершин належить лише початкова вершина.

*повністю досліджені* вершини (ClosedList): до цих вершин вже відомий найкоротший шлях. Повністю досліджені вершини додаються до так званого замкненого списку, аби запобігти багаторазовому дослідженню вже досліджених вершин. Список повністю досліджених вершин на початку роботи алгоритму порожній.

Кожна відома або повністю досліджена вершина має вказівник на попередні вершини. Завдяки цьому вказівникові, можна пройти шляхом від цієї до початкової вершини.

Коли вершину *x* буде повністю досліджено (або розкрито, релаксована), суміжні з нею вершини додаються до списку відомих вершин, а сама вершина додається в список повністю досліджених. Вказівники на попередню вершину встановлюються на *x*. Суміжні вершини, які вже знаходяться в списку повністю досліджених вершин, до списку відомих не додаються, а зворотні вказівники не змінюються. Суміжні вершини, які вже знаходяться в списку відомих, лише оновлюються (значення *f* та вказівник на попередню вершину), якщо знайдений до них шлях коротший за вже відомий.

Алгоритм зупиняється коли кінцева вершина потрапляє до списку повністю досліджених вершин. Знайдений шлях відтворюється за допомогою вказівників на попередню вершину. Якщо список відомих вершин порожніє, то розв'язку задачі не існує і алгоритм припиняє пошук.

Відтворений за зворотніми вказівниками знайдений шлях починається з кінцевої вершини та прямує до початкової. Аби одразу отримати шлях в правильному напрямі, з початкової вершини до кінцевої, в умовах задачі слід переставити місцями початок та кінець. Якщо шукати шлях починаючи з кінцевої вершини, відтворений список починатиметься з початкової вершини й прямуватиме до кінцевої.

* + 1. Алгоритм Флойда-Уоршелла

Алгоритм Воршелла порівнює всі можливі шляхи в графі між кожною парою вершин. Він виконується за Θ(|V|³) порівнянь. Це доволі примітивно, враховуючи, що в графі може бути до Ω(|V |²) ребер, і кожну комбінацію буде перевірено. Він виконує це шляхом поступового поліпшення оцінки по найкоротшому шляху між двома вершинами, поки оцінка не стає оптимальною.

Розгляньмо граф G з ребрами V, пронумерованими від 1 до N. Крім того розгляньмо функцію shortestPath(i, j, k), яка повертає найкоротший шлях від i до j, використовуючи вершини з множини {1,2,…,k} як внутрішні у шляху. Тепер, маючи таку функцію, нам потрібно знайти найкоротший шлях від кожного i до кожного j, використовуючи тільки вершини від 1 до k + 1.

Для кожної з цих пар вершин, найкоротший шлях може бути або (1)- шлях, у якому є тільки вершини з множини {1, …, k}, або (2)- шлях, який проходить від i до k + 1 а потім відk + 1 до j.Найкоротший шлях від i to j that only uses vertices 1 через k визначається функцією shortestPath(i, j, k), і якщо є коротший шлях відi до (k + 1 до j), то довжина цього шляху буде сумою(конкатенацією) найкоротшого шляху відi до k + 1 (використовуючи вершини{1, …, k}) і найкоротший шлях від k + 1 до j (також використовуючи вершини з {1, …, k}).

w(i,j) — це вага ребра між i та j. Можна визначити

shortestPath(i, j, k + 1)

наступною рекурсивною формулою база:

shortestPath(i,j,0)=w(i,j)

рекурсивна частина:

shortestPath(i,j,k+1)=min(shortestPath(i,j,k),shortestPath(i,k+1,k)

+shortestPath(k+1,j,k))

Ця формула є основною частиною алгоритму Флойда-Воршелла. Алгоритм спочатку обчислює shortestPath(i, j, k) для всіх пар(i, j) де k = 1, потім k = 2, і т. д. Цей процес продовжується, поки k = N, і поки не знайдено всі найкоротші шляхи для пар (i, j).

Алгоритм виконується на рисунку нижче:

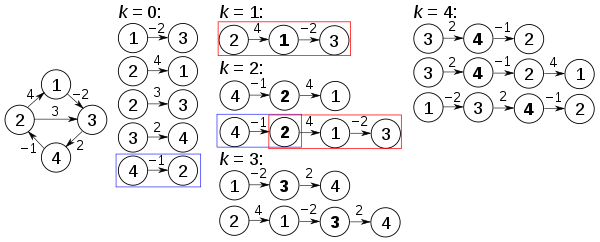


Рис. 8 - Приклад роботи алгоритму Флойда-Уоршелла

Перед першою ітерацією циклу k=0 і відомі шляхи відповідають одиночним ребрам у графі. Коли k=1, знайдено шляхи, яку проходять через вершину 1, зокрема: шлях 2→1→3 , замінить шлях 2→3, що проходить через меншу кількість ребер, але є довшим. При k=2, знаходяться шляхи, що проходять через вершини {1,2} . Червоні та блакитні квадратики показують, як шлях 4→2→1→3 складається з 4→2 і 2→1→3, визначеними на попередніх ітераціях. Шлях 4→2→3 не розглядається, бо 2→1→3 поки що найкоротший шлях. При k=3,знаходяться шляхи, що проходять через {1,2,3}. Нарешті, при k=4, знайдено всі найкоротші шляхи.

* 1. Висновки

В даному розділі дипломної роботи було описано математичну постановку задачі трасування маршрутів, визначена область використання теми:

у системах проектування печатних платформ;

комп'ютерних іграх;

у робототехніці;

навігаційних задачах.

Детально описані найпоширеніші алгоритми пошуку найближчої відстані з прикладами та ілюстраціями конкретних задач. Ці алгоритми мають як переваги так і недоліки, але темою моєї дипломної роботи було вибрано саме алгоритм трасування маршрутів на основі методів фронту хвилі. Оскільки, на моє глибоке переконання, саме цей алгоритм – найбільш оптимізований під цільові призначення.

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ (плейсхолдер)

1. Зорин С. Ф. Разработка автоматизированной системы контроля знаний./ С.Ф. Зорин – М.:2007. – 36 с.

2. Тертышная Т. И. Автоматизированная система контроля знаний / Т. И. Тертышная, Е. В. Колесникова, В. Д. Гогунский – Одесса: Тр. Одес. политехн. ун-та., 2001. – Вып. 1(13). – С. 125 – 128.

3. Челышкова М. Б., Теория и практика конструирования тестов: Уч. пособие / М. Б. Челышкова – М: Логос, 2002. – 432 с.

4. Майоров, А. Н. Теория и практика создания тестов для системы образования: Как выбирать, создавать и использовать тесты для целей образования./ А. Н. Майоров – М: Интеллект-Центр, 2002. – 296 с.

5. Линда Крокер, Джеймс Алгина. Введение в классическую и соврменную теорию тестов. / Линда Крокер – М.:Логос, 2010. – 668

6. http://uss.dvfu.ru/static/kim\_testing\_monograph/src/glava\_3\_8.html

7. mypsychologysite.ru/index/nadezhnost\_testa\_vidy\_i\_sposoby\_ opredelenija\_ nadez hnosti\_oshibka\_izmerenija/0-79

8. http://med-books.info/nadjnost-metodom-rasschepleniya.html

9. http://med-books.info/nadjnost-otdelnogo-zadaniya.html

10. Русинова Н.В., Методика проведения анализа тестов по результатам тестирования. / Н. В. Русинова, Т. В. Онищенко, В. А. Крисилов – Одесса: Тр. Одес. политехн. ун-та., 2007. – Вып. 1(13). – С. 125 –128.

11. Ватутин В.А. Теория вероятностей и математическая статистика в задачах: Учеб. посо-бие для вузов / В. А. Ватутин, Г. И. Ивченко, Ю. И. Медведев, В. П. Чистяков— 2-е изд., испр. – М.: Дрофа, 2003.– 328 с.